EXTENDED CLUSTER WEIGHTED MODELING METHODS FOR TRANSIENT RECOGNITION CONTROL

by

Tao Zhu

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

June 2006

APPROVAL

of a dissertation submitted by

Tao Zhu

This dissertation has been read by each member of the dissertation committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the Division of Graduate Education.

Dr. Steven R. Shaw

Approved for the Department of Electrical and Computer Engineering

Dr. James Peterson

Approved for the Division of Graduate Education

Dr. Joseph J. Fedock

STATEMENT OF PERMISSION TO USE

In presenting this dissertation in partial fulfillment of the requirements for a doctoral degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library. I further agree that copying of this dissertation is allowable only for scholarly purposes, consistent with fair use as prescribed in the U.S. Copyright Law. Requests for extensive copying or reproduction of this dissertation should be referred to ProQuest Information and Learning, 300 North Zeeb Road, Ann Arbor, Michigan 48106, to whom I have granted the exclusive right to reproduce and distribute my dissertation in and from microform along with the non-exclusive right to reproduce and distribute my abstract in any format in whole or in part.

Tao Zhu

June 2006

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. Steven R. Shaw for his kind advice and supervision throughout this process. I am highly thankful for the time he spent in this project and his understanding.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

LIST OF FIGURES – CONTINUED

LIST OF FIGURES – CONTINUED

ABSTRACT

This dissertation considers cluster weighted modeling (CWM), a novel nonlinear modeling technique for the electric load transient recognition problem. The original version of CWM is extended with a new training algorithm and a real-time CWM prediction method.

In the training process, a new training algorithm is derived that is a mixture of expectation maximization (EM) – least mean squares (LMS). The algorithm addresses the singular matrix inversion problem in EM. A recursive EM-LMS algorithm is developed that allows the CWM to adapt to time varying systems.

In the prediction process, a sequential version of CWM prediction based on the novel idea of tail prediction is proposed to improve the real-time performance of load transient recognition. This idea also gives rise to a real-time transient overlapping resolution method that is critical for robust online operation. Other aspects of real-time transient processing methods, such as transient scaling, detection under conditions of transient overlap, and off-training set transient indication are also developed and combined into the sequential CWM model.

The sequential CWM technique is applied to an electric load transient recognition model for hybrid fuel cell system control. The model provides real-time information about the steady-state behavior of incoming load transients to control and allocate power between the fast sources and the fuel cell. An implementation of this system in a Xilinx FPGA is discussed.

1

INTRODUCTION

Systems combining fuel cells with fast energy storage devices are good solutions to the fuel cell load-following problem that may also offer efficiency and reliability advantages. Existing power control schemes for hybrid fuel cell systems do not always optimize the power control between the fuel cell and the fast sources during the load transient periods. These approaches assume constant power output for the fuel cell and treat the fuel cell as simple as a battery charger [25, 28, 38, 24], or passively determine the power allocation depending on the dynamics of the fuel cell and the fast sources, and the interactions between the load transients and hybrid power system [3].

The transient performance of a hybrid fuel cell system can be improved if the control system can determine the future behavior of the transient. Motivated by a load transient recognition model in the Non-Intrusive Load Monitor (NILM) [49], the load transient steady state behavior can be predicted by recognizing the transient current waveform using a library of recorded transient "finger prints". This dissertation proposes a new load transient recognition model using a sequential modification of cluster weighted modeling (CWM) in [17, 18, 48]. The new model differs from NILM in that the identification of the transient behavior happens quickly, on a time-scale shorter than the transient itself.

Work and Contributions

Cluster-Weighted Modeling (CWM) was improved with newly developed training algorithms that are a mixture of Expectation Maximization (EM) – Least Mean Square (LMS) algorithms to replace the old (EM) algorithm. The new training algorithm enhances the numerical stability of CWM during the training process. It also gives the CWM with online adaptation capabilities.

A new electric load transient recognition model is developed from the sequential modification of cluster weighted modeling (SCWM) based on the approach of un-received transient tail prediction from received transient segments. SCWM can achieve, in real-time, transient recognition under multiple transient overlapping situation, overlapped transients separation, and adaptive transient scaling. The SCWM assists with transient detection under situation of multiple transient overlap.

The developed transient recognition model was used to achieve the transient recognition control for hybrid fuel cells system, in which the transient recognition model provides the feed-forward information about long range behaviors of the coming load transients and accordingly controls the fuel cell power output optimally with respect to each individual transient.

The transient recognition model was developed into a pipelined RTL design and implemented in Xilinx Virtex4 FPGA.

Background

Gaussian Mixture Model

Mixture models are often used in probability density estimation to approximate a density distribution function with the superposition of a number of kernel distributions. A mixture model consists of a number of distribution kernels with predetermined forms. The number of kernels is often pre-determined with the assumption that the model size only reflects the complexity of the problem being modeled, and does not necessarily grow with the size of training data set. Although it is not required, the kernels are often assumed to be Gaussian because the Gaussian distribution has a concise structure with only two parameters to be estimated. It needs the least prior knowledge (as measured in terms of entropy) in estimating an unknown

density [30], and it has the capability of smoothly approximating arbitrarily shaped densities [14].

A Gaussian mixture model (GMM) consists of a set of M clusters. Each cluster is assumed to be in Gaussian form,

$$p(\vec{x}_n|c_m) = \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma_{m,d}^2}} \exp\left[\frac{-(x_{n,d} - \mu_{m,d})^2}{2\sigma_{m,d}^2}\right], \tag{1}$$

where D is the dimension of the input vector $\vec{x}_n$, $\sigma_{m,d}^2$ is the variance of the d'th dimension of the m'th cluster, $\mu_{m,d}$ is the center of the d'th dimension of the m'th cluster, and c_m is the label of m'th cluster. The separable Gaussian form is often used in GMM because it requires much less computation cost that would be required for a full covariance matrix. A weight $P(c_m)$ is assigned to each cluster with the restriction that,

$$0 \leq P(c_m) \leq 1, \quad and \quad \sum_{m=1}^{M} P(c_m) = 1. \tag{2}$$

The density at the n'th data point $\vec{x}_n$ is approximated by the superposition of the M clusters,

$$p(\vec{x}_n) = \sum_{m=1}^{M} p(\vec{x}_n|c_m) \cdot P(c_m). \tag{3}$$

Three prior parameters $\{\vec{\mu}_m, \sigma_m^2, P(c_m)\}$ are associated with each cluster. The goal of training is to find optimal estimates of the parameters in the sense of maximizing the joint log likelihood,$J = \sum_{n=1}^{N} \log\ p(\vec{x}_n)$, over the training data set $\{\vec{x}_n\}_{n=1}^{N}$. Let $\frac{\partial J}{\partial \theta} = 0$ (where θ is a parameter), we have

$$\hat{\vec{\mu}}_m = \frac{\sum_{n=1}^{N} p(c_m|\vec{x}_n) \cdot \vec{x}_n}{\sum_{n=1}^{N} p(c_m|\vec{x}_n)}, \tag{4}$$

$$\hat{\sigma}^2_{m,d} = \frac{\sum_{n=1}^{N} p(c_m|\vec{x}_n) \cdot (x_{n,d} - \hat{\mu}_{m,d})^2}{\sum_{n=1}^{N} p(c_m|\vec{x}_n)}, \tag{5}$$

$$\hat{P}(c_m) = \frac{\sum_{n=1}^{N} p(c_m|\vec{x}_n)}{N}, \tag{6}$$

where $p(c_m|\vec{x}_n)$ is the posterior likelihood that an cluster represents input data $\vec{x}_n$. This can be evaluated by the Bayes theorem as,

$$p(c_m|\vec{x}_n) = \frac{p(\vec{x}_n|c_m) \cdot P(c_m)}{\sum_{m=1}^{M} p(\vec{x}_n|c_m) \cdot P(c_m)}. \tag{7}$$

Equation (6) is derived with the help of softmax transformation [7],

$$P(c_m) = \frac{\exp \gamma_m}{\sum_{m=1}^{M} \exp \gamma_m}, \tag{8}$$

where γ_m is a un-restricted auxiliary variable that helps transform a restricted optimization problem of $P(c_m)$ to an un-restricted one. Equations (1) through (7) are nonlinearly coupled and can not be solved directly. Instead, a iterative process, called Expectation Maximization (EM), is used to find the optimal parameters values [20, 21]. The iteration starts from a initial guess of the parameters. In each step, the new posterior likelihood $p(c_m|\vec{x}_n)$ is evaluated based upon the priors $\{\vec{\mu}_m, \sigma^2_m, P(c_m)\}$ evaluated in last step, and the new evaluation of priors is updated by the newly updated posterior likelihood. The guaranteed convergence of the parameters with the direction of minimizing $-J$ is proven in [7].

The GMM finds important applications in many areas, such as radial basis functions in functional approximation [51], soft splitting (soft weight) in divide-and-conquer nonlinear modeling [34], and a wide range of signal processing areas which involve signal characterization recognition and signal density estimation, e.g., speech processing [52], image segmentation [10], and sensor array processing [8].

Divide and Conquer

Nonlinear modeling problems are often solved with model-based methods through supervised training with sample inputs and the corresponding desired outputs. The model parameters are adjusted in the direction of optimizing some criterion over the training set, such as maximum likelihood or minimum mean square error. Conventional modeling methods use a single complex model to fit with the whole training set. These methods adjust the model parameters based on innovations appearing from all of the training data. The global methods face a critical problem of non-convex optimization in a large scale modeling task that is probably nonlinear. These methods suffer such difficulties as long training time, local minima, and model over-fitting / under-fitting conflicts. As shown in Fig. 1, if a regularized model is assumed, the global mapping may be smooth; however the approximation of local details may suffer from under-fitting and loss of precision. If a flexible model is assumed, the local details approximation may be satisfied; however the model will also probably over-fit the noise and fail to generalize.

Divide-and-conquer is a methodology addressing the difficulties of the complex nonlinear modeling problems using only a single model. The basic idea of divide-and-conquer is to: (1) Divide the original training set (the original complex modeling problem) into a number of subsets (simple sub-problems); (2) Fit the data in each subset by a simple local model; and (3) Recombine the local models for the final global modeling generation. A general formulation of divide-and-conquer is shown as

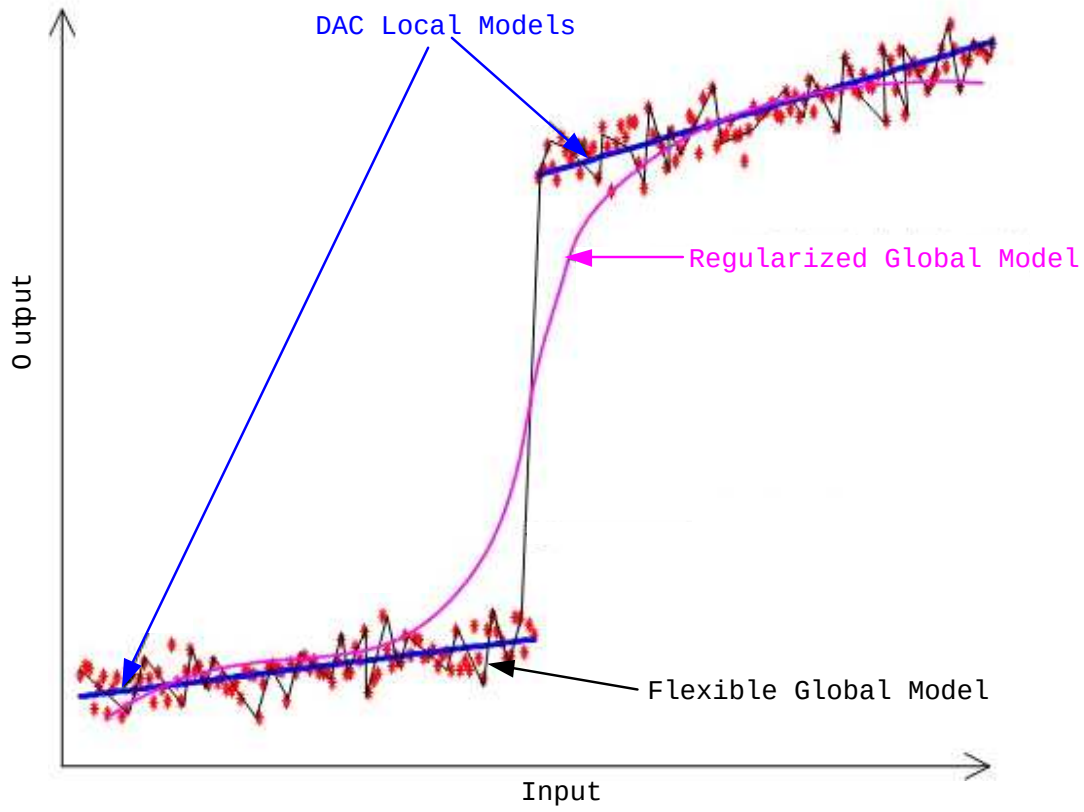

Figure 1: Nonlinear over-fitting/under-fitting modeling problems and divide-and-conquer solution.

below,

$$\Phi(x) = \sum_{m=1}^{M} \gamma_m(x) \cdot \phi(x, \beta_m), \tag{9}$$

$$0 \leq \gamma_m(x) \leq 1 \quad and \quad \sum_{m=1}^{M} \gamma_m(x) = 1, \tag{10}$$

where $\gamma_m(x)$ is the membership function of the m'th local model $\phi(x, \beta_m)$ (with parameters β_m) with respect to input x, and $\Phi(x)$ is the final global model generated from the combination of the local models. In these equations, each local model

$\phi(x, \beta_m)$ only works properly in a finite region of the input space. The membership function $\gamma_m(x)$ select an appropriate local model or combination of local models.

Most divide-and-conquer methods assume linear local models to take advantage of fast and easy linear local optimization. Some modeling problems may require more complex local models including the generalized linear model (GLM) (which has linear input networks and a single output nonlinearity) with the forms of Poisson, Gamma, Normal, and Binomial [2]; or more complex local models such as radial basis functions, multi-layer perceptrons [57], or recurrent neural networks [29].

The "dividing" step of the problem can happen in two ways: hard splitting where $\gamma_m(x)$ is either one or zero, or soft splitting where $\gamma_m(x)$ is a continuous and smooth function of x. Soft splitting is often adopted because it not only helps maintain smoothness and continuity over the global mapping [33], but also generally leads to less estimation variances than does hard splitting [34]. The membership function $\gamma_m(x)$ can be modeled naturally from the probability intuition as a prior or posterior likelihood ($p(x|m)$ or $p(m|x)$) [17, 18] to reflect the correlation between the input and a specific local model. Some methods model the local model switching sequence by a Markov process [56] or Monte Carlo method [12]. Alternatively, the membership function can even be modeled by a higher level model such as the recurrent neural network [29] or adaptive spline model [9, 42].

Divide-and-conquer methods have been widely applied to attack the problems met in complex modeling because this methodology can often speed up model training [44, 53, 43], lead to a more compact model [43, 31], reduce the parameter estimation bias [34], and may achieve the nonlinear modeling task through a combination of tractable local linear models [31, 54, 56, 9, 42, 18].

Cluster Weighted Modeling

Cluster weighted modeling (CWM) is a divide-and-conquer method including a bank of local models organized by a GMM method that was first developed for digital music time series analysis and synthesis [17, 18, 48]. CWM attacks the modeling problem through a mixture of experts, and softly splits the global modeling task according to clustering on the input space with respect to different local models. The notational convention is as follows: $\vec{x}_n$ is the input vector, y_n the corresponding desired outputs, and c_m the cluster label. Three quantities, named the input prior $p(\vec{x}_n|c_m)$, input-output dependency prior $p(y_n|\vec{x}_n, c_m)$, and cluster weight $P(c_m)$, are needed to describe a cluster and measure the likelihood between the data and a cluster. The priors are assumed to be Gaussian. A local model is embedded in each cluster as the mean function of $p(y_n|\vec{x}_n, c_m)$, which approximates local function mapping over one cluster. The CWM output is combined from the local models' outputs averaged by the input likelihood and cluster weight. The model parameters are iteratively evaluated through an expectation maximization (EM) process that maximizes the joint log-likelihood over the whole training set. However, the training samples are effectively divided into several parts by the posterior likelihood $p(c_m|\vec{x}_n, y_n)$ (which is directly evaluated from the cluster attributes $p(\vec{x}_n|c_m)$, $p(y_n|\vec{x}_n, c_m)$, and $P(c_m)$) between a cluster and one sample pair. Only samples that are strongly associated with a cluster will contribute to tuning the local model of that cluster. CWM was originally proposed for stochastic time series analysis and synthesis [46, 45], but is also appealing for non-linear modeling [47] and pattern recognition problems.

Extensions to CWM

The EM training algorithm for CWM may have a singular matrix inversion problem in updating the local model parameters when the model order is large or the

training data are sparse. Singular value decomposition (SVD) method has been applied to deal with poor conditioned problems [48]. However, SVD involves a large computational and storage cost for a high dimensional problem. This dissertation proposes a Cluster Weighted Normalized Least Mean Squares method (CWNLMS). The least mean square method does not have matrix inversion computation and requires much less computational effort than the SVD method. The cluster weighted LMS method is further extended to an online adaptation method for all of the CWM parameters allowing the online parameter adaptation for time varying systems.

CWM is considered in this dissertation for electric load transient recognition for hybrid fuel cell system control. This application requires the delay of the transient recognition model to be as small as possible. However, in general no prior information about the minimal dimensionality for effective transient recognition is available. Therefore, in the CWM training process the modeling length of the transient pattern is assumed to be a large value to assure reliable transient recognition. This value is very possibly not optimal or is even greater than the actual length of some transients, and it introduces a large delay in conventional CWM prediction. Empirical observation shows that the transient variances in each dimension (indexed by time point) decreases approximately with time, hinting that the information (uncertainty) contained in each dimension becomes less and less as time passes. According to principle component analysis (PCA) [7], transient recognition can converge fast by "picking up" the transient information sequentially. Another empirical observation shows that the shape and magnitude differences between the initial parts of different type transients are big, which further suggests fast convergency of sequential transient recognition. This dissertation proposes to minimize the prediction delay by re-formulating the CWM prediction process in sequential way, named *sequential CWM* (SCWM). A novel aspect of SCWM is that the *tail prediction* for the partially received transient

segment is used to complete the transient pattern for CWM modeling. This idea is extended to real-time detection and separation of overlapped transients.

Application Background

Electric Load Transient Recognition

Electric loads experience load turn-on transients which are of high level peaks of power compared to steady state operations. Observations from load monitoring systems reveal that many important load classes exhibit similar and repeatable transient profiles (sequence of transient sections) for the loads in the same class and sufficiently distinct transient shapes for different classes of transients, suggesting that load transients can be used as reliable signatures for identifying / recognizing load types [13].

The Non-Intrusive Load Monitor (NILM) is a near real-time field electric load monitoring and diagnostic system utilizing the load transients information sensed from a current waveform at central point in a building or system [13]. The NILM detects transient events by a change-of-mean detector. The change-of-mean detector filters the power with a low pass filter, and compares the power stream level with the filter output. Transient events are detected if the difference is greater than a predefined threshold. The NILM disaggregates and recognizes individual loads using a library of transient signatures or exemplars. An exemplar is composed of a sequence of transient sections. Each transient section is a waveform segment with a relatively high derivative. Sections in a transient exemplar can be shifted both in time and offset, and all of the sections in one exemplar share an adjustable global scaling factor. The transient signature library model is pre-trained to record repeatable load transients observed in a system. The incoming aggregate events are compared to the exemplars with a least squares criterion to select appropriate gains and shifts. The best matched exemplar with the least residual is then compared to a threshold, and is considered

"recognized" if a threshold criterion is satisfied. The measured transient information can be further analyzed for the purposes such as load parameter identification and/or load "health" diagnostics.

Hybrid Fuel Cell Systems Control

Fuel cells have attracted much attention as an efficient, scalable, low-polluting means of generating electrical power. Potential fuel cell applications include distributed generation, auxiliary and primary generation in transportation systems, consumer electronics, and backup generation. Load transients in fuel cell systems often draw significant peaks in power relative to the steady-state consumption. These peaks may impact lifetime and efficiency of fuel cells [16, 1]. The effects of load transients on the fuel cell can be reduced by combining fuel cells with energy storage devices such as capacitors or batteries to form a hybrid system as in [25, 40, 3, 28, 38, 24, 26, 23]. Transient peak power requirements can be provided by the energy storage devices, which typically have much faster dynamics than fuel cells. Hybrid fuel cell systems have been proposed for improved transient response in several scenarios. Proton exchange membrane (PEM) fuel cells are considered in [25, 28, 24, 26, 23] in combination with lead-acid batteries, Li-ion batteries, and capacitors as energy storage elements for portable military electronics and communication applications. A fuel cell system coupled with a superconducting magnetic energy storage system (SMES) is proposed in [38] for a distributed generation system. The authors of [40] investigate the combination of a direct methanol fuel cell (DMFC) with an all-solid-state super-capacitor. The authors in [19, 32, 27, 5, 37] consider combining fuel cells with batteries or super-capacitors for electric vehicle applications.

Transient Recognition Control for Hybrid Fuel Cell Systems

Simple hybrid fuel cell systems connect the energy storage device in parallel with the fuel cell. During a transient, the portion of current delivered from the fuel cell is determined implicitly by the impedances of the fuel cell and the storage device. An example can be found in [3]. Some hybrid systems control the fuel cell to output approximately constant power as in [25, 28, 38, 24]. In these applications the storage device handles all transient energy and the fuel cell operates like a battery charger.

The transient performance of a hybrid fuel cell system can be improved if the control system can determine the future behavior of the transient. The fuel cell can then be controlled to avoid responding to large transient currents. In some very limited, fixed-load scenarios, it may be possible for an external communications network to alert a hybrid system to the startup behavior of key loads. This dissertation proposes a more flexible system motivated by the near real-time transient recognition capabilities of the Non-Intrusive Load Monitor (NILM) in [49]. This dissertation proposes a power control scheme using the feed-forward information about the transient future behavior provided by the transient recognition model that is constructed by a low-latency Sequential CWM (SCWM) model mentioned above. This control scheme manage the flow of energy between components of a hybrid fuel cell system, and may allow designers to minimize energy storage requirements, improve system reliability, reduce internal loss, and extend the lifetime of fuel cell systems.

Organization of the Dissertation

The dissertation is organized as follows. In chapter 2, the improvements to CWM are developed and discussed, including the new training algorithm developed, and the sequential CWM and its contribution to the transient overlapping resolution. In chapter 3, a power control scheme for hybrid fuel cell systems based on the SCWM

is developed. The implementation of transient recognition control is discussed and developed in detail, including transient detection, scaling under overlapping, and off-training set transient indication. Both simulation and hardware experiments were conducted to demonstrate the proposed methods. In chapter 4, the proposed TRC model is implemented in a register transfer level design and verified in an FPGA system. Design and implementation details are presented. Finally, in the last chapter, the work in the dissertation is summarized and discussed. Potential future work is outlined.

EXTENDED CLUSTER WEIGHTED MODELING METHODS

Cluster Weighted Modeling

Cluster-weighted modeling (CWM) [17, 18, 48] is a divide-and-conquer method for estimating the unconditional joint probability density $p(y, \vec{x})$ and constructing a function mapping between input patterns $\vec{x}_n$ and desired outputs y_n over a sample set $\{\vec{x}_n, y_n\}_{n=1}^{N}$ drawn from the problem being studied. The output y_n is assumed to be scalar in the dissertation, but generalization to the vector case is straightforward. CWM solves the modeling problem with a mixture of clusters (experts). The structure of a cluster is described by three priors, $\{P(c_m),\ p(\vec{x}|c_m),\ p(y|\vec{x}, c_m)\}$, and $p(y, \vec{x})$ is estimated from the priors by

$$p(y, \vec{x}) = \sum_{m=1}^{M} p(y|\vec{x}, c_m) \cdot p(\vec{x}|c_m) \cdot P(c_m), \tag{11}$$

where M is the pre-determined number of clusters. The cluster probability $P(c_m)$ can be viewed as the cluster weight, reflecting the relative importance of one cluster to the modeling problem. The input likelihood $p(\vec{x}|c_m)$ describes the global distribution of clusters in the input space and the local grouping information of input patterns around each individual cluster. $p(y|\vec{x}, c_m)$ is the likelihood that describes the local functional relationship between the input patterns and the desired outputs in one cluster. For the electric load transient modeling and recognition problem, CWM can be applied by assigning the load transient pattern as the CWM input vector $\vec{x}_n$ and the corresponding load transient steady-state value as the desired output y_n. A cluster c_m could represent a specific transient or a class of transients.

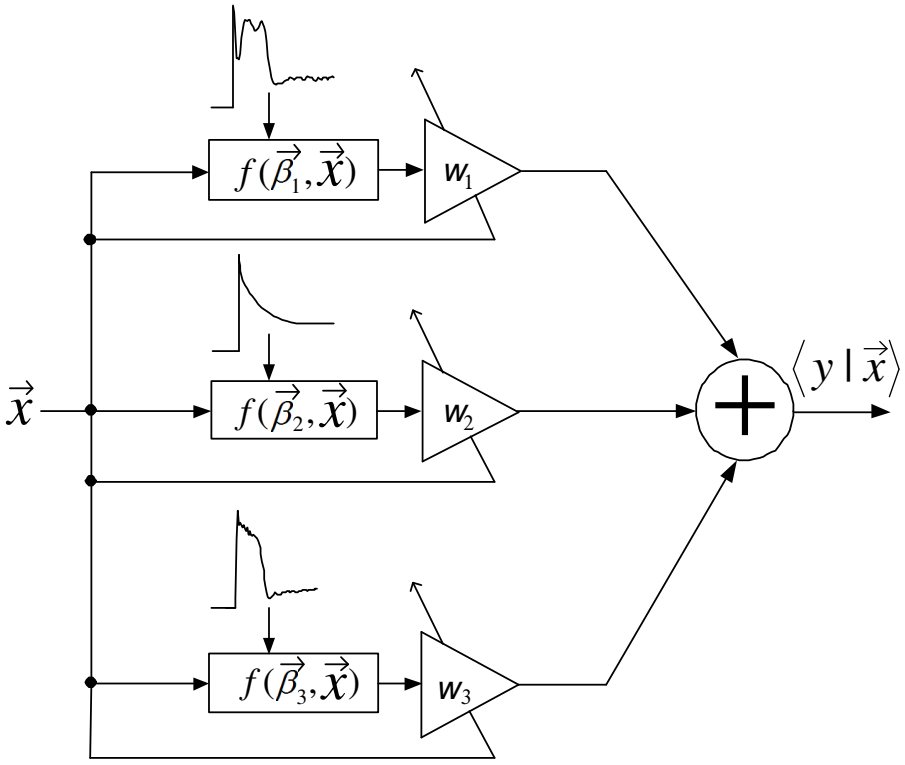

Figure 2: Graphical explanation of CWM. CWM achieves the complex non-linear modeling problem by breaking the global modeling problem into a set of sub-modeling problems corresponding to clusters. A simple local model $f(\vec{\beta}_m, *)$ is assumed for each cluster, and the model's parameters are adjusted based on the samples that are close to that cluster. The global solution is synthesized by combining the local models with weights $W_m = \frac{p(\vec{x}|c_m) \cdot P(c_m)}{\Sigma_{m=1}^{M} p(\vec{x}|c_m) \cdot P(c_m)}$.

The prior probability densities are assumed to be Gaussian distributions:

$$p(\vec{x}|c_m) = \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma_{m,d}^2}} \exp\left[\frac{-(x_d - \mu_{m,d})^2}{2\sigma_{m,d}^2}\right], \tag{12}$$

$$p(y|\vec{x}, c_m) = \frac{1}{\sqrt{2\pi\sigma_{m,y}^2}} \exp\left[\frac{-(y - f(\vec{x}, \vec{\beta}_m))^2}{2\sigma_{m,y}^2}\right]. \tag{13}$$

In (12), D is the pre-defined input pattern CWM modeling dimension. A diagonal variance matrix, instead of the full covariance matrix, is assumed for $p(\vec{x}|c_m)$ because the pattern dimension may be large. Although the full covariance matrix allows

the CWM to use full correlation information for modeling, the size of computation and storage for the covariance matrix grows quadratically with the pattern size. In contrast, the burden for a diagonal variance matrix grows linearly with pattern dimension. A local model (expert), $f(\vec{x}, \vec{\beta}_m)$, is embedded in each cluster as the mean value for $p(y|\vec{x}, c_m)$. The role of the local model $f(\vec{x}, \vec{\beta}_m)$ is illustrated in Fig. 2, i.e., the CWM output (prediction) for a given input pattern is

$$\begin{aligned} \langle \hat{y}|\vec{x} \rangle &= \int y \cdot p(y|\vec{x}) dy \\ &= \frac{\sum_{m=1}^{M} f(\vec{x}, \vec{\beta}_m) p(\vec{x}|c_m) P(c_m)}{\sum_{m=1}^{M} p(\vec{x}|c_m) P(c_m)}. \end{aligned} \tag{14}$$

(14) is a "cluster weighted" combination of the local model outputs for each cluster. The contribution of each local model output to the CWM output is weighted by the cluster-weighted likelihood $p(\vec{x}|c_m)P(c_m)$ of the input pattern to the associated cluster. Distant local models for an input pattern have little effect on the CWM output because their likelihood is small. The complexity of the local model $f(\vec{x}, \vec{\beta}_m)$ may be determined according to the prior information about the functional relationship between input patterns and desired outputs for each cluster [18]. In this dissertation, $f(\vec{x}, \vec{\beta}_m)$ is assumed to be a linear model,

$$f(\vec{x}, \vec{\beta}_m) = \vec{\beta}_m^T \cdot \vec{x} = \sum_{d=1}^{D} \beta_{m,d} \cdot x_d. \tag{15}$$

This model may suffice even for relatively complicated nonlinearities because the Gaussian clusters have good localization properties. The linear local model also helps simplify the parameter optimization process.

The training algorithm was motivated by Gershenfeld in [17] from probability theory with some key approximations. However, the reasonability of the approximations

that are important in the derivation were not clearly explained. The equations in [17] actually optimize the cluster parameters, $\vec{\mu}_m, \vec{\sigma_m^2}, \vec{\beta}_m, \sigma_{m,y}^2$ and $P(c_m)$ in the direction of maximizing the joint log likelihood $J = \sum_{n=1}^{N} \log p(y_n, \vec{x}_n)$ over the sample set. The derivation is straightforward except that the result of $P(c_m)$ is achieved with the assistance of *softmax* transformation [7]

$$P(c_m) = \frac{\exp(\gamma_m)}{\sum_{k=1}^{M} \gamma_k}, \quad -\infty < \gamma_k < +\infty, \tag{16}$$

to transform a constrained optimization variable $P(c_m)$ ($0 \leq P(c_m) \leq 1$, $\Sigma_{m=1}^{M} P(c_m) = 1$) to an un-constrained one γ_k ($-\infty < \gamma_k < +\infty$). Below the closed form result of $P(c_m)$ is derived as an example and the results of other parameters are listed for future reference.

1. *Derivation of Maximum Likelihood Estimate for* $P(c_m)$:

$$\begin{aligned} 0 &= \frac{\partial \sum_{n=1}^{N} \log p(y_n, \vec{x}_n)}{\partial \gamma_m} \\ &= \sum_{k=1}^{M} \sum_{n=1}^{N} \frac{\partial \log p(y_n, \vec{x}_n)}{\partial P(c_k)} \cdot \frac{\partial P(c_k)}{\partial \gamma_m}, \end{aligned} \tag{17}$$

and from (16), we have

$$\frac{\partial P(c_k)}{\partial \gamma_m} = \delta_{m,k} P(c_m) - P(c_m) P(c_k), \tag{18}$$

where $\delta_{m,k} = 1$ if $m = k$ and $\delta_{m,k} = 0$ if $m \neq k$. Substituting (18) into (17), we have

$$
\begin{aligned}
0 &= \sum_{k=1}^{M}\sum_{n=1}^{N} \frac{\partial \log p(y_n, \vec{x}_n)}{\partial P(c_k)} \cdot (\delta_{m,k} - P(c_k)) \\
&= \sum_{k=1}^{M}\sum_{n=1}^{N} \frac{p(y_n|\vec{x}_n, c_k)p(\vec{x}_n|c_k)}{p(y_n, \vec{x}_n)} \cdot (\delta_{m,k} - P(c_k)) \\
&= \sum_{k=1}^{M}\sum_{n=1}^{N} \frac{p(c_k|y_n, \vec{x}_n)}{P(c_k)} \cdot (\delta_{m,k} - P(c_k)). \qquad (19)
\end{aligned}
$$

Moving the first item in the right hand side of (19) to the left, we have

$$
\begin{aligned}
\sum_{n=1}^{N} \frac{p(c_m|y_n, \vec{x}_n)}{P(c_m)} &= \sum_{k=1}^{M}\sum_{n=1}^{N} p(c_k|y_n, \vec{x}_n) \\
&= \sum_{n=1}^{N} 1 = N. \qquad (20)
\end{aligned}
$$

Therefore

$$
P(c_m) = \frac{1}{N}\sum_{n=1}^{N} p(c_m|y_n, \vec{x}_n) \qquad (21)
$$

is the maximum likelihood update of $P(c_m)$.

2. *Maximum Likelihood Updates of Other CWM Parameters ($\vec{\mu}_m, \vec{\sigma^2_m}, \vec{\beta}_m, \sigma^2_{m,y}$)*:

$$
\vec{\mu}_m = \langle \vec{x}_n \rangle_m, \qquad (22)
$$

$$
\sigma^2_{m,d} = \langle (x_{n,d} - \mu_{m,d})^2 \rangle_m, \ 1 \le d \le D, \qquad (23)
$$

$$\vec{\beta}_m = B^{-1} \cdot \vec{a},\ B = \langle \vec{x}_n \cdot \vec{x}_n^T \rangle_m,\ \vec{a} = \langle y_n \cdot \vec{x}_n \rangle_m, \tag{24}$$

$$\sigma_{m,y}^2 = \langle (y_n - f(\vec{x}_n, \vec{\beta}_m))^2 \rangle_m, \tag{25}$$

where

$$\langle *_n \rangle_m \equiv \frac{\sum_{n=1}^{N} *_n p(c_m|y_n, \vec{x}_n)}{\sum_{n=1}^{N} p(c_m|y_n, \vec{x}_n)}, \tag{26}$$

The posterior likelihood $p(c_m|y_n, \vec{x}_n)$ of one cluster to a sample pair can be evaluated from the cluster's priors by the Bayesian rule,

$$p(c_m|y_n, \vec{x}_n) = \frac{p(y_n|\vec{x}_n, c_m) \cdot p(\vec{x}_n|c_m) \cdot P(c_m)}{\sum_{m=1}^{M} p(y_n|\vec{x}_n, c_m) \cdot p(\vec{x}_n|c_m) \cdot P(c_m)}. \tag{27}$$

The calculations of the priors' parameters $\vec{\mu}_m, \vec{\sigma_m^2}, \vec{\beta}_m, \sigma_{m,y}^2, P(c_m)$ and the posteriors $p(c_m|\vec{x}_n, y_n)$ are non-linearly coupled and can not be solved directly. Instead, an expectation maximization (EM) procedure is applied in the training to iteratively search for optimal values of the parameters. The training process starts from an initial guess at the parameters. In the "E-step," the posterior $p(c_m|y_n, \vec{x}_n)$ is evaluated based on the priors $\{P(c_m), p(\vec{x}_n|c_m), p(y_n|\vec{x}_n, c_m)\}$ updated in the last iteration, and then in the "M-step," the posterior is used to maximize the priors by updating the priors' parameters according to (21) through (25).

The EM algorithm may diverge because the joint log likelihood could go to infinity when a cluster shrinks to one sample point and causes the variance to be zero [7].

That problem can be resolved by adding a small constant to the variances of $p(\vec{x}_n|c_m)$ and $p(y_n|\vec{x}_n, c_m)$ [18].

Another possible problem in the EM training is that a cluster may converge locally with samples that are close to the initial position and the resultant clusters do not span the sample set well. This situation may be mitigated by better cluster initialization with the help of some cluster pre-initialization techniques such as rough-set theory [41], initial condition refinement techniques [39], split and merge approaches [11], or unsupervised competitive clustering techniques [58]. This dissertation avoids the local convergence problem by intentionally initializing the cluster centers according to the prior information about load transient distribution.

Cluster-weighted modeling may overcome some of the difficulties associated with conventional non-linear modeling using a single nonlinear model and global optimization criterion, e.g., multi-layer perceptron. In the load transient recognition problem, the mapping for load transients and their associated steady state values is likely to be locally smooth near each load transient cluster and quite different (or discontinuous) between different clusters. It is difficult to integrate all of these locally smooth but globally discontinous sub-mappings into an unique model. Two extreme cases could happen. A flexible model could cause local over-fitting to the sample data which is often coupled with noise, while a regularized model may achieve globally smooth mapping deviating from the true mapping at the boundaries of clusters. In contrast, CWM partitions a global modeling problem into several sub-problems. In each sub-problem, a simple local model can be applied to capture smooth local mapping. Global mapping is then synthesized by the cluster weighted combination of local mappings, as defined in (14). Each local model corresponds to one class of load transient, and can only be significantly activated by the transient from that class. In training, the sample set is effectively separated into M sets by the posterior $p(c_m|y_n, \vec{x}_n)$. As

indicated by (21) to (25), only the samples that are close to a given cluster will affect the update of the parameters of that cluster.

From a signal processing point of view, CWM can be thought of as a bank of filters. Each filter can be tuned individually to a specific group (cluster) of signals with similar features. The filter bank output is a probabilistic combination of the outputs of the activated filters that match the input signal. The criterion of choosing the activated filters is the cluster weighted likelihood of input signals to the associated filter.

Improvements on CWM Training Algorithm

Cluster Weighted Normalized Least Mean Squares

A numerical problem in the CWM training process is that a badly conditioned or singular matrix inversion may occur for matrix B in (24). If the CWM modeling dimension D is large, matrix B is very likely to be nearly or exactly singular. This can happen if the number of parameters of the local model is more than the number of available sample data, i.e., the problem is under-determined. Singular value decomposition (SVD) is suggested in [48] as a solution for calculating the inverse matrix. However, SVD involves high computational effort for large D.

Equation (24) is of the form of batch least squares estimate for $\vec{\beta}_m$ in "cluster weighted" sense. Similar to avoiding the direct matrix inversion in the conventional batch least squares, recursive least squares (RLS) or least mean squares (LMS) algorithms in the "cluster weighted" sense are expected to recursively/iteratively search the optimal value of $\vec{\beta}_m$.

A new criterion, named cluster weighted mean square error, is defined on the local model output residual of a given cluster,

$$J_m = \langle (y_n - f(\vec{x}_n, \vec{\beta}_m))^2 \rangle_m$$

$$= \frac{\sum_{n=1}^{N} (y_n - f(\vec{x}_n, \vec{\beta}_m))^2 \cdot p(c_m | y_n, \vec{x}_n)}{\sum_{n=1}^{N} p(c_m | y_n, \vec{x}_n)}. \quad (28)$$

which is equivalent to the joint log likelihood $\sum_{n=1}^{N} \log p(\vec{x}_n, y_n)$ in [17] with respect to optimizing the local model parameters. The same updating result for $\vec{\beta}_m$ can be derived by minimizing the new output error criterion or by maximizing the old joint log-likelihood criterion. When the order of the local model is large or the sample data are highly noisy, an additional regularization criterion may be added into J_m to achieve smooth filtering [6, 51]. Note that in (28), only the samples with high posterior likelihood with respect to a cluster will contribute significantly to the updating of the parameters of that cluster. Irrelevant residuals will not affect the parameter update of local model.

A LMS algorithm is derived in the this and combined with the EM updating routine for CWM training. A problem for LMS modeling of the electrical load transients is that the transients normally have significant initial peaks in power relative to the transient tails. Similar to the gradient estimation noise defined in [4], the estimate of $\beta_{m,j}$ suffers gradient noise (variance)

$$N_j = -2(y - \vec{\beta}_m^T \vec{x}) x_j, \ j = 1, 2, ..., D, \quad (29)$$

that is proportional to the magnitude of x_j. Therefore, the $\hat{\beta}_{m,j}$ for large x_j may have large convergence noise that causes long convergence time. The problem can be mitigated by decoupling the vector case of LMS into D scalar LMS sub-routines with respect to each individual $\beta_{m,j}$ and normalizing the updating step size for each

$\hat{\beta}_{m,j}$ by the power of the corresponding x_j. A scalar cluster weighted normalized LMS (CWNLMS) algorithm with respect to an individual $\beta_{m,j}$ can be derived as below.

Calculate the derivative of J_m with respect to $\beta_{m,j}$ and let it equal zero,

$$\begin{aligned} 0 &= \frac{\partial}{\partial \beta_{m,j}} J_m \\ &= \langle -2(y_n - \vec{\beta}_m^T \cdot \vec{x}_n) \cdot x_{n,j} \rangle_m \\ &= -2\langle y_n \cdot x_{n,j} \rangle_m + 2\vec{\beta}_m^T \langle \vec{x}_n \cdot x_{n,j} \rangle_m. \end{aligned} \tag{30}$$

Separating $\langle x_{n,j}^2 \rangle_m$ from $\langle \vec{x}_n \cdot x_{n,j} \rangle_m$ in the second item of the right hand side of (30), we have,

$$0 = -\langle y_n x_{n,j} \rangle_m + \beta_{m,j} \langle x_{n,j}^2 \rangle_m + \sum_{i=1, i \neq j}^{D} \beta_{m,i} \langle x_{n,i} x_{n,j} \rangle_m. \tag{31}$$

Moving $\beta_{m,j}$ to the left of (31), we get

$$\begin{aligned} \beta_{m,j} &= \frac{1}{\langle x_{n,j}^2 \rangle_m} \left[\langle y_n x_{n,j} \rangle_m - \sum_{i=1, i \neq j}^{D} \beta_{m,i} \langle x_{n,i} x_{n,j} \rangle_m \right] \\ &= \beta_{m,j} + \frac{1}{\langle x_{n,j}^2 \rangle_m} \left[\langle y_n x_{n,j} \rangle_m - \beta_m^T \langle \vec{x}_n x_{n,j} \rangle_m \right]. \end{aligned} \tag{32}$$

Substituting $\beta_{m,j}$ at the left side of (32) with the estimate at the current step and the parameters at the right side of (32) with the estimate at the last step, and adding a step size ρ yields the cluster weighted normalized least squares algorithm,

$$\hat{\beta}_{m,j}^{(k)} = \hat{\beta}_{m,j}^{(k-1)} +$$

$$\frac{\rho}{\langle x_{n,j}^2 \rangle_m} \left[\langle y_n x_{n,j} \rangle_m - \hat{\beta}_m^{T,(k-1)} \langle \vec{x}_n x_{n,j} \rangle_m \right]. \tag{33}$$

Note in (33), the update step size for $\beta_{m,j}$ is normalized by the cluster weighted power $\langle x_{n,j}^2 \rangle_m$. Consequently, the factor of the transient component magnitude in (29) is eliminated by normalization, such that normalized LMS converges faster than does LMS [7].

A difference between the LMS algorithm derived here and the original LMS algorithm proposed in [22, 4] is that the NLMS proposed in this dissertation is a batch update routine using all the samples in each iteration. The batch feature is appropriate to the essentially batch-updating EM routine for CWM training.

Recursive Extension of CWM Training Algorithm

Time-varying systems are often encountered in real world applications, and may require that the modeling tool be adjusted to track changing dynamics. However, the batch calculation aspect of the EM algorithm makes the CWM poorly suited to adapt to new input data. The SVD method proposed in [48] further complicates adaptive CWM.

This section develops a recursive CWM training algorithm that can be used for online adaption. To avoid the SVD calculation, the algorithm is based on the least mean squares method proposed in the last sub-section. The algorithm adjusts the full set of CWM parameters ($\vec{\mu}_m,\ \vec{\sigma}_m^2,\ \vec{\beta}_m,\ \sigma_{m,y}^2$) based on a cluster weighted least mean squares routine.

The derivation begins with the model optimization criterion as below, which is the joint log likelihood over a sample set of size K,

$$J \ = \ \sum_{n=1}^{K} \log p(y_n, \vec{x}_n)$$

$$= \sum_{n=1}^{K} \log \left[\sum_{m=1}^{M} p(y_n|\vec{x}_n, c_m) p(\vec{x}_n|c_m) P(c_m) \right]. \quad (34)$$

The LMS routine is a gradient descent method which adjusts the parameters in the negative gradient direction of the mean square error criterion. However, the criterion defined here is the *maximum* likelihood. Therefore the CWM parameters should be adjusted in the positive gradient direction, i.e.,

$$\theta^{(K)} = \theta^{(K-1)} + \rho \nabla_\theta J, \quad (35)$$

where θ is the parameter and ρ is the adjusting step size. The LMS adaptation for $\vec{\beta}_m$ is derived in the following. The derivations for other parameters are similar.

1. *Cluster local model parameters* $\vec{\beta}_m$:

The gradient of the criterion J (refer to (34)) with respect to $\vec{\beta}_m$ is

$$\begin{aligned} \nabla_{\vec{\beta}_m} J &= \sum_{n=1}^{K} \frac{1}{p(y_n, \vec{x}_n)} \cdot \frac{\partial p(y_n, \vec{x}_n)}{\partial \vec{\beta}_m}, \\ &= \sum_{n=1}^{K} \frac{p(y_n, \vec{x}_n, c_m)}{p(y_n, \vec{x}_n)} \cdot \frac{y_n - \vec{\beta}_m^T \cdot \vec{x}_n}{\sigma_{m,y}^2} \cdot \vec{x}_n, \\ &= \frac{1}{\sigma_{m,y}^2} \sum_{n=1}^{K} p(c_m|y_n, \vec{x}_n) \cdot \left[y_n \vec{x}_n - \vec{x}_n \vec{x}_n^T \cdot \vec{\beta}_m \right]. \end{aligned} \quad (36)$$

Recall that

$$\sum_{n=1}^{K} p(c_m|y_n, \vec{x}_n) = K \cdot P(c_m). \quad (37)$$

Substituting (37) into (36), therefore,

$$\nabla_{\vec{\beta}_m} J = \frac{K \cdot P(c_m)}{\sigma^2_{m,y}} \sum_{n=1}^{K} \left\{ \frac{p(c_m|y_n, \vec{x}_n)}{\sum_{n=1}^{K} p(c_m|y_n, \vec{x}_n)} \cdot \left[y_n \vec{x}_n - \vec{x}_n \vec{x}_n^T \cdot \vec{\beta}_m \right] \right\}$$

$$= \frac{K \cdot P(c_m)}{\sigma^2_{m,y}} \left[\langle y_n \vec{x}_n \rangle_m - \langle \vec{x}_n \vec{x}_n^T \rangle_m \cdot \vec{\beta}_m \right]. \tag{38}$$

Substituting (38) into (35), the LMS update for $\vec{\beta}_m$ at the Kth step is

$$\begin{aligned} \vec{\beta}_m^{(K)} &= \vec{\beta}_m^{(K-1)} + \rho \frac{K \cdot P(c_m)}{\sigma^{2,(K-1)}_{m,y}} \left[\langle y_n \vec{x}_n \rangle_m - \langle \vec{x}_n \vec{x}_n^T \rangle_m \cdot \vec{\beta}_m^{(K-1)} \right], \\ &= \vec{\beta}_m^{(K-1)} + \rho \frac{K \cdot P(c_m)}{\sigma^{2,(K-1)}_{m,y}} \langle \vec{x}_n \rangle_m \langle y_n - \vec{x}_n^T \cdot \vec{\beta}_m^{(K-1)} \rangle_m, \\ &= \vec{\beta}_m^{(K-1)} + \tilde{\rho}_1 \langle \vec{x}_n \rangle_m \langle y_n - \vec{x}_n^T \cdot \vec{\beta}_m^{(K-1)} \rangle_m, \end{aligned} \tag{39}$$

$$\tag{40}$$

where

$$\tilde{\rho}_1 = \rho \frac{K \cdot P(c_m)^{(K)}}{\sigma^{2,(K-1)}_{m,y}} \tag{41}$$

is the effective LMS updating step size.

2. *LMS update of other Cluster parameters*:

The LMS update and the corresponding effective updating step size $\tilde{\rho}$ for other parameters can be derived similarly. The results are listed below for reference.

Cluster mean vector LMS update:

$$\mu_{m,d}^{(K)} = \mu_{m,d}^{(K-1)} + \tilde{\rho}_2 \left[\langle x_{n,d} \rangle_m - \mu_{m,d}^{(K-1)} \right], \tag{42}$$

$$\tilde{\rho}_2 = \rho \frac{K P(c_m)}{\sigma^2_{m,d}}, \tag{43}$$

$$\tag{44}$$

where $d = 1, 2, \cdots, D$. D is the dimension of $\vec{x}_n$ (same for below).

Cluster input vector variance LMS update:

$$\sigma_{m,d}^{2,(K)} = \sigma_{m,d}^{2,(K-1)} + \tilde{\rho}_3 \left[\langle (x_{n,d} - \mu_{m,d})^2 \rangle_m - \sigma_{m,d}^{2,(K-1)} \right], \quad (45)$$

$$\tilde{\rho}_3 = \rho \frac{KP(c_m)}{2\sigma_{m,d}^4}, \quad (46)$$

Cluster output variance LMS update:

$$\sigma_{m,y}^{2,(K)} = \sigma_{m,y}^{2,(K-1)} + \tilde{\rho}_4 \left[\langle (y_n - \vec{x}_n^T \vec{\beta}_m)^2 \rangle_m - \sigma_{m,y}^{2,(K-1)} \right], \quad (47)$$

$$\tilde{\rho}_4 = \rho \frac{KP(c_m)}{2\sigma_{m,y}^4}. \quad (48)$$

In the above equations, $\langle \theta(y_n, \vec{x}_n) \rangle_m$ is the cluster weighted average of the variable $\theta(y_n, \vec{x}_n)$. This value can also be evaluated recursively, i.e., the new value at the K'th step is evaluated from the old value at the $K-1$'th step and the new data $(y_K, \vec{x}_K)$ being fed to the model. The method proposed in [36] is used in the dissertation to recursively update the $\langle \theta \rangle_m^{(K)}$. To see how the recursive update occurs, recall that, by definition,

$$\langle \theta(y_n, \vec{x}_n) \rangle_m^{(K)} = \frac{\sum_{n=1}^K p(c_m | y_n, \vec{x}_n)^{(K)} \theta(y_n, \vec{x}_n)}{\sum_{n=1}^K p(c_m | y_n, \vec{x}_n)^{(K)}}$$

$$= \frac{\sum_{n=1}^K p(c_m | y_n, \vec{x}_n)^{(K)} \theta(y_n, \vec{x}_n)}{KP(c_m)^{(K)}} \quad (49)$$

Separating the K'th item $p(c_m | y_K, \vec{x}_K)^{(K)} \theta(y_K, \vec{x}_K)$ out of the summation in the numerator of (49),

$$\langle\theta(y_n,\vec{x}_n)\rangle_m^{(K)} = \frac{\sum_{n=1}^{K-1} p(c_m|y_n,\vec{x}_n)^{(K)}\theta(y_n,\vec{x}_n) + p(c_m|y_K,\vec{x}_K)^{(K)}\theta(y_K,\vec{x}_K)}{KP(c_m)^{(K)}}$$

$$= \frac{(K-1)P(c_m)^{(K-1)}}{KP(c_m)^{(K)}} \cdot \frac{\sum_{n=1}^{K-1} p(c_m|y_n,\vec{x}_n)^{(K)}\theta(y_n,\vec{x}_n)}{(K-1)P(c_m)^{(K-1)}} + \frac{p(c_m|y_k,\vec{x}_K)^{(K)}}{KP(c_m)^{(K)}} \cdot \theta(y_K,\vec{x}_K). \quad (50)$$

In (50), assuming that the posterior likelihood of existing data is not affected by the new input data $(y_K,\vec{x}_K)$, then $p(c_m|y_n,\vec{x}_n)^{(K)}$ is approximated by $p(c_m|y_n,\vec{x}_n)^{(K-1)}$ for $n \leq K-1$, and therefore

$$\langle\theta(y_n,\vec{x}_n)\rangle_m^{(K)} = \frac{(K-1)P(c_m)^{(K-1)}}{KP(c_m)^{(K)}} \cdot \frac{\sum_{n=1}^{K-1} p(c_m|y_n,\vec{x}_n)^{(K-1)}\theta(y_n,\vec{x}_n)}{(K-1)P(c_m)^{(K-1)}} + \frac{p(c_m|y_k,\vec{x}_K)^{(K)}}{KP(c_m)^{(K)}} \cdot \theta(y_K,\vec{x}_K),$$

$$= \frac{(K-1)P(c_m)^{(K-1)}}{KP(c_m)^{(K)}} \cdot \langle\theta(y_n,\vec{x}_n)\rangle_m^{(K-1)} + \frac{p(c_m|y_k,\vec{x}_K)^{(K)}}{KP(c_m)^{(K)}} \cdot \theta(y_K,\vec{x}_K). \quad (51)$$

Equation (51) is the recursive update of $\langle\theta(y_n,\vec{x}_n)\rangle_m^{(K)}$ from the old value $\langle\theta(y_n,\vec{x}_n)\rangle_m^{(K-1)}$ and the innovation contributed by $(y_K,\vec{x}_K)$. In (51), $P(c_m)^{(K)}$ can be also recursively updated using the similar method,

$$P(c_m)^{(K)} = \frac{K-1}{K}P(c_m)^{(K-1)} + \frac{1}{K}p(c_m|y_K,\vec{x}_K)^{(K)}. \quad (52)$$

The posterior likelihood $p(c_m|y_K,\vec{x}_K)^{(K)}$ in (51) and (52) is estimated with the new data $(y_K,\vec{x}_K)$ and the current model parameters.

Note that the assumption

$$p(c_m|y_n,\vec{x}_n)^{(K)} = p(c_m|y_n,\vec{x}_n)^{(K-1)}, \quad n \leq K-1 \tag{53}$$

is important for the recursive fabrication. Otherwise, $p(c_m|y_n,\vec{x}_n)^{(K)}$ is re-evaluated for data up to K and the training algorithm degenerates to the batch calculation case. Using the approximation may result in sub-optimal estimates compared to the batch EM algorithm. In applications, the batch EM method should be used to initialize the model. The recursive algorithm can only be expected to adapt the model parameters to slight variations. When a brand new transient happens, $p(c_m|y_K,\vec{x}_K)$ could evaluate to 0/0, i.e., the prior likelihood for all of the clusters are 0. In this case, $(y_K,\vec{x}_K)$ should not be used to update the model parameters. Un-recognized transients could be collected for off-line analysis and used to periodically re-initialize the model. Another practical consideration for the recursive algorithm is that, depending on the excitation, the model may adapt so that performance for uncommon events is poor. One strategy to avoid this problem is to update only the champion and/or rival clusters that best match the data.

There is a theoretical upper bound of the step size to guarantee the stability of the LMS algorithm [22]. However, the effective step size of the proposed algorithm (e.g., $\rho\frac{K\cdot P(c_m)^{(K)}}{\sigma^{2,(K-1)}_{m,y}}$ for $\vec{\beta}_m$) may increase with K increasing. To prevent the algorithm from diverging, a new step size $\tilde{\rho}$ is used to replace the original ρ. In application, a safe small value for $\tilde{\rho}$ is used to ensure that the model remains stable.

Another issue with the recursive method is that adaptation may saturate with time. In (51) and (52), the contributions of the innovations $\theta(y_K,\vec{x}_K)$ and $p(c_m|y_K,\vec{x}_K)$ go to zero as $K \to \infty$, and the values of $\langle\theta(y_n,\vec{x}_n)\rangle_m$ and $P(c_m)$ will remain unchanged. In order to continuously excite the model to the new incoming data, old data may be discarded or weakened by using a moving window or a forgetting factor

in the model optimization object. For example,

$$J = \sum_{n=K-N_1+1}^{K} \log p(y_n, \vec{x}_n) \tag{54}$$

for the moving window implementation with window size N_1 or

$$J = \sum_{n=1}^{K} \lambda^{K-n} \cdot \log p(y_n, \vec{x}_n) \tag{55}$$

for the forgetting factor implementation where λ is slightly less than unity to ensure stability.

Sequential CWM for Overlapped Load Transients Recognition

This section considers the real-time application of CWM to load transient recognition for hybrid fuel cell control. The CWM prediction (14) is rederived sequentially to satisfy the real-time requirement of transient recognition control. The method is then extended to resolve overlapping transients.

Sequential CWM Prediction

The CWM modeling dimension D is a critical parameter for transient recognition control. Conventionally, CWM requires a full set of D data points of an input pattern to make a prediction. This results in a delay of at least D steps after a transient occurs before a prediction is available. If D is set to be small, the transient pattern may not be effectively modeled and the resulting prediction is not reliable. If D is large, the output delay may render the prediction useless. The real-time performance of the CWM prediction can be improved by conducting the CWM prediction sequentially. The sequential CWM (SCWM) can generate a prediction when the first data point of a transient is detected, and can update the prediction sequentially through innovations received from the subsequent transient data points. The prediction is complete after the Dth step. Under the SCWM prediction scenario, the modeling dimension D can

be set to be a large value, so that reliable prediction is ensured for any transient before the Dth data point is received. However, SCWM will automatically find for each transient $\vec{x}_n$ a minimum delay d_n^{min} when the sequential prediction converges accurately. The delay d_n^{min} is therefore the effective modeling dimension of $\vec{x}_n$ to guarantee reliable output. Suppose L_n is the physical length of $\vec{x}_n$; it is expected that d_n^{min} is generally much less than both D and L_n, so that the delay in transient recognition control is tolerable.

Before deriving the SCWM prediction, one notation needs to be defined in advance, i.e.,

$$\vec{\theta}_{(i:j)} = (\, \theta_i \quad \theta_{i+1} \cdots \theta_j \,)^T, \tag{56}$$

where the subscript index $(i:j)$ of the vector $\vec{\theta}$ marks the segment from the ith point to the jth point.

Suppose a transient $\vec{x}$ is detected at time one and the transient recorded at the k'th step is $\vec{x}_{(1:k)}$. For $k \leq D$, the SCWM prediction at the k'th step can be approximated by

$$\langle \hat{y}|\vec{x}\rangle = \langle \hat{y}|\vec{x}_{(1:D)}\rangle \cong \langle \hat{y}|\vec{x}_{(1:k)}\rangle. \tag{57}$$

The sequential CWM is derived below from the definition of $\langle \hat{y}|\vec{x}_{(1:k)}\rangle$:

$$\langle \hat{y}|\vec{x}_{(1:k)}\rangle = \int y p(y|\vec{x}_{(1:k)})dy = \int y \frac{p(y,\vec{x}_{(1:k)})}{p(\vec{x}_{(1:k)})}dy$$

$$= \frac{\int\int y p(y,\vec{x}_{(1:k)},\vec{x}_{(k+1:D)})dyd\vec{x}_{(k+1:D)}}{\int p(\vec{x}_{(1:k)},\vec{x}_{(k+1:D)})d\vec{x}_{(k+1:D)}}$$

$$= \frac{\int\int y p(y,\vec{x}_{(1:D)})dyd\vec{x}_{(k+1:D)}}{\int p(\vec{x}_{(1:D)})d\vec{x}_{(k+1:D)}}. \tag{58}$$

Substitute

$$p(y, \vec{x}_{(1:D)}) = \sum_{m=1}^{M} p(y|\vec{x}_{(1:D)}, c_m) p(\vec{x}_{(1:D)}|c_m) P(c_m), \tag{59}$$

$$p(\vec{x}_{(1:D)}) = \sum_{m=1}^{M} p(\vec{x}_{(1:D)}|c_m) P(c_m) \tag{60}$$

into (58), and note that

$$\int y p(y|\vec{x}_{(1:D)}, c_m) dy = \vec{\beta}_{m,(1:D)}^T \cdot \vec{x}_{(1:D)}, \tag{61}$$

then

$$\langle \hat{y}|\vec{x}_{(1:k)} \rangle = \frac{\int \sum_{m=1}^{M} \left[\vec{\beta}_{m,(1:D)}^T \cdot \vec{x}_{(1:D)} p(\vec{x}_{(1:D)}|c_m) P(c_m) \right] d\vec{x}_{(k+1:D)}}{\sum_{m=1}^{M} \left[\int p(\vec{x}_{(1:D)}|c_m) P(c_m) d\vec{x}_{(k+1:D)} \right]}. \tag{62}$$

Separate the variables $\vec{\beta}_{m,(1:D)}$ and $\vec{x}_{(1:D)}$ at the right hand side of (62) with respect to the segment intervals $(1:k)$ and $(k+1:D)$, we have

$$\begin{aligned}
&\langle \hat{y}|\vec{x}_{(1:k)} \rangle = \\
&\frac{\sum_{m=1}^{M} \left[\vec{\beta}_{m,(1:k)}^T \vec{x}_{(1:k)} p(\vec{x}_{(1:k)}|c_m) P(c_m) \right]}{\sum_{m=1}^{M} \left[p(\vec{x}_{(1:k)}|c_m) P(c_m) \right]} + \\
&\frac{\sum_{m=1}^{M} \left\{ \vec{\beta}_{m,(k+1:D)}^T \int [\vec{x}_{(k+1:D)} p(\vec{x}_{(k+1:D)}|\vec{x}_{(1:k)}, c_m) \cdot p(\vec{x}_{(1:k)}|c_m) P(c_m)] d\vec{x}_{(k+1:D)} \right\}}{\sum_{m=1}^{M} \left[p(\vec{x}_{(1:k)}|c_m) P(c_m) \right]} \\
= \quad &\frac{\sum_{m=1}^{M} \left[\vec{\beta}_{m,(1:k)}^T \vec{x}_{(1:k)} p(\vec{x}_{(1:k)}|c_m) P(c_m) \right]}{\sum_{m=1}^{M} \left[p(\vec{x}_{(1:k)}|c_m) P(c_m) \right]} + \\
&\frac{\sum_{m=1}^{M} \left\{ \vec{\beta}_{m,(k+1:D)}^T E \left[\vec{x}_{(k+1:D)}|\vec{x}_{(1:k)}, c_m \right] \cdot p(\vec{x}_{(1:k)}|c_m) P(c_m) \right\}}{\sum_{m=1}^{M} \left[p(\vec{x}_{(1:k)}|c_m) P(c_m) \right]}.
\end{aligned} \tag{63}$$

Because the Gaussian model in CWM is assumed to be separable, i.e., $\vec{x}_{(k+1:D)}$ is

independent of $\vec{x}_{(1:k)}$,

$$E\left[\vec{x}_{(k+1:D)}|\vec{x}_{(1:k)}, c_m\right] = E\left[\vec{x}_{(k+1:D)}|c_m\right] = \vec{\mu}_{m,(k+1:D)}. \tag{64}$$

Substituting (64) into (63), we get

$$\langle \hat{y}|\vec{x}\rangle = \frac{\sum_{m=1}^{M}\left[f(\vec{q}_m, \vec{\beta}_m)p(\vec{x}_{(1:k)}|c_m)P(c_m)\right]}{\sum_{m=1}^{M}\left[p(\vec{x}_{(1:k)}|c_m)P(c_m)\right]}, \tag{65}$$

where

$$f(\vec{q}_m, \vec{\beta}_m) = \begin{pmatrix} \vec{\beta}^T_{m,(1:k)} & \vec{\beta}^T_{m,(k+1:D)} \end{pmatrix} \cdot \vec{q}_m \tag{66}$$

is a linear local model, and

$$\vec{q}_m = \begin{pmatrix} \vec{x}^T_{n,(1:k)} & \vec{\mu}^T_{m,(k+1:D)} \end{pmatrix}^T. \tag{67}$$

Equation (65) is the sequential CWM (SCWM) prediction at the k'th step, $k \leq D$. $\vec{\mu}_{m,(k+1:D)}$ is the prediction by cluster c_m for the pattern segment $\vec{x}_{(k+1:D)}$ not yet received. A reconstruction $\vec{q}_m$ of the complete input pattern $\vec{x}_{(1:D)}$ by cluster c_m is available at any step of the sequential prediction process. The reasonableness of each cluster's prediction is evaluated by the corresponding partial likelihood $p(\vec{x}_{(1:k)}|c_m)$. Implausible reconstructions (such as $\vec{x}_{(1:k)}$ being the first points of an incoming bulb transient and $\vec{\mu}_{m,(k+1:D)}$ being the last points of a lathe's cluster center) may significantly affect the model output accuracy at the beginning. However, the prediction converges quickly as more data points arrive. Two factors contribute to the rapid convergence of the sequential prediction. First, the initial impulse magnitude and shape difference between various transients help sort transients quickly. $p(\vec{x}_{(1:k)}|c_m)$ will rise quickly as time passes for relevant clusters and vanishes rapidly for irrelevant ones. After $k \geq d^{min}$, only relevant clusters can "survive," and irrelevant clusters

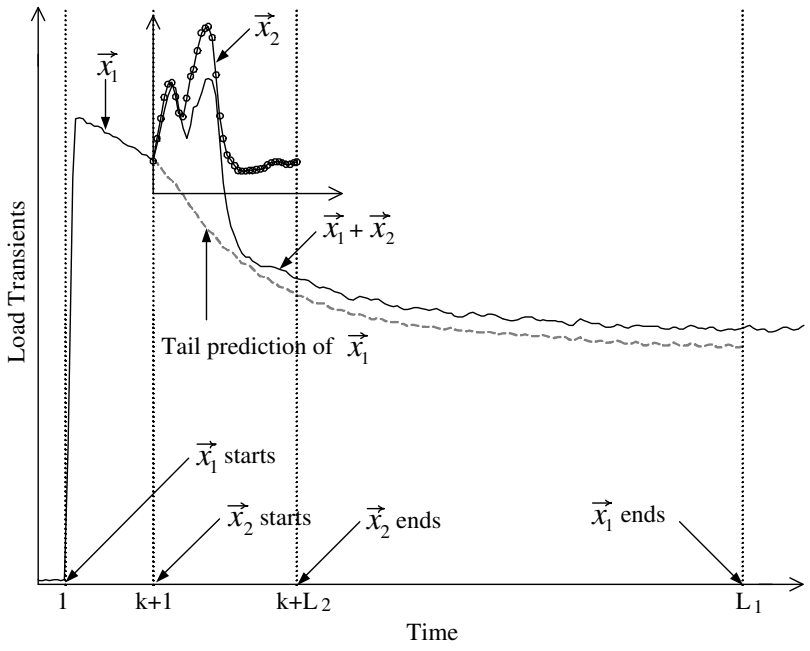

Figure 3: Transient overlap elimination. Transient $\vec{x}_1$ (with physical length L_1) is detected at time 1. Transient $\vec{x}_2$ (with physical length L_2) is detected at time $k+1$. $\vec{x}_2$ is overlapped and therefore deformed by the tail of $\vec{x}_1$. The true $\vec{x}_2$ vector (shown by the line marked with circles) can be recovered by subtracting the tail of $\vec{x}_1$ from the overlapped signals. $\vec{x}_1$'s tail can be estimated by the cluster weighted tail prediction defined in (73) and is shown by the dashed line in the figure.

have almost no effect in (65); Second, the uncertainty (variance) in each dimension (indexed by time points) of the transient pattern is conventionally descending with time, suggesting that the data with more information are received first and the uncertainty in predicting the un-received transient segment becomes less and less. The pattern dimensions can be effectively decreased by picking up dimensions with more information [15, 7], i.e., sequentially for load transients. Another merit of the SCWM prediction is that the optimal (minimal) modeling dimension for each individual transient can be automatically detected by conducting the sequential prediction procedure. Empirically, only a small initial part of a load transient is need for accurate modeling, and the resultant control action delay of the transient recognition is tolerable.

Sequential Resolution of Transient Overlapping

Overlapping transients pose a challenge for SCWM in applications. As shown in Fig. 3, transient $\vec{x}_2$ overlaps and is significantly deformed by the tail of transient $\vec{x}_1$ after time k. This could cause the predictions for both $\vec{x}_1$ and $\vec{x}_2$ to fail. Inspired by the idea of predicting the un-received pattern segment through the clusters' prior predictions explored in the last sub-section, $\vec{x}_1$'s tail can be accurately predicted by synthesizing all of the clusters' tail predictions in the same way as the SCWM output. The final synthesized result is named *tail prediction*. The true $\vec{x}_2$ vector can be recovered by subtracting $\vec{x}_1$'s tail prediction from the overlapped signals. This transient overlapping resolution is illustrated in Fig. 3. Analogous to the derivation of SCWM prediction, the cluster weighted tail prediction $\hat{\vec{x}}_{1,(k+1:D)}$ given the received segment $\vec{x}_{(1:k)}$ is,

$$\begin{aligned}\langle\hat{\vec{x}}_{1,(k+1:L_1)}|\vec{x}_{1,(1:k)}\rangle &= \int \vec{x}_{1,(k+1:L_1)} p(\vec{x}_{1,(k+1:L_1)}|\vec{x}_{1,(1:k)}) d\vec{x}_{1,(k+1:L_1)}, \\ &= \int \vec{x}_{1,(k+1:L_1)} \frac{p(\vec{x}_{1,(k+1:L_1)}, \vec{x}_{1,(1:k)})}{p(\vec{x}_{1,(1:k)})} d\vec{x}_{1,(k+1:L_1)}. \qquad (68)\end{aligned}$$

Substitute

$$\begin{aligned}p(\vec{x}_{1,(k+1:L_1)}, \vec{x}_{1,(1:k)}) &= p(\vec{x}_{1,(k+1:L_1)}|\vec{x}_{1,(1:k)}, c_m) p(\vec{x}_{1,(1:k)}|c_m) P(c_m), \qquad (69) \\ p(\vec{x}_{1,(1:k)}) &= p(\vec{x}_{1,(1:k)}|c_m) P(c_m) \qquad (70)\end{aligned}$$

into (68), we have

$$\langle\hat{\vec{x}}_{1,(k+1:L_1)}|\vec{x}_{1,(1:k)}\rangle =$$

$$\frac{\sum_{m=1}^{M} \left[\int \vec{x}_{1,(k+1:L_1)} p(\vec{x}_{1,(k+1:L_1)}|\vec{x}_{1,(1:k)}, c_m) d\vec{x}_{1,(k+1:L_1)} p(\vec{x}_{1,(1:k)}|c_m) P(c_m)\right]}{\sum_{m=1}^{M} \left[p(\vec{x}_{1,(1:k)}|c_m) P(c_m)\right]}. \qquad (71)$$

Note that in (71),

$$\int \vec{x}_{1,(k+1:L_1)} p(\vec{x}_{1,(k+1:L_1)}|\vec{x}_{1,(1:k)}, c_m) d\vec{x}_{1,(k+1:L_1)} = E[\vec{x}_{1,(k+1:L_1)}|\vec{x}_{1,(1:k)}, c_m], \qquad (72)$$

resulting in

$$\begin{aligned} \langle \hat{\vec{x}}_{1,(k+1:L_1)}|\vec{x}_{1,(1:k)}\rangle &= \frac{\sum_{m=1}^{M}\left[E[\vec{x}_{1,(k+1:L_1)}|\vec{x}_{1,(1:k)}, c_m]p(\vec{x}_{1,(1:k)}|c_m)P(c_m)\right]}{\sum_{m=1}^{M}\left[p(\vec{x}_{1,(1:k)}|c_m)P(c_m)\right]}, \\ &= \frac{\sum_{m=1}^{M}\left[\vec{\mu}_{m,(k+1:L_1)}p(\vec{x}_{1,(1:k)}|c_m)P(c_m)\right]}{\sum_{m=1}^{M}\left[p(\vec{x}_{1,(1:k)}|c_m)P(c_m)\right]}. \end{aligned} \qquad (73)$$

Equation (73) is the definition of cluster-weighted tail prediction. A difference between the tail prediction here and the ones in the last sub-section is that the prediction goes to dimension D (the pre-determined modeling dimension) in the last sub-section and goes to dimension L_n (the actual length of transient $\vec{x}_n$) in this sub-section. As shown in (73), the cluster weighted tail prediction is determined by weighting all of the clusters' predictions with corresponding partial likelihood, with the same structure as (14) and (65). When $k \geq d_1^{min}$, the optimal modeling dimension of $\vec{x}_1$, the irrelevant predictions vanish in (73) because of the very small likelihood, and the tail prediction $\langle \hat{\vec{x}}_{1,(k+1:L_1)}|\vec{x}_{1,(1:k)}\rangle$ approximates the true signal $\vec{x}_{1,(k+1:L_1)}$. Therefore the transient $\vec{x}_{2,(1:L_2)}$ can be recovered by subtracting $\langle \hat{\vec{x}}_{1,(k+1:k+L_2)}|\vec{x}_{1,(1:k)}\rangle$ from the overlapped signals. The dashed line in Fig.3 is the predicted tail of $\vec{x}_{1,(1:k)}$ calculated by (73). The recovered $\vec{x}_2$ shape is shown by the line marked with circles. Like (65), (73) is also formulated sequentially, therefore the transient overlapping resolution can be performed online, along with the SCWM prediction process. This is valuable for real-time field applications. The tail prediction error at the k'th step for i'th component $x_{1,i}(k < i \leq L_1)$ can be estimated by

$$\begin{aligned}\varepsilon_i^k &= \int (x_{1,i} - \langle x_{1,i}|\vec{x}_{1,(1:k)}\rangle)^2 p(x_{1,i}|\vec{x}_{1,(1:k)})dx_{1,i}, \\ &= \int (x_{1,i}^2 - \langle x_{1,i}|\vec{x}_{1,(1:k)}\rangle^2) p(x_{1,i}|\vec{x}_{1,(1:k)})dx_{1,i}. \end{aligned} \tag{74}$$

Substitute

$$p(x_{1,i}|\vec{x}_{1,(1:k)}) = \frac{\sum_{m=1}^M p(x_{1,i}|\vec{x}_{1,(1:k)}, c_m)p(\vec{x}_{1,(1:k)}|c_m)P(c_m)}{\sum_{m=1}^M p(\vec{x}_{1,(1:k)}|c_m)P(c_m)} \tag{75}$$

into (74), we have

$$\varepsilon_i^k = \frac{\sum_{m=1}^M \left[\int x_{1,i}^2 p(x_{1,i}|\vec{x}_{1,(1:k)})dx_{1,i} p(\vec{x}_{1,(1:k)}|c_m)P(c_m)\right]}{\sum_{m=1}^M \left[p(\vec{x}_{1,(1:k)}|c_m)P(c_m)\right]} - \langle x_{1,i}|\vec{x}_{1,(1:k)}\rangle^2. \tag{76}$$

Note that

$$\int x_{1,i}^2 p(x_{1,i}|\vec{x}_{1,(1:k)})dx_{1,i} = \sigma_{m,i}^2 + \mu_{m,i}^2, \tag{77}$$

therefore

$$\begin{aligned}\varepsilon_i^k &= \frac{\sum_{m=1}^M \left[(\sigma_{m,i}^2 + \mu_{m,i}^2) p(\vec{x}_{1,(1:k)}|c_m)P(c_m)\right]}{\sum_{m=1}^M \left[p(\vec{x}_{1,(1:k)}|c_m)P(c_m)\right]} - \langle x_{1,i}|\vec{x}_{1,(1:k)}\rangle^2, \\ &= \langle \sigma_{m,i}^2|\vec{x}_{1,(1:k)}\rangle + \langle \mu_{m,i}^2|\vec{x}_{1,(1:k)}\rangle - \langle x_{1,i}|\vec{x}_{1,(1:k)}\rangle^2. \end{aligned} \tag{78}$$

Define

$$\langle \mu_{m,i}^2|\vec{x}_{1,(1:k)}\rangle - \langle x_{1,i}|\vec{x}_{1,(1:k)}\rangle^2 \equiv \langle \mathtt{var}\left[x_{1,i}\right]|\vec{x}_{1,(1:k)}\rangle, \tag{79}$$

then

$$\varepsilon_i^k = \langle \sigma_{m,i}^2|\vec{x}_{1,(1:k)}\rangle + \langle \mathtt{var}\left[x_{1,i}\right]|\vec{x}_{1,(1:k)}. \tag{80}$$

The error approximately reflects the essential error features of the model ($\langle \sigma^2_{m,i} | \vec{x}_{1,(1:k)} \rangle$) and transient signal ($\langle \mathrm{var}[x_{1,i}] | \vec{x}_{1,(1:k)} \rangle$) respectively. The fact that the variances of the transient tails are often very small enhances the accuracy of the tail prediction.

Because the true transient's length is normally greater than the CWM modeling dimension, the conventional CWM training process is modified to include an update for transient information beyond the D'th dimension. Suppose L is the maximum true transient length in the sample set $\{\vec{x}_n\}_{n=1}^N$, then (22) and (23) in the training process should be modified to

$$\vec{\mu}_{m,(1:L)} = \langle \vec{x}_{n,(1:L)} \rangle_m, \tag{81}$$

and

$$\sigma^2_{m,i} = \langle (x_{n,i} - \mu_{m,i})^2 \rangle_m, \qquad 1 \le i \le L. \tag{82}$$

where the additional points of $\vec{x}_n$ that are beyond the L_n'th dimension are complemented with the dummy steady-state value.

In online applications, the length of the current transient needs to be estimated to determine whether the current transient is still active when a new transient is detected, and accordingly to turn on/off the transient overlapping resolution procedure. The expected transient length of $\vec{x}_n$ can be estimated by an equation similar to (73), i.e.,

$$\langle \hat{L}_n | \vec{x}_{n,(1:k)} \rangle = \frac{\sum_{m=1}^M \left[L_m p(\vec{x}_{n,(1:k)} | c_m) P(c_m) \right]}{\sum_{m=1}^M \left[p(\vec{x}_{n,(1:k)} | c_m) P(c_m) \right]}, \tag{83}$$

where L_m is the expected "transient length" of cluster c_m that is evaluated in the training process. Similar to (21)-(25) in the training process, L_m can be estimated by

$$L_m = \frac{\sum_{n=1}^{N} L_n \cdot p(c_m|y_n, \vec{x}_{n,(1:D)})}{\sum_{n=1}^{N} p(c_m|y_n, \vec{x}_{n,(1:D)})} \tag{84}$$

$$= \langle L_n \rangle_m.$$

Illustrated by Fig.3, if $k+1 < \langle \hat{L}_1 | \vec{x}_{1,(1:k)} \rangle$, then the transient overlapping resolution procedure will be turned on to recover transient $\vec{x}_2$; if $k+1 > \langle \hat{L}_1 | \vec{x}_{1,(1:k)} \rangle$, then $\vec{x}_2$ is not overlapped by $\vec{x}_1$ and can be processed by SCWM directly.

Tail prediction method for transient overlapping resolution, illustrated in Fig. 3, may not work perfectly in all situations. A worst case for tail prediction is that two different transients have identical initial transient segments (e.g., $\vec{x}_{1,(1:k)}$ in Fig. 3), and different remaining tails from each other. The tail prediction in this situation could be false, and cause the transient overlapping resolution fail. However, the possibility of worst case of tail prediction can be predicted from the CWM model cluster mean parameters. Any two clusters having the identical initial segments and different tails could be marked.

TRANSIENT RECOGNITION CONTROL

The SCWM method studied in the dissertation is mainly for the purpose of implementing a electric load transient recognition model for hybrid fuel cell systems power control. In this chapter, transient recognition control for power control of hybrid fuel cells system is described. Then the issues and considerations of implementing the transient recognition control model based on SCWM are illustrated. The performance of the transient recognition model and the contributions to hybrid fuel cell system control are demonstrated by several benchmark electric load transient examples and a prototype hybrid fuel cell system.

Novel Power Control Scheme For Hybrid Fuel Cell Systems

Figure 4 shows a novel transient-based control scheme for a hybrid system using energy storage elements and fuel cells or other critical sources. The power electronic circuit in Fig. 4 regulates the voltage on the DC bus and uses the input command from the transient recognition control (TRC) module to adjust energy flow from the storage devices and the fuel cells during a transient. The TRC model provides a prediction of the transient long range behavior based on recognizing the initial part of the input load transient. The architecture in Fig. 4 allows the fuel cell to respond to the estimate of the steady-state behavior of the load supplied by the TRC module, and uses the auxiliary fast source compensate the remaining load transient requirement. In contrast, the response of a conventional control depends on the initial transient behavior. A conventional control may unnecessarily accelerate the reactions in the fuel cell, leading to thermal consequences and system inefficiency. The system in Fig. 4 is particularly useful when the magnitude of the load transients is significant compared to the capacity of the critical source.

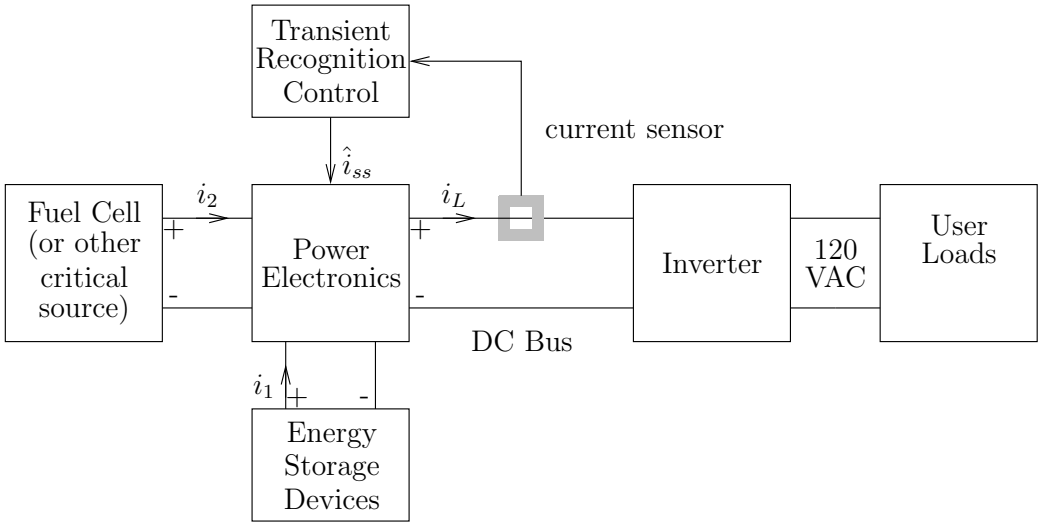

Figure 4: Hybrid power control scheme combining fuel cells or other critical source with energy storage devices. The current required from the fuel cell is determined by the recognition of the load transient. This system shows an inverter and AC load, but DC loads could also be used.

Figure 5 shows the potential advantages of the transient recognition control scheme in Fig. 4. The load current transient on the DC bus, in this case from an incandescent light bulb, has initial values that are large relative to the steady-state value. The "conventional control" response shows how a hybrid system with a linear controller might control the fuel cell for this transient. Storage devices provide the difference between the load current and the fuel cell output current at the beginning of the transient. However, the fuel cell response overshoots the demand before reaching steady state, vigorously accelerating the generating process. In contrast, the transient recognition control response in Fig. 5 shows how the fuel cell could respond to the transient given the estimate of the long-range transient behavior $\hat{i}_{ss}$.

Given a class of transients, a conventional control system could be designed to minimize the overshoot, fuel cell thermal excursions, or any other criterion of interest. However, that control would be a compromise solution over all possible loads. A control that recognizes the "fingerprint" of an incoming load can provide a response

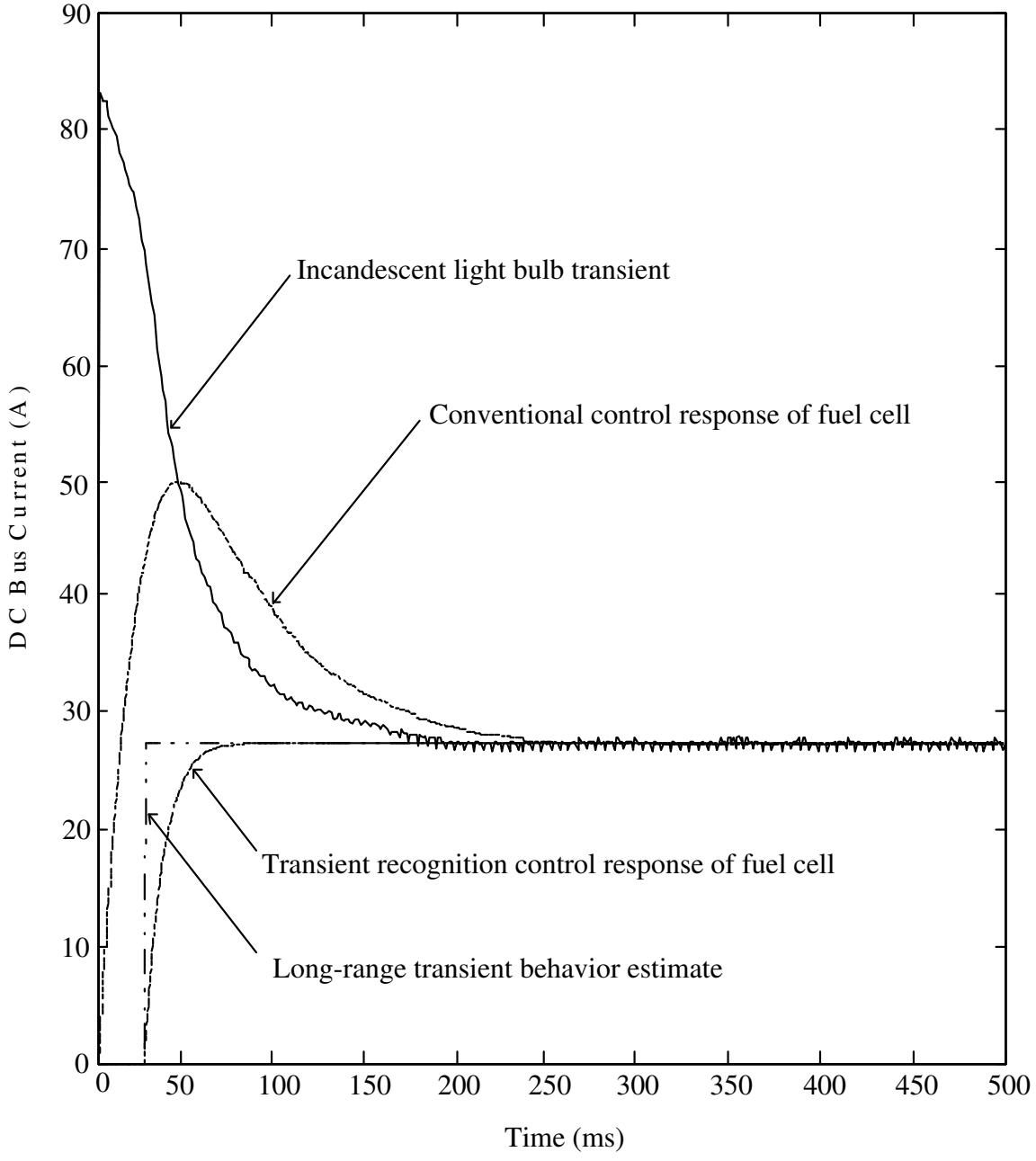

Figure 5: Theoretical comparison of conventional control and transient recognition control responses to a load transient.

that is optimal for that load. The delay in estimating a useful $\hat{i}_{ss}$ must be as short as possible. This problem is addressed by adapting a sequential modification of the cluster-weighted modeling (SCWM) developed in last chapter to the hybrid control problem. In practical systems it may be desirable to combine the TRC with some conventional control to handle, for example, small offset errors in $\hat{i}_{ss}$ in the steady state.

Transient Recognition Control Implementation

The transient recognition control (TRC) model is implemented with a sequential cluster weighted model (SCWM) described in last chapter. In the training process, the load transient current waveforms are assigned as the input vector $\vec{x}_n$, and the corresponding load steady-state current levels are assigned as the desired model output y_n. One load transient type can be modeled by one or more clusters. In the online load transient recognition process, the TRC model sequentially recognizes the incoming transient data with the SCWM algorithm described in last chapter. The TRC also separates the overlapped transients with the developed transient tail prediction technique.

In this section, other important aspects of implementing the TRC model are described. The developed techniques allow sequential processing and can be naturally combined into the SCWM scheme. Finally an operation flowchart outlines the overall functionality of the TRC model in the last section.

Transient V-section Utilization

Load transients can be characterized by one or more narrow segments with relatively high derivative or mean value variation information, called v-sections [50].

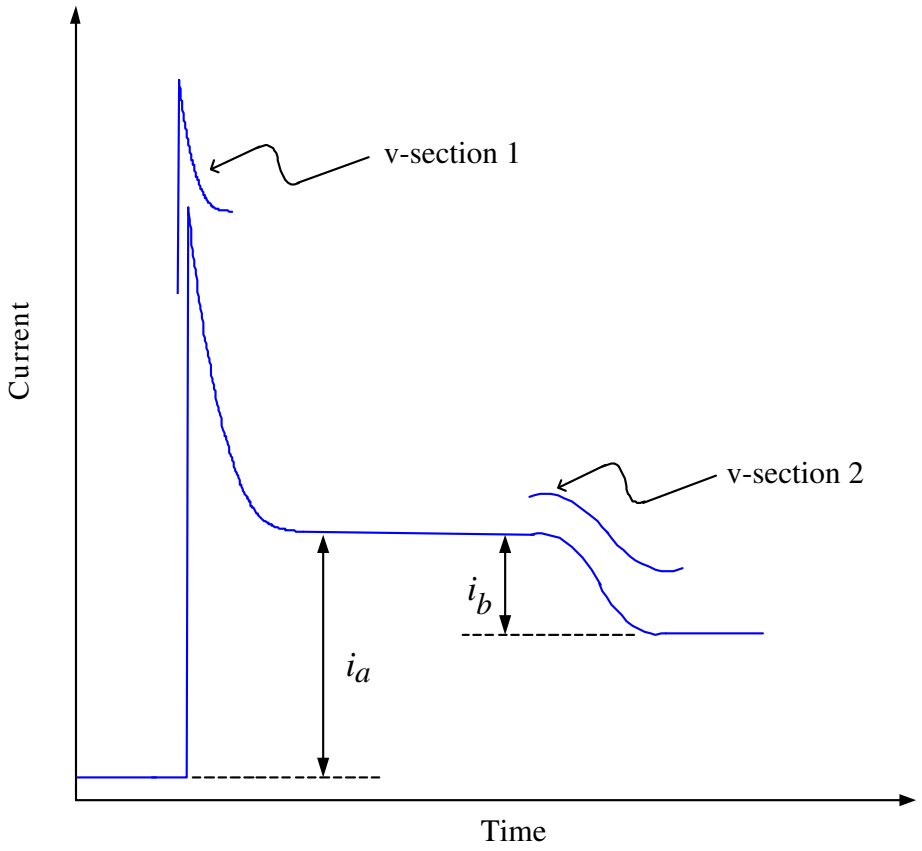

Figure 6: Informative segments (v-sections) in a load transient. A load transient can have one or more v-sections. Each v-section represents a significant variation in the long-term power consumption, and can be modeled individually by CWM.

Figure 6 shows v-sections in the context of a sample transient. Using v-sections instead of the entire transient to model the load transients is preferred because the v-sections involve less computation and storage requirements. In the non-intrusive load monitoring system, the principle purpose of transient recognition is to identify the load. In this context, it is important to apply the same scale factor / gain to each "v-section" or significant feature of the input stream. In transient recognition control the v-sections can be treated separately. Each v-section is used to predict the effective long range power change. For instance, the first v-section in Fig. 6 can be used to predict the initial change i_a in a long-duration transient. Unlike NILM,

there is no need to link v-sections to estimate the exact load type. The V-sections are found in the input transient stream by a change of mean detector, as in [49].

Online Load Transient Scaling

An empirical observation important to the non-intrusive load monitor is that transients from differently sized but physically similar loads tend to be similar up to scale factors in amplitude and time [49]. The storage and computation efficiency of the transient recognizor can be improved by adopting a transient scaling scheme that finds an offset b and an amplitude scale factor a between the detected transient and the cluster centers. The offset b can be continuously estimated by a unit gain low pass filter. The scale factor a for the most relevant cluster is determined by sequentially solving the following maximum likelihood problem

$$a = \texttt{argmax}_{a_m} \log p(\tilde{\vec{x}}_{(1:k)}|c_m), \tag{85}$$

$$\tilde{\vec{x}}_{(1:k)} = a_m \cdot (\vec{x}_{(1:k)} - b), \tag{86}$$

where a_m is the maximum likelihood scale factor between $\vec{x}_{(1:k)}$ and $\vec{\mu}_{m,(1:k)}$ for cluster c_m. a_m is determined by solving the following maximum likelihood problem for $\log p(\tilde{\vec{x}}_{(1:k)}|c_m)$ with respect to a_m, i.e.,

$$0 = \frac{\partial \log p(\tilde{\vec{x}}_{(1:k)}|c_m)}{\partial a_m}. \tag{87}$$

Substitute (86) and the Gaussian distribution for $p(\tilde{\vec{x}}_{(1:k)}|c_m)$ into (87), we have

$$0 = \frac{\partial \log p(a_m \cdot (\vec{x}_{(1:k)} - b)|c_m)}{\partial a_m}$$

$$
\begin{aligned}
&= \sum_{i=1}^{k} \frac{[a_m(x_i - b) - \mu_{m,i}] \cdot (x_i - b)}{\sigma_{m,i}^2} \\
&= a_m \cdot \sum_{i=1}^{k} \frac{(x_i - b)^2}{\sigma_{m,i}^2} - \sum_{i=1}^{k} \frac{\mu_{m,i}(x_i - b)}{\sigma_{m,i}^2}.
\end{aligned} \tag{88}
$$

Move the second item at the right hand side of (88) to the left and therefore the maximum likelihood solution for a_m is

$$
a_m = \frac{\sum_{i=1}^{k}[\mu_{m,i}(x_i - b)/\sigma_{m,i}^2]}{\sum_{i=1}^{k}[(x_i - b)^2/\sigma_{m,i}^2]}. \tag{89}
$$

Load transients are pre-scaled by the coefficients a and b before being utilized for the SCWM prediction. There are limits imposed on the scaling range for each a_m to prevent scaling that may lead to poor results, especially at the beginning of the sequential prediction. The pre-scaling operation is conducted in a sequential way and therefore can be integrated into the SCWM process.

Online Transient Event Detection and New Type Transient Indication

Electric load transient typically experiences a sudden change in mean value at the beginning of the transient period, making them relatively easy to detect. The detection delay can be ignored with respect to the time scale of a transient. The change-of-mean detector proposed in [49] is applied in the dissertation for transient detection. The detector compares the difference between the current transient value and the output of a low pass filter with transient signals as input. If the difference is greater than a threshold, then a new transient is detected.

The change-of-mean detector may not work properly when transients overlap. The interaction between the dynamics of the low pass filter and the overlapped signals may render the threshold ineffective for detecting the second transient in an overlapping

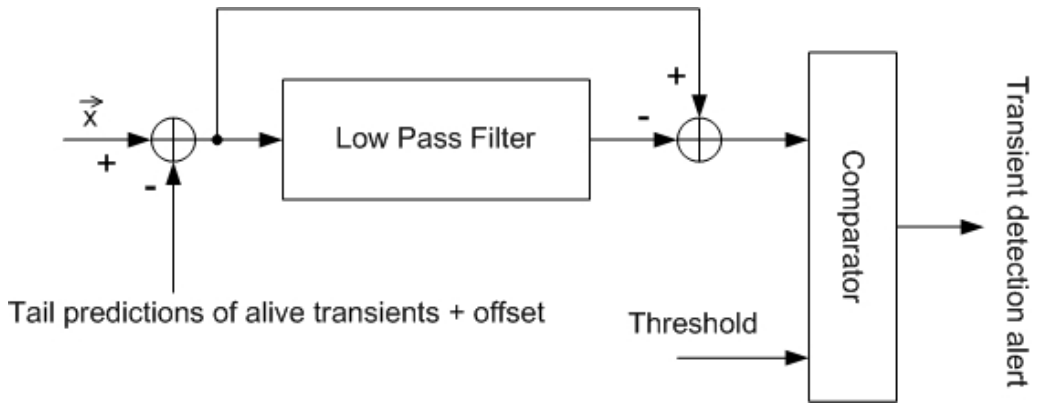

Figure 7: Change of mean detector combining tail prediction method for transient detection in overlapping situation.

situation. One way to solve the problem, illustrated in Fig. 7, is to combine the change of mean detector with the tail prediction method to subtract the tail prediction and offset of the first transient from the overlapped signal stream before it is processed by the change-of-mean detector, as if the second transient happened in zero offset and non-overlap.

Another method for detecting the overlapped transient is to monitor the likelihood curve of the first transient estimated by the SCWM model. A lock out period following the time when a transient is detected may be setup. The width of the lock out period is determined so that the likelihood will converge to the satisfied values for the recognized transients. In the overlapping situation, the likelihood estimate curve for the first transient is continually monitored. After the lock out period, if the likelihood converges to a value greater than a designed threshold, the first transient is recognized. If the likelihood of first transient suddenly decreases while the first transient is still in duration, the second transient is detected in the overlapping situation.

A reliable transient detector should have the ability to identify the off-training set transients. One merit of CWM over conventional alternatives is that CWM not only approximates the mapping from input to output, but also estimates the likelihood for

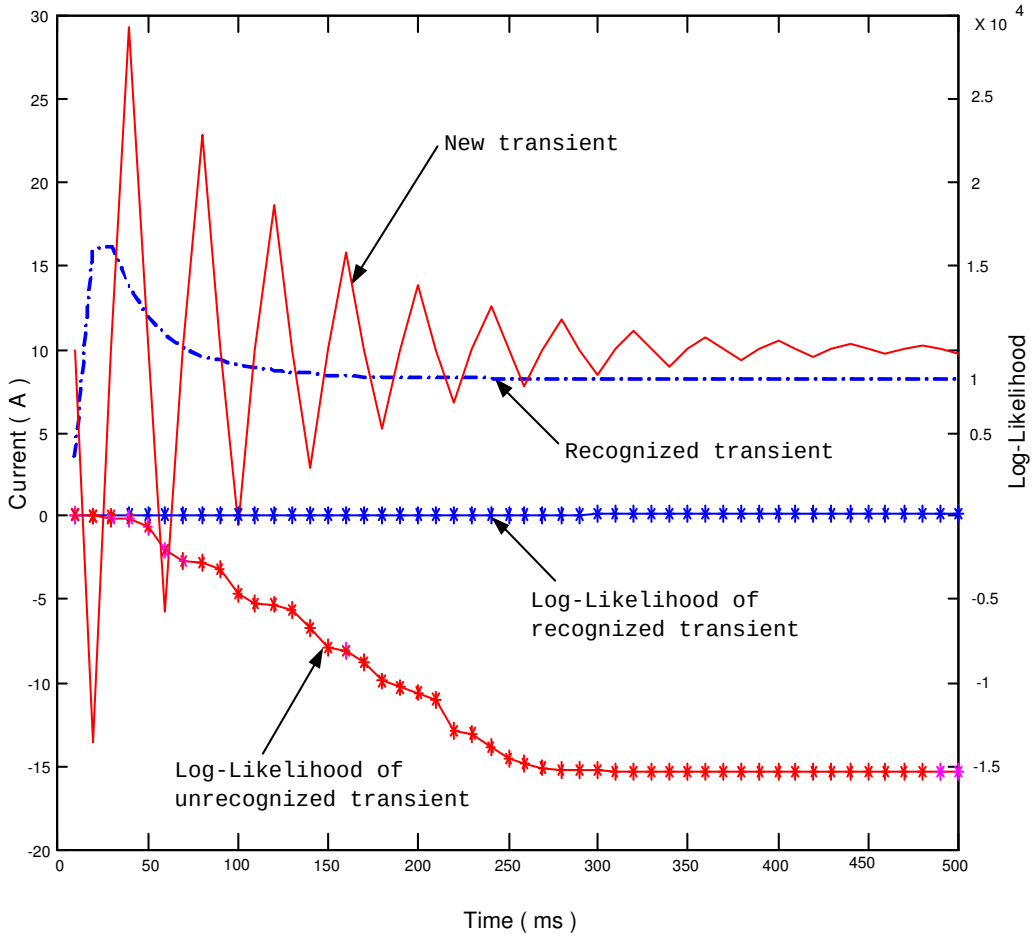

Figure 8: New transient indication by monitoring the likelihood of input.

the input signal. A neural network, for example, may give poor outputs to off-training set inputs and gives no indication that a problematic input has been presented. As shown in Fig. 8, the likelihood of a new type of transient is much less than the likelihood for a recognized transient in the example. A threshold can be set in the TRC model. If the monitored likelihood for a detected transient event never satisfies the designed threshold after the lock out period, it is indicated as a new type of transient. If the likelihood decreases from the converged value during the transient active period, the TRC will combine the result from the transient detector to determine whether a new type of transient happens or a second transient overlaps the first transient. An indicator of a new transient helps the system switch to a conservative control solution to protect the fuel cell.

Transient Recognition Control Flowchart

A flow-chart of the transient recognition control (TRC) module is shown below including signal filtering, transient detection, scaling, overlap resolution, and long-range transient behavior prediction. A new transient processing line is set up for each newly detected transient event, and any processing lines are marked as active if the associated transients are still evolving. The active status is checked by the associated transient starting point and expected transient length calculated by (83). The outcomes of each active processing line (such as the tail prediction, the expected transient length, and the long-range behavior prediction) when a new transient is detected remain available for future use, and the outcomes of the inactive processing line is used to update the power bus offset one time before it is closed. It is assumed that a new transient does not overlap the current transient which is still in the lock-out period.

Step 1: *Pre-filter the input signal stream to get rid of the 120Hz coupled noise from power electronics.*

Step 2: *Determine the status of each transient processing line. Turn off the expired transient processing lines and update the offset accordingly. Subtract the tail predictions of the active transient processing lines from the signal stream.*

Step 3: *Monitor the signal stream processed in step 2 by change-of-mean detector. Set up a new transient processing line if a new transient is detected and stop updating the outcomes of the current processing line.*

Step 4: *Block the outcomes of the newly set up processing line until the "lock-out" period expires. Check the likelihood of the newly detected transient immediately after the "lock-out" period. If the likelihood increases to a reasonable non-zero value, then go to the next step. Otherwise maintain the current SCWM output and reset the SCWM after all transients are finished, and report to the system that a new type of transient has been detected.*

Step 5: *Calculate the outcomes of the current processing line, including scale factor, tail prediction, expected transient length, and transient long-range behavior prediction. Calculate SCWM output depending on the long-range behavior predictions from each active processing line and the power bus offset.*

Step 6: *Go back to step 1.*

Load Transient Recognition Simulations

Simulations were conducted in Matlab. Five types of benchmark load transients (including transients from a lathe, computer monitor, bulb, drill, and vacuum cleaner) were gathered from a DC/DC converter–DC/AC inverter and AC load system. Current signals were measured from the DC bus between the converter and inverter. The recorded data stream was pre-processed to eliminate the 120Hz ripple from the inverter. The v-sections of the load transients were extracted, sorted to different transient classes, and synchronized with others within the same class. A total of

Transient	Number of training transients	Number of testing transients	Number of clusters
Lathe	40	8	3
Monitor v-section1	30	6	6
Monitor v-section2	30	6	6
Bulb	51	17	2
Drill	30	10	3
Vacuum	28	9	2
Total	209	56	22

Table 1: Assignments of training / testing samples and clusters with respect to different load transients.

265 transients were recorded. Each class of transients is split into two parts, one subset used for CWM modeling (training), and the other subset used for testing of the SCWM prediction performance after training. The assignments of load transients with respect to training and testing subsets are shown in Table 1. Finally a fully functional SCWM module was implemented using the properly trained CWM model and verified with the original raw transient signal stream under noise coupling, scaling, and overlapping situations.

CWM Training

The new CWM training algorithm developed in last chapter was used in the training process to find the proper values of the parameters $\{\vec{\mu}_m,\ \vec{\sigma}_m^2,\ \vec{\beta}_m,\ \sigma_{m,y}^2,\ P(c_m)\}_{m=1}^{M}$. In the training process, a divide-by-zero problem may occur in (27) if the dimension of the input pattern is large and the clusters scatter over a wide range. In this case, the evaluations of $p(\vec{x}_n|c_m)$ and $p(y_n|\vec{x}_n, c_m)$ at the boundaries of clusters (or at the beginning of the training process before the parameters converge properly) could be very small, which could cause both the numerator and denominator of (27) to be very small. These values may be too small to be represented on a machine and might

be rounded to zero. Therefore the EM iteration would fail because of the divide-by-zero computation in (27). To help solve the problem, the Gaussian likelihood can be estimated in the log domain to increase the numerical range so that the values are not rounded to zero. If the involved likelihood $p(\vec{x}_n|c_m)$ and $p(y_n|\vec{x}_n, c_m)$ in (27) are enlarged by the same scale factor, the evaluation of the posterior $p(c_m|y_n, \vec{x}_n)$ and the EM adjusting of the parameters is not affected. A suitable scale is the maximum value of $p(\vec{x}_n|c_m) \cdot p(y_n|\vec{x}_n, c_m) \cdot P(c_m)$ so that at least one item in the summation in the denominator of (27) has a non-zero value that can be represented in the machine.

Training data is first shifted so that the starting offset is approximately zero. The transient length is set to $D = 50$ given the 100Hz sampling rate. Cluster probabilities $\{P(c_m)\}_{m=1}^{M}$ are initialized as $1/M$, where M is the predefined total number of clusters. The initial positions of clusters $\{\vec{\mu}_m\}_{m=1}^{M}$ should not be initialized randomly, because this will necessitate an unreasonably long time for the clusters to converge upon the expected positions. More importantly, the CWM algorithm only ensures that the clusters converge to a local likelihood maximum [48]. Therefore random initialization may not guarantee that the clusters converge according to the variations of each transient class. A practical way of initializing the clusters is to predefine how many clusters are needed for each transient class. This is mainly determined by the variability of the transient class. More than one cluster may be used to model a class of transients if the class is highly variable. Twenty-two clusters were used in this example to model six classes of transients / v-sections. The allocation of clusters to different transient classes is summarized in Table 1. Then the set of clusters used to model one class is initialized to the average center of that class, adding small random disturbances to effectively dispatch clusters to span the transient variability. Local models' parameters $\{\vec{\beta}_m\}_{m=1}^{M}$ are initialized with small random numbers. The variances of two Gaussian distributions are initialized to be a constant 10. During training, it is also necessary to add small constants to the variances to prevent them

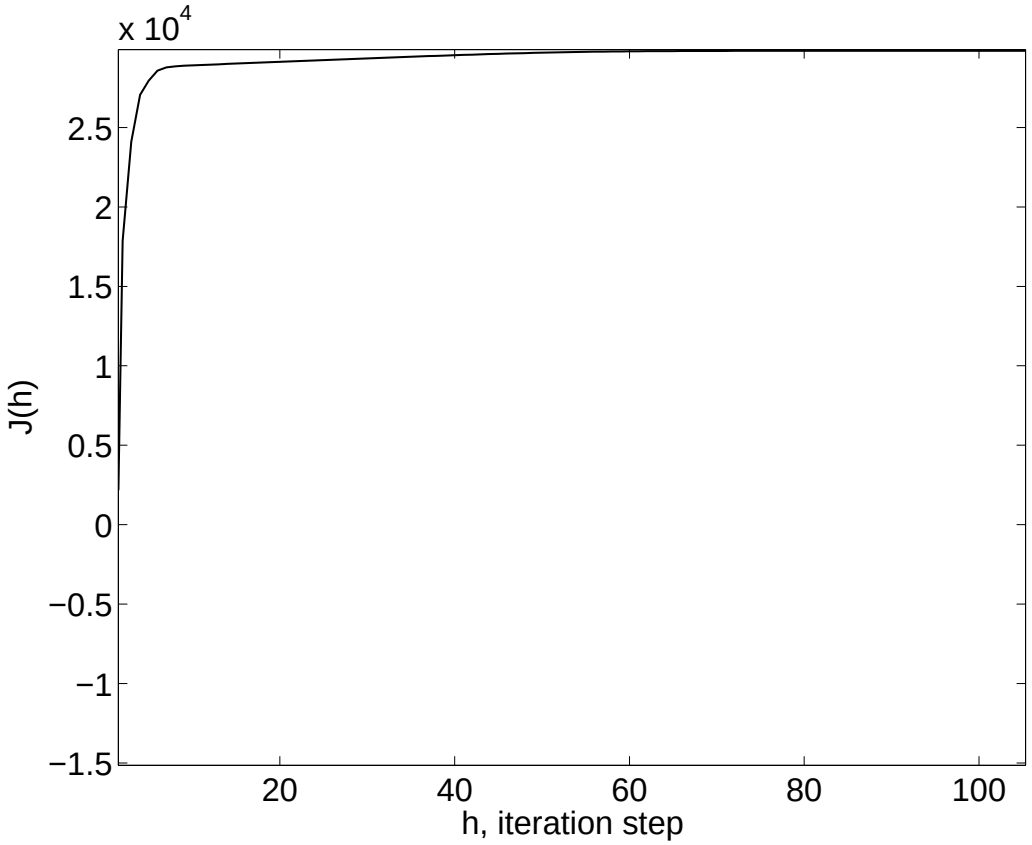

Figure 9: The convergence curve of the joint log-likelihood over the training samples. The variable $J(h) = \sum_{n=1}^{N} \log p(y_n, \vec{x}_n)$, as defined in section 3. The variable $J(h)$ extends beyond zero because the two Gaussian likelihood $p(\vec{x}_n|c_m)$ and $p(y_n|\vec{x}_n, c_m)$ are re-scaled to prevent division by zero

from shrinking to zero [18]. The adaptation step size for CWNLMS is 0.002. One thousand iterations were performed for training.

After training, the cluster probability was checked to ensure that no clusters had infinitesimal probability. If that was the case, the number of clusters and the clusters allocation for transient classes were adjusted, and the training was repeated. The converged clusters' positions were also checked relative to the position for each transient class. If any cluster converged far from the transients, the training was also repeated. The final convergence curve of the log-likelihood criterion is shown in Figure 9, which suggests that $50 - 100$ iterations are sufficient in this example.

SCWM Testing

A separate set of transient data, *not* used for training, was used for testing. Two kinds of testing were conducted. The first test used 56 primitive v-sections, similar to the ones used in training, to verify whether the SCWM prediction converged accurately and quickly in the transient scaling situation. The second test used the continuous transient stream data to evaluate the full functional SCWM module implemented in Matlab, including the pseudo-online signal pre-filtering, transient detection, scaling, recognition, and elimination of transient overlap according to the TRC operation flowchart described in sub-section 3.

The results of the first test are shown in Figure 10. The dashed lines shown in each sub-figure describe the family of load transients under scales different from the default scale used in training. The solid lines show the SCWM prediction of the transients' steady state behavior, and the broken-dashed lines show the tail prediction errors of the corresponding partially received transients. The tail prediction error is defined as a root mean square distance between the true signal and the prediction,

$$e_i(k) = \|\vec{x}_{i,(k+1:D)} - \langle \hat{\vec{x}}_{i,(k+1:D)} | \vec{x}_{i,(1:k)} \rangle \|. \tag{90}$$

Early in the observation of a transient, there are significant errors in the tail predictions and consequently in the SCWM outputs. This is expected since it is impossible to predict the future behavior of a transient depending on only one or two data points. However, in almost all cases, the tail prediction and the SCWM output settle quickly and accurately compared to the transient length. In practice, a "lock-out" interval can help block the initial behavior.

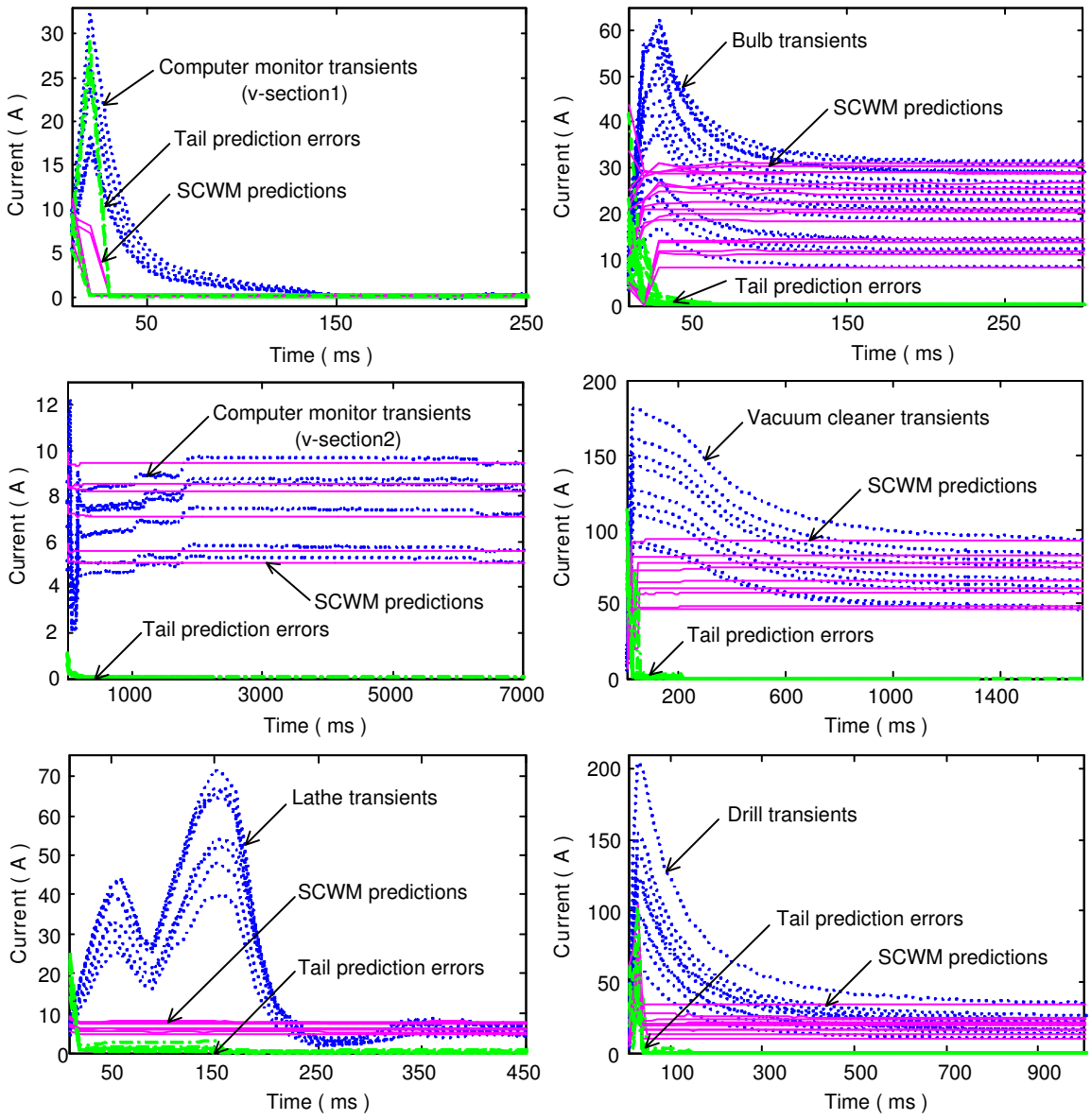

Figure 10: SCWM test of prototype electric load transients with different scale factors

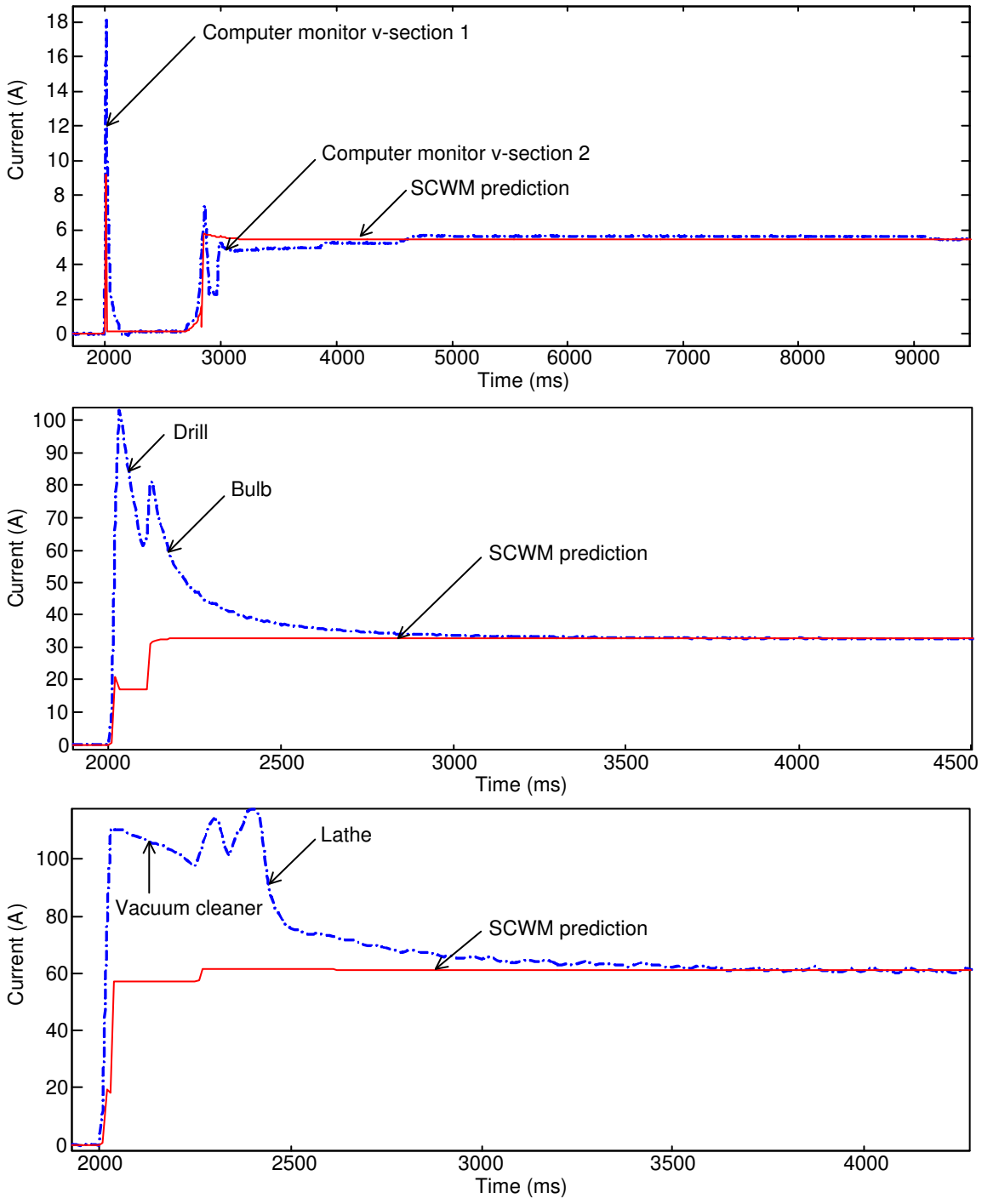

Figure 11: SCWM test of concatenated and overlapped load transients

Three simulations were conducted to verify the fully functional TRC module, including a full computer monitor transient (including two v-sections), a mixed drill-bulb transients stream, and a mixed vacuum cleaner-lathe transients stream. The results are shown in Fig. 11. The transient signals after pre-filtering were shown in figures for the purpose of visibility. The TRC output within the "lock-out" period is not blocked in the figure in order to show the full details of the TRC action. The performance of the fully functional TRC module, especially the transient overlapping resolution through the idea of tail prediction, were verified by the presented results. The long-range behavior prediction for the second overlapping transient is accurate, and there are no significant accumulated errors as reported in the simulation results.

Hybrid Fuel Cell System Results

Figure 12 shows a diagram of a prototype fuel cell hybrid system built to test the transient recognition control concept. The system in Fig. 12 is a simple representation of the general system in Fig. 4. In particular, the battery is connected to the DC bus, providing both voltage regulation and a fast source. The responses of the system in Fig. 12 under TRC are compared to the direct connection of the inverter to the fuel cell without a battery for an incandescent light bulb transient and a lathe transient. A Tektronix TDS3054B oscilloscope and TCP202 current probes were used for all measurements. Other instrumentation details are provided in Table 2.

Figure 13 shows the response of the system to an incandescent light bulb transient. All of the currents in Fig. 13 are measured on the DC side of the inverter. The load transient starts at about 70 ms. The current is not zero before the transient because the inverter consumes some power. The current supplied to the inverter is equal to the battery current until about 90 ms, when the TRC control provides an estimate of the steady state current value, which is output from the DC to DC converter. After

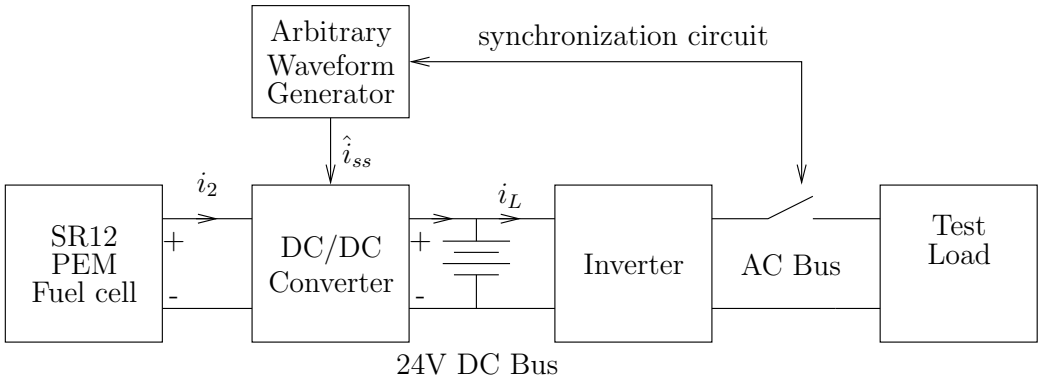

Figure 12: Multi-source test system implementation. This system can be compared to Fig. 4, except that the control signal is synthesized off-line and output at the appropriate time by the arbitrary waveform generator.

Component	Specifications
Fuel cell	Avista Labs SR-12 Modular PEM Generator
DC/DC converter	Kollmorgen KXA-80-10-20 PWM servo amplifier
Battery	17 AH, 12V lead acid
DC/AC inverter	EXELTECH XP600 600W Inverter 1100W surge
Arbitrary waveform generator	Tektronix AFG320

Table 2: Instrumentation used in hybrid fuel cell system experiment

the fuel cell takes over the load, the battery current drops to an average value of zero. Note that the battery continues to isolate the fuel cell from the ripple of the inverter.

Figure 14 shows the same incandescent light bulb transient as seen by the fuel cell. The hybrid system responses are on the left. The current from the fuel cell increases at about 90 ms when the DC to DC converter receives the command from the TRC. This step response could be adjusted to meet specific requirements for the fuel cell. The fuel cell voltage in the lower left of Fig. 14 drops in a controlled manner as the load increases. In contrast, the voltage and current responses on the right of

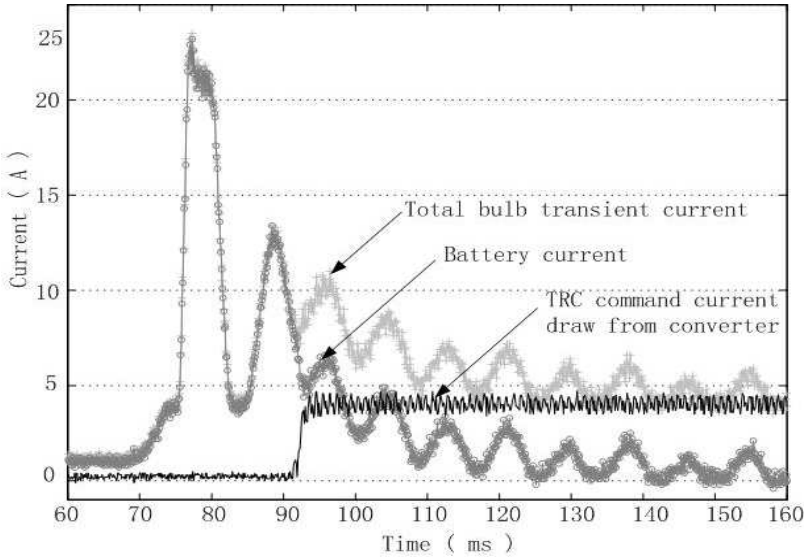

Figure 13: Responses on DC bus of hybrid fuel cell system to incandescent light bulb transient.

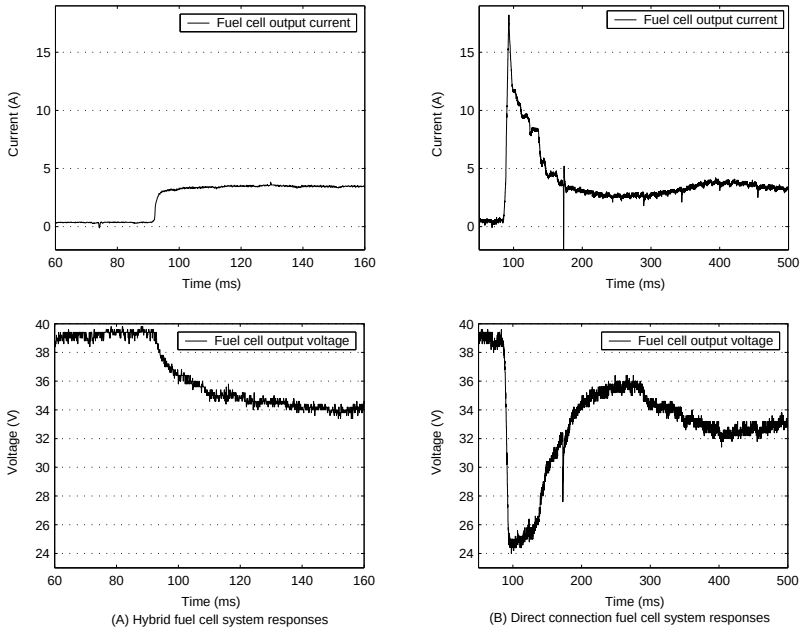

Figure 14: Responses of hybrid and simple systems to incandescent light bulb transient.

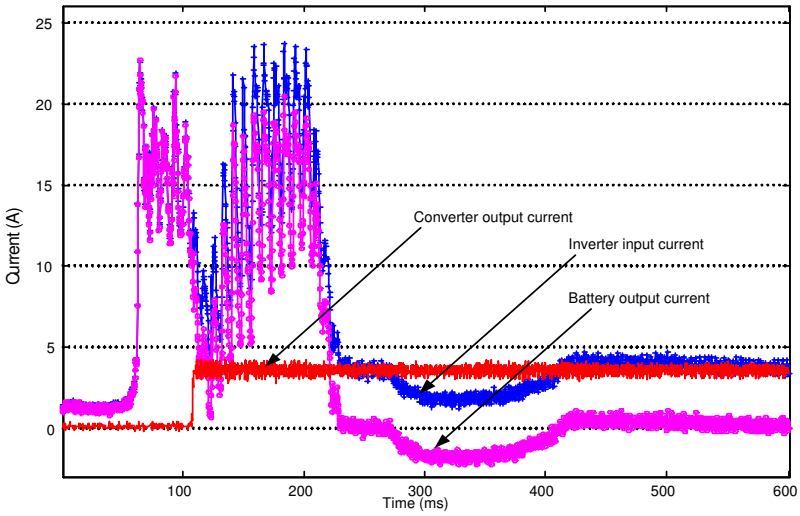

Figure 15: Responses on DC bus of hybrid fuel cell system to lathe transient.

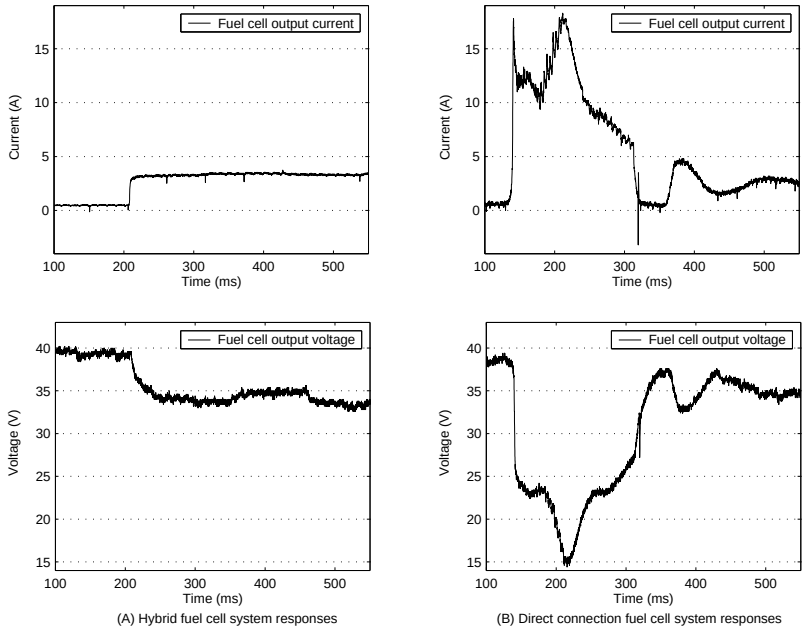

Figure 16: Responses of hybrid and simple systems to lathe transient.

Fig. 14 vary greatly and may adversely affect the fuel cell. The peak fuel cell current in response to the light bulb transient reaches nearly 20A, while the voltage collapses. The transient takes longer to evolve on the graphs to the right because the fuel cell is unable to maintain its output. The steady-state requirement of the light bulb is about 120 W – well within the 500 W rated capacity of the fuel cell and inverter system.

Figures 15 and 16 show similar results, but for a lathe transient. Figure 15 shows the simulated TRC response to the lathe transient. The effective recognition time occurs at 200 ms, when the fuel cell picks up the long range transient demand. As a result, the fuel cell voltage and current are well-behaved, as shown on the left of Fig. 16. Without TRC there are uncontrolled excursions in voltage and current as seen on the right of Fig. 16. Internal losses in the fuel cell are close to the power delivered for most of the transients on the right side of Fig. 16.

FPGA IMPLEMENTATION OF TRC MODEL

The developed TRC model is implemented on a Xilinx Virtex4 FPGA. The FPGA was selected as an experimental platform because it has the features of parallel processing and is scalable according to the implemented system size. In this chapter, the register transfer level (RTL) design and functionality details are described. Then the design is implemented with Verilog coding. Testing is conducted on FPGA hardware. The test results measured from the real FPGA hardware demonstrate successful FPGA implementation of the TRC model.

System Functionality Setup

Load transients may vary from transients collected for training the TRC model. Real transients will probably have non-zero offset, different scale factors from the template, and be coupled with noise. More importantly, transients may overlap with each other.

A robust TRC implementation should handle these real world application problems. Also an implementation should consider computational cost, economy, and low power operation, and low computational cost. The cluster calculation should be recursive and scalar at each step. Large vector data calculations, large amount of data transfers between different FPGA blocks, and re-calculation for old data should be avoided or minimized. The data should be represented with the necessary and sufficient resolution.

A top level functional block diagram is shown in Fig. 17. Six functional blocks in the RTL design are accordingly set up to achieve the transient recognition control problem, as well as to address the issues mentioned above. The blocks' functions

are reviewed briefly below. The implementations and functions of each block are described in detail in the following respect sections.

Block1: Noise cancellation and down sampling pre-filters the coupled noise in raw transient data and down-samples the data to the necessary resolution to improve the computation and storage efficiency of the following blocks.

Block2: Transient pre-processing and detection can detect transients in overlap situations, shift the transient to zero offset, separate the multiple overlapped transients, and prepare three parallel transient data paths with different time lag to the following blocks. The pre-processed transient segments look the same as the transient template used for training the SCWM model.

Block3: Cluster calculation array includes a cluster local model bank, a local model weight and transient scaling factor calculation array, and a local model output competitive selection and processing unit. The local model unit designates a "champion" and "rival" cluster to be used in place of the full SCWM computation. Irrelevant clusters are not used to save computational resources. The local model output processing unit provides the final SCWM estimate of the steady state value of the current input transient segment, as well as the current transient tail prediction information and transient duration estimate.

Block4: Post-processing array includes multiple transient processing lines. In the situation of transient overlap, each processing line corresponds to a currently active transient, updating the line status (e.g., transient active) according to the transient duration information received from the block3, recover the tail prediction of active transient, and feed the recovered tail, transient steady state output, and offset information to the output processing block and pre-processing block for further using.

Block5: Output processing uses the information about the transient steady state output, offset, and duration from each transient processing line and calculate

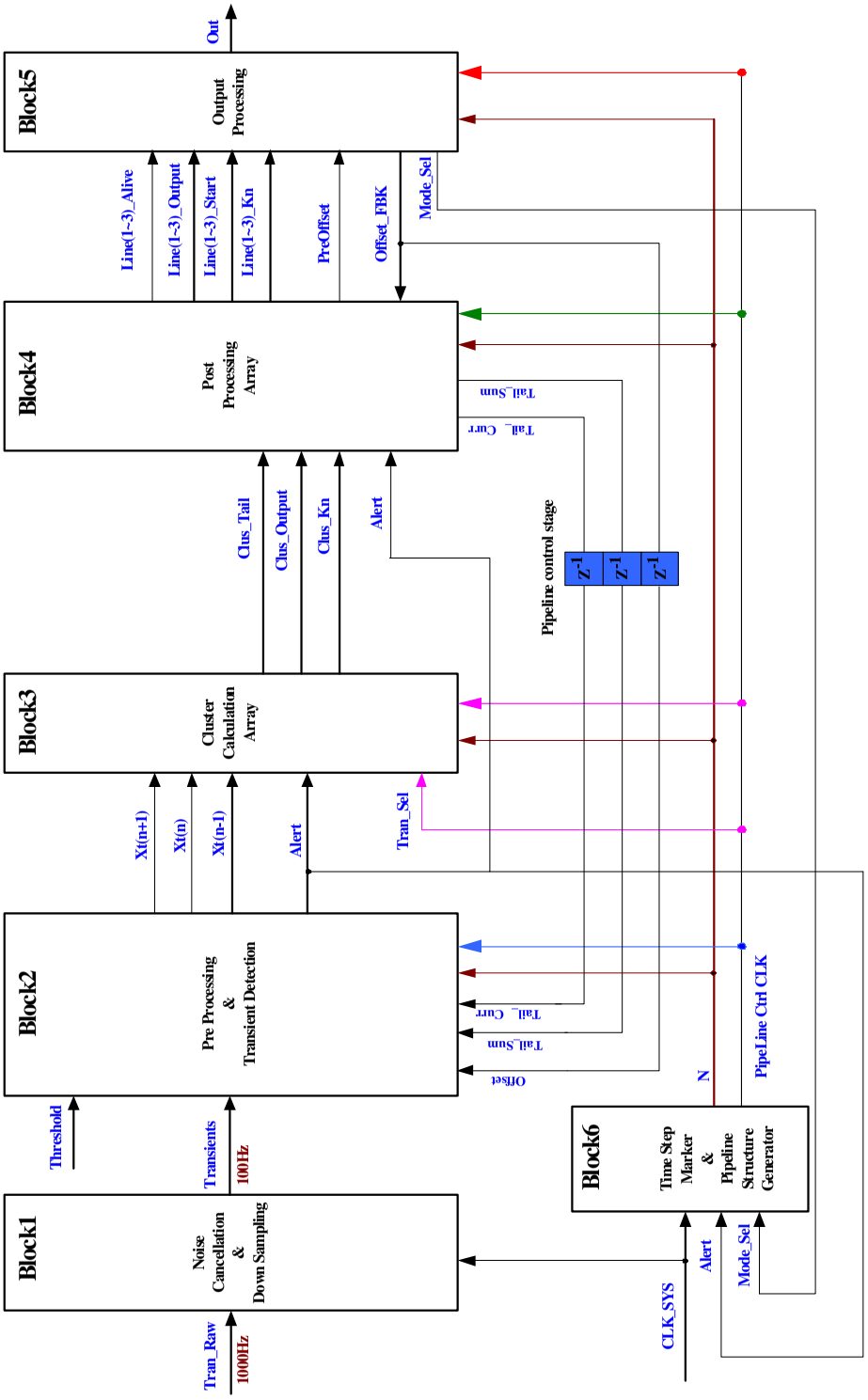

Figure 17: TRC model RTL design top level functionality structure diagram.

the final model output. This block also determines the system status of "during transient period" or "in steady state". This information is fed to block6 (pipeline control generation) for further utilization.

Block6: Pipeline control generation generates the pipeline control clocks to control the calculations and transfer of data. The pipelined operation also offer the low power feature because the intermediate status flipping in one combinational block will not affect the other blocks until the designated clock when the output of this block is stable. The system status feedback from the output processing block is used to change the pipeline structure to turn off the cluster calculation array block during the system steady status and waken it whenever a new transient is detected.

Design details of each block are described in the following individual sections. The corresponding Verilog codes are included in the appendix.

Block1: Transient Pre-filtering and Down-Sampling

This block is used to pre-filter the inverter noise in transient signals and down-sample the signal to the necessary resolution to improve the computation and storage efficiency.

Raw transient data are coupled with high power level 120Hz switching noise from the DC/AC inverter. An example for vacuum cleaner is shown by the gray line in Fig. 18. The transient signal should be properly filtered before it can be used for transient recognition model. A comb FIR filter is used to achieve the filtering problem. As shown in Fig. 19, the notches of the filter are set to the positions of the noise harmonics. The resultant filter is a 17-tap moving average FIR assuming that the input transient sampling rate is 1kHz. The resultant output from the FIR for vacuum cleaner transient is shown by the dark solid line in Fig. 18.

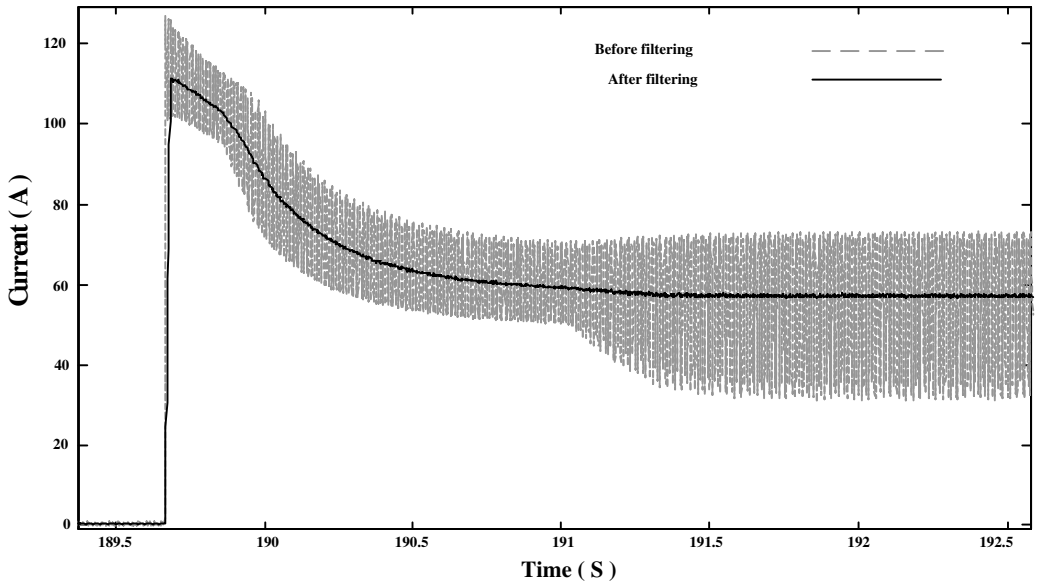

Figure 18: Typical load transient before and after pre-filtering inverter noise

In order to save finite multiplier resources, the comb filter is implemented with the multiplier-accumulator structure because a multiplier-accumulator FIR needs only one multiplier and one adder. Fig. 20 shows the FIR structure and the operation pipeline stages organization.

A mod-256 synchronized counter, a 3/8 decoder, and the necessary combinational logics are used to generate the pipeline control clocks. The highest 5 bits of the counter output $S[7:3]$ are used to switch the data path between the different taps calculation stages, $Tap[0:16]$. The lower 3 bits, $S[2:0]$, combined with the 3/8 decoder, are used to generate 8 separate clocks with different phases. Each tap stage includes 5 sub-stages of operation, $T[0:4]$ using the clocks $CLK[1:5]$ respectively within the tap stage. The first clock $CLK[0]$ is used to wait for the valid tap switch signals $S[7:3]$. The clock $T[0]$ is used to shift the taps data one step to the right, input one new data, and clear the accumulator. The clock $T[0]$ can only trigger the $Tap[0]$ stage; therefore the combinational logic for $T[0]$ is

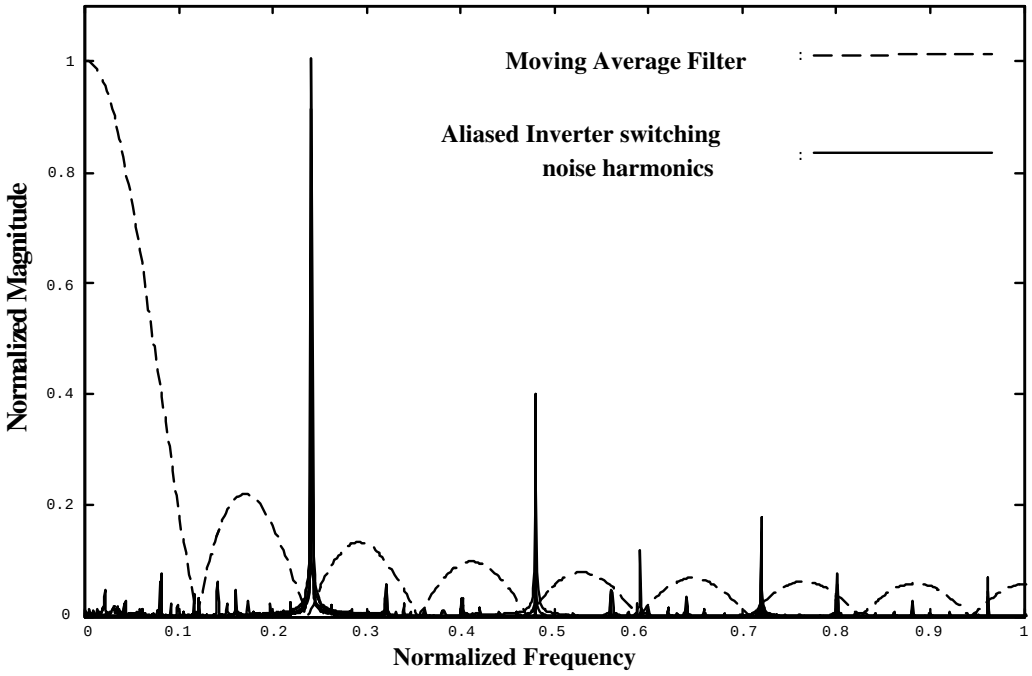

Figure 19: Scaled frequency spectrum of the inverter switching noise and frequency response of the comb FIR filter. Note, the notches of the filter are set to the positions of the noise harmonics

$$T[0] = (S[7:3] = Tap0) \cdot CLK[1]. \tag{91}$$

The signal *AccuRst* is used to clear the accumulator and has the same timing as $T[0]$. The clocks $T[1:3]$ is used to trigger multiplication, addition, and accumulation operations for each tap data. The clocks $T[1:3]$ should be activated at each tap stages through $Tap0$ to $Tap16$, and should be blocked during the time slot from 17 to 31 that are also generated by $S[7:3]$. Therefore the combinational logic for $T[1:3]$ is

$$T[1:3] = (Tap0 + Tap1 + \cdots + Tap16) \cdot CLK[2:4]. \tag{92}$$

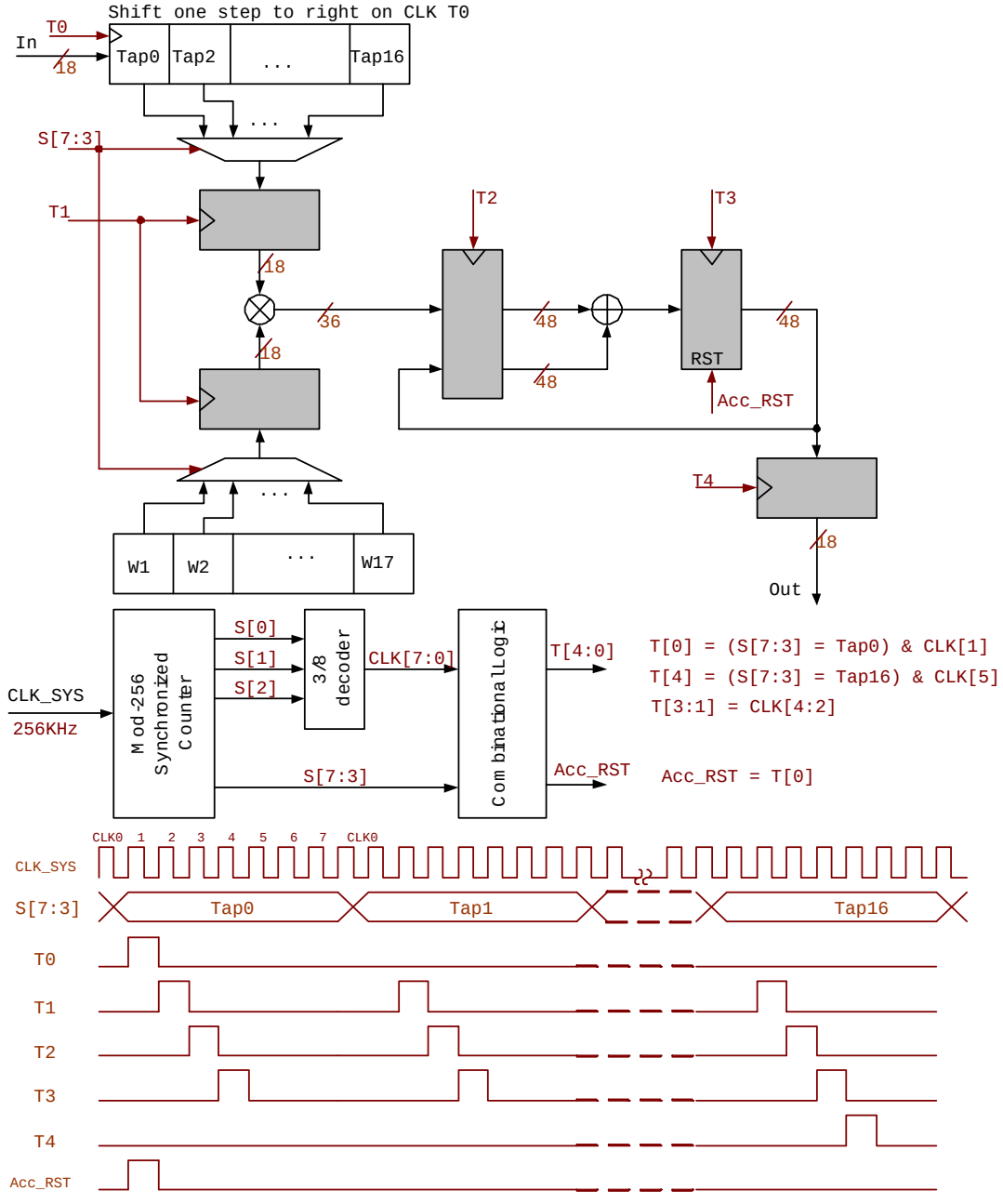

Figure 20: Multiplier - Accumulator FIR filter structure and pipeline stages organization.

The clock T[4] is used to output the result for one step convolution calculation conducted through Tap0 stage to Tap16 state, and should be triggered only in the Tap16 stage. Therefore the combinational logic for T[4] is

$$T[4] = (S[7:3] = Tap16) \cdot CLK[5]. \tag{93}$$

The output of the FIR filter is further down sampled by 10 since the amount of data computation and storage for transients is large assuming the 1khz sampling rate, and also the resolution of 100Hz is already enough for transient pattern modeling.

Block2: Transient Pre-Processing and Detection

Transient preprocessing and detection block achieve three functions, transient preprocessing, transient detect, and output preprocessing in the pipeline stage 2.1-2.3 respectively, shown in Fig. 21.

Stage2.1: Before the input data stream $Transients$ from block1 (Transient Pre-filtering and Down-Sampling) can be used to detect the transient event, it should be shifted to zero offset. To prevent false transient detection event in the multiple transients overlapping situation, the currently active transient tails should be cancelled. This is achieved by subtracting the signals $TailCur$, $TailSum$, and $Offset$ from the input data stream. These signals are calculated by the block4 (Post-processing array) and block5 (output processing) at last step and feedback to block2 at current step. The signal $TailSum$ is defined to be the summation of one step transient tail predictions of all active but not currently processed transients. The signal $TailCur$ is defined to be the one step transient tail prediction of the currently processed transient. After pre-processing, the output signal Xt looks like the currently processed

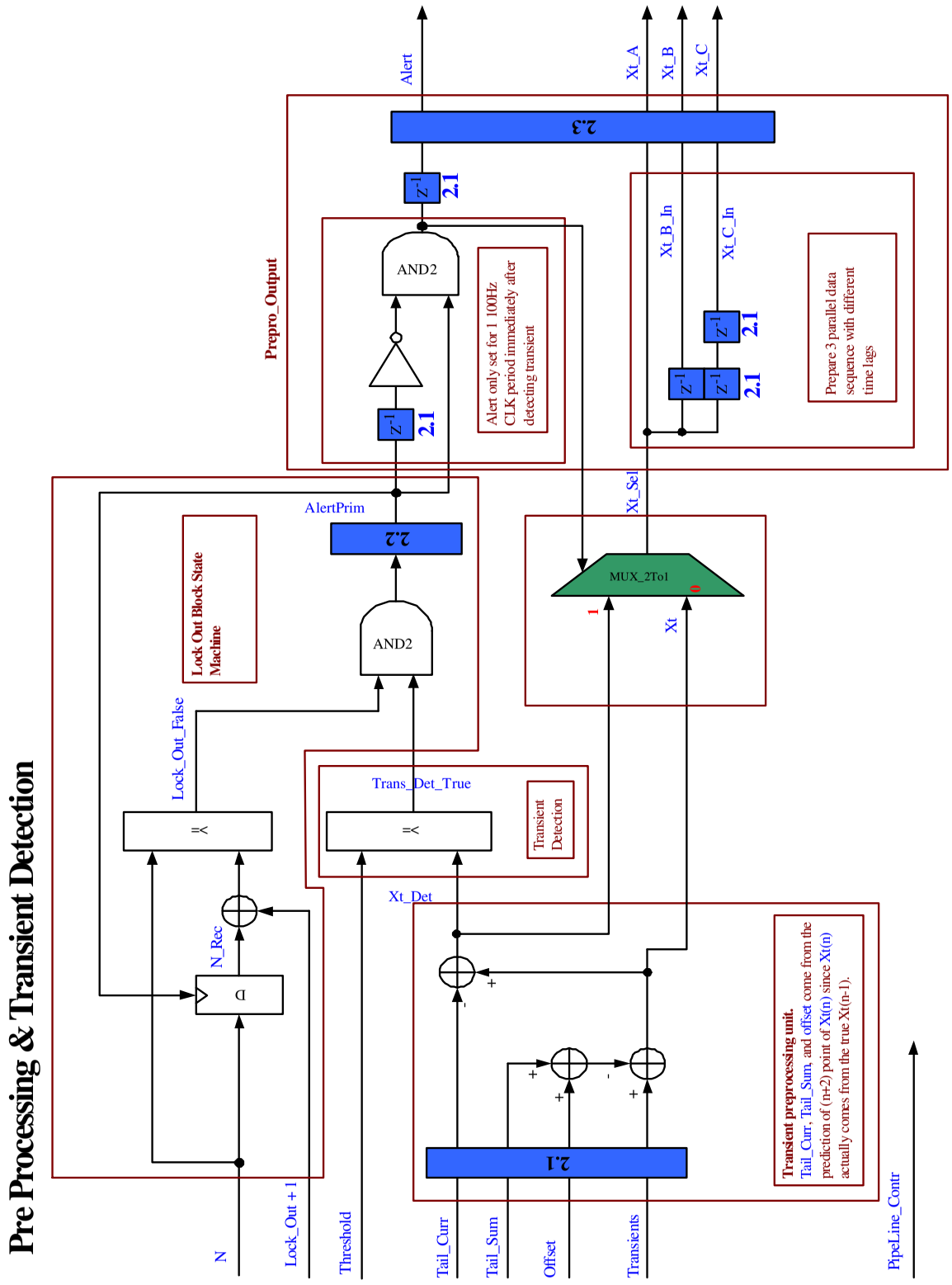

Figure 21: Transient preprocessing and detect block

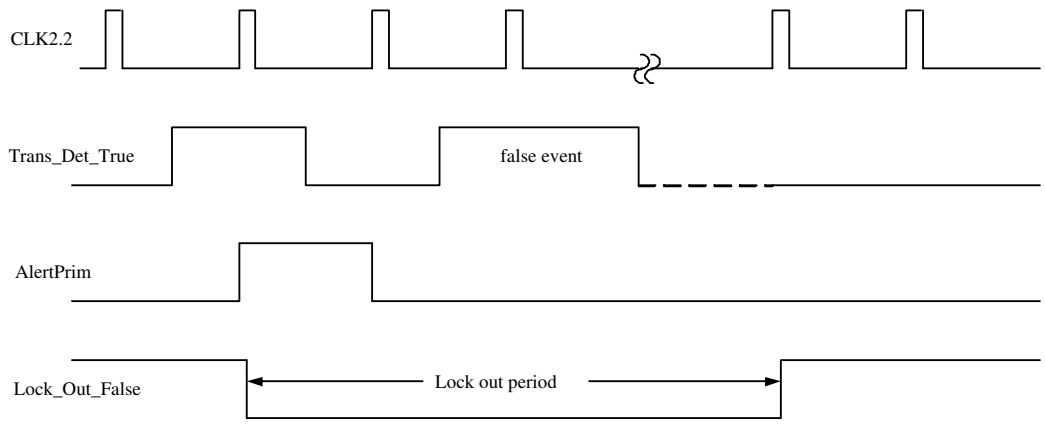

Figure 22: Lock out state machine timing chart

transient happens in zero offset and non-overlapping, which simplified the calculation in block3, the cluster calculation array. The signal $XtDet$ looks like the DC current around zero offset, which is required for effective transient detection with fixed threshold value.

Stage2.2: In this stage, the signal $XtDet$ is used for transient detection. Because the $TailCur$ feedback immediately after a new transient detection is generally not accurate because of not enough information gathered for currently processed transient, there should be a lock out period to prevent strange transient event detection happening. The lock out function is achieved by the lock out state machine as shown in Fig. 21, in which a D flip-flop is used to record the current detect time and accordingly block out $TransDetTrue$ to function until the lock out period finished. The timing of the relevant signals is shown in Fig. 22 for illustration. An important assumption made here is that there is no new transient happening during the lock out period, such that the lock out action is reasonable. The multiplexer in this stage is used to switch the pre-processed transient output data from Xt to $XtDet$ at the time of $Alert$ setting since when a new transient is detected, the "currently" processed transient becomes the "previous" active transient, and its transient tail $TailCur$ should be also cancelled from the output data stream.

Stage2.3: The transient detection signal *Alert* is required by the following blocks to be set only one step of SCWM processing. The combinational logic in (94) is used to generate the *Alert* signal that only set for one clock (100Hz) at the rising edge of $AlertPrim$,

$$Alert = \overline{(AlertPrim \cdot Z^{-1})} \cdot AlertPrim. \tag{94}$$

Three parallel paths of data XtA,XtB, and XtC are prepared to copy Xt with time lags of 0-1 respectively. These copies of Xt are used to find best match between the Xt and the templates stored in model since the data Xt may be not perfectly synchronized (aligned) to the template and could degenerate the recognition performance. *Alert* is delayed one step to be synchronized to XtB(one step delay to Xt) as if XtA (the actual Xt) is one step ahead and XtC(two steps delay to Xt) is one step delay.

Block3: SCWM Cluster Calculation Array

The cluster calculation array is the primary functional block implementing the sequential cluster weighted modeling algorithm as well as recursively finding a optimal scaling factor between the input transient data Xt and the clusters' templates $\vec{\mu}_m$. A top level block diagram is shown in Fig. 23. The input Xt is pre-processed in block2 with zero offset and with no previous transient tail overlapping, similar to the ideal training exemplars used for pre-initializing the SCWM model. Three parallel paths of data streams with different time-lag copies of Xt are present to the model at the same time, noted as $Xt[a:c]$. The reason for using multiple time-lag copies of the original data are that the data measured may be not aligned (synchronized) to the template data. The best matches between the data and the templates could be

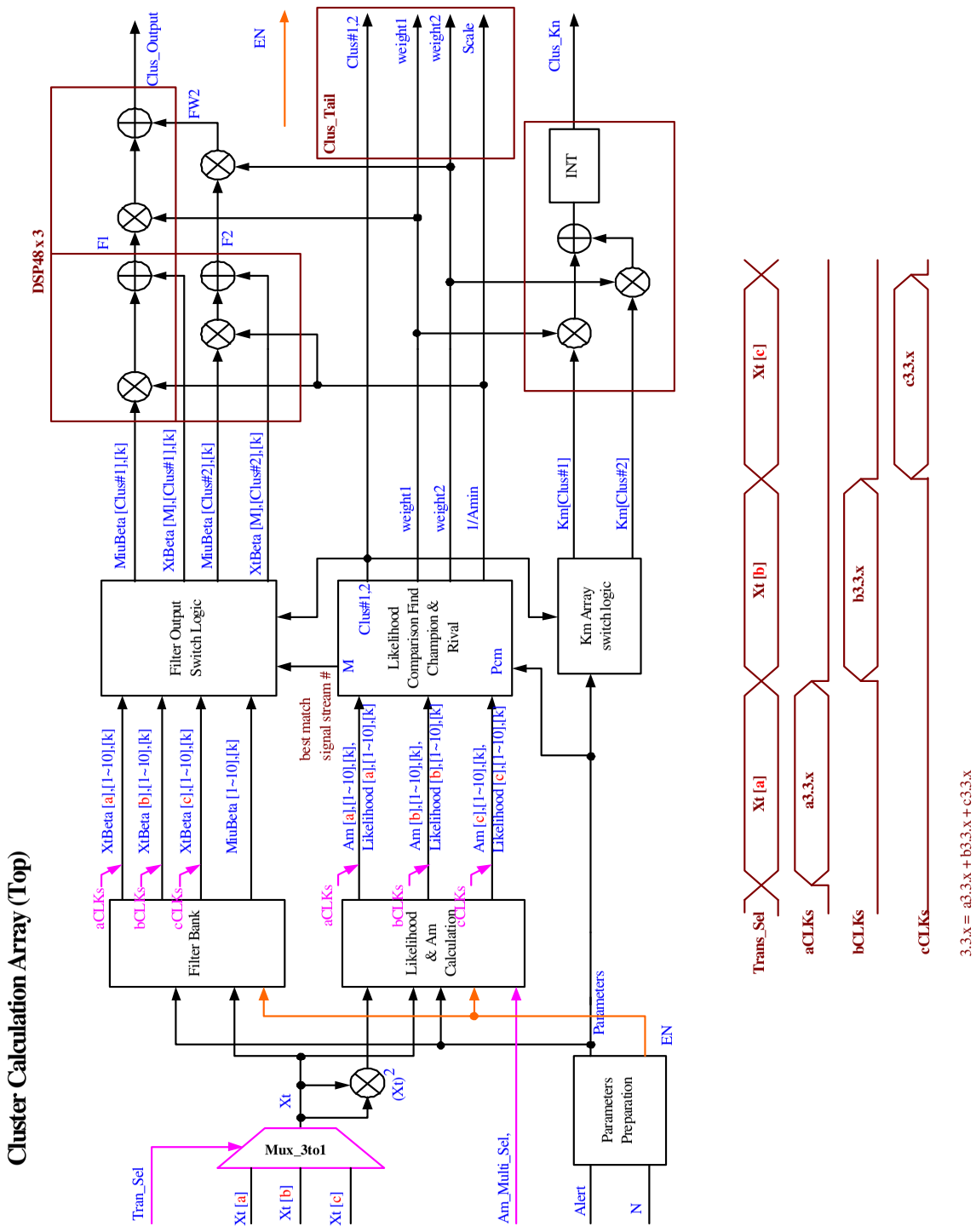

Figure 23: Cluster calculation array block (Top level)

found with multiple time-lag copies of the data. Three data streams share the same data path and resources at different time slots controlled by the multiplexing signal $TranSel$. The operations for different data streams are almost the same. However, some registers functioning as the accumulators must be separated for different data streams. These registers are clocked by $aCLK$, $bCLK$, $cCLK$ respectively. These clock signals are same to the original pipeline clocks (e.g., $aCLK3.2.2 = CLK3.2.2$ in Fig. 25) but are only triggered at their designated time slots.

Instead of doing the tail prediction, block3 only generates necessary tail prediction information, such as the best match cluster numbers, local model weights, and associated scale factor. The tail prediction task is transferred to the next block, post-processing array. If the tail prediction was performed in block3, then at each step of processing, the full vector of the remaining tail prediction would need to be transferred to the next block for storage, since it's not known when the next new transient will happen. The amount of data transfer may be large. If the tail prediction task is transferred to the next block, then all of these tail predictions involve scalar data. Block3 needs only to update these scalars stored in the next block for future use.

There are six functional sub-blocks, named "parameter preparation and reset signal generation" (shown in Fig. 24), "local model bank" (shown in Fig. 25), "likelihood and scaling factor calculation" (shown in Fig. 26, 27, 28), "likelihood comparison to choose best clusters" (shown in Fig. 29), "filter output switching and weights generation" (shown in Fig. 30), and a model output finalization shown in the top level diagram. Three pipeline operation stages are set up. The first stage is used to wait for the *Alert* signal and accordingly generate the reset signals for clearing the accumulators in the block. The model parameters are also prepared in the first stage. The second stage is used to compute the local model outputs and the cluster' likelihood and scaling factors. The third stage is used to find the best match (champion and rival) clusters and compute the local model weights and finalize the SCWM outputs.

Unlike the SCWM algorithm described in Chapter 2, in this design, only the two best matched clusters are used for generating the SCWM output because the contributions of the irrelevant clusters are extremely small. The sub-blocks will be addressed in the following sub-sections.

Parameter Preparation and Reset Signal Generation

The block diagram of this sub-block is shown in Fig. 24. The first function of this sub-block is to generate the reset signal. The reset signal is triggered by the *Alert*, which indicates that a new transient has been detected. There are two parallel *Alert* monitoring and reset generation signals, $EN1$ and $EN2$. The signal $EN1$ reset between the interval of *Alert* setting and the second pipeline clock $CLK3.1.2$ using the same circuit of generating *Alert* in block2. The signal $EN1$ is an absolute reset signal to clear the accumulators whenever a new transient comes. The signal $EN2$ generates a relative reset signal by monitoring the effective parameter memory access address $AddrIn$ which is the difference between the current time and the start time of a new transient. The signal $EN2$ guarantees that block3 only work (when $EN2$ is high) in the period within the effective SCWM modeling dimension (recall the dimension D for SCWM modeling mentioned in Chapter 2). If there is no transient overlap, $EN2$ is sufficient for resetting the accumulators. However, if overlapping transients occur, $EN2$ will not reset when the new transient comes because the effective modeling dimension D has not expired. Therefore, $EN1$ finds important function for reset logic in the transient overlapping situation. The combined reset signal EN guarantees the correct action in both the overlapping and non-overlapping situations. The reset logic for non-overlapping and overlapping cases are all shown in the time diagram in Fig. 24.

Another function of this sub-block is to prepare the model parameters for other sub-blocks. Because the memory resource of Virtex4 is sufficient, any variables that

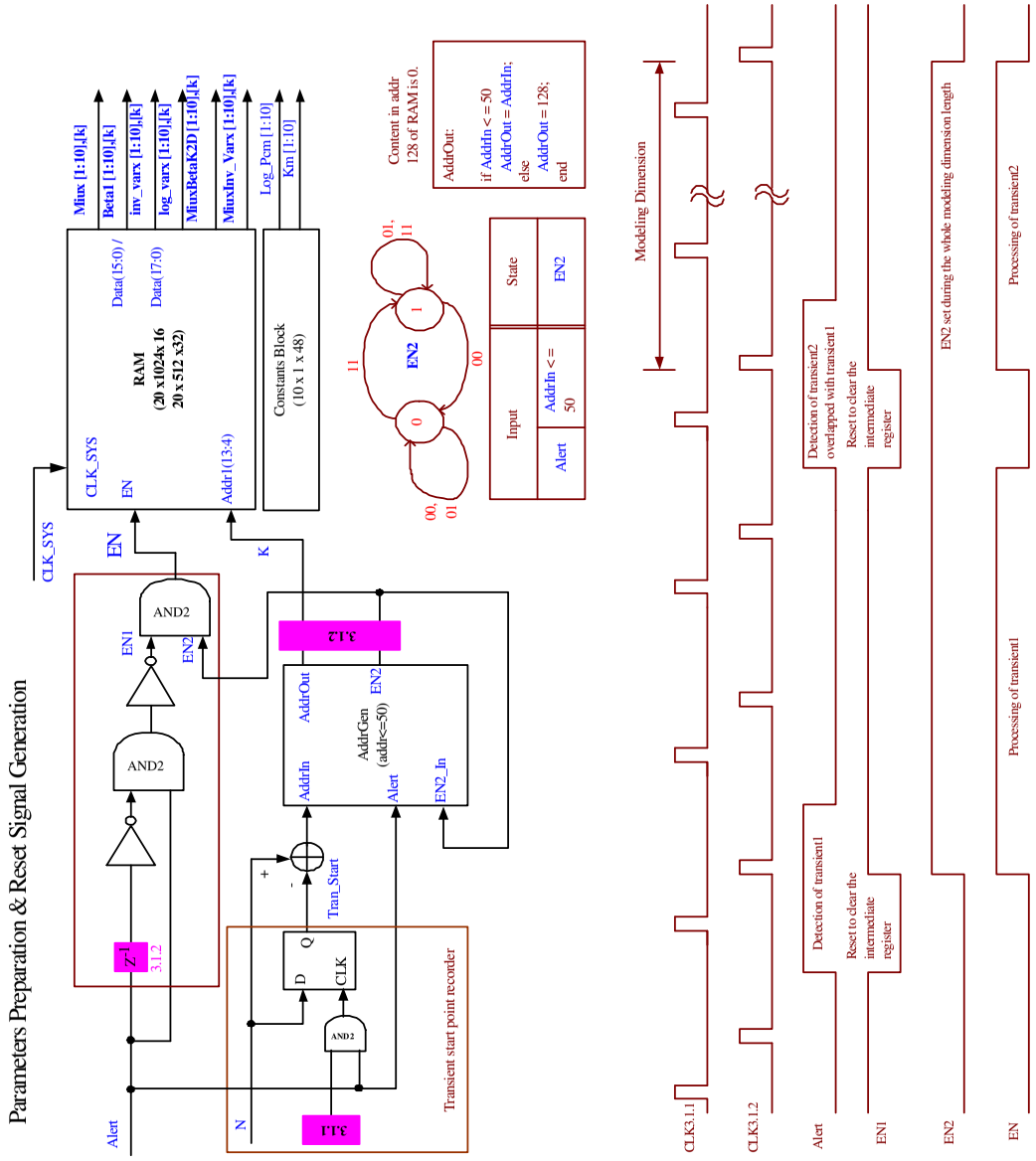

Figure 24: Cluster calculation array block (2): Block reset logic and model parameters preparation.

can be pre-calculated are stored in memory. The parameters are organized in parallel in the memory. Although this is not a effective usage of the memory block, it reduces the complexity of timing as long as the memory resource is sufficient.

Local Model Bank

The local model bank block diagram is shown in Fig. 25. This bank achieves the SCWM local modeling recursively, i.e.,

$$XtBeta[k] = XtBeta[k-1] + Xt[k] \cdot Beta[K], \tag{95}$$

where $Beta$ represents the local model parameter $\vec{\beta}_m$ and $XtBeta$ is a accumulator to store the accumulated results of $Xt[k] \cdot Beta[K]$. Note that the tail prediction item $\vec{\beta}^T_{m,(k+1:D)} \vec{\mu}_m^{(k+1:D)}$ in the original algorithm, referenced as the signal $MiuxBetaK2D$ in the figure, is not calculated here since the value does not depend on the input signals and can be pre-calculated and stored. Also note that these two signals $XtBeta$ and $MiuxBetaK2D$ are not added here which is different from the original algorithm. The reason is that only the best matched local models will be picked up in the following operation stages to add these two number and finalize the local model output. These two signals are left separate here since there is still no evidence that the associated cluster is one of the best matched clusters to the input transient. One DSP48 with accumulators is used to implement (95). The accumulators are separated for different data streams $Xt[a]$, $Xt[b]$, and $Xt[c]$, and clocked by $aCLK$3.2.2, $bCLK$3.2.2, $cCLK$3.2.2 separately.

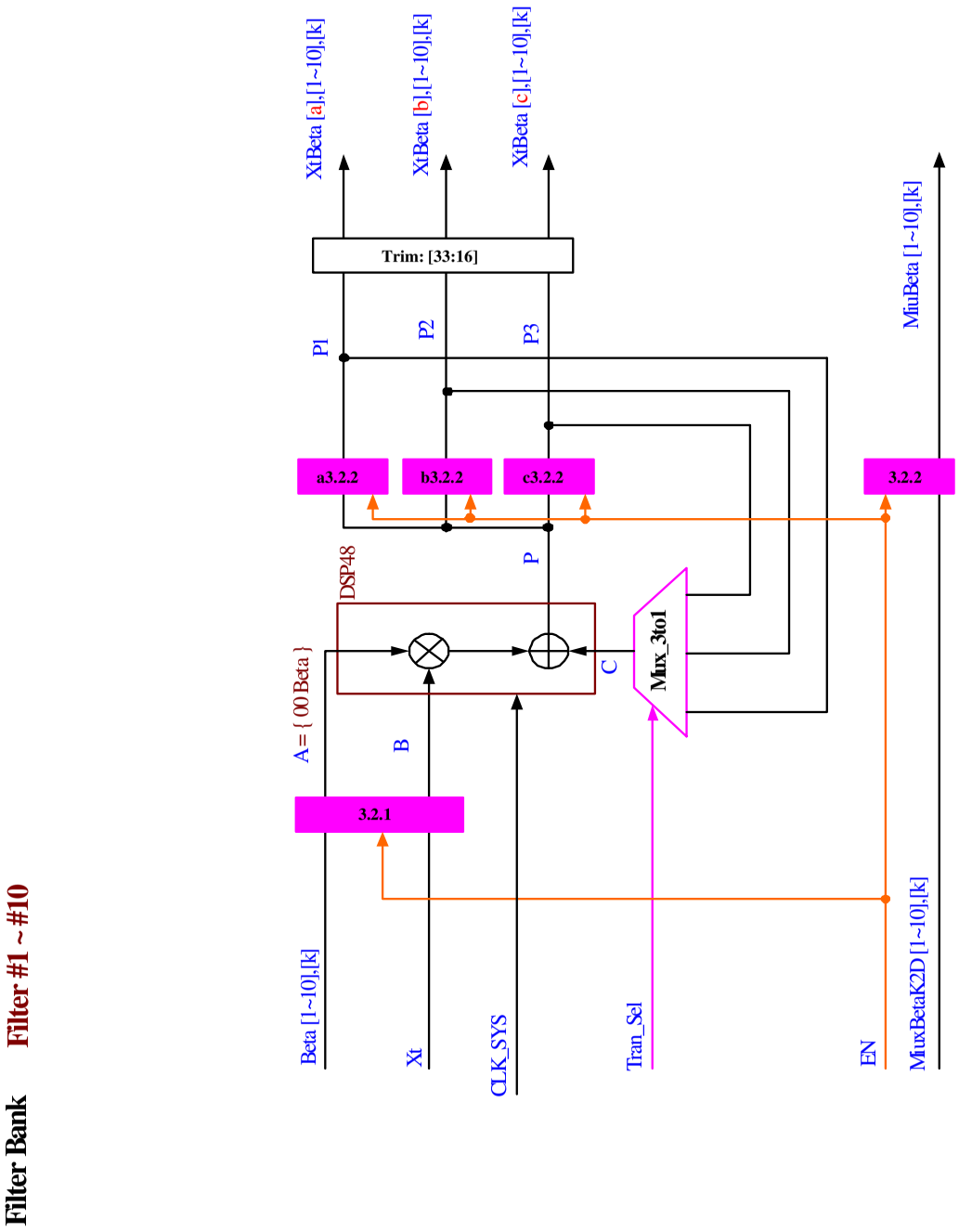

Figure 25: Cluster calculation array block (3): filter bank.

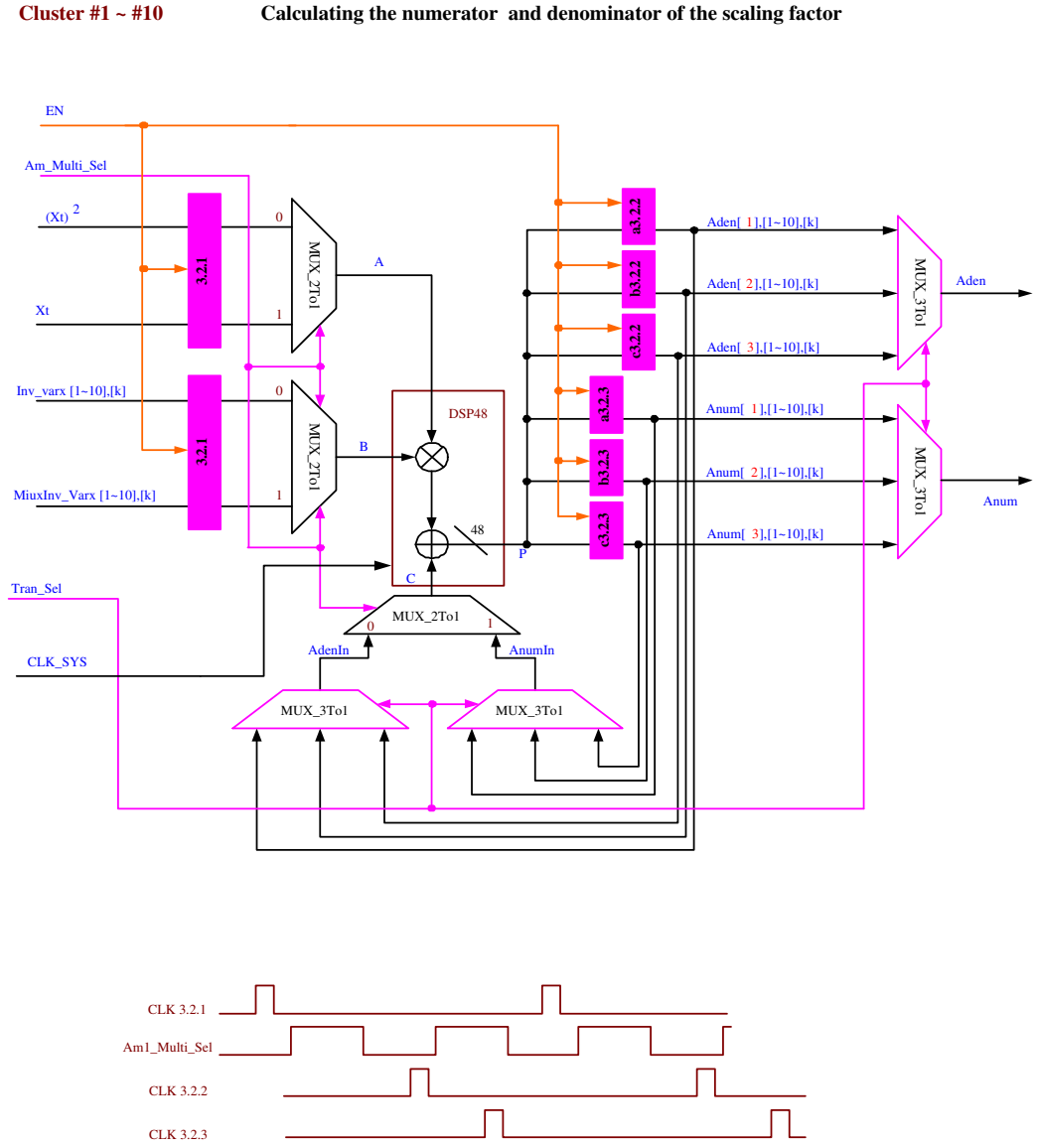

Figure 26: Cluster calculation array block (4): Transient scaling factor calculation (1).

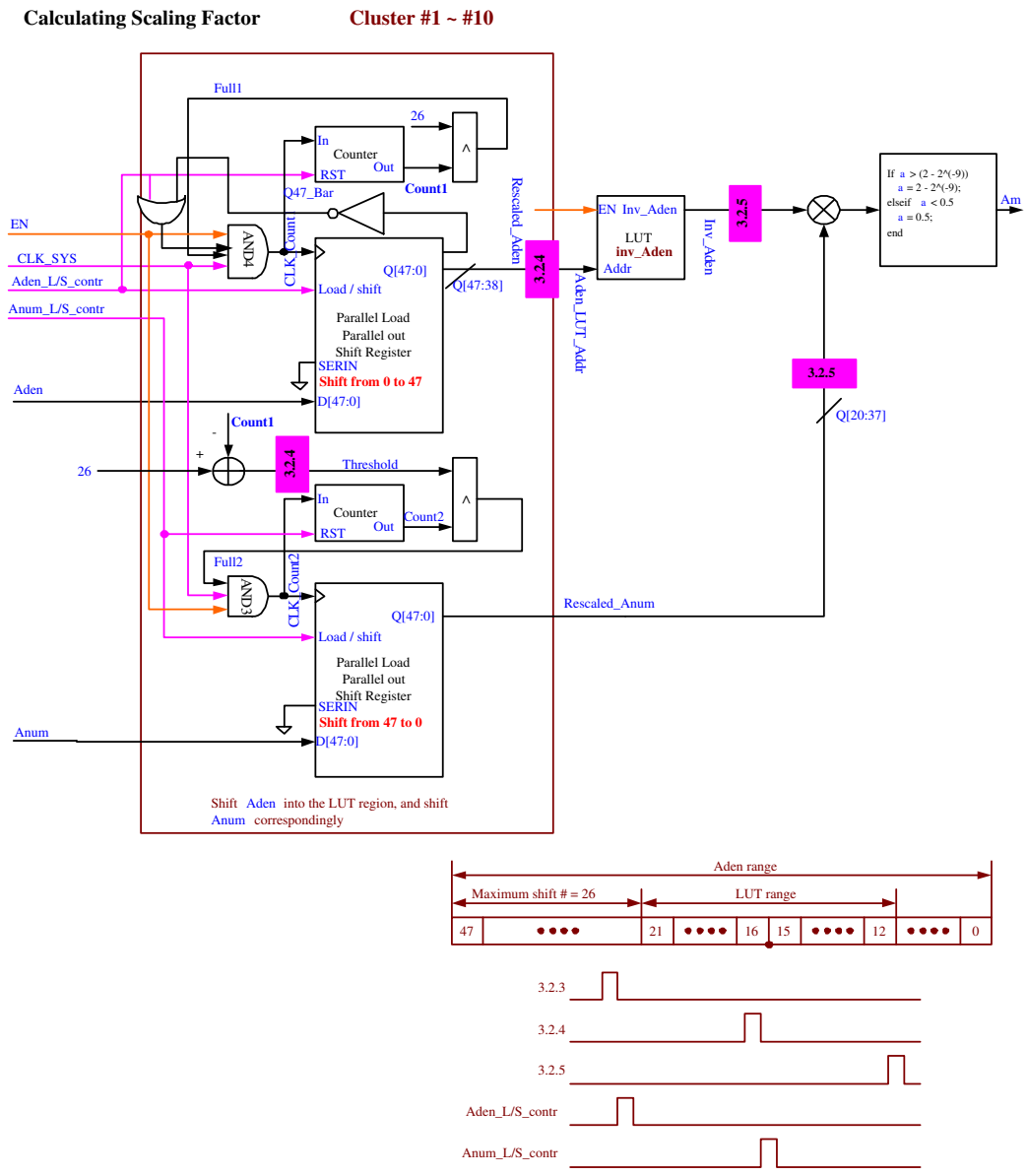

Figure 27: Cluster calculation array block (5): Transient scaling factor calculation (2).

Likelihood and Scaling Factor Calculation

The block diagram of this sub-block is shown in Fig. 26 and Fig. 27 for scaling factor ($Amin$) calculation (refer to equation (89) in last chapter), and shown in Fig. 28 for log-likelihood $p(\vec{x}_{(1:k)}|c_m)$ calculation. There are three steps in this sub-block in calculating the relevant numbers. The first step (between $CLK3.2.1$ and $CLK3.2.3$) is to calculate the numerator ($Anum$) and denominator ($Aden$) of scale factor $Amin$, as shown in Fig. 26. The second step (between $CLK3.2.3$ and $CLK3.2.5$) is to calculate the scaling factor $Amin$ using a look up table (LUT) and a point shifting state machine to match the denominator to the LUT range, as shown in Fig. 27. The third step is to calculate the log likelihood for the scaled transient data between $CLK3.2.5$ and $CLK3.2.10$, as shown in Fig. 28. Because of the finite multiplier resource, the DSP48 is used in multiplexing for multiple calculation requirements, which is controlled by the signal $AmMultiSel$.

As shown in Fig. 27, the calculated scaling factor denominator $Aden$ needs to be re-scaled the value to match the LUT range. The range of $Aden$ value can extended to 2^{30} and can not be effectively expressed by any LUT with finite memory resources. The idea of fitting $Aden$ to the LUT range meanwhile does not affect the final result is to shift the point of $Aden$ to suitable position, and accordingly shift the point of the numerator $Anum$. A state machine is used to find the suitable shift offset of $Aden$ and shift $Aden$ to the range of LUT. The shifting offset is further transferred to another state machine to accordingly shift the point of $Anum$. The state machine is composed of a parallel load, parallel output shift register, a counter, and a clock enable logic. After loading the data (the counter is also cleared at the loading time), the shift register will continuously work to shift the number to MSB direction until the highest bit $Q[47]$ is one, i.e., the first non-zero bit is found. Then the highest 10 bits vector is used as the LUT address (our LUT is with resolution of 1024). The shift offset is recorded in counter output $Count1$, and is transferred to another state

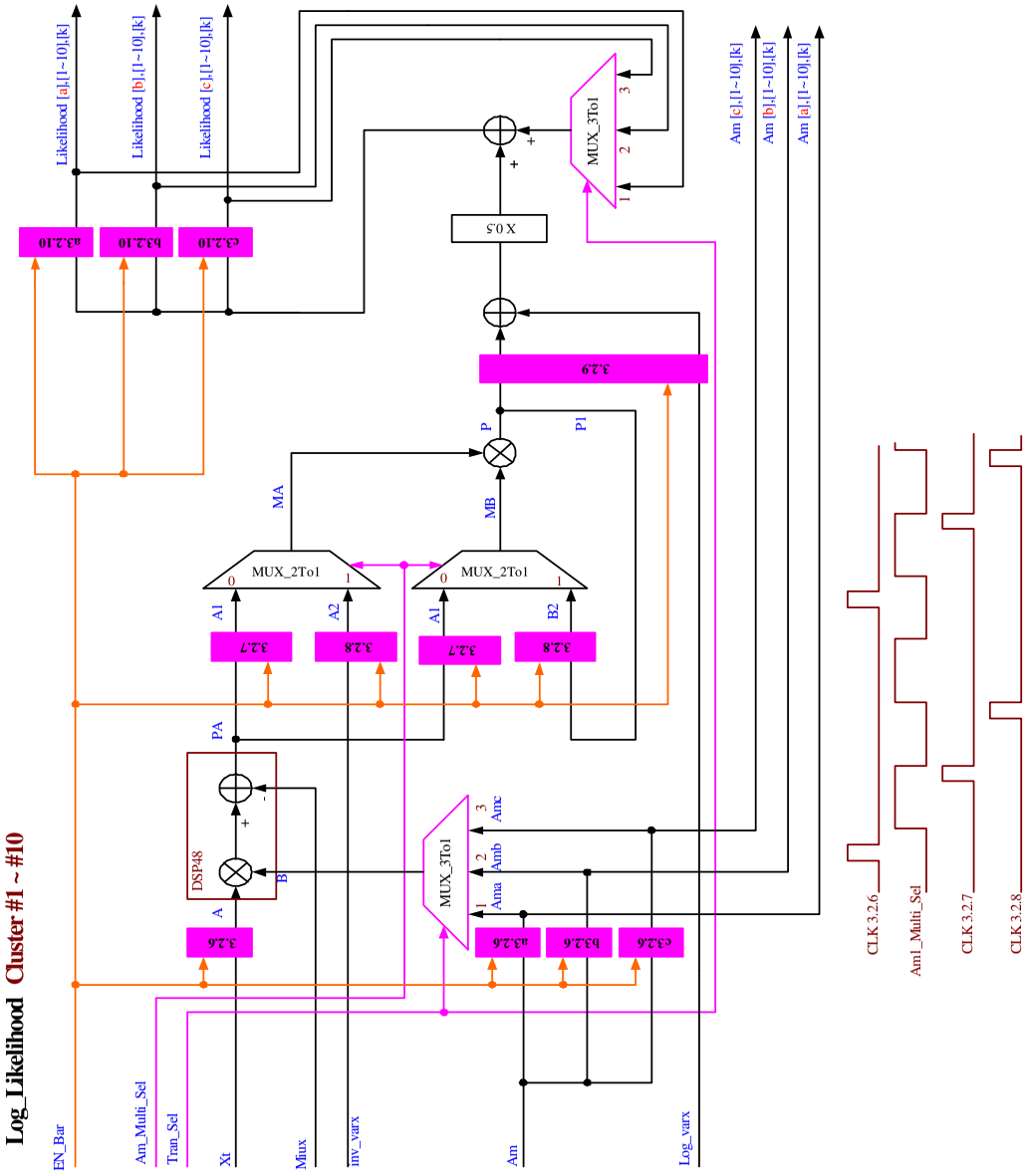

Figure 28: Cluster calculation array block (6): Cluster likelihood calculation.

machine to shift *Anum* accordingly. Note that the action of taking the highest 10 bits of the shift register output for *Aden* as the LUT address actually shifts the point of *Aden* in the LSB direction until entering into the LUT range, which is between 21st and 15th bits (Look at the illustration chart in the lower right of Fig. 27). Therefore *Anum* should be accordingly shift to LSB direction. The offset of shifting *Anum* is derived below.

Suppose the largest number of non-zero bits for *Aden* is K. The action of taking the highest 10 bits output of shift register as the LUT address(after the K'th bit being shift to 47th bit) actually shifts the K'th bit to the 21st bit (MSB of LUT range) Therefore *Anum* should be shifted to LSB direction with offset

$$K - 21 = (47 - 21) - Count1 = 26 - Count1. \tag{96}$$

An upper bound of 26 should be set for $Count1$ in the case if the *Aden* is already in the LUT range (less than 21st bit) before shifting.

Likelihood Comparison to Choose Best Clusters

The block diagram of this sub-block is shown in Fig. 29. Usually only a few clusters are relevant to a specific transient input. If the original CWM algorithm is implemented directly, all of the irrelevant clusters use computational resources (e.g., multiplier) and contribute nothing to the final model output. In order to save resources and improve computational efficiency, it is worth of setting up a separate sub-block to find the best matched clusters. In the dissertation, two best clusters, named "champion" and "rival", are used for generating the final model output.

The sub-block uses three steps to achieve the problem. The first step (between $CLK3.3.1$ to $CLK3.3.2$) is to input the updated likelihood data sets and find the champion using a parallel comparison tree. Each data set waiting to be compared

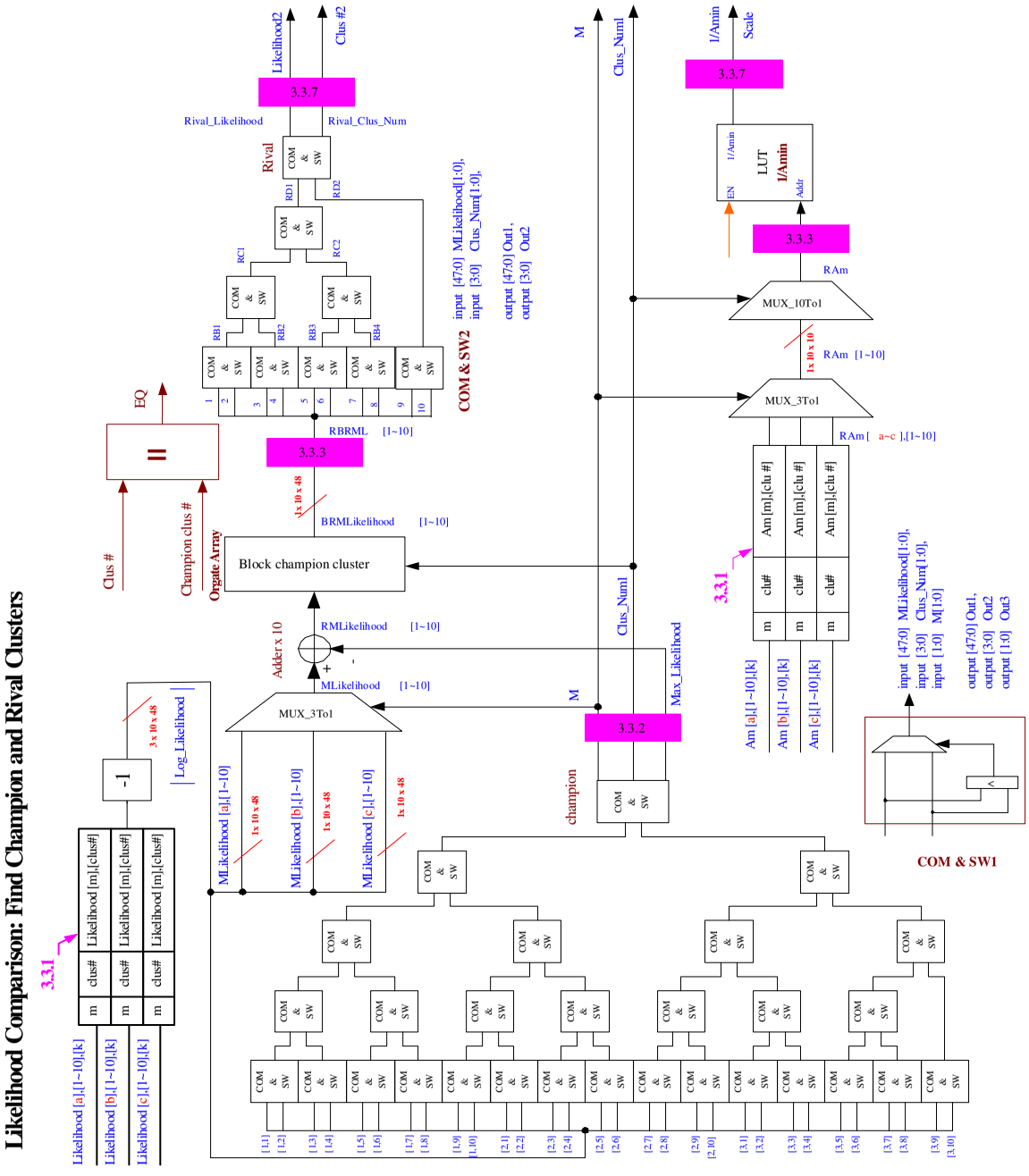

Figure 29: Cluster calculation array block (7): Compare and choose champion and rival clusters with highest likelihood to transient.

has the index of cluster number and data stream number corresponding to the data stream $Xt[a : c]$. The champion is recorded with the associated log likelihood and the cluster number and data stream number.

At the second step (between $CLK3.3.2$ to $CLK3.3.3$), the remaining data are prepared for finding the rival cluster. First the best data stream information is used to select all of the data sets on that data path (corresponding to all of the clusters). Then the champion cluster number information is used to set the champion cluster's likelihood to all one values (note the absolute value of the log likelihood is used. Therefore, the biggest value of likelihood imply the worst clusters.) This is achieved by using a equality comparator and an *or* gate array added to the data path. Also in this step, the relevant clusters' likelihood are re-scaled by the maximum likelihood to prevent singular values when translating the log likelihood to the normal probability densities.

At the third step (after $CLK3.3.3$), the rival is selected using the comparison tree. And the final scaling factor for model output $Scale = 1/Amin$ is generated with the help of champion cluster information and a reciprocal LUT. All of the information about the champion and rival (cluster number, likelihood, data stream path number) and the scale factor are ready for using to the following blocks.

Local Model Output Switching and Weights Generation

The block diagram of this sub-block is shown in Fig. 30. This block serves two functions. First the champion and rival local model outputs are selected with the champion and rival cluster information. The other function is to generated the local model weights using the likelihood information. Assume the champion is cluster 1 and the rival is cluster 2. In this case, the weight of the rival local model is

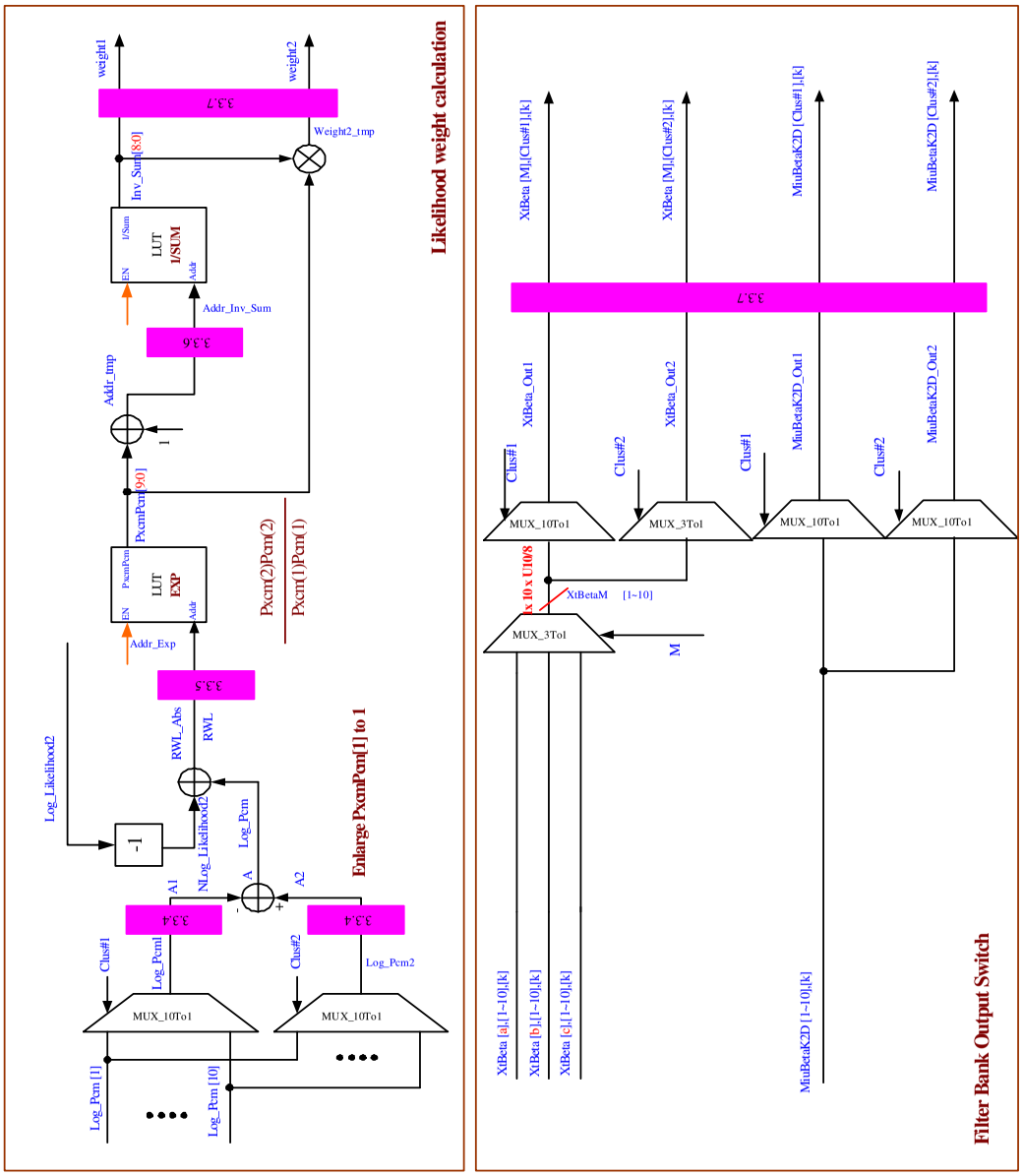

Figure 30: Cluster calculation array block (8): Select champion and rival filters, and generate filters' weights.

$$
\begin{aligned}
w_2 &= \frac{p(x|c_2)P(C_2)}{p(x|c_1)P(c_1)+p(x|c_2)P(c_2)} \\
&= \frac{\frac{p(x|c_2)P(C_2)}{p(x|c_1)P(c_1)}}{1+\frac{p(x|c_2)P(C_2)}{p(x|c_1)P(c_1)}}.
\end{aligned} \tag{97}
$$

Remember that the likelihood of the rival has been scaled by the champion likelihood in last sub-block and has the value of $\frac{p(x|c_2)}{p(x|c_1)}$. It is natural to estimate the rival's relative cluster-weighted likelihood as $\frac{p(x|c_2)P(C_2)}{p(x|c_1)P(c_1)}$. The relative cluster-weighted likelihood of cluster 1 is one at this time. To save multipliers, the relative cluster-weighted likelihood $\frac{p(x|c_2)P(C_2)}{p(x|c_1)P(c_1)}$ is calculated in log domain using adders. Another advantage of calculating the relative cluster-weighted likelihood is that only one LUT is needed to calculate the champion cluster weight $w_1 = \frac{1}{1+\frac{p(x|c_2)P(C_2)}{p(x|c_1)P(c_1)}}$, and the rival cluster weight is $w_2 = w_1 \cdot \frac{p(x|c_2)P(C_2)}{p(x|c_1)P(c_1)}$.

After the champion and rival local models outputs and weights are prepared, the SCWM output is finally generated (see in the top level diagram Fig. 23) and fed with other useful information to block4 for further processing. It should be noted here that the generation of the transient duration information $ClusKn$ in the top level of the block3 diagram can be simplified to only record the champion cluster's parameter Km since the TRC model is not sensitive to the exact value of the transient duration around the time when the transient is over. The transient tail values is almost the same as the transient steady state value when a transient will finish. Therefore the transient tail contribution can be either added to the offset estimation or to the transient tail predictions (either $TailSum$ or $TailCur$).

Block4: Transient Post-Processing

The Post-processing block is used to track the active transient status, store and update the intermediate transient processing information, and generate the tail prediction and pre-calculated offset information that will be feed to block2 and block5 for further processing.

Three transient processing lines are set up assuming that no more than three transients will happen in overlapping. Each transient processing line record the transient status ($PreActive, PostActive$) and the transient data input from block3, including transient steady state estimate ($Output$), transient start time($Start$), transient length (Kn), and transient tail prediction information ($TailClusNum1, TailClusNum2$, $TailWeight1, TailWeight2, TailScale$). A pointer $Curr[1:0]$ is used to indicate which transient processing line corresponds to the transient currently being processed in block3.

The functions of post processing block are separated and organized into four pipeline stages, including "new processing line initialization" (stage 4.1), "transient processing line data update " (stage 4.2), "transient tail recovering and state update" (stage 4.3), and "transient processing line data output" (stage 4.4), as shown in Fig. 31, Fig. 32, Fig. 33, and Fig. 34 respectively.

Stage 4.1: The first stage of operation (Fig. 31) is to determine whether a new transient is detected by monitoring *Alert*, and accordingly to initialize an empty processing line to respond to the new detected transient. It is required that *Alert* can only be set for one step when the new transient is detected (This is achieved in block2). Otherwise, block4 will continuously open new processing lines to respond to the same transient. A processing line is determined to be empty or not by monitoring the input *PostActive* updated at last step. There is also a priority order between different empty lines for opening a new processing line (assume Line1 > Line2 > Line3).

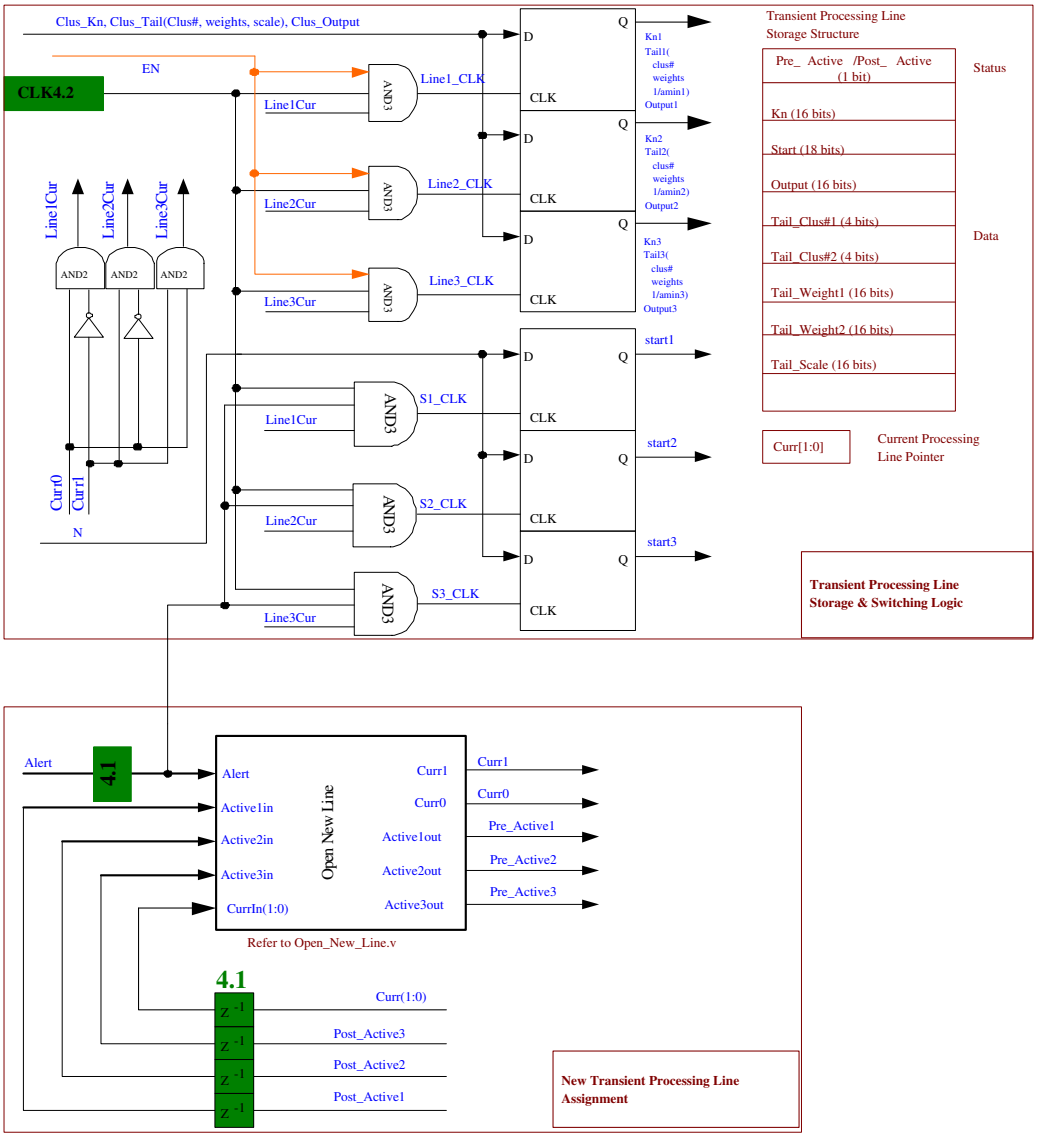

Figure 31: Post processing block diagram 1 – Stage 4.1 new processing line initialization, and Stage 4.2 transient processing line data update.

Otherwise multiple empty lines may be initialized for the same detected transient. When an empty line is found, the status bit $PreActive$ is set and the pointer is set to point to this line. The resultant combinational logic of first stage is

$$PreActive1 = Alert + PostActive1 \quad (98)$$

$$PreActive2 = Alert \cdot PostActive1 + PostActive2 \quad (99)$$

$$PreActive3 = Alert \cdot PostActive1 \cdot PostActive2 + PostActive3 \quad (100)$$

$$Curr1 = (\overline{A1} \cdot \overline{A2} \cdot \overline{A3} \cdot \overline{CurrIn1})) \cdot (A1 \cdot \overline{A2} \cdot \overline{A3}) \quad (101)$$

$$Curr0 = (\overline{A1} \cdot \overline{A2} \cdot \overline{A3} \cdot \overline{CurrIn0}) \cdot (\overline{A1} \cdot A2 \cdot \overline{A3}) \quad (102)$$

$$A1 = \overline{PreActive1} \cdot PostActive1 \quad (103)$$

$$A2 = \overline{PreActive2} \cdot PostActive2 \quad (104)$$

$$A3 = \overline{PreActive1} \cdot PostActive3 \quad (105)$$

Stage 4.2: (Fig. 31) In this stage, the updated transient data information from block3 is input to a transient processing line indicated by the pointer $Curr[1:0]$. When the signal $Alert$ is set, the signal $LineCur$, which is generated from the pointer signal $Curr[1:0]$, functions as a clock enable signal to allow the "current" processing line to record the start time of the transient. The signal EN is generated from block3 functioning as an enable or disable signal to turn on or terminate data transferring from block3 to block4.

Stage 4.3: This stage implements two functions, updating the processing lines status according to updated transient duration information (Fig. 32) and recovering the transient tail prediction using the cluster number index, the cluster weight, and the scale factors (Fig. 33).

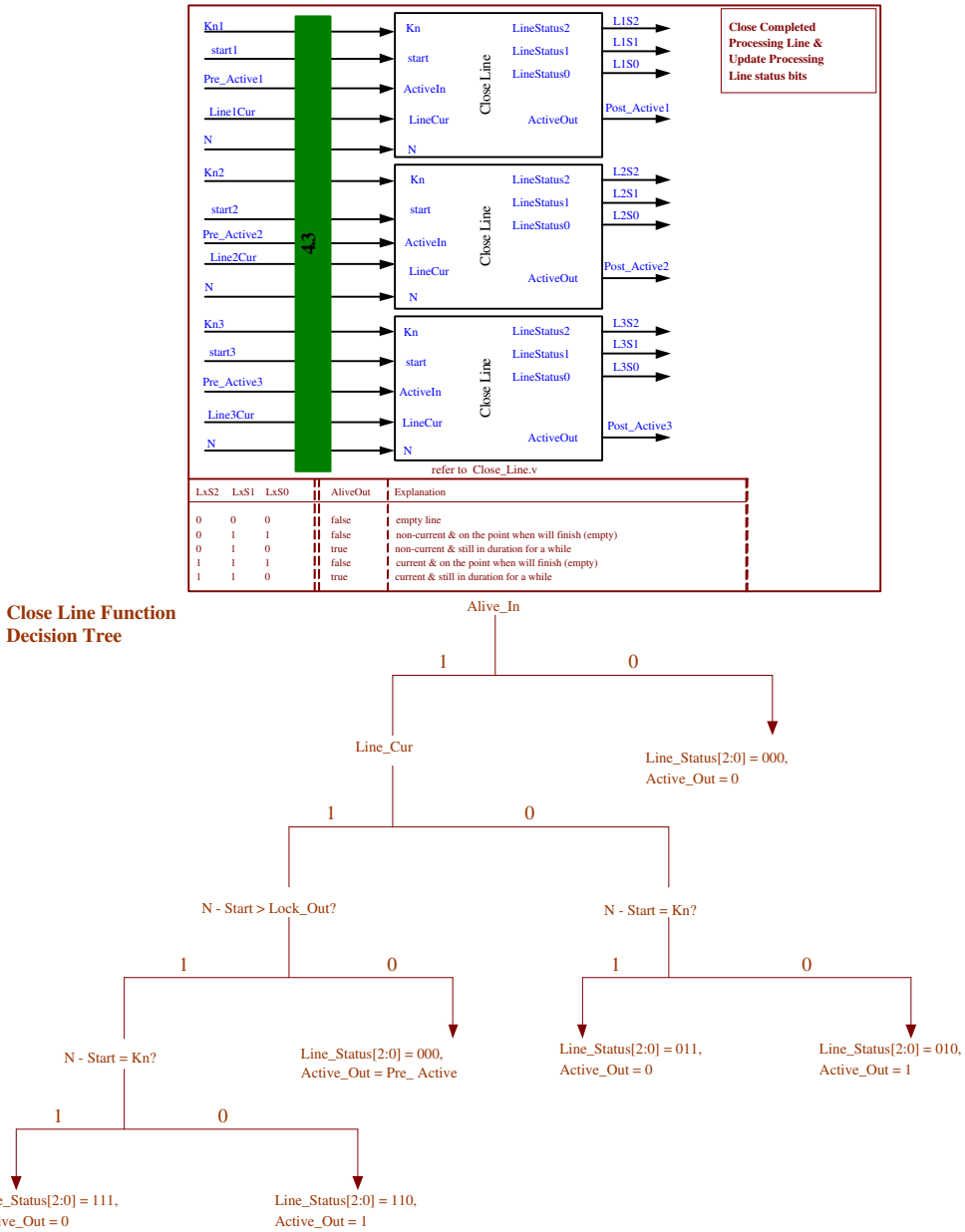

Figure 32: Post processing block diagram 2 – Stage 4.3 transient tail recovering and state update.

LineStatus2	LineStatus1	LineStatus0	PostActive	Description
0	0	0	0	empty line
0	0	0	PreActive	in lock out period
0	1	1	0	non-current and finish at this step
0	1	0	1	non-current and still in duration
1	1	1	0	current and finish at this step
1	1	0	1	current and still in duration

Table 3: Close line function outputs encoding and description.

In the first function, the transient status is updated depending on the transient duration information, such as "$N - Start > LockOut$?" and "$N - Start < Kn$?", and the processing lines' current statuses ($PreActive$). The output transient status is detailed into six situations and encoded with the output line status code $LineStatus[2:0]$ and $PostActive$. The status codes are listed and described in table 3, and are also shown in Fig. 32. A close line function decision tree, as shown in Fig. 32, is used to map the input line status information to the output line status codes. In the tree, note that the condition "$N - Start = Kn$?" is equivalent to "$N - Start \geq Kn$?" since whenever "$N - Start = Kn$" is satisfied, the signal $PostActive$ will be cleared and drive the following processing line status to the top right-most leaf at the next step. Therefore the condition "$N - Start > Kn$" will never happen. The encoded bits $LineStatus[2:0]$ (also noted as $LxS[2:0]$) are

used in the next stage to generate the processing line selecting signals for calculating $TailSum$, $TailCur$, and offset.

In the second function, one step of tail prediction for each processing line is performed using the transient tail information including cluster number index, cluster weight, and the scale factor. The advantage of recovering the tail prediction in block4 instead of in block3 is that the amount of information transferred from block3 is significantly less. For example, imagine that transient A is currently being processed in block3. If the transient tail prediction is performed in block3, a full vector of the remaining transient tail data needs to be transferred to block4 at each step because the currently being processed transient is terminated when a new transient is detected. Meanwhile, if the tail prediction is recovered in block4, all of the information needed to be transferred at each step are scalars. The tail recovery unit implements one step tail prediction,

$$\begin{aligned} Tail(N-start+2) &= Scale \cdot (TailWeight1 \cdot \vec{\mu}_{TailClusNum1}(N-start+2) \\ &\quad + TailWeight2 \cdot \vec{\mu}_{TailClusNum2}(N-start+2)). \end{aligned} \tag{106}$$

Unlike the tail prediction defined in Chapter 2 where full set of clusters are needed, only the two most relevant clusters with the highest likelihood are used to generate the output for the purpose of saving computation cost and resources. The data path and calculation resources (parameter memory block and multiplier) for (106) are shared between two clusters and three transient processing lines. The timing of multiplexing the data path and the pipeline clocks organization is shown in Fig. 33. The clocks allocation is also summarized in Table 4. The recovered tails are used in next stage for calculating $TailSum$, $TailCur$.

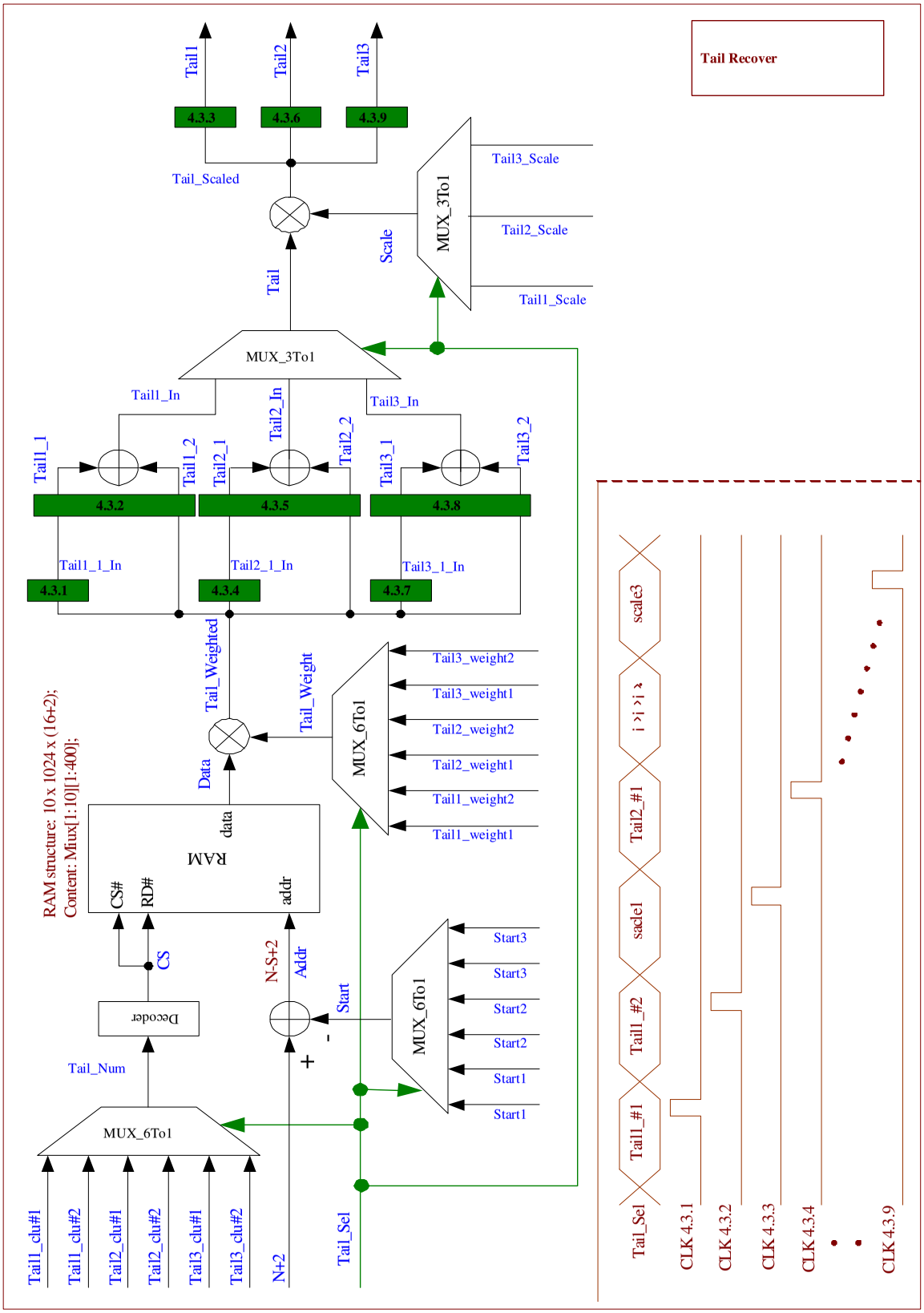

Figure 33: Post processing block diagram 3 – 4.3 transient processing line data output.

	$\vec{\mu}_{ClusNum1} \cdot Weight1$	$\vec{\mu}_{ClusNum1} \cdot Weight1$	Adding and Scaling
Line1	CLK4.3.1	CLK4.3.2	CLK4.3.3
Line2	CLK4.3.4	CLK4.3.5	CLK4.3.6
Line3	CLK4.3.7	CLK4.3.8	CLK4.3.9

Table 4: Tail recovery: Data path multiplexing and pipeline clocks allocation between different clusters and lines.

Stage 4.4: In this stage, the number such as the summation of all "active but non-currently being processed" transient tail predictions ($TailSum$), the "active and currently being processed" transient tail prediction ($TailCur$), and the pre-calculated offset ($PreOffset$) are generated depending on the information of the recovered $tail[1:3]$ and the processing line status $LxS[2:0]$ calculated at stage 4.3, and also depends on the transient steady state estimate ($Output[1:3]$) stored in the processing line. A processing line's tail will contribution to the summation of $TailSum$ if the associated transient is still in progress and is not currently being processed, corresponding to line status code $LxS[2:0] = [0\ 1\ 0]$. Meanwhile the tail will contribute to the signal $TailCur$ if the associated transient is in progress and is currently being processed by block3, corresponding to line status code $LxS[2:0] = [0\ 0\ 1]$.

The offset is summed from the $Output$ of each processing line if the associated transient will finish at current step. A processing line's *output* can only be added to the offset one time at this time point, and the resultant $PostActive$ is cleared (note in the leaf in the decision shown in Fig.32 where the condition "$N-Start = Kn$" is true). Afterwards, the line cease to function and is empty at next stage. The added $Output$ is further accumulated with the $OffsetFB$ feedback from block5 which represents the current system offset information.

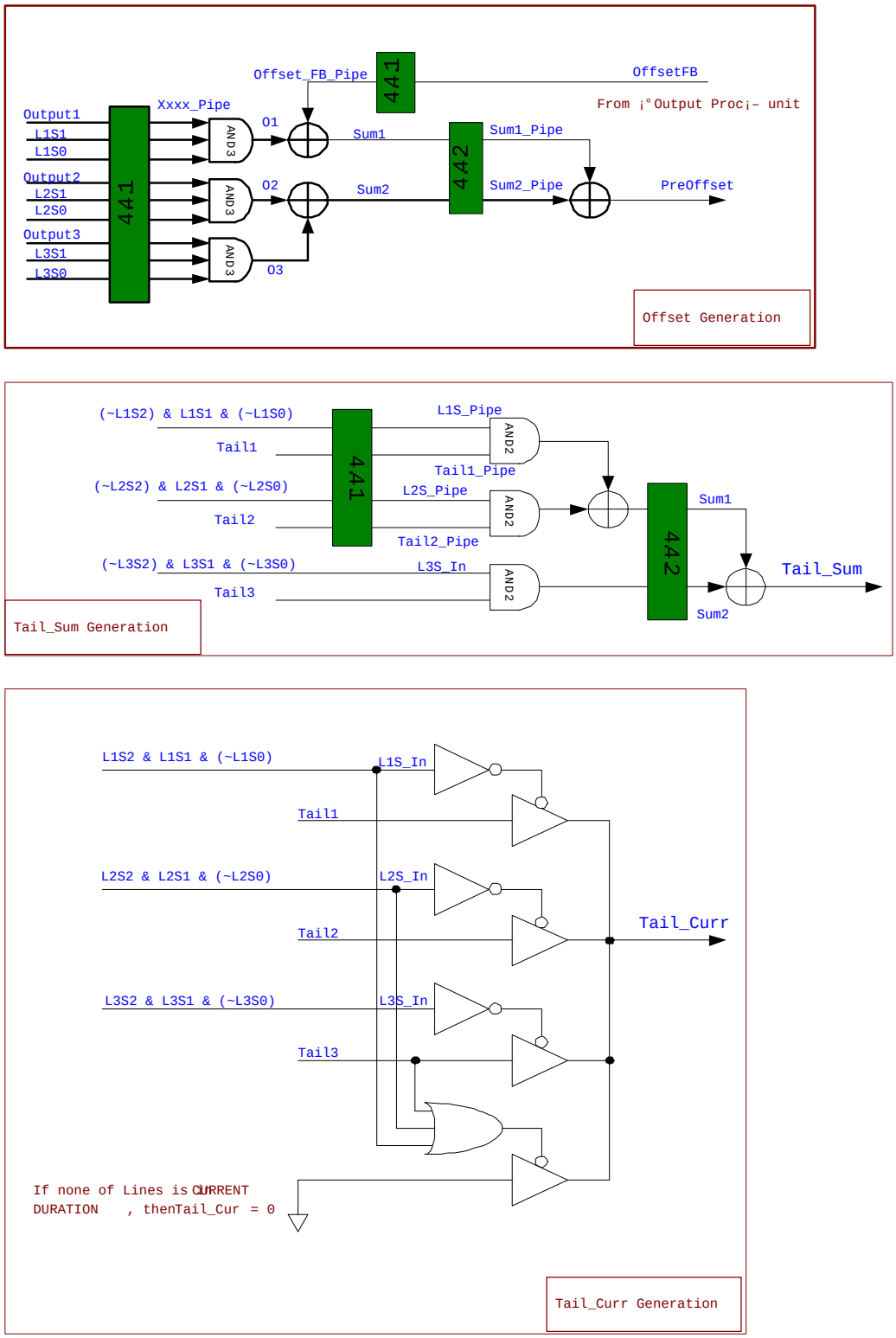

Figure 34: Post processing block diagram 4. 4.4 Offset, TailSum, TailCur generation

Block5: TRC Model Output Processing

As shown in Fig. 35, the output processing block is used to generate the final estimates of TRC model output, system offset, and system operation mode using information received from each of the three transient processing lines in block4. The necessary information includes transient active statuses ($PostActive1, PostActive2, PostActive3$), transient start time record ($Start1$, $Start2$, $Start3$), transient length estimations ($Kn1, Kn2, Kn3$), and a pre-calculated offset estimation $PreOffset$ from block4. A 10-tap moving average filter is also used to estimate the system offset during the steady status. Three stages of operations are performed in this block.

Stage 5.1: In this stage,the system operation mode is estimated to determine whether the system is in "transient period" or "far from transient period (steady state)". This is done by checking the transient active status ($PostActive1$, $PostActive2$, $PostActive3$) from each transient processing lines, as well as by checking whether each transient has been finished over than time threshold (e.g., "$N - (Kn + Start) \geq 10$?" in this design). The reason for checking the later condition is that the offset estimation will be switched to the 10-taps (therefore the threshold is set to 10) moving average filter during the system steady status. The delay effect the filter may affect the accuracy of the offset estimation if the switching happens immediately after transients finishing.

Stage 5.2: In this stage, the final system offset output is generated, choosing either the $PreOffset$ estimate from block4 or $XtSS$ from the moving average filter, according to the system status ($SSTrue$) determined in stage 5.1. The techniques used to implement the moving average filter is also a multiplier-accumulator structure, the same as the one used for pre-filtering in block1.

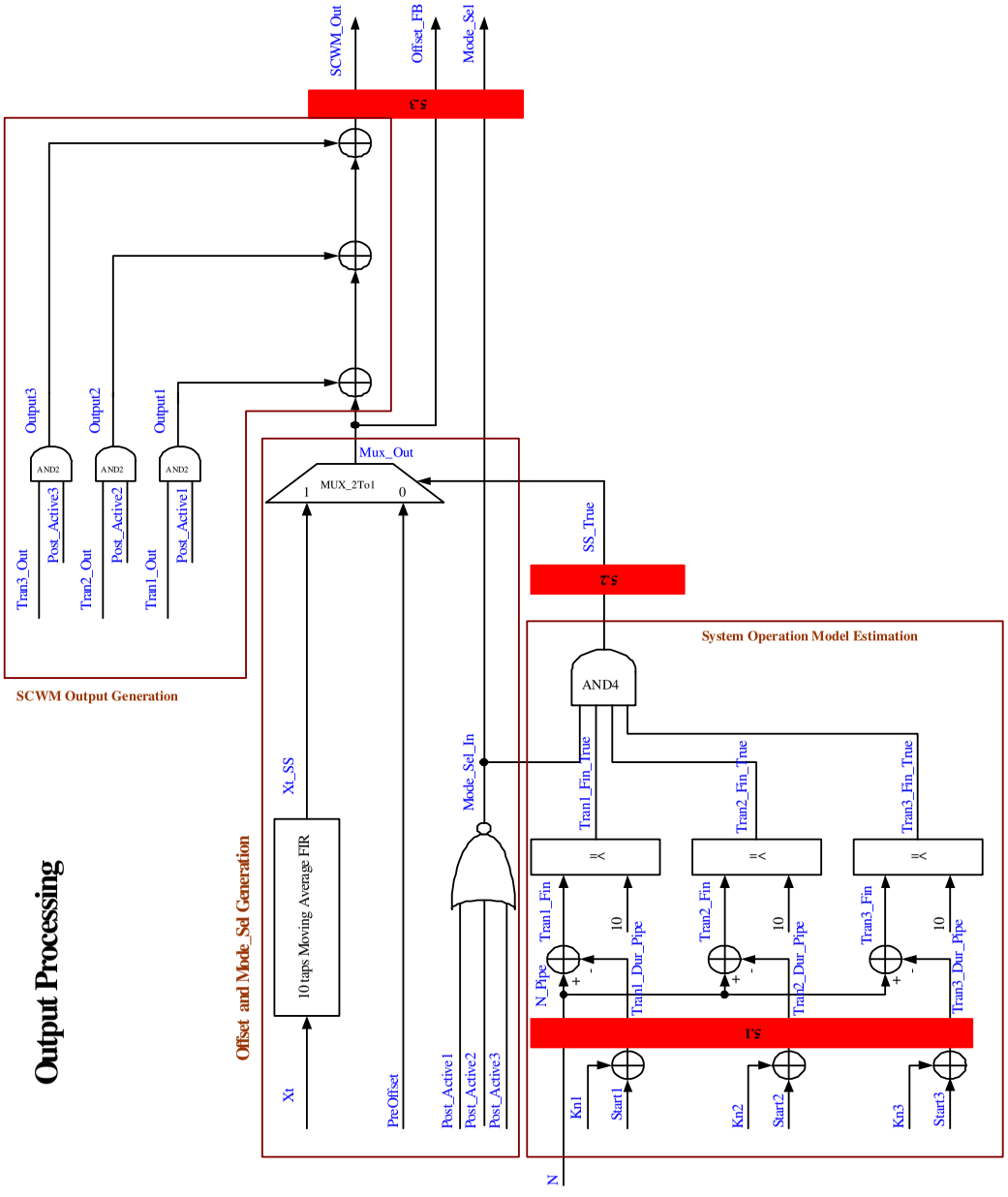

Figure 35: Output processing block diagram

Stage 5.3: In this stage, the SCWM output is generated by summing the offset estimate and the output contributions from each transient processing line if the associated transient is still active. The output contributions of the expired transient has been added to the pre-offset estimate in the last block and therefore has no further usefulness. The system mode select signal $ModeSel$ is used for pipeline control generation block to accordingly turn on/off the functionality of cluster calculation array block. This signal is important for low power and high efficiency operation since block3 consumes most of the power and computational effort and is only needed during transient periods.

Block6: Pipeline Control Generation and Time Step Marker

This block is used to generated all of the pipeline control clocks and the time step marker N for block2 – block5. Block1 (FIR) is clocked separately since the FIR works at a different frequency from the other blocks. The down sampling between FIR and other blocks is implemented by block2 to sample the output of FIR at 100Hz.

The block diagram and the time slot allocation plan is shown in Fig. 36. 256 pipeline control clocks per 0.01 second (100Hz) are setup. The 256 clocks are organized by 16×16 matrix. Each individual clock is represented by $GCLK(x, y)$, where x represents the clock group number, and the y represents the clock number within the group. As shown in the figure, both x and y are arranged from 0 to 15 along the time, i.e., the clocks along the time are $GCLK(0, 0)$, $GCLK(0, 1)$ $\ldots GCLK(15, 15)$. The time step marker counts at 100Hz, the processing frequency of the TRC model. The time resolution (the system clock) is 1/32 per pipeline control clock, which is the working frequency of the clocked devices other than the pipeline control D flip-flops, such as the shift register, synchronous RAM, and the DSP48. The resultant system

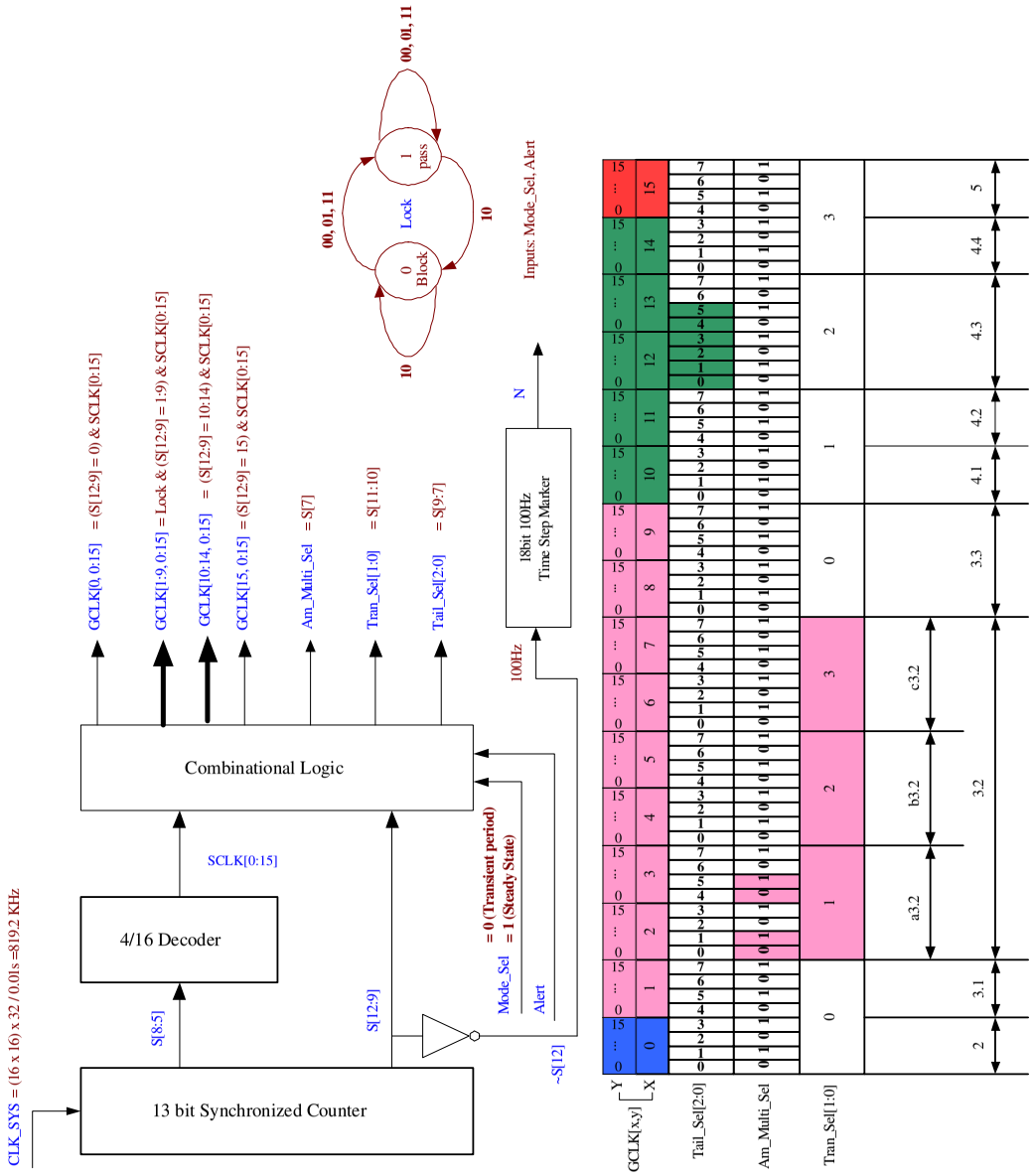

Figure 36: System pipeline control clocks generation and time step marker.

clock rate is $CLKSYS = 819.2kHz$. This clock is generated from the digital clock manager inside the Virtex4.

The circuit that generates the pipeline control clocks and the time step marker includes a 13-bit synchronous counter, a 4/16 decoder, and the necessary combinational logics. The $CLKSYS$ is the system clock generated by digital clock manager. In order to acquire the 1/32 resolution, the lower 5 bits of the counter outputs $S[4:0]$ are left open. The next 4 outputs $S[8:5]$ are feed to the 4/16 decoder to generate the 16 clocks $SCLK[0:15]$ within the group. And the outputs $S[12:9]$ are used to generate the combinational logic of switching the $SCLK[0:15]$ into the clock groups from $GCLK[0, 0:15]$ to $GCLK[15, 0:15]$ in time order, i.e.,

$$GCLK[x, 0:15] = (S[12:9] = x)\ and\ SCLK[x, 0:15], x = 0, 1, \ldots, 15 \qquad (107)$$

The signal $S[12]$ has the frequency of 100Hz and is input to a 18bits counter to generate the time step marker. The multiplexing control signals ($TranSel[1:0]$ for multiple data streams $Xt[a:c]$ multiplexing, $AmMultiSel$ for local DSP48 multiplexing in the likelihood and scale factor calculation sub-block, and $TailSel[2:0]$ for multiplier and RAM multiplexing in post-processing block)are generated directly from the 13-bit counter outputs, which is summarizing as below,

$$\begin{aligned} AmMultiSel &= S[7], \\ TranSel[1:0] &= S[11:10], \\ TailSel[2:0] &= S[9:7]. \end{aligned} \qquad (108)$$

The phase relationships between the multiplexes and the pipeline control clocks are

shown in the time slot allocation chart in the figure, where the shaded slots of the multiplexing signals indicate the functioning time of these signals.

The time slot allocation plan for each pipeline controlled operations is shown in Fig. 36. There are some constrains in allocating the time slots. First, the next clock after the multiplex switching is used to wait for the stability of the switched signals. Specifically, the $GCLK[x, 0], x = 0, \ldots 15$ is not assigned to any pipeline control registers, and $GCLK[0, 0]$ is used to wait for the stable value of time step marker N. Second, the operations associated to the multiplexes by $AmMultiSel$, $TranSel[1 : 0]$, $TailSel[2 : 0]$ are assigned into corresponding time slots indicated by the shaded slots of the multiplex control signals. Third, all of the clocks in stage 3.2 should be repeated three times for the data path multiplexing of $Xt[a : c]$ respectively. A full set of final time slots allocated to every pipeline operation steps is shown in Table 5.

Recall that a signal $ModeSel$ is defined in block5 that indicates whether the system is in steady state (True) or in transient (False). This signal combined with *Alert* is used to change the pipeline structure to accordingly turn on/off the functions of block3. To reduce the power consumption by digital CMOS circuitry, it is only necessary to maintain the MOSFET's status unchanged. The only power consumption of a MOSFET in steady state is the leakage current. A state machine is designed to block or pass the control clocks $GCLK[1 : 9, 0 : 15]$ for block3. The status variable *Lock* in the state machine becomes false (block) whenever the $ModeSel$ is true and *Alert* is false, and becomes true (pass) whenever the *Alert* is true. The *Lock* variable is added to the combinational switch control logics of $G[1 : 9, 0 : 15]$ as

$$GCLK[x, 0 : 15] = Lock\ and\ (S[12 : 9] = x)\ and\ SCLK[0 : 15]. \qquad (109)$$

Remember that the EN reset / enable signal in block3 has a similar function. The

Pipeline stage	Clock	Pipeline stage	Clock
2.1	GCLK[0,1]	a3.2.10	GCLK[3,9]
2.2	GCLK[0,10]	b3.2.10	GCLK[5,9]
2.3	GCLK[0,12]	c3.2.10	GCLK[7,9]
3.1.1	GCLK[1,4]	3.3.1	GCLK[8,1]
3.1.2	GCLK[1,8]	3.3.2	GCLK[8,10]
3.2.1	GCLK[2,1],GCLK[4,1],GCLK[6,1]	3.3.3	GCLK[8,14]
3.2.2	GCLK[2,3],GCLK[4,3],GCLK[6,3]	3.3.4	GCLK[9,1]
a3.2.2	GCLK[2,3]	3.3.5	GCLK[9,3]
b3.2.2	GCLK[4,3]	3.3.6	GCLK[9,8]
c3.2.2	GCLK[6,3]	3.3.7	GCLK[9,13]
3.2.3	GCLK[2,6],GCLK[4,6],GCLK[6,6]	4.1	GCLK[10,1]
a3.2.3	GCLK[2,6]	4.2	GCLK[10,3]
b3.2.3	GCLK[4,6]	4.3.1	GCLK[12,3]
c3.2.3	GCLK[6,6]	4.3.2	GCLK[12,6]
AdenLS	GCLK[2,7],GCLK[4,7],GCLK[6,7]	4.3.3	GCLK[12,7]
3.2.4	GCLK[2,9],GCLK[4,9],GCLK[6,9]	4.3.4	GCLK[12,11]
AnumLS	GCLK[2,10],GCLK[4,10],GCLK[6,10]	4.3.5	GCLK[12,14]
3.2.5	GCLK[2,12],GCLK[4,12],GCLK[6,12]	4.3.6	GCLK[12,15]
3.2.6	GCLK[2,14],GCLK[4,14],GCLK[6,14]	4.3.7	GCLK[13,3]
a3.2.6	GCLK[2,14]	4.3.8	GCLK[13,6]
b3.2.6	GCLK[4,14]	4.3.9	GCLK[13,7]
c3.2.6	GCLK[6,14]	4.4.1	GCLK[14,4]
3.2.7	GCLK[3,1],GCLK[5,1],GCLK[7,1]	4.4.2	GCLK[14,8]
3.2.8	GCLK[3,3],GCLK[5,3],GCLK[7,3]	5.1	GCLK[15,4]
3.2.9	GCLK[3,6],GCLK[5,6],GCLK[7,6]	5.2	GCLK[15,8]
3.2.10	GCLK[3,9],GCLK[5,9],GCLK[7,9]	5.3	GCLK[15,12]

Table 5: Time slots allocation to pipelined operation steps.

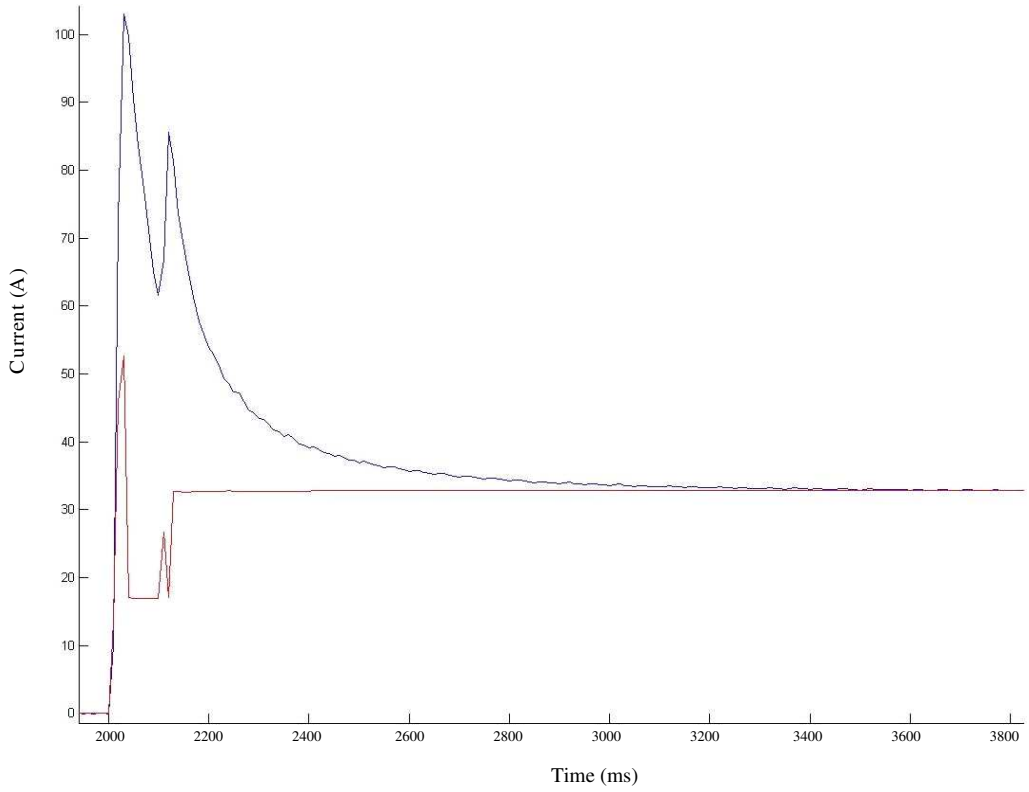

Figure 37: Overlapped transients (vacuum cleaner followed by lathe) recognition with the finite resolution fix point implementation of TRC model in Matlab.

Lock function added here helps further reducing the power of block3 by stopping clocking the devices.

FPGA Implementation Testing Results

To test the RTL design, the TRC model implemented with the same operation flow and fixed point data as the RTL design was tested in Matlab. Figure 37 and 38 show the simulation results for two prototype overlapped transients. The results verify that the operation flow and the finite fixed point data used in RTL design are proper. Next, the RTL design of TRC model was implemented with Verilog coding on Xilinx Virtex4 FPGA system. The primary functional blocks including cluster calculation array (block3) which implements the SCWM prediction as well

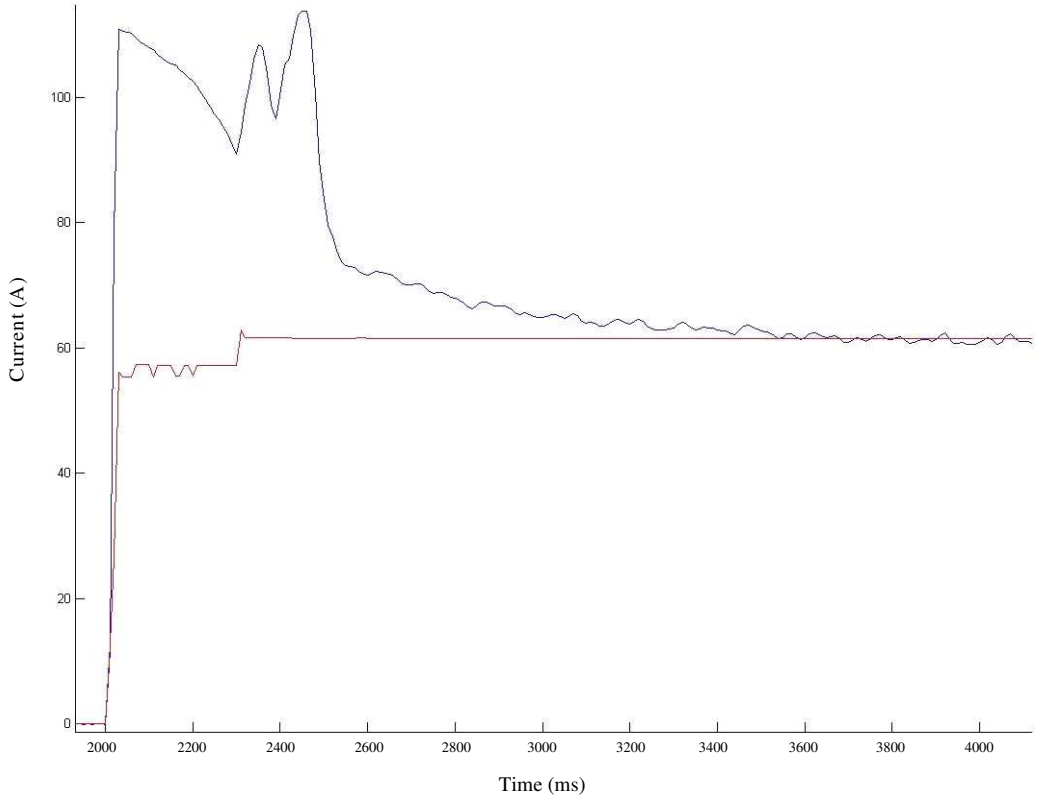

Figure 38: Overlapped transients (drill followed by bulb) recognition with the finite resolution fixed point implementation of TRC model in Matlab.

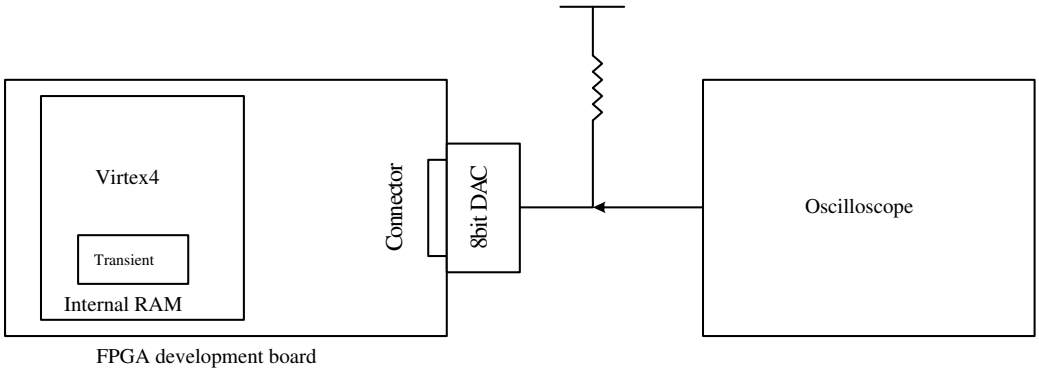

Figure 39: Experiment setup for testing the RTL design of TRC in FPGA hardware. Transient is pre-programmed into the internal RAM of FPGA.

as the sequential transient scaling, and the pipeline clocks generation (block6) have been combined together to form a primary SCWM prediction model and verified on FPGA hardware. Other blocks were verified with ModelSim "post place and route simulations" separately and are yet to be connected with the primary functional block and verified on FPGA hardware.

The experiment setup is shown in Fig. 39, including Virtex4(XC4VLX60) MB development board (Memec), DAC0808 8bit D/A converter and oscilloscope. The SCWM model was implemented with two clusters, representing lathe and bulb transients respectively. Two transients of lathe and bulb types, which are out of the training set, were used for test. The transient data was pre-programmed into the RAM inside the FPGA and was read out by SCWM module to simulate the transient data from an A/D converter. The digital transient and SCWM prediction outputs were converted to analog signal by the 8bit D/A converters and measured by the oscilloscope. The system clock rate is 100MHz, and a pipeline control clock width is 32 system clocks. The resultant experimental SCWM processing rate is $100\text{MHz}/(32 \times 256) = 12.2\text{kHz}$, instead of the 100Hz used for processing the true transient data. The test results for lathe and bulb transients are shown in Fig. 40 and Fig. 41 respectively. Note in the figures that the measured outputs are the voltage

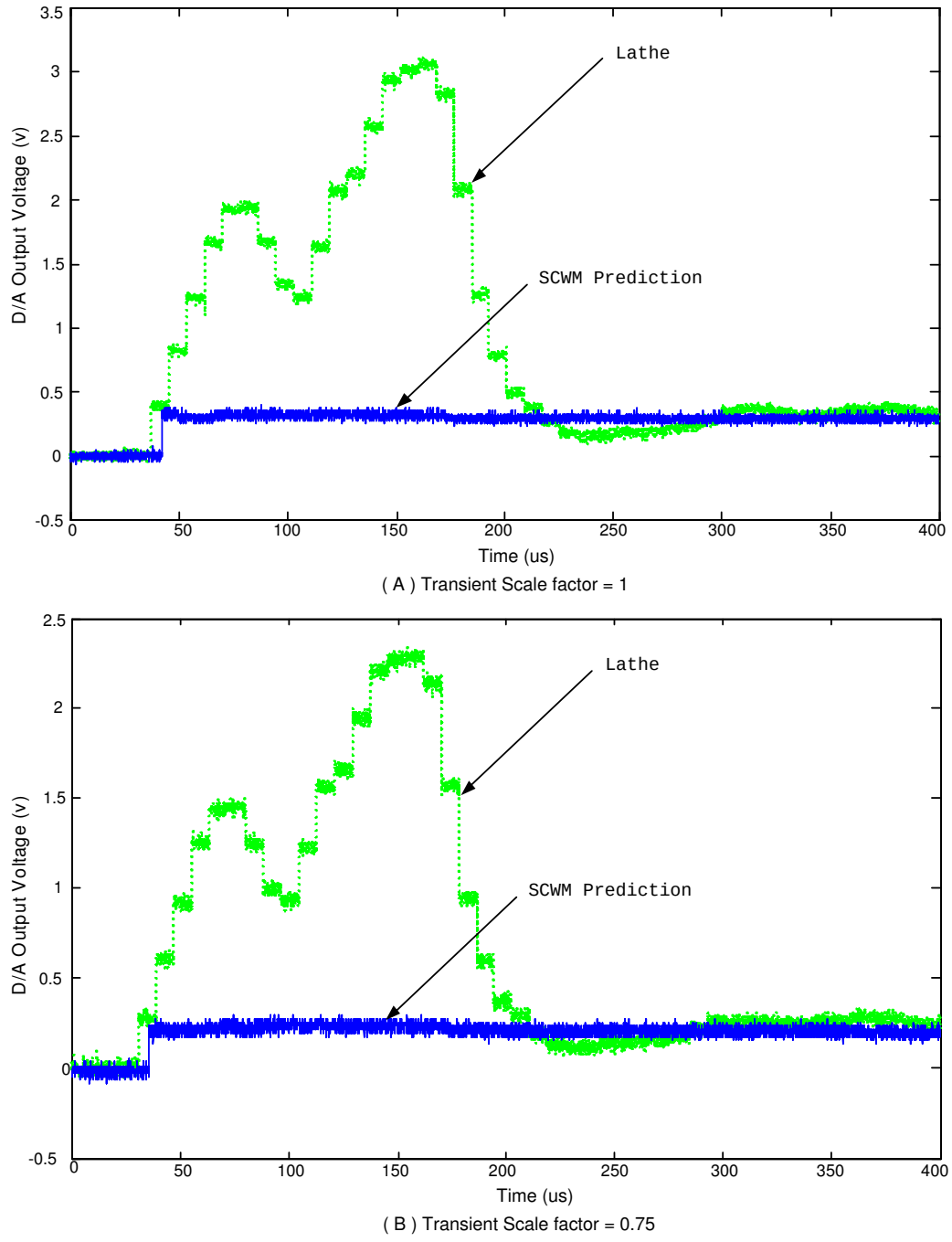

Figure 40: FPGA hardware testing result 1: SCWM prediction of lathe transient long range behavior.

converted from D/A output current, and the time resolution reflects the experimental 12.2kHz SCWM processing rate. The sub-figure (A) in each figure shows the result for the input transient of the same scale in magnitude as the corresponding cluster mean. Sub-figure (B) shows the result that the input transient is of different scale from cluster mean. The input lathe was pre-scaled by 0.75, and the input bulb transient was pre-scaled by 1.5.

As shown in Fig. 40, the SCWM predictions of the long range behavior of the lathe transients on two different scales are accurate and converge quickly. The results demonstrate the successful implementation of the SCWM model in RTL level, which includes the cluster calculation array and pipeline clock generation blocks. The results of other important internal variables, including local model weights in SCWM output averaging, champion cluster number, the calculated scaling factor, and the individual local model output were also checked with the ModelSim "post place and route simulations" to be accurate and converge fast.

The results for bulb transients on two different scales are shown in Fig. 41. The SCWM predictions again converge quickly but with prediction errors. Other related internal variables were checked with ModelSim simulations. The simulation results indicate that the champion cluster number, scaling factor, and the individual local model output were accurate and converge fast. However, the result for the local model weight is not accurate, and weight1 + weight2 $<$ 1. The reason for these inaccurate weights may be due to improperly designed lookup tables used for the exponential and reciprocal calculations. The lookup tables are implemented linearly with respect to the input range. The output resolution may be too coarse for the related variable range of the bulb transient cases. The problem can be solved by redesigning the lookup tables to be nonlinear with respect to input range such that the input range corresponding to highly varying output range has improved resolution.

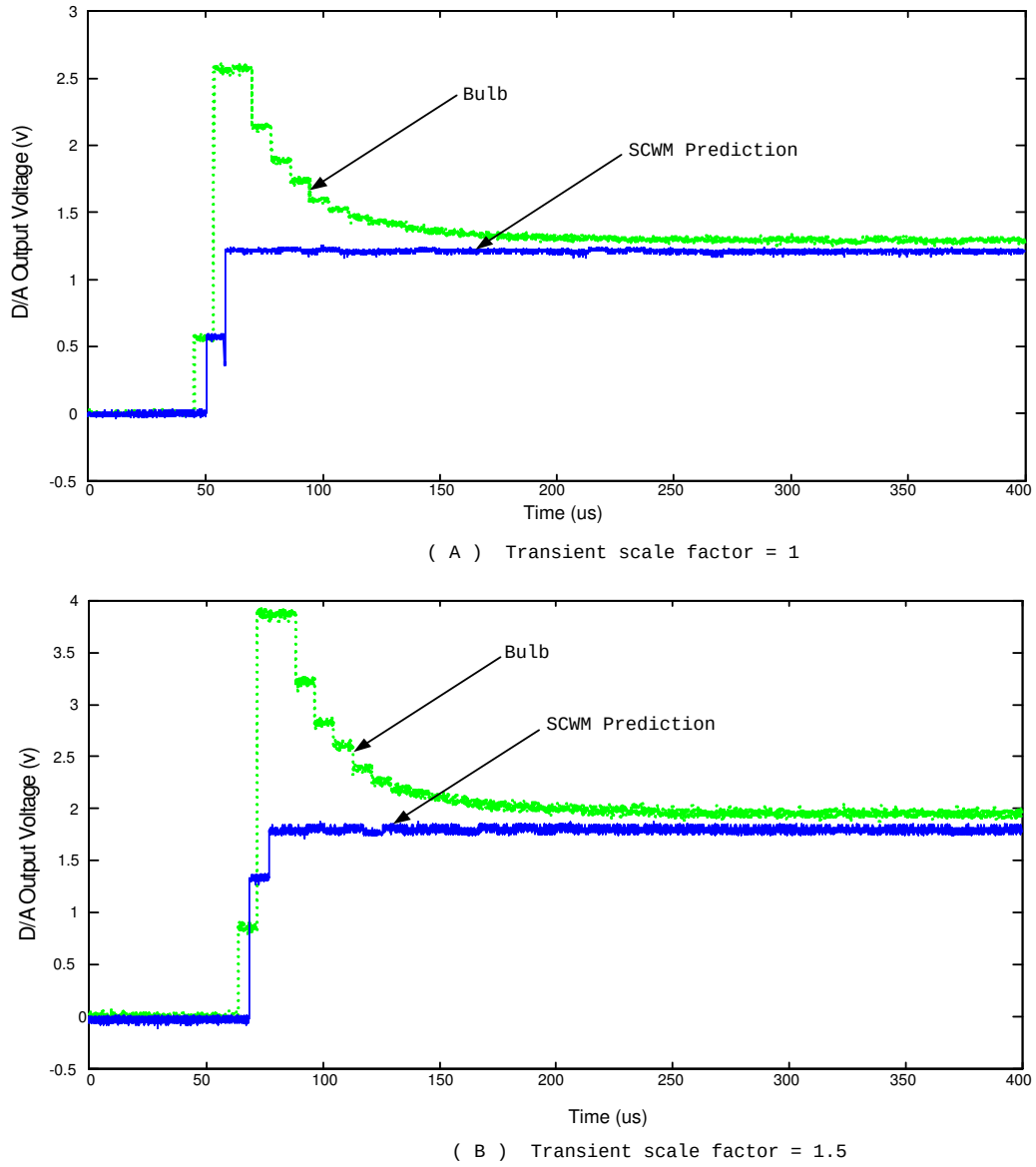

Figure 41: FPGA hardware testing result 2: SCWM prediction of bulb transient long range behavior.

Another possible reason for prediction error may be due to truncation of the 16-bit SCWM output by the 8-bit D/A converter.

DISCUSSION

Work Summary

This thesis proposes several extensions of cluster-weighted modeling (CWM) for electric load transient recognition and control. Mixed EM-LMS algorithms were developed to solve the singular matrix inversion problems in the model training process to improve numerical stability and save computational cost. A recursive training method was introduced to give CWM online adaptation capabilities for tracking time varying features of load transients. A sequential modification of CWM prediction (SCWM) was developed to solve the real-time transient recognition and prediction problem. SCWM also contributes to the load transient recognition problem by sequentially resolving the multiple transient overlapping problem with the idea of tail prediction for incomplete transient segments. Several benchmark examples, including both overlapped and scaled transient data, were used in the simulation to verify the methods. Experimental results of SCWM suggest that the effective recognition delay of the SCWM for load transient recognition is often significantly less than the length of transient itself. For the transients in this thesis, SCWM typically provides good estimates using about one-fifth of the transient length. The tail prediction method successfully solved the transient overlapping problem, giving accurate transient long range behavior predictions.

A transient recognition control (TRC) was developed based on the SCWM method along with the transient detection and scaling techniques. TRC was applied to the power control task for a proposed hybrid fuel cell system that may improve the reliability, efficiency, and service life of fuel cells and other critical sources. TRC predicts the long range behavior of the load transients and prevents the fuel cell from responding to the power requirement of the load transient. The technique is demonstrated

by emulation of a real-time control using an arbitrary waveform generator in a real hybrid system. The training process for the TRC was performed once, prior to testing the prediction steps. In practice the training can occur in near real-time to adapt the control to new loads. It may be that a generic initialization can be performed for standard loads before installation. Some combination of on-site and off-site initialization may be desirable. An advantage of TRC is that new transients are easy to detect by examining the likelihood of the input relative to the pre-installed transients template in the model.

The proposed TRC model was implemented at the register transfer level and tested on Virtex4 FPGA. Testing results demonstrate the successful design. TRC model could also be implemented on conventional platforms, e.g., the DSP. An advantage of the FPGA implementation is that the details of TRC can be implemented at the lowest level on the hardware. Conversely, the FPGA implementation involves many low level design complexities that could increase the design time compared to conventional implementations using high level design tools.

Future Work Discussion

In future work, the whole TRC model could be verified on FPGA hardware. An A/D converter will be used to sense the true transient data from the system. The currently verified model was placed and routed on FPGA automatically without placing and routing constraints. A larger model with more clusters could be implemented and verified, which would involve more effort and work on FPGA floor planning, routing, and resource allocation.

The EM training algorithm has the so-called local-minimum convergence problem, i.e., the clusters converge where the likelihood is only locally maximum. Some articles discuss pre-initialization techniques in this context to properly choose the

initial positions and numbers of clusters. Examples includes rough-set theory [41], initial condition refinement techniques [39], split and merge approaches [11], or unsupervised competitive clustering techniques [58]. Another way of overcoming the local minima trapping problem is to improve the parameter searching and updating algorithms for EM by exploiting heuristic information reflected in the data. The authors in [55] used a genetic algorithm to help search more optimal parameters in the mixture of Kalman filter model they proposed. An annealing method was proposed in [35], combining with the EM method for searching for parameters. Future work might combine both methods, pre-initialization and improved search algorithms, into consideration to develop new training methods for CWM that perform better.

The issue of detecting off-training set transients is discussed in Chapter 3. The advantage of CWM model is that the on-site TRC model can generate and monitor the likelihood of the input data. If the likelihood is below a threshold after the lock out time, a new transient is detected. For the reasons of operation stability and robustness, it is not suggested that the on-site TRC be updated in real-time, although a recursive CWM training algorithm was developed to adapt the TRC parameters to the slightly changed transient features. Future TRC system could include a higher level, PC-based load monitoring system. This system could detect new transient events periodically update TRC parameters.

Another potential application is to combine CWM with Kalman filter methods for nonlinear and non-stationary state estimation. The optimality of the Kalman filter depends on the exact knowledge of the model parameters and the statistics of the process and measurement noise. These assumptions are often not true in applications. A Kalman filter bank could be developed to address the nonlinear and non-stationary state estimation problem. Each Kalman filter could be embedded in the CWM as a local model. The clusters would be described by the likelihood $p(y_k|x_k, c_m), p(x_k|x_{k-1}, c_m)$ that models the state space equations of each local Kalman

filter. The weight that combines the local Kalman filter output to generate the final model output can be evaluated from the posterior likelihood between a cluster and the sequence of measurements. A similar EM-LMS method can be developed for training the model. Some initial results show that this CWM/Kalman filter has promise for some simple nonlinear and non-stationary systems.

REFERENCES

[1] Acharya, K., Mazumder, S. K., and Burra, P. K., "System interaction analyses of solid oxide fuel cell(sofc) power conditioning system," pp. 2026–2032, IEEE Industry Applications Conference, 38th IAS Annual Meeting, Oct. 2003.

[2] Alexandre X. CARVALHO, M. A. T., "Modeling nonlinear time series with local mixtures of generalized linear models," *The Canadian Journal of Statistics*, vol. 33, 2005.

[3] Amphlett, J. C., de Oliveira, E. H., Mann, R. F., Roberge, P. R., Rodrigues, A., and Salvador, J. P., "Dynamic interaction of a proton exchange membrane fuel cell and a lead-acid battery," *Journal of Power Sources*, vol. 65, pp. 173–178, 1997.

[4] B. Widrow, S. D. S., "Adaptive signal processing," *Proceedings of the IEEE*, 1985.

[5] Bertoni, L., Gualous, H., Bouquain, D., Hissel, D., Pera, M. C., and Kauffmann, J. M., "Hybrid auxiliary power unit (apu) for automotive applications," pp. 1840–1845, Vehicular Technology Conference 2002. Proceedings. VTC 2002-Fall, 2002 IEEE 56th, Sept 2002.

[6] Bishop, C., "Improving the generalization properties of radial basis function neural networks," *Neural Computation*, vol. 1, pp. 579–588, 1991.

[7] Bishop, C. M., *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[8] Brahim-Belhouari S., Bermak A., M. S. C. P., "Fast and robust gas identification system using an integrated gas sensor technology and gaussian mixture models," *Sensors Journal, IEEE*, vol. 5, pp. 1433 – 1444, 2005.

[9] C. J. Harris, Q. G., "State estimation and multi-sensor data fusion using data-based neurofuzzy local linearization process models,," *Information Fusion*, vol. 2, pp. 17–29, 2001.

[10] Caillol H., Pieczynski W., H. A., "Estimation of fuzzy gaussian mixture and unsupervised statistical image segmentation," *IEEE Transactions on Image Processing,*, vol. 6, pp. 425 – 440, 1997.

[11] Charalampidis, D., "A modified k-means algorithm for circular invariant clustering," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, pp. 1856 – 1865, 2005.

[12] Chen R., L. J. S., "Mixture kalman filters," *J. R. Statist. Soc. B*, vol. 62, pp. 493–508, 2000.

[13] Christopher Laughman, Douglas Lee, R. C. S. S. S. L. L. N. P. A., "Advanced nonintrusive monitoring of electric loads," *IEEE Power and Energy Magazine*, vol. March/April, pp. 56–63, 2003.

[14] D. A. Reynolds, R. C. R., "Robust text-independent speaker identification using gaussian mixture speaker models," *IEEE Transactions on Speech and Audio Processing*, vol. 3, pp. 72–83, 1995.

[15] Fukunaga, K., "Introduction to statistical pattern recognition," *Academic Press*, 1990.

[16] Gemmen, R. S., "Analysis for the effect of inverter ripple current on fuel cell operating condition," *Transactions of the ASME, Journal of Fluids Engineering.*, vol. 125, pp. 576–585, May 2003.

[17] Gershenfeld, N., Schoner, B., and Metois, E., "Cluster weighted modeling for time-series analysis," *Nature*, vol. 397, pp. 329–332, Jan 1999.

[18] Gershenfeld, N., *The Nature of Mathematical modeling.* Cambridge University Press, 1999.

[19] Gokdere, L. U., Benlyazid, K., Dougal, R. A., Santi, E., and Brice, C. W., "A virtual prototype for a hybrid electric vehicle," *Mechatronics*, vol. 12, pp. 575–593, 2002.

[20] Hartley, H., "Maximum likelihood estimation from incomplete data," *Biometrics*, vol. 14, pp. 174–194, 1958.

[21] Hartley, H., "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society, Series B,*, vol. 39, pp. 1–38, 1977.

[22] Haykin, S., "Adaptive filter theory," 2002.

[23] Jarvis, L. P., Atwater, T. B., and Cygan, P. J., "Hybrid power sources for land warrior scenario," *IEEE Aerospace and Electronic Systems Magazine*, vol. 15, pp. 37–41, Sept 2000.

[24] Jarvis, L. P., Cygan, P. J., and Roberts, M. P., "Hybrid power source for manportable applications," *IEEE Aerospace and Electronic Systems Magazine*, vol. 18, pp. 13–16, Jan 2003.

[25] Jarvis, L. P., Atwater, T. B., and Cygan, P. J., "Fuel cell / electrochemical capacitor hybrid for intermittent high power applications," *Journal of Power Sources*, vol. 79, pp. 60–63, 1999.

[26] Jarvis, L. P., Atwater, T. B., Plichta, E. J., and Cygan, P. J., "Power assisted fuel cell," *Journal of Power Sources*, vol. 70, pp. 253–257, 1998.

[27] Jeong, K. S. and Oh, B. S., "Fuel economy and life-cycle cost analysis of a fuel cell hybrid vehicle," *Journal of Power Sources*, vol. 105, pp. 58–65, 2002.

[28] Jones, P. B., Lakeman, J. B., Mepsted, G. O., and Moore, J. M., "A hybrid power source for pulse power applications," *Journal of Power Sources*, vol. 80, pp. 242–247, 1999.

[29] Jun Tani, e., "Learning to percive the world as articulated: An approch for hierarchycal learning in sensory motor system," *Proc. of 5th conf. on Simulation of adaptive behavior, MIT press*, pp. 270–279.

[30] Kapur, J. N., "Maximum entropy models in science and engineering," *New York: Wiley*, 1989.

[31] Kim, S. P., "Divide-and-conquer approach for brain machine interfaces: nonlinear mixture of competitive linear models," *Neural Networks*, vol. 1, pp. 865–871, 2003.

[32] Lee, H. S., Jeong, K. S., and Oh, B. S., "An experimental study of controlling strategies and drive forces for hydrogen fuel cell hybrid vehicles," *International Journal of Hydrogen Energy*, vol. 28, pp. 215–222, 2003.

[33] M. Banan, K. H., "A monte carlo strategy for data-based mathematical modeling," *Mathl. Comput. Modeling*, vol. 22, pp. 73–90, 1995.

[34] M. I. Jordan, R. A. J., "Hierarchical mixture of experts and the em algorithm," *Neural Computation*, vol. 6, pp. 181–214, 1994.

[35] Murphy., K., "Switching kalman filters.," *Technical report, U. C. Berkeley*, 1998.

[36] N. Gershenfeld, B. Schoner, E. M., "Cluster-weighted modeling for time series prediction and characterization," *Preprint, http://citeseer.ist.psu.edu/136695.html*, 1997.

[37] Nadal, M. and Barbir, F., "Development of a hybrid fuel cell / battery powered electric vehicle," *International Journal of Hydrogen Energy*, vol. 21, no. 6, pp. 497–505, 1996.

[38] Nojima, R., Takano, I., and Sawada, Y., "Transient performance of a new-type hybrid electric power distribution system with fuel cell and smes," pp. 1303–1308, IECON'01, the 27th Annual Conference of the IEEE Industrial Electronics Society, 2001.

[39] P. S. Bradley, U. M. F., "Refining initial points for k-means clustering," *Proc. 15th Intl Conf. Machine Learning,*, pp. 91–99, 1998.

[40] Park, K. W., Ahn, H. J., and Sung, Y. E., "All-solid-state supercapacitor using a nafion polymer membrane and its hybridization with a direct methanol fuel cell," *Journal of Power Sources*, vol. 109, pp. 500–506, 2002.

[41] Pawlak, Z., "Rough sets, theoretical aspects of reasoning about data.," *Dordrecht: Kluwer Academic*, 1991.

[42] Q. Gan, C. J. H., "A hybrid learning scheme combinning em and masmod algorithm for fuzzy local linearization modeling," *IEEE transactions on neural networks*, vol. 12, pp. 43–53, 2001.

[43] S. K. Pal, S. Mitra, P. M., "Rough-fuzzy mlp: Modular evolution, rule generation, and evaluation," *IEEE transactions on knowledge and data engineering*, vol. 15, pp. 14–25, 2003.

[44] S. Kumar, J. G., "Gamls: a generalized framework for associative modular learning systems," *Conference on applications and science of computational intelligence II*, vol. 3722, pp. 24–35, 1999.

[45] Schoner, B., Cooper, C., Douglas, C., and Gershenfeld, N., "Data-driven modeling and synthesis of acoustical instruments," (Ann Arbor), International Computer Music Conference, Oct 1998.

[46] Schoner, B., Cooper, C., and Gershenfeld, N., "Cluster-weighted sampling for synthesis and cross-synthesis of violin family instruments," (Berlin, Germany), Proc. International Computer Music Conference, Aug 2000.

[47] Schoner, B. and Gershenfeld, N., "Data-driven modeling of nonlinear microwave devices," (Anaheim, California), Digest 53rd ARFTG Conference on Nonlinearity Characterization, June 1999.

[48] Schoner, B. and Gershenfeld, N., "Cluster-weighted modeling: Probabilistic time series prediction, characterization and synthesis," pp. 365–385, Alistair Mees(Ed.), Birkhaeuser, Boston, 2000.

[49] Shaw, S. R., *System Identification Techniques and Modeling for Non-intrusive Load Diagnostics*. Ph.d., MIT, February 2000.

[50] Shaw, S. R., Abler, C. B., Lepard, R. F., Luo, D., Leeb, S. B., and Norford, L. K., "Instrumentation for high performance nonintrusive electrical load monitoring," *ASME Journal of Solar Energy Engineering*, vol. 120, pp. 224–229, August 1998.

[51] T. Bishop, F. G., "Networks for approximation and learning," *Proceedings of the IEEE*, vol. 1, pp. 1481–1197, 1991.

[52] Tiago H. Falk, W.-Y. C., "Nonintrusive speech quality estimation using gaussian mixture models," *IEEE SIGNAL PROCESSING LETTERS*, vol. 13, pp. 108–111, 2006.

[53] Uykan, Z., "Clustering-based algorithms for single-hidden-layer signoid perceptron," *IEEE transactions on neural networks*, vol. 14, pp. 708–715, 2003.

[54] W. S. Chaer, R. H. Bishop, J. G., "Hierarchical adaptive kalman filtering for interplanetary orbit determination," *IEEE transactions on aerospace and electronic systems*, vol. 3, pp. 883–896, 1998.

[55] Wassim S. Chaer, Robert H. Bishop, J. G., "A mixture-of-experts framework for adaptive kalman filtering," *IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICSPART B: CYBERNETICS,*, vol. 27, pp. 452–464, 1997.

[56] Wu, W., "Modeling and decoding motor cortical activity using a switching kalman filtering," *IEEE transactions on biomedical engineering*, vol. 51, pp. 993–942, 2004.

[57] Xin Wang, Peter Whigham, D. D. M. P., ""time-line" hidden markov experts for time series prediction," *Neural Information Processing - Letters and Reviews*, vol. 3, pp. 39–47, 2004.

[58] YIU-MING CHEUNG, L. X., "Rival penalized competitive learning based approach for discrete-valued source separation," *International Journal of Neural Systems*, vol. 10, pp. 483–490, 2000.

APPENDIX A

VERILOG CODES OF THE TRC RTL MODEL

```
////////////////////////////////////////////////////////////////
//
// Transient recognition control (TRC) top level file.
//  TRC model can detect, scale and predict the long range
// behavior of load transient in non-overlapping / overlapping
//  situations. There are 5 blocks under top level:
// Block2:  Transient detection and pre-processing
// Block3:  Cluster Calculation array
// Block4:  Post processing
// Block5:  Output processing
// Block6:  Pipeline control clock generation
//
//--------------------------------------------------------------
//  TRC model modules hierarchy:
//==============================================================
//
//  SCWM_Top
//    PrePro_TransDet
//       Trans_Prepro
//        Trans_Det
//        Lock_Out
//        Prepro_Output
//    Clus_Array
//       CWM_Para
//        Am_and_Likelihood
//          Am1
//           Am2
//             Parallel_Shift_Reg
//                Shift_Reg
//              Syn_Counter_4bit_Asyn_RST
//           Likelihood
//        Filter_Bank
//          SCWM_filter
//        Likelihood_Comp
//          COMSW1
//           COMSW2
//        filter_switch_and_weight
//    Post_Proc
//      Open_New_Line
//      Post_Data_Input
//      Close_Line_array
//          Close_Line
//      Tail_Recover
//        Tail_Recover_Mem_Block
//      Tail_Sum_Generation
//      Tail_Curr_Generation
//      Offset_Generation
//    Output_Proc
//    Pipeline_generation
//      syn_counter_14
//      syn_counter_18
//--------------------------------------------------------------
//
```

```
////////////////////////////////////////////////////////////////


//*****************************************************
//         TRC TOP Level file
//*****************************************************


//-----------------------------------------------------
//  CLK_SYS: Input system clock
//  Trans:   Input transients
//  SCWM_Out: Output TRC prediction
//-----------------------------------------------------


module SCWM_TOP( input                CLK_SYS,
                 input      [17:0]    Trans,
                 output     [15:0]    SCWM_Out
                );


//***********************************************
//    internal signals
//***********************************************

//-----------------------------------------------
//  Clock signal connections
//-----------------------------------------------

wire [15:0]

GCLK0, GCLK1,  GCLK2,  GCLK3,  GCLK4,  GCLK5, GCLK6, GCLK7,GCLK8,
GCLK9, GCLK10, GCLK11, GCLK12, GCLK13, GCLK14,GCLK15;

wire

CLK_2_1,    CLK_2_2,     CLK_2_3,     CLK_3_1_1,   CLK_3_1_2,
CLK_3_2_1,  aCLK_3_2_2,  bCLK_3_2_2,  cCLK_3_2_2,  aCLK_3_2_3,
bCLK_3_2_3, cCLK_3_2_3,  CLK_3_2_4,   CLK_3_2_5,   CLK_3_2_6,
aCLK_3_2_6, bCLK_3_2_6,  cCLK_3_2_6,  CLK_3_2_7,   CLK_3_2_8,
CLK_3_2_9,  aCLK_3_2_10, bCLK_3_2_10, cCLK_3_2_10, CLK_3_3_1,
CLK_3_3_2,  CLK_3_3_3,   CLK_3_3_4,   CLK_3_3_5,   CLK_3_3_6,
CLK_3_3_7,  CLK_4_1,     CLK_4_2,     CLK_4_3_1,   CLK_4_3_2,
CLK_4_3_3,  CLK_4_3_4,   CLK_4_3_5,   CLK_4_3_6,   CLK_4_3_7,
CLK_4_3_8,  CLK_4_3_9,   CLK_4_4_1,   CLK_4_4_2,   CLK_5_1,
CLK_5_2,    CLK_5_3;

//-----------------------------------------------
//   Definitions of connections between blocks
//-----------------------------------------------

 wire [17:0] N;
 wire        Aden_LS, Anum_LS;
```

```
 wire [1:0]  Tran_Sel;
 wire        Am_Multi_Sel;
 wire [2:0]  Tail_Sel;
 wire [17:0] Tail_Curr;
 wire [17:0] Tail_Sum;
 wire [15:0] Offset;
 wire        Alert;
 wire [17:0] Xt_A;
 wire [17:0] Xt_B;
 wire [17:0] Xt_C;
 wire        EN;
 wire [15:0] Clus_Scale;
 wire [3:0]  Clus_Num1;
 wire [3:0]  Clus_Num2;
 wire [8:0]  Clus_Weight1;
 wire [8:0]  Clus_Weight2;
 wire [8:0]  Clus_Kn;
 wire [15:0] Clus_Output;
 wire [15:0] Tail1_Output;
 wire [8:0]  Tail1_Kn;
 wire [17:0] Tail1_Start;
 wire        Tail1_Post_Alive;
 wire [15:0] Tail2_Output;
 wire [8:0]  Tail2_Kn;
 wire [17:0] Tail2_Start;
 wire        Tail2_Post_Alive;
 wire [15:0] Tail3_Output;
 wire [8:0]  Tail3_Kn;
 wire [17:0] Tail3_Start;
 wire        Tail3_Post_Alive;
 wire [15:0] Pre_Offset;
 wire        Mode_Sel;


//*****************************************************
//                BLOCK2
//*****************************************************

//-----------------------------------------------------
// Block2 achieves transient detection in non-overlapping
// overlapping situations, and preparing multiple paths of
// data with different time lag
//-----------------------------------------------------

PrePro_TransDet Block2(.Trans(Trans),
                       .Tail_Curr(Tail_Curr),
                       .Tail_Sum(Tail_Sum),
                       .Offset(Offset),
                       .N(N),
                       .CLK_2_1(CLK_2_1),
                       .CLK_2_2(CLK_2_2),
                       .CLK_2_3(CLK_2_3),
                       .Alert(Alert),
```

```
                        .Xt_A(Xt_A),
                        .Xt_B(Xt_B),
                        .Xt_C(Xt_C)
                       );


//*********************************************
//                   Block3
//*********************************************

//---------------------------------------------
//  Block3 implement the SCWM prediction as well
//  as transient scaling
//---------------------------------------------


 Clus_Array     Block3(.CLK_SYS(CLK_SYS),
                       .CLK_3_1_1(CLK_3_1_1),
                       .CLK_3_1_2(CLK_3_1_2),
                       .CLK_3_2_1(CLK_3_2_1),
                       .aCLK_3_2_2(aCLK_3_2_2),
                       .bCLK_3_2_2(bCLK_3_2_2),
                       .cCLK_3_2_2(cCLK_3_2_2),
                       .aCLK_3_2_3(aCLK_3_2_3),
                       .bCLK_3_2_3(bCLK_3_2_3),
                       .cCLK_3_2_3(cCLK_3_2_3),
                       .CLK_3_2_4(CLK_3_2_4),
                       .CLK_3_2_5(CLK_3_2_5),
                       .CLK_3_2_6(CLK_3_2_6),
                       .aCLK_3_2_6(aCLK_3_2_6),
                       .bCLK_3_2_6(bCLK_3_2_6),
                       .cCLK_3_2_6(cCLK_3_2_6),
                       .CLK_3_2_7(CLK_3_2_7),
                       .CLK_3_2_8(CLK_3_2_8),
                       .CLK_3_2_9(CLK_3_2_9),
                       .aCLK_3_2_10(aCLK_3_2_10),
                       .bCLK_3_2_10(bCLK_3_2_10),
                       .cCLK_3_2_10(cCLK_3_2_10),
                       .CLK_3_3_1(CLK_3_3_1),
                       .CLK_3_3_2(CLK_3_3_2),
                       .CLK_3_3_3(CLK_3_3_3),
                       .CLK_3_3_4(CLK_3_3_4),
                       .CLK_3_3_5(CLK_3_3_5),
                       .CLK_3_3_6(CLK_3_3_6),
                       .CLK_3_3_7(CLK_3_3_7),
                       .Aden_LS(Aden_LS),
                       .Anum_LS(Anum_LS),
                       .Tran_Sel(Tran_Sel),
                       .Am_Multi_Sel(Am_Multi_Sel),
                       .Alert(Alert),
                       .N(N),
                       .Xta(Xt_A),
                       .Xtb(Xt_B),
```

```
                        .Xtc(Xt_C),
                        .EN(EN),
                        .Scale(Clus_Scale),
                        .Clus_Num1(Clus_Num1),
                        .Clus_Num2(Clus_Num2),
                        .Weight1(Clus_Weight1),
                        .Weight2(Clus_Weight2),
                        .Clus_Kn(Clus_Kn),
                        .Clus_Output(Clus_Output)
                        );

//************************************************
//            Block4
//************************************************

//------------------------------------------------
// Block4 is used to store and update the status of
// multiple alive transients and generate the tail
// prediction.
//------------------------------------------------

Post_Proc       Block4(.CLK_4_1(CLK_4_1),
                       .CLK_4_2(CLK_4_2),
                       .CLK_4_3_1(CLK_4_3_1),
                       .CLK_4_3_2(CLK_4_3_2),
                       .CLK_4_3_3(CLK_4_3_3),
                       .CLK_4_3_4(CLK_4_3_4),
                       .CLK_4_3_5(CLK_4_3_5),
                       .CLK_4_3_6(CLK_4_3_6),
                       .CLK_4_3_7(CLK_4_3_7),
                       .CLK_4_3_8(CLK_4_3_8),
                       .CLK_4_3_9(CLK_4_3_9),
                       .CLK_4_4_1(CLK_4_4_1),
                       .CLK_4_4_2(CLK_4_4_2),
                       .CLK_SYS(CLK_SYS),
                       .Tail_Sel(Tail_Sel),
                       .Clus_Kn(Clus_Kn),
                       .Clus_Num1(Clus_Num1),
                       .Clus_Num2(Clus_Num2),
                       .Clus_Weight1(Clus_Weight1),
                       .Clus_Weight2(Clus_Weight2),
                       .Clus_Scale(Clus_Scale),
                       .Clus_Output(Clus_Output),
                       .Tail1_Output(Tail1_Output),
                       .Tail1_Kn(Tail1_Kn),
                       .Tail1_Start(Tail1_Start),
                       .Tail1_Post_Alive(Tail1_Post_Alive),
                       .Tail2_Output(Tail2_Output),
                       .Tail2_Kn(Tail2_Kn),
                       .Tail2_Start(Tail2_Start),
                       .Tail2_Post_Alive(Tail2_Post_Alive),
                       .Tail3_Output(Tail3_Output),
                       .Tail3_Kn(Tail3_Kn),
```

```
                        .Tail3_Start(Tail3_Start),
                        .Tail3_Post_Alive(Tail3_Post_Alive),
                        .EN(EN),
                        .Alert(Alert),
                        .N(N),
                        .Offset_FB(Offset),
                        .Tail_Curr(Tail_Curr),
                        .Tail_Sum(Tail_Sum),
                        .Pre_Offset(Pre_Offset),
                        .Addr_Null()
                       );

//*************************************************
//         Block5
//*************************************************

//-------------------------------------------------
// Block5 generate the final TRC output and determine
// the system future status depending on the info from
// block4
//-------------------------------------------------

 Output_Proc    Block5(.Xt(Xt_A),
                       .Pre_Offset(Pre_Offset),
                       .N(N),
                       .Kn1(Tail1_Kn),
                       .Kn2(Tail2_Kn),
                       .Kn3(Tail3_Kn),
                       .Start1(Tail1_Start),
                       .Start2(Tail2_Start),
                       .Start3(Tail3_Start),
                       .Tran1_Out(Tail1_Output),
                       .Tran2_Out(Tail2_Output),
                       .Tran3_Out(Tail3_Output),
                       .Post_Alive1(Tail1_Post_Alive),
                       .Post_Alive2(Tail2_Post_Alive),
                       .Post_Alive3(Tail3_Post_Alive),
                       .CLK_5_1(CLK_5_1),
                       .CLK_5_2(CLK_5_2),
                       .CLK_5_3(CLK_5_3),
                       .SCWM_Out(SCWM_Out),
                       .Offset_FB(Offset),
                       .Mode_Sel(Mode_Sel)
                       );

//*************************************************
//            Block6
//*************************************************

//-------------------------------------------------
// Block6 generate all of the pipeline stage control
// CLKs and data path multiplexing signals
//-------------------------------------------------
```

```
pipeline_generation Pipe_CLK
                     (.CLK_SYS(CLK_SYS),
                      .Mode_Sel(Mode_Sel),
                      .Alert(Alert),
                      .N(N),
                      .Am_Multi_Sel(Am_Multi_Sel),
                      .Tran_Sel(Tran_Sel),
                      .Tail_Sel(Tail_Sel),
                      .GCLK0(GCLK0),
                      .GCLK1(GCLK1),
                      .GCLK2(GCLK2),
                      .GCLK3(GCLK3),
                      .GCLK4(GCLK4),
                      .GCLK5(GCLK5),
                      .GCLK6(GCLK6),
                      .GCLK7(GCLK7),
                      .GCLK8(GCLK8),
                      .GCLK9(GCLK9),
                      .GCLK10(GCLK10),
                      .GCLK11(GCLK11),
                      .GCLK12(GCLK12),
                      .GCLK13(GCLK13),
                      .GCLK14(GCLK14),
                      .GCLK15(GCLK15)
                      );

//-------------------------------------------------
//           Pipeline Control CLK Connection
//-------------------------------------------------

assign CLK_2_1     = GCLK0[1],
       CLK_2_2     = GCLK0[10],
       CLK_2_3     = GCLK0[12],
       CLK_3_1_1   = GCLK1[4],
       CLK_3_1_2   = GCLK1[8],
       CLK_3_2_1   = GCLK2[1] | GCLK4[1] | GCLK6[1],
       aCLK_3_2_2  = GCLK2[3],
       bCLK_3_2_2  = GCLK4[3],
       cCLK_3_2_2  = GCLK6[3],
       aCLK_3_2_3  = GCLK2[6],
       bCLK_3_2_3  = GCLK4[6],
       cCLK_3_2_3  = GCLK6[6],
       CLK_3_2_4   = GCLK2[9]  | GCLK4[9]  | GCLK6[9],
       CLK_3_2_5   = GCLK2[12] | GCLK4[12] | GCLK6[12],
       CLK_3_2_6   = GCLK2[14] | GCLK4[14] | GCLK6[14],
       aCLK_3_2_6  = GCLK2[14],
       bCLK_3_2_6  = GCLK4[14],
       cCLK_3_2_6  = GCLK6[14],
       CLK_3_2_7   = GCLK3[1] | GCLK5[1] | GCLK7[1],
       CLK_3_2_8   = GCLK3[3] | GCLK5[3] | GCLK7[3],
       CLK_3_2_9   = GCLK3[6] | GCLK5[6] | GCLK7[6],
       aCLK_3_2_10 = GCLK3[9],
```

```
        bCLK_3_2_10 = GCLK5[9],
        cCLK_3_2_10 = GCLK7[9],
        CLK_3_3_1   = GCLK8[1],
        CLK_3_3_2   = GCLK8[10],
        CLK_3_3_3   = GCLK8[14],
        CLK_3_3_4   = GCLK9[1],
        CLK_3_3_5   = GCLK9[3],
        CLK_3_3_6   = GCLK9[8],
        CLK_3_3_7   = GCLK9[13],
        Aden_LS     = GCLK2[7]  | GCLK4[7]  | GCLK6[7],
        Anum_LS     = GCLK2[10] | GCLK4[10] | GCLK6[10],
        CLK_4_1     = GCLK10[1],
        CLK_4_2     = GCLK10[3],
        CLK_4_3_1   = GCLK12[3],
        CLK_4_3_2   = GCLK12[6],
        CLK_4_3_3   = GCLK12[7],
        CLK_4_3_4   = GCLK12[11],
        CLK_4_3_5   = GCLK12[14],
        CLK_4_3_6   = GCLK12[15],
        CLK_4_3_7   = GCLK13[3],
        CLK_4_3_8   = GCLK13[6],
        CLK_4_3_9   = GCLK13[7],
        CLK_4_4_1   = GCLK14[4],
        CLK_4_4_2   = GCLK14[8],
        CLK_5_1     = GCLK15[4],
        CLK_5_2     = GCLK15[8],
        CLK_5_3     = GCLK15[12];

endmodule

////////////////////////////////////////////////////////////
//
//  transient preprocessing and detection. detect and recover
//  the transient with no overlapping and zero offset, prepare
//  3 data path of transient with different time lag
//  Four modules under this level:
//  1.  trans_prepro: transient preprocessing
//  2.  trans_det: transient detection (comparator)
//  3.  lock_out: lock out state machine to block false
//      detection in lock out period
//  4.  prepro_output: prepare multiple output data path with
//      time lags
//
////////////////////////////////////////////////////////////

module PrePro_TransDet(input    [17:0]    Trans,
                       input    [17:0]    Tail_Curr,
                       input    [17:0]    Tail_Sum,
                       input    [15:0]    Offset,
                       input    [17:0]    N,
                       input              CLK_2_1,
                       input              CLK_2_2,
                       input              CLK_2_3,
```

```
                        output              Alert,
                        output   [17:0]     Xt_A,
                        output   [17:0]     Xt_B,
                        output   [17:0]     Xt_C  );


//*************************************************
// Internal variable and constant definition.
//*************************************************

 wire   [17:0] Threshold = 18'b00_0000_0100_1000_0000;
 wire   [17:0] Xt, Xt_Det, Xt_Sel;
 wire          Trans_Det_True, Alert_Prim, Xt_Mux;

//*************************************************
//   Xt  Mux  to switch the inputs to following block
//   when new transient detect
//*************************************************

MUX_2_1 Xt_SW [17:0] (.D0(Xt),
                      .D1(Xt_Det),
                      .S0(Xt_Mux),
                      .O(Xt_Sel)  );



//*******************************************************
//  transient preprocessing to cancel the tail overlapping
//  and offset, generate Xt for processing and Xt_det for
//  transient detection
//*******************************************************

Trans_Prepro     Trans_Prepro_Inst(  .Tail_Sum(Tail_Sum),
                                     .Tail_Curr(Tail_Curr),
                                     .Offset(Offset),
                                     .Trans(Trans),
                                     .CLK_2_1(CLK_2_1),
                                     .Xt_Det(Xt_Det),
                                     .Xt(Xt) );

//*******************************************************
//     transient detection (comparator)
//*******************************************************

Trans_Det        Trans_Det_Inst(     .Xt_Det(Xt_Det[15:0]),
                                     .Threshold(Threshold[15:0]),
                                     .Trans_Det_True(Trans_Det_True));

//*************************************************************
//  Lock out state machine to prevent false detection in
//  lock out period
//*************************************************************
```

```
Lock_Out         Lock_Out_Inst(      .N(N),
                                     .Trans_Det_True(Trans_Det_True),
                                     .CLK_2_2(CLK_2_2),
                                     .Alert_Prim(Alert_Prim)     );


//*********************************************************
//  Prepare 3 data path of Xt with different time lag to
//  find best match between the data and cluster means
//*********************************************************

Prepro_Output    Prepro_Output_Inst (.Alert_Prim(Alert_Prim),
                                     .Xt(Xt_Sel),
                                     .CLK_2_1(CLK_2_1),
                                     .CLK_2_3(CLK_2_3),
                                     .Alert(Alert),
                                     .Xt_Mux(Xt_Mux),
                                     .Xt_A(Xt_A),
                                     .Xt_B(Xt_B),
                                     .Xt_C(Xt_C)   );
endmodule

////////////////////////////////////////////////////////////
//
//  transient preprocessing to cancel the tail overlapping
//  and offset, generate Xt for processing and Xt_det for
//  transient detection
//
//
////////////////////////////////////////////////////////////
module Trans_Prepro(input      [17:0]   Tail_Sum,
                    input      [17:0]   Tail_Curr,
                    input      [15:0]   Offset,
                    input      [17:0]   Trans,
                    input               CLK_2_1,
                    output     [17:0]   Xt_Det,
                    output     [17:0]   Xt         );

 reg [17:0] Tail_Sum_In = 18'h00000,
            Tail_Curr_In = 18'h00000,
            Trans_In = 18'h00000;
 reg [15:0] Offset_In = 16'h0000;

always @ (posedge CLK_2_1)
  begin
    Tail_Sum_In  <= Tail_Sum;
    Tail_Curr_In <= Tail_Curr;
    Offset_In    <= Offset;
    Trans_In     <= Trans;
  end

assign Xt = Trans_In - (Tail_Sum_In + {2'b00, Offset_In}),
       Xt_Det = Xt - Tail_Curr_In;
```

```
endmodule

////////////////////////////////////////////////////
//  Lock out state machine to prevent false detection in
//  lock out period
//
////////////////////////////////////////////////////
module Lock_Out(input      [17:0]  N,
                input              Trans_Det_True,
                input              CLK_2_2,
                output reg         Alert_Prim = 0     );


 wire [17:0]  LockOut = 18'b00_0000_0000_0000_0110;
 reg  [17:0]  N_Rec   = 18'b00_0000_0000_0000_0000;

// record the start point of transient
 wire         Lock_Out_False;
 always @ (posedge Alert_Prim)  N_Rec <= N;
 assign  Lock_Out_False =  (N > (N_Rec + LockOut)) ? 1 : 0;
 always @ (posedge CLK_2_2)
 Alert_Prim<= Trans_Det_True &&Lock_Out_False;

endmodule


`timescale 1ns / 1ps
////////////////////////////////////////////////////////
//  Prepare 3 data path of Xt with different time lag to
//  find best match between the data and cluster means
//
////////////////////////////////////////////////////////

module Prepro_Output(input                 Alert_Prim,
                     input       [17:0]    Xt,
                     input                 CLK_2_1,
                     input                 CLK_2_3,
                     output reg            Alert = 0,
                     output                Xt_Mux,
                     output reg  [17:0]    Xt_A = 18'h00000,
                     output reg  [17:0]    Xt_B = 18'h00000,
                     output reg  [17:0]    Xt_C = 18'h00000   );

 reg          Alert_Prim_In = 0,
              Alert_In = 0;
 reg [17:0]   Xt_B_In = 18'h00000,
              Xt_C_In = 18'h00000;

assign Xt_Mux = ~Alert_Prim_In & Alert_Prim;

always @ (posedge CLK_2_1)
  begin
```

```
     Alert_Prim_In <= Alert_Prim;
     Xt_B_In       <= Xt;
  end

always @ (posedge CLK_2_1)
  begin
        Xt_C_In <= Xt_B_In;
        Alert_In <= ~Alert_Prim_In & Alert_Prim;
  end

always @ (posedge CLK_2_3)
  begin
     Xt_A  <= Xt;
     Xt_B  <= Xt_B_In;
     Xt_C  <= Xt_C_In;
     Alert <= Alert_In;
  end

endmodule

////////////////////////////////////////////////////////////
//     Block3: Cluster Processing Array Top file
//
//   This block achieves transient scaling and SCWM prediction
//   for current processed transient, and generate the tail
//   prediction information for the using by block4. 5 blocks
//   and input / output interface and processing under this
//   hierarchy level:
//   1. Parameter ppreparation and reset signal generation
//   2. scaling factor and likelihood calculation
//   3. local filter bank
//   4. likelihood comparation
//   5. filter output switch and filter weights generation
//    6. Input:  data path multiplexing
//   7. Output: SCWM output finalization
//
////////////////////////////////////////////////////////////


module Clus_Array( input                        CLK_SYS,
                   input                        CLK_3_1_1,
                   input                        CLK_3_1_2,
                   input                        CLK_3_2_1,
                   input                       aCLK_3_2_2,
                   input                       bCLK_3_2_2,
                   input                       cCLK_3_2_2,
                   input                       aCLK_3_2_3,
                   input                       bCLK_3_2_3,
                   input                       cCLK_3_2_3,
                   input                        CLK_3_2_4,
                   input                        CLK_3_2_5,
                   input                        CLK_3_2_6,
                   input                       aCLK_3_2_6,
```

```
                    input                       bCLK_3_2_6,
                    input                       cCLK_3_2_6,
                    input                        CLK_3_2_7,
                    input                        CLK_3_2_8,
                    input                        CLK_3_2_9,
                    input                       aCLK_3_2_10,
                    input                       bCLK_3_2_10,
                    input                       cCLK_3_2_10,
                    input                        CLK_3_3_1,
                    input                        CLK_3_3_2,
                    input                        CLK_3_3_3,
                    input                        CLK_3_3_4,
                    input                        CLK_3_3_5,
                    input                        CLK_3_3_6,
                    input                        CLK_3_3_7,
                    input                        Aden_LS,
                    input                        Anum_LS,
                    input           [1:0]        Tran_Sel,
                    input                        Am_Multi_Sel,
                    input                        Alert,
                    input           [17:0]       N,
                    input           [17:0]       Xta,
                    input           [17:0]       Xtb,
                    input           [17:0]       Xtc,
                    output                       EN,
                    output          [15:0]       Scale,
                    output          [3:0]        Clus_Num1,
                    output          [3:0]        Clus_Num2,
                    output          [8:0]        Weight1,
                    output          [8:0]        Weight2,
                    output          [8:0]        Clus_Kn,
                    output          [15:0]       Clus_Output
                      );

//************************************************************
//              Internal Signals
//************************************************************

// control signals

wire  [1:0]  M;

//  Data  processing connection wires

 wire [17:0] Xt_Beta_F1, Xt_Beta_F2, Miu_Beta_K2D_F1, Miu_Beta_K2D_F2;
 wire [47:0] F1, F2;
 wire [26:0] FW2, FW1;
 wire [47:0] Likelihood2;
 wire [17:0] Xt, Xt2;
 wire [35:0] Xt2_Full;

 wire [17:0]
 Xt_Beta_A0, Xt_Beta_A1, Xt_Beta_A2, Xt_Beta_A3, Xt_Beta_A4,
```

```
 Xt_Beta_A5, Xt_Beta_A6, Xt_Beta_A7, Xt_Beta_A8, Xt_Beta_A9,
 Xt_Beta_B0, Xt_Beta_B1, Xt_Beta_B2, Xt_Beta_B3, Xt_Beta_B4,
 Xt_Beta_B5, Xt_Beta_B6, Xt_Beta_B7, Xt_Beta_B8, Xt_Beta_B9,
 Xt_Beta_C0, Xt_Beta_C1, Xt_Beta_C2, Xt_Beta_C3, Xt_Beta_C4,
 Xt_Beta_C5, Xt_Beta_C6, Xt_Beta_C7, Xt_Beta_C8, Xt_Beta_C9;

 wire [9:0]
 Am1a, Am2a, Am3a, Am4a, Am5a, Am6a, Am7a, Am8a, Am9a, Am10a,
 Am1b, Am2b, Am3b, Am4b, Am5b, Am6b, Am7b, Am8b, Am9b, Am10b,
 Am1c, Am2c, Am3c, Am4c, Am5c, Am6c, Am7c, Am8c, Am9c, Am10c;

 wire  [47:0]
 Likelihood1a, Likelihood2a, Likelihood3a, Likelihood4a,
 Likelihood5a, Likelihood6a, Likelihood7a, Likelihood8a,
 Likelihood9a, Likelihood10a,
 Likelihood1b, Likelihood2b, Likelihood3b, Likelihood4b,
 Likelihood5b, Likelihood6b, Likelihood7b, Likelihood8b,
 Likelihood9b, Likelihood10b,
 Likelihood1c, Likelihood2c, Likelihood3c, Likelihood4c,
 Likelihood5c, Likelihood6c, Likelihood7c, Likelihood8c,
 Likelihood9c, Likelihood10c;

// Parameters connection wires

 wire  [15:0]
 Miux0, Miux1, Miux2, Miux3, Miux4,
 Miux5, Miux6, Miux7, Miux8, Miux9,
 Beta0, Beta1, Beta2, Beta3, Beta4,
 Beta5, Beta6, Beta7, Beta8, Beta9;

 wire  [17:0]
 Inv_Varx0, Inv_Varx1, Inv_Varx2, Inv_Varx3, Inv_Varx4,
 Inv_Varx5, Inv_Varx6, Inv_Varx7, Inv_Varx8, Inv_Varx9,

 wire  signed [15:0]
 Log_Varx0, Log_Varx1, Log_Varx2, Log_Varx3, Log_Varx4,
 Log_Varx5, Log_Varx6, Log_Varx7, Log_Varx8, Log_Varx9;

 wire  [17:0]
 Miux_Beta_K2D0, Miux_Beta_K2D1, Miux_Beta_K2D2,
 Miux_Beta_K2D3, Miux_Beta_K2D4, Miux_Beta_K2D5,
 Miux_Beta_K2D6, Miux_Beta_K2D7, Miux_Beta_K2D8,
 Miux_Beta_K2D9,
 Miux_Inv_Varx0, Miux_Inv_Varx1, Miux_Inv_Varx2,
 Miux_Inv_Varx3, Miux_Inv_Varx4, Miux_Inv_Varx5,
 Miux_Inv_Varx6, Miux_Inv_Varx7, Miux_Inv_Varx8,
 Miux_Inv_Varx9;

 wire  signed [15:0]
 Log_Pcm0, Log_Pcm1, Log_Pcm2, Log_Pcm3, Log_Pcm4,
 Log_Pcm5, Log_Pcm6, Log_Pcm7, Log_Pcm8, Log_Pcm9,
 Km0, Km1, Km2, Km3, Km4, Km5, Km6, Km7, Km8, Km9;
```

```
//*************************************************************
//                  Input data interface
//*************************************************************


Mux_4_1 Xt_Mux [17:0](.D0(18'h0_0000),
                      .D1(Xta),
                      .D2(Xtb),
                      .D3(Xtc),
                      .S0(Tran_Sel[0]),
                      .S1(Tran_Sel[1]),
                      .O(Xt)
                      );

 assign  Xt2_Full = Xt * Xt;
 assign  Xt2 = Xt2_Full[30:13];



//*************************************************************
//               Output finalization: Clus_Kn
//*************************************************************


Mux_16_1 Clus_Kn_Mux [8:0](        .D1(Km0[8:0]),
                                   .D2(Km1[8:0]),
                                   .D3(Km2[8:0]),
                                   .D4(Km3[8:0]),
                                   .D5(Km4[8:0]),
                                   .D6(Km5[8:0]),
                                   .D7(Km6[8:0]),
                                   .D8(Km7[8:0]),
                                   .D9(Km8[8:0]),
                                   .D10(Km9[8:0]),
                                   .S0(Clus_Num1[0]),
                                   .S1(Clus_Num1[1]),
                                   .S2(Clus_Num1[2]),
                                   .S3(Clus_Num1[3]),
                                   .O(Clus_Kn)
                                  );


//*************************************************************
//                  Output finalization: Clus_Output
//*************************************************************

 wire EN_Bar;
 assign EN_Bar = ~EN;



 DSP48
```

```
DSP48_MA1(
.BCOUT(),                           // 18-bit B cascade output
.P(F1),                             // 48-bit product output
.PCOUT(),                           // 38-bit cascade output
.A(Miu_Beta_K2D_F1),                // 18-bit A data input
.B({2'b00, Scale}),                 // 18-bit B data input
.BCIN(18'h00000),                   // 18-bit B cascade input
.C({13'h0000, Xt_Beta_F1,17'h00000}), // 48-bit cascade input
.CARRYIN(1'b0),                     // Carry input signal
.CARRYINSEL(2'b00),                 // 2-bit carry input select
.CEA(1'b1),                         // A data clock enable input
.CEB(1'b1),                         // B data clock enable input
.CEC(1'b1),                         // C data clock enable input
.CECARRYIN(1'b0),                   // CARRYIN clock enable input
.CECINSUB(1'b0),                    // CINSUB clock enable input
.CECTRL(1'b0),                      // Clock Enable input for CTRL registers
.CEM(1'b1),                         // Clock Enable input for multiplier
                                    // registers
.CEP(1'b1),                         // Clock Enable input for P registers
.CLK(CLK_SYS),                      // Clock input
.OPMODE(7'b011_01_01),              // 7-bit operation mode input
.PCIN(48'h0000_0000_0000),          // 48-bit PCIN input
.RSTA(EN_Bar),                      // Reset input for A pipeline registers
.RSTB(EN_Bar),                      // Reset input for B pipeline registers
.RSTC(EN_Bar),                      // Reset input for C pipeline registers
.RSTCARRYIN(1'b0),                  // Reset input for CARRYIN registers
.RSTCTRL(1'b0),                     // Reset input for CTRL registers
.RSTM(EN_Bar),                      // Reset input for multiplier registers
.RSTP(EN_Bar),                      // Reset input for P pipeline registers
.SUBTRACT(1'b0)                     // SUBTRACT input
);

defparam DSP48_MA1.AREG = 1;
defparam DSP48_MA1.BREG = 1;
defparam DSP48_MA1.B_INPUT = "DIRECT";
defparam DSP48_MA1.CARRYINREG = 0;
defparam DSP48_MA1.CARRYINSELREG = 0;
defparam DSP48_MA1.CREG = 1;
defparam DSP48_MA1.LEGACY_MODE = "MULT18X18S";
defparam DSP48_MA1.MREG = 1;
defparam DSP48_MA1.OPMODEREG = 0;
defparam DSP48_MA1.PREG = 1;
defparam DSP48_MA1.SUBTRACTREG = 0;
// End of DSP48_inst instantiation

DSP48
DSP48_MA2(
.BCOUT(),     // 18-bit B cascade output
.P(F2),                             // 48-bit product output
.PCOUT(),                           // 38-bit cascade output
.A(Miu_Beta_K2D_F2),                // 18-bit A data input
.B({2'b00, Scale}),                 // 18-bit B data input
.BCIN(18'h00000),                   // 18-bit B cascade input
```

```
 .C({13'h0000, Xt_Beta_F2,17'h00000}), // 48-bit cascade input
 .CARRYIN(1'b0),                      // Carry input signal
 .CARRYINSEL(2'b00),                  // 2-bit carry input select
 .CEA(1'b1),                          // A data clock enable input
 .CEB(1'b1),                          // B data clock enable input
 .CEC(1'b1),                          // C data clock enable input
 .CECARRYIN(1'b0),                    // CARRYIN clock enable input
 .CECINSUB(1'b0),                     // CINSUB clock enable input
 .CECTRL(1'b0),                       // Clock Enable input for CTRL registers
 .CEM(1'b1),                          // Clock Enable input for multiplier
                                      // registers
 .CEP(1'b1),                          // Clock Enable input for P registers
 .CLK(CLK_SYS),                       // Clock input
 .OPMODE(7'b011_01_01),               // 7-bit operation mode input
 .PCIN(48'h0000_0000_0000),           // 48-bit PCIN input
 .RSTA(EN_Bar),                       // Reset input for A pipeline registers
 .RSTB(EN_Bar),                       // Reset input for B pipeline registers
 .RSTC(EN_Bar),                       // Reset input for C pipeline registers
 .RSTCARRYIN(1'b0),                   // Reset input for CARRYIN registers
 .RSTCTRL(1'b0),                      // Reset input for CTRL registers
 .RSTM(EN_Bar),                       // Reset input for multiplier registers
 .RSTP(EN_Bar),                       // Reset input for P pipeline registers
 .SUBTRACT(1'b0)                      // SUBTRACT input
 );
 defparam DSP48_MA2.AREG = 1;
 defparam DSP48_MA2.BREG = 1;
 defparam DSP48_MA2.B_INPUT = "DIRECT";
 defparam DSP48_MA2.CARRYINREG = 0;
 defparam DSP48_MA2.CARRYINSELREG = 0;
 defparam DSP48_MA2.CREG = 1;
 defparam DSP48_MA2.LEGACY_MODE = "MULT18X18S";
 defparam DSP48_MA2.MREG = 1;
 defparam DSP48_MA2.OPMODEREG = 0;
 defparam DSP48_MA2.PREG = 1;
 defparam DSP48_MA2.SUBTRACTREG = 0;
 // End of DSP48_inst instantiation

 assign FW2 = F2[34:17] * Weight2;
 assign FW1 = F1[34:17] * Weight1;
 assign Clus_Output = FW2[23:8] + FW1[23:8];

//************************************************************
//       filter output switch and weight generation
//************************************************************

filter_switch_and_weight filter_out_weight (
                                        .CLK_SYS(CLK_SYS),
                                        .CLK_3_3_4(CLK_3_3_4),
                                        .CLK_3_3_5(CLK_3_3_5),
                                        .CLK_3_3_6(CLK_3_3_6),
                                        .CLK_3_3_7(CLK_3_3_7),
                                        .EN(EN),
                                        .M(M),
```

```
.Clus_Num1(Clus_Num1),
.Clus_Num2(Clus_Num2),
.Log_Likelihood2(Likelihood2),
.Log_Pcm1(Log_Pcm0),
.Log_Pcm2(Log_Pcm1),
.Log_Pcm3(Log_Pcm2),
.Log_Pcm4(Log_Pcm3),
.Log_Pcm5(Log_Pcm4),
.Log_Pcm6(Log_Pcm5),
.Log_Pcm7(Log_Pcm6),
.Log_Pcm8(Log_Pcm7),
.Log_Pcm9(Log_Pcm8),
.Log_Pcm10(Log_Pcm9),
.Xt_Beta1a(Xt_Beta_A0),
.Xt_Beta2a(Xt_Beta_A1),
.Xt_Beta3a(Xt_Beta_A2),
.Xt_Beta4a(Xt_Beta_A3),
.Xt_Beta5a(Xt_Beta_A4),
.Xt_Beta6a(Xt_Beta_A5),
.Xt_Beta7a(Xt_Beta_A6),
.Xt_Beta8a(Xt_Beta_A7),
.Xt_Beta9a(Xt_Beta_A8),
.Xt_Beta10a(Xt_Beta_A9),
.Xt_Beta1b(Xt_Beta_B0),
.Xt_Beta2b(Xt_Beta_B1),
.Xt_Beta3b(Xt_Beta_B2),
.Xt_Beta4b(Xt_Beta_B3),
.Xt_Beta5b(Xt_Beta_B4),
.Xt_Beta6b(Xt_Beta_B5),
.Xt_Beta7b(Xt_Beta_B6),
.Xt_Beta8b(Xt_Beta_B7),
.Xt_Beta9b(Xt_Beta_B8),
.Xt_Beta10b(Xt_Beta_B9),
.Xt_Beta1c(Xt_Beta_C0),
.Xt_Beta2c(Xt_Beta_C1),
.Xt_Beta3c(Xt_Beta_C2),
.Xt_Beta4c(Xt_Beta_C3),
.Xt_Beta5c(Xt_Beta_C4),
.Xt_Beta6c(Xt_Beta_C5),
.Xt_Beta7c(Xt_Beta_C6),
.Xt_Beta8c(Xt_Beta_C7),
.Xt_Beta9c(Xt_Beta_C8),
.Xt_Beta10c(Xt_Beta_C9),
.Miu_Beta_K2D1(Miux_Beta_K2D0),
.Miu_Beta_K2D2(Miux_Beta_K2D1),
.Miu_Beta_K2D3(Miux_Beta_K2D2),
.Miu_Beta_K2D4(Miux_Beta_K2D3),
.Miu_Beta_K2D5(Miux_Beta_K2D4),
.Miu_Beta_K2D6(Miux_Beta_K2D5),
.Miu_Beta_K2D7(Miux_Beta_K2D6),
.Miu_Beta_K2D8(Miux_Beta_K2D7),
.Miu_Beta_K2D9(Miux_Beta_K2D8),
.Miu_Beta_K2D10(Miux_Beta_K2D9),
```

```
                                        .Xt_Beta_F1(Xt_Beta_F1),
                                        .Xt_Beta_F2(Xt_Beta_F2),
                                        .Miu_Beta_K2D_F1(Miu_Beta_K2D_F1),
                                        .Miu_Beta_K2D_F2(Miu_Beta_K2D_F2),
                                        .Weight1(Weight1),
                                        .Weight2(Weight2)
                                       );



//************************************************************
//             Likelihood comparison
//************************************************************

Likelihood_Comp Likelihood_Comp_inst(
                        .EN(EN),
                        .CLK_SYS(CLK_SYS),
                        .CLK_3_3_1(CLK_3_3_1),
                        .CLK_3_3_2(CLK_3_3_2),
                        .CLK_3_3_3(CLK_3_3_3),
                        .CLK_3_3_7(CLK_3_3_7),
                        .Likelihood1a(Likelihood1a),
                        .Likelihood2a(Likelihood2a),
                        .Likelihood3a(Likelihood3a),
                        .Likelihood4a(Likelihood4a),
                        .Likelihood5a(Likelihood5a),
                        .Likelihood6a(Likelihood6a),
                        .Likelihood7a(Likelihood7a),
                        .Likelihood8a(Likelihood8a),
                        .Likelihood9a(Likelihood9a),
                        .Likelihood10a(Likelihood10a),
                        .Likelihood1b(Likelihood1b),
                        .Likelihood2b(Likelihood2b),
                        .Likelihood3b(Likelihood3b),
                        .Likelihood4b(Likelihood4b),
                        .Likelihood5b(Likelihood5b),
                        .Likelihood6b(Likelihood6b),
                        .Likelihood7b(Likelihood7b),
                        .Likelihood8b(Likelihood8b),
                        .Likelihood9b(Likelihood9b),
                        .Likelihood10b(Likelihood10b),
                        .Likelihood1c(Likelihood1c),
                        .Likelihood2c(Likelihood2c),
                        .Likelihood3c(Likelihood3c),
                        .Likelihood4c(Likelihood4c),
                        .Likelihood5c(Likelihood5c),
                        .Likelihood6c(Likelihood6c),
                        .Likelihood7c(Likelihood7c),
                        .Likelihood8c(Likelihood8c),
                        .Likelihood9c(Likelihood9c),
                        .Likelihood10c(Likelihood10c),
                        .Am1a(Am1a),
```

```
                        .Am2a(Am2a),
                        .Am3a(Am3a),
                        .Am4a(Am4a),
                        .Am5a(Am5a),
                        .Am6a(Am6a),
                        .Am7a(Am7a),
                        .Am8a(Am8a),
                        .Am9a(Am9a),
                        .Am10a(Am10a),
                        .Am1b(Am1b),
                        .Am2b(Am2b),
                        .Am3b(Am3b),
                        .Am4b(Am4b),
                        .Am5b(Am5b),
                        .Am6b(Am6b),
                        .Am7b(Am7b),
                        .Am8b(Am8b),
                        .Am9b(Am9b),
                        .Am10b(Am10b),
                        .Am1c(Am1c),
                        .Am2c(Am2c),
                        .Am3c(Am3c),
                        .Am4c(Am4c),
                        .Am5c(Am5c),
                        .Am6c(Am6c),
                        .Am7c(Am7c),
                        .Am8c(Am8c),
                        .Am9c(Am9c),
                        .Am10c(Am10c),
                        .Scale(Scale),
                        .M(M),
                        .Clus_Num1(Clus_Num1),
                        .Clus_Num2(Clus_Num2),
                        .Likelihood2(Likelihood2) );



//************************************************************
//         Scaling Factor and Likelihoods
//************************************************************


Am_and_Likelihood Am_Likelihood (
                     .CLK_SYS(CLK_SYS),
                     .CLK_3_2_1(CLK_3_2_1),
                     .aCLK_3_2_2(aCLK_3_2_2),
                     .bCLK_3_2_2(bCLK_3_2_2),
                     .cCLK_3_2_2(cCLK_3_2_2),
                     .aCLK_3_2_3(aCLK_3_2_3),
                     .bCLK_3_2_3(bCLK_3_2_3),
                     .cCLK_3_2_3(cCLK_3_2_3),
                     .CLK_3_2_4(CLK_3_2_4),
```

```
.CLK_3_2_5(CLK_3_2_5),
.CLK_3_2_6(CLK_3_2_6),
.aCLK_3_2_6(aCLK_3_2_6),
.bCLK_3_2_6(bCLK_3_2_6),
.cCLK_3_2_6(cCLK_3_2_6),
.CLK_3_2_7(CLK_3_2_7),
.CLK_3_2_8(CLK_3_2_8),
.CLK_3_2_9(CLK_3_2_9),
.aCLK_3_2_10(aCLK_3_2_10),
.bCLK_3_2_10(bCLK_3_2_10),
.cCLK_3_2_10(cCLK_3_2_10),
.Am_Multi_Sel(Am_Multi_Sel),
.Tran_Sel(Tran_Sel),
.EN(EN),
.Aden_LS(Aden_LS),
.Anum_LS(Anum_LS),
.Xt2(Xt2),
.Xt(Xt),
.Inv_Varx0(Inv_Varx0),
.Inv_Varx1(Inv_Varx1),
.Inv_Varx2(Inv_Varx2),
.Inv_Varx3(Inv_Varx3),
.Inv_Varx4(Inv_Varx4),
.Inv_Varx5(Inv_Varx5),
.Inv_Varx6(Inv_Varx6),
.Inv_Varx7(Inv_Varx7),
.Inv_Varx8(Inv_Varx8),
.Inv_Varx9(Inv_Varx9),
.Miux_Inv_Varx0(Miux_Inv_Varx0),
.Miux_Inv_Varx1(Miux_Inv_Varx1),
.Miux_Inv_Varx2(Miux_Inv_Varx2),
.Miux_Inv_Varx3(Miux_Inv_Varx3),
.Miux_Inv_Varx4(Miux_Inv_Varx4),
.Miux_Inv_Varx5(Miux_Inv_Varx5),
.Miux_Inv_Varx6(Miux_Inv_Varx6),
.Miux_Inv_Varx7(Miux_Inv_Varx7),
.Miux_Inv_Varx8(Miux_Inv_Varx8),
.Miux_Inv_Varx9(Miux_Inv_Varx9),
.Miux0(Miux0),
.Miux1(Miux1),
.Miux2(Miux2),
.Miux3(Miux3),
.Miux4(Miux4),
.Miux5(Miux5),
.Miux6(Miux6),
.Miux7(Miux7),
.Miux8(Miux8),
.Miux9(Miux9),
.Log_Varx0(Log_Varx0),
.Log_Varx1(Log_Varx1),
.Log_Varx2(Log_Varx2),
.Log_Varx3(Log_Varx3),
.Log_Varx4(Log_Varx4),
```

```
.Log_Varx5(Log_Varx5),
.Log_Varx6(Log_Varx6),
.Log_Varx7(Log_Varx7),
.Log_Varx8(Log_Varx8),
.Log_Varx9(Log_Varx9),
.Am1a(Am1a),
.Am2a(Am2a),
.Am3a(Am3a),
.Am4a(Am4a),
.Am5a(Am5a),
.Am6a(Am6a),
.Am7a(Am7a),
.Am8a(Am8a),
.Am9a(Am9a),
.Am10a(Am10a),
.Am1b(Am1b),
.Am2b(Am2b),
.Am3b(Am3b),
.Am4b(Am4b),
.Am5b(Am5b),
.Am6b(Am6b),
.Am7b(Am7b),
.Am8b(Am8b),
.Am9b(Am9b),
.Am10b(Am10b),
.Am1c(Am1c),
.Am2c(Am2c),
.Am3c(Am3c),
.Am4c(Am4c),
.Am5c(Am5c),
.Am6c(Am6c),
.Am7c(Am7c),
.Am8c(Am8c),
.Am9c(Am9c),
.Am10c(Am10c),
.Likelihood1a(Likelihood1a),
.Likelihood2a(Likelihood2a),
.Likelihood3a(Likelihood3a),
.Likelihood4a(Likelihood4a),
.Likelihood5a(Likelihood5a),
.Likelihood6a(Likelihood6a),
.Likelihood7a(Likelihood7a),
.Likelihood8a(Likelihood8a),
.Likelihood9a(Likelihood9a),
.Likelihood10a(Likelihood10a),
.Likelihood1b(Likelihood1b),
.Likelihood2b(Likelihood2b),
.Likelihood3b(Likelihood3b),
.Likelihood4b(Likelihood4b),
.Likelihood5b(Likelihood5b),
.Likelihood6b(Likelihood6b),
.Likelihood7b(Likelihood7b),
.Likelihood8b(Likelihood8b),
```

```
                    .Likelihood9b(Likelihood9b),
                    .Likelihood10b(Likelihood10b),
                    .Likelihood1c(Likelihood1c),
                    .Likelihood2c(Likelihood2c),
                    .Likelihood3c(Likelihood3c),
                    .Likelihood4c(Likelihood4c),
                    .Likelihood5c(Likelihood5c),
                    .Likelihood6c(Likelihood6c),
                    .Likelihood7c(Likelihood7c),
                    .Likelihood8c(Likelihood8c),
                    .Likelihood9c(Likelihood9c),
                    .Likelihood10c(Likelihood10c)
                   );




//************************************************************
//                      Filter Bank
//************************************************************


Filter_Bank Filters(     .CLK_SYS(CLK_SYS),
                         .CLK_3_2_1(CLK_3_2_1),
                         .aCLK_3_2_2(aCLK_3_2_2),
                         .bCLK_3_2_2(bCLK_3_2_2),
                         .cCLK_3_2_2(cCLK_3_2_2),
                         .EN(EN),
                         .Tran_Sel(Tran_Sel),
                         .Xt(Xt),
                         .Beta0(Beta0),
                         .Beta1(Beta1),
                         .Beta2(Beta2),
                         .Beta3(Beta3),
                         .Beta4(Beta4),
                         .Beta5(Beta5),
                         .Beta6(Beta6),
                         .Beta7(Beta7),
                         .Beta8(Beta8),
                         .Beta9(Beta9),
                         .Xt_Beta_A0(Xt_Beta_A0),
                         .Xt_Beta_A1(Xt_Beta_A1),
                         .Xt_Beta_A2(Xt_Beta_A2),
                         .Xt_Beta_A3(Xt_Beta_A3),
                         .Xt_Beta_A4(Xt_Beta_A4),
                         .Xt_Beta_A5(Xt_Beta_A5),
                         .Xt_Beta_A6(Xt_Beta_A6),
                         .Xt_Beta_A7(Xt_Beta_A7),
                         .Xt_Beta_A8(Xt_Beta_A8),
                         .Xt_Beta_A9(Xt_Beta_A9),
                         .Xt_Beta_B0(Xt_Beta_B0),
                         .Xt_Beta_B1(Xt_Beta_B1),
```

```
                        .Xt_Beta_B2(Xt_Beta_B2),
                        .Xt_Beta_B3(Xt_Beta_B3),
                        .Xt_Beta_B4(Xt_Beta_B4),
                        .Xt_Beta_B5(Xt_Beta_B5),
                        .Xt_Beta_B6(Xt_Beta_B6),
                        .Xt_Beta_B7(Xt_Beta_B7),
                        .Xt_Beta_B8(Xt_Beta_B8),
                        .Xt_Beta_B9(Xt_Beta_B9),
                        .Xt_Beta_C0(Xt_Beta_C0),
                        .Xt_Beta_C1(Xt_Beta_C1),
                        .Xt_Beta_C2(Xt_Beta_C2),
                        .Xt_Beta_C3(Xt_Beta_C3),
                        .Xt_Beta_C4(Xt_Beta_C4),
                        .Xt_Beta_C5(Xt_Beta_C5),
                        .Xt_Beta_C6(Xt_Beta_C6),
                        .Xt_Beta_C7(Xt_Beta_C7),
                        .Xt_Beta_C8(Xt_Beta_C8),
                        .Xt_Beta_C9(Xt_Beta_C9)
                       );

//************************************************************
//        Para Mem block and EN generation
//************************************************************


CWM_Para Para_Mem (     .CLK_3_1_1(CLK_3_1_1),
                        .CLK_3_1_2(CLK_3_1_2),
                        .CLK_SYS(CLK_SYS),
                        .Alert(Alert),
                        .N(N),
                        .EN(EN),
                        .Miux0(Miux0),
                        .Miux1(Miux1),
                        .Miux2(Miux2),
                        .Miux3(Miux3),
                        .Miux4(Miux4),
                        .Miux5(Miux5),
                        .Miux6(Miux6),
                        .Miux7(Miux7),
                        .Miux8(Miux8),
                        .Miux9(Miux9),
                        .Beta0(Beta0),
                        .Beta1(Beta1),
                        .Beta2(Beta2),
                        .Beta3(Beta3),
                        .Beta4(Beta4),
                        .Beta5(Beta5),
                        .Beta6(Beta6),
                        .Beta7(Beta7),
                        .Beta8(Beta8),
                        .Beta9(Beta9),
                        .Inv_Varx0(Inv_Varx0),
                        .Inv_Varx1(Inv_Varx1),
```

```
.Inv_Varx2(Inv_Varx2),
.Inv_Varx3(Inv_Varx3),
.Inv_Varx4(Inv_Varx4),
.Inv_Varx5(Inv_Varx5),
.Inv_Varx6(Inv_Varx6),
.Inv_Varx7(Inv_Varx7),
.Inv_Varx8(Inv_Varx8),
.Inv_Varx9(Inv_Varx9),
.Log_Varx0(Log_Varx0),
.Log_Varx1(Log_Varx1),
.Log_Varx2(Log_Varx2),
.Log_Varx3(Log_Varx3),
.Log_Varx4(Log_Varx4),
.Log_Varx5(Log_Varx5),
.Log_Varx6(Log_Varx6),
.Log_Varx7(Log_Varx7),
.Log_Varx8(Log_Varx8),
.Log_Varx9(Log_Varx9),
.Miux_Beta_K2D0(Miux_Beta_K2D0),
.Miux_Beta_K2D1(Miux_Beta_K2D1),
.Miux_Beta_K2D2(Miux_Beta_K2D2),
.Miux_Beta_K2D3(Miux_Beta_K2D3),
.Miux_Beta_K2D4(Miux_Beta_K2D4),
.Miux_Beta_K2D5(Miux_Beta_K2D5),
.Miux_Beta_K2D6(Miux_Beta_K2D6),
.Miux_Beta_K2D7(Miux_Beta_K2D7),
.Miux_Beta_K2D8(Miux_Beta_K2D8),
.Miux_Beta_K2D9(Miux_Beta_K2D9),
.Miux_Inv_Varx0(Miux_Inv_Varx0),
.Miux_Inv_Varx1(Miux_Inv_Varx1),
.Miux_Inv_Varx2(Miux_Inv_Varx2),
.Miux_Inv_Varx3(Miux_Inv_Varx3),
.Miux_Inv_Varx4(Miux_Inv_Varx4),
.Miux_Inv_Varx5(Miux_Inv_Varx5),
.Miux_Inv_Varx6(Miux_Inv_Varx6),
.Miux_Inv_Varx7(Miux_Inv_Varx7),
.Miux_Inv_Varx8(Miux_Inv_Varx8),
.Miux_Inv_Varx9(Miux_Inv_Varx9),
.Log_Pcm0(Log_Pcm0),
.Log_Pcm1(Log_Pcm1),
.Log_Pcm2(Log_Pcm2),
.Log_Pcm3(Log_Pcm3),
.Log_Pcm4(Log_Pcm4),
.Log_Pcm5(Log_Pcm5),
.Log_Pcm6(Log_Pcm6),
.Log_Pcm7(Log_Pcm7),
.Log_Pcm8(Log_Pcm8),
.Log_Pcm9(Log_Pcm9),
.Km0(Km0),
.Km1(Km1),
.Km2(Km2),
.Km3(Km3),
.Km4(Km4),
```

```
                        .Km5(Km5),
                        .Km6(Km6),
                        .Km7(Km7),
                        .Km8(Km8),
                        .Km9(Km9)   );

endmodule


////////////////////////////////////////////////////////////////////////
//
//       CWM model parameter storage and preparation and RST / EN signal
//       generation
//
////////////////////////////////////////////////////////////////////////

module CWM_Para(     input                       CLK_3_1_1,
                     input                       CLK_3_1_2,
                     input                       CLK_SYS,
                     input                       Alert,
                     input           [17:0]      N,
                     output                      EN,
                     output          [15:0]      Miux0,
                     output          [15:0]      Miux1,
                     output          [15:0]      Miux2,
                     output          [15:0]      Miux3,
                     output          [15:0]      Miux4,
                     output          [15:0]      Miux5,
                     output          [15:0]      Miux6,
                     output          [15:0]      Miux7,
                     output          [15:0]      Miux8,
                     output          [15:0]      Miux9,
                     output          [15:0]      Beta0,
                     output          [15:0]      Beta1,
                     output          [15:0]      Beta2,
                     output          [15:0]      Beta3,
                     output          [15:0]      Beta4,
                     output          [15:0]      Beta5,
                     output          [15:0]      Beta6,
                     output          [15:0]      Beta7,
                     output          [15:0]      Beta8,
                     output          [15:0]      Beta9,
                     output          [17:0]      Inv_Varx0,
                     output          [17:0]      Inv_Varx1,
                     output          [17:0]      Inv_Varx2,
                     output          [17:0]      Inv_Varx3,
                     output          [17:0]      Inv_Varx4,
                     output          [17:0]      Inv_Varx5,
                     output          [17:0]      Inv_Varx6,
                     output          [17:0]      Inv_Varx7,
                     output          [17:0]      Inv_Varx8,
                     output          [17:0]      Inv_Varx9,
                     output    signed [15:0]     Log_Varx0,
```

```
                output    signed [15:0]      Log_Varx1,
                output    signed [15:0]      Log_Varx2,
                output    signed [15:0]      Log_Varx3,
                output    signed [15:0]      Log_Varx4,
                output    signed [15:0]      Log_Varx5,
                output    signed [15:0]      Log_Varx6,
                output    signed [15:0]      Log_Varx7,
                output    signed [15:0]      Log_Varx8,
                output    signed [15:0]      Log_Varx9,
                output           [17:0]      Miux_Beta_K2D0,
                output           [17:0]      Miux_Beta_K2D1,
                output           [17:0]      Miux_Beta_K2D2,
                output           [17:0]      Miux_Beta_K2D3,
                output           [17:0]      Miux_Beta_K2D4,
                output           [17:0]      Miux_Beta_K2D5,
                output           [17:0]      Miux_Beta_K2D6,
                output           [17:0]      Miux_Beta_K2D7,
                output           [17:0]      Miux_Beta_K2D8,
                output           [17:0]      Miux_Beta_K2D9,
                output           [17:0]      Miux_Inv_Varx0,
                output           [17:0]      Miux_Inv_Varx1,
                output           [17:0]      Miux_Inv_Varx2,
                output           [17:0]      Miux_Inv_Varx3,
                output           [17:0]      Miux_Inv_Varx4,
                output           [17:0]      Miux_Inv_Varx5,
                output           [17:0]      Miux_Inv_Varx6,
                output           [17:0]      Miux_Inv_Varx7,
                output           [17:0]      Miux_Inv_Varx8,
                output           [17:0]      Miux_Inv_Varx9,
                output    signed [15:0]      Log_Pcm0,
                output    signed [15:0]      Log_Pcm1,
                output    signed [15:0]      Log_Pcm2,
                output    signed [15:0]      Log_Pcm3,
                output    signed [15:0]      Log_Pcm4,
                output    signed [15:0]      Log_Pcm5,
                output    signed [15:0]      Log_Pcm6,
                output    signed [15:0]      Log_Pcm7,
                output    signed [15:0]      Log_Pcm8,
                output    signed [15:0]      Log_Pcm9,
                output           [15:0]      Km0,
                output           [15:0]      Km1,
                output           [15:0]      Km2,
                output           [15:0]      Km3,
                output           [15:0]      Km4,
                output           [15:0]      Km5,
                output           [15:0]      Km6,
                output           [15:0]      Km7,
                output           [15:0]      Km8,
                output           [15:0]      Km9      );

//*********************************************************
//                  internal signals
//*********************************************************
```

```
 reg         Alert_Pipe = 0;
 reg         EN2 = 0;
 reg  [17:0] Tran_Start = 0;
 reg  [9:0]  Addr_Out = 0;
 wire        N_CLK, EN1, EN2_In, AddrLE50;
 wire [17:0] Addr_In;

//************************************************************
// EN1: Reset signal generated at the start of Alert, to
//      clear out the SCWM accummulators during overlap
//************************************************************

 always @ (posedge CLK_3_1_2) Alert_Pipe <= Alert;
 assign EN1 = ~( (~Alert_Pipe) & Alert );

//************************************************************
//               Tran start point recoder
//************************************************************

 assign N_CLK = CLK_3_1_1 & Alert;
 always @ (posedge N_CLK) Tran_Start <= N;
 assign Addr_In = N - Tran_Start;

//************************************************************
//     EN2: Set for 50 steps after Alert set
//          and generate parameter MEM address
//************************************************************

 assign EN2_In = EN2;
 assign AddrLE50 = (Addr_In < 18'b00_0000_0000_0011_0010) ? 1 : 0;
 always @ (posedge CLK_3_1_2)
  begin
    Addr_Out <= (AddrLE50) ? (Addr_In) : (10'b00_1000_0000);
    EN2      <=  Alert | ( AddrLE50 & EN2_In);
  end

//************************************************************
//  EN : Global reset/enable signal that
//************************************************************
assign EN = EN1 & EN2;



//*********************************************************************
//  SCWM Parameters MEM Blocks Initialization
//  Width : 16/18(32) x Depth : 50
//*********************************************************************

// ----------------------------------------------------------
// (1) Miux         U8/8
//  ----------------------------------------------------------
```

```
//Miux0
RAMB16_S18 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"), // "WRITE_FIRST", "READ_FIRST", or
"NO_CHANGE"
// The following generic INIT_xx declarations specify the initial contents of
// the RAM
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000000423040c),
.INIT_02(256'h03ec03d703df03f9041104230436042e041e040804090421042c0431042803f1),
.INIT_01(256'h0390030a028b022c01f601dd01dc02080282039804ec067b0abd10de1b7c235e),
.INIT_00(256'h253a255823b520271c00176e12370efb0f7112a5162a16ae13760eea09d10471)
)
 RAM16_Miux0(
 .DO(Miux0),                    // 16-bit A port data output
 .DOP(),                        // 2-bit A port parity data output
 .ADDR(Addr_Out),               // 10-bit A port address input
 .CLK(CLK_SYS),                 // 1-bit A port clock input
 .DI(16'hFFFF),                 // 16-bit A port data input
 .DIP(2'b11),                   // 2-bit A port parity data input
 .EN(EN),                       // 1-bit A port enable input
 .SSR(1'b0),                    // 1-bit A port set/reset input
 .WE(1'b0)                      // 1-bit A port write enable input
 );

//Miux1
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000004360419),
.INIT_02(256'h041d043404330429041204 0c040e040e041a0439045d046b047d048e044b03c2),
.INIT_01(256'h033802c7026f0233020a01ee01f0023602e5042c056c07ad0c69132d1dea23df),
.INIT_00(256'h24d8246321911e2e1a81164f115a0ecb0f9c1309162d168b13170e900998044c)
) RAM16_Miux1(.DO(Miux1),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
              .DI(16'hFFFF),
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)
            );

//Miux2
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000042e0448),
.INIT_02(256'h045d046504404100400040a0411041e04340454045e0452044e044d0442041b),
```

```
.INIT_01(256'h03ca035002c4024601f401d501d201f20250034004ad060709820ee4183121df),
.INIT_00(256'h24a6250023aa20ba1d0317fd12df0fdf0fdf13c71796179013ba0ee209d4047e)
) RAM16_Miux2(.DO(Miux2),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
              .DI(16'hFFFF),
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)
             );

//Miux3
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000fd80fd5),
.INIT_02(256'h0fda0fe10fdc0fd90fd60fda0fe20fdd0fda0fd70fdb0fe30fde0fdb0fd80fdd),
.INIT_01(256'h0fe50fe00fde0fdc0fe20fea0fe80fe80fe80ff00ffc0fff1005100c101d1036),
.INIT_00(256'h10491060107d10a810ec1139118c11f3128313841509172f1a381f5b1e8706db)
) RAM16_Miux3(.DO(Miux3),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
              .DI(16'hFFFF),
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)
            );

//Miux4
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000fd60fd6),
.INIT_02(256'h0fd80fda0fdb0fd60fd70fd90fda0fdb0fd70fd80fd90fdc0fdc0fd80fd90fdb),
.INIT_01(256'h0fde0fdf0fdc0fdd0fe00fe30fe60fe50fe90fee0ff50ffd1001100c101b102e),
.INIT_00(256'h1045105b107c10a610de112a117f11ee1283136514c616be19cf1f181cbd06a7)
)RAM16_Miux4( .DO(Miux4),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
              .DI(16'hFFFF),
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)
            );
```

```
//Miux5
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000015111523),
.INIT_02(256'h15a315dc163916151611169e17051765173117501806188419141 8dd191c19e5),
.INIT_01(256'h1a871b621b421b841c9b1d8e1ec51ead1f3e20b021f423ac23e924e72708293e),
.INIT_00(256'h2b742c9e2e3f31a1356338cd3bb33edd45634ade50cb571560ce65da46b30870)
)RAM16_Miux5( .DO(Miux5),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
              .DI(16'hFFFF),
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)
            );

//Miux6
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000001516151d),
.INIT_02(256'h155115a515eb15fe160d164916b4171117301746179618 1b189d18d018f9195c),
.INIT_01(256'h1a091acb1b1d1b5a1bea1cd91de71e6d1edb1fae20f92272235c242c258627a6),
.INIT_00(256'h29c12b5b2cce2f1232a835fb39073bf740bb46594bdd51a958a05dce3e120770)
) RAM16_Miux6(  .DO(Miux6),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
              .DI(16'hFFFF),
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)   );

//Miux7
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000015491560),
.INIT_02(256'h158315e5161b163b165a168a170a1755178317ae17ee188f18fb193c197d19dc),
.INIT_01(256'h1aac1b541bb91c221ca61dbf1eb31f581ffa20c6225523c124de25fc276c29d0),
.INIT_00(256'h2c062ddb2fc532a536873a513dd9419547484db654645b72636464fd420d081a)
)RAM16_Miux7(  .DO(Miux7),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
              .DI(16'hFFFF),
```

```
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)
            );

//Miux8
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000049da4a5b),
.INIT_02(256'h4b204bec4c724cde4d7f4e5f4f515000508e5152526053825465552b56315790),
.INIT_01(256'h58d559e45af05c6a5dfc5f3d603f6132625a6368648665846660670e6799684d),
.INIT_00(256'h68e6694569a96a2a6ae56b826be16c436cbf6d6d6df26e276e3f6db348c008aa)
)RAM16_Miux8(  .DO(Miux8),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
              .DI(16'hFFFF),
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)
            );

//Miux9
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000004c754d2a),
.INIT_02(256'h4e124ee84f654ffd50e251fa52f253895458559b56f8580d58fb5a1e5bac5d46),
.INIT_01(256'h5e8f5f72608a61b962c563f264dd65d666a0675d6827689268f2695a69fd6ac3),
.INIT_00(256'h6b2b6b906c056ca46d6e6dd76e366e986f276fdf701b704c706570b64a75080b)
)RAM16_Miux9(  .DO(Miux9),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
              .DI(16'hFFFF),
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)
            );

// ------------------------------------------------------
// (2) Beta         U0/16
//  ------------------------------------------------------

//Beta0
RAMB16_S18 #(
```

```
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000005360539),
.INIT_02(256'h0566058e0589056f0558054f053605350544055e055f0543053e053905440580),
.INIT_01(256'h05fd0704085a09a30a8f0b120b340a51085705da0459034b01fd014c00ca009e),
.INIT_00(256'h00940093009c00ae00c600ee013501700163011f00f300f4011c0174023204d6)
) RAM16_Beta0(  .DO(Beta0),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
              .DI(16'hFFFF),
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)
            );


//Beta1
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h000000000000000000000000000000000000000000000000000000000547055f),
.INIT_02(256'h053504fc051e054905950591056c057f059c0593055e05460529050105530616),
.INIT_01(256'h06e60805097f0ad40baa0c120bca09d3073c050e03ed02c701bb011d00b8009a),
.INIT_00(256'h00940093009d00af00d300fc01450171015d011600f100f30121017a024004f6)
) RAM16_Beta1(  .DO(Beta1),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
              .DI(16'hFFFF),
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)
            );

//Beta2
RAMB16_S18 #(

 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000005410532),
.INIT_02(256'h0526052e05520581057e0565056b056f05680542051e052305340541054c0572),
.INIT_01(256'h05e206b107e409690ad10b8e0bb90adb08d50626047f03830233016b00e100a5),
.INIT_00(256'h00980095009e00b200c600e9013101700164011e00f200f50124017e024204f5)
)RAM16_Beta2(  .DO(Beta2),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
```

```
              .DI(16'hFFFF),
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)
            );

//Beta3
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000005200520),
.INIT_02(256'h051f051e051f05200520051d051e0520052005 1f051b051d051f051f051f),
.INIT_01(256'h051b051e051f051d051c051a051c051a0519051a05150513051205 0f050a0501),
.INIT_00(256'h04fd04f504ed04e004cc04b904a104860464042a03df0383031a029802a40ae8)
)RAM16_Beta3(  .DO(Beta3),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
              .DI(16'hFFFF),
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)
            );

//Beta4
RAMB16_S18 #(

 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000005230524),
.INIT_02(256'h0523052305210523052405230522052105220524052205200521052205240522),
.INIT_01(256'h0521051f052205210521051f051d051e051e051c0519051805150513050f0508),
.INIT_00(256'h050004f904f004e304d204bd04a504890465043203e8038e031f029602bb0a9a)
)RAM16_Beta4(  .DO(Beta4),
              .DOP(),
              .ADDR(Addr_Out),
              .CLK(CLK_SYS),
              .DI(16'hFFFF),
              .DIP(2'b11),
              .EN(EN),
              .SSR(1'b0),
              .WE(1'b0)
            );

//Beta5
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
```

```
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000004150412),
.INIT_02(256'h03fb03f003e003e703e503cf03bd03ae03b403af03940384037003750 36c0353),
.INIT_01(256'h033f03250328031e030402e902cc02cc02c002a30289026a0263025302350217),
.INIT_00(256'h01fb01ed01dc01bc019c01840170015e013e0126011100fc00e500d9013b0a30)
)RAM16_Beta5(  .DO(Beta5),
               .DOP(),
               .ADDR(Addr_Out),
               .CLK(CLK_SYS),
               .DI(16'hFFFF),
               .DIP(2'b11),
               .EN(EN),
               .SSR(1'b0),
               .WE(1'b0)
            );

//Beta6
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000420041c),
.INIT_02(256'h0413040103f603f703ed03e503d103c203c103b703ae0398038803830377036c),
.INIT_01(256'h0354033c03340328031a02fe02e402dc02cc02bd02a0028302740262024d022b),
.INIT_00(256'h021001fe01ec01d601b201970181016e01520137012 1010c00f800e901560a09)
)RAM16_Beta6(  .DO(Beta6),
               .DOP(),
               .ADDR(Addr_Out),
               .CLK(CLK_SYS),
               .DI(16'hFFFF),
               .DIP(2'b11),
               .EN(EN),
               .SSR(1'b0),
               .WE(1'b0)
            );

//Beta7
RAMB16_S18 #(

 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000413040d),
.INIT_02(256'h040703f603e803e703de03d803c303b303b003a8039f03870375036f03660358),
.INIT_01(256'h0340032903200313030502ea02d202c402b502a50286026c025a024702320213),
.INIT_00(256'h01f601e301d001b80197017d016501520136011d010600f300df00dc01480932)
)RAM16_Beta7(  .DO(Beta7),
               .DOP(),
               .ADDR(Addr_Out),
               .CLK(CLK_SYS),
               .DI(16'hFFFF),
               .DIP(2'b11),
               .EN(EN),
               .SSR(1'b0),
```

```
              .WE(1'b0)
            );

//Beta8
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000003f703f1),
.INIT_02(256'h03e703dc03d403d003c703bd03b003a803a1039b038e037f0377036f03650356),
.INIT_01(256'h034a034003370329031b0312030a030202f802f302ea02e302dc02d802d602d0),
.INIT_00(256'h02cb02ca02c702c402be02ba02b802b602b202ae02ac02a902a902ac03ec1c49)
)RAM16_Beta8(  .DO(Beta8),
               .DOP(),
               .ADDR(Addr_Out),
               .CLK(CLK_SYS),
               .DI(16'hFFFF),
               .DIP(2'b11),
               .EN(EN),
               .SSR(1'b0),
               .WE(1'b0)
            );

//Beta9
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000003d703d2),
.INIT_02(256'h03c503ba03b403ae03a60399038c0385037d037203640357034f034503370327),
.INIT_01(256'h031d0315030e030302fb02f202ec02e402e002db02d402d102cd02cc02c802c1),
.INIT_00(256'h02be02bc02bb02b502af02ad02aa02aa02a702a2029f029c029e029d03b21754)
)RAM16_Beta9(  .DO(Beta9),
               .DOP(),
               .ADDR(Addr_Out),
               .CLK(CLK_SYS),
               .DI(16'hFFFF),
               .DIP(2'b11),
               .EN(EN),
               .SSR(1'b0),
               .WE(1'b0)
            );


// ------------------------------------------------------
// (3) Log_Varx          S5/11
//  ------------------------------------------------------

// Log_Varx0
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
```

```
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000000000000ed94ed94),
.INIT_02(256'hed94ed94ed94ed94ed94ed94ed94ed94eee0ed94ed94ed94ed94ed94ed94ed94),
.INIT_01(256'hed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ef84f8fbfaeefe47f189),
.INIT_00(256'hf681fab3fdaa00b9064e01bf01a6f140f20d0157fe94fc53f710ed94ed94ed94)
)RAM16_Log_Varx0(  .DO(Log_Varx0),
                   .DOP(),
                   .ADDR(Addr_Out),
                   .CLK(CLK_SYS),
                   .DI(16'hFFFF),
                   .DIP(2'b11),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                 );


// Log_Varx1
RAMB16_S18 #(

 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000000000000ed94ed94),
.INIT_02(256'hed94ed94ed94ed94ef46f0f1ed94ed94eee8f0caee95ed94eecbeedbf1a5f1c1),
.INIT_01(256'heeb3eef5f110ef80ed94ed94ed94ed94ed94ed94ed94f265f5db021102b7ed94),
.INIT_00(256'hf4eb0582098203c1f9310467fe45f7dfef2cfd34fcf3ed94ef65ed94ed94ed94)
)RAM16_Log_Varx1(  .DO(Log_Varx1),
                   .DOP(),
                   .ADDR(Addr_Out),
                   .CLK(CLK_SYS),
                   .DI(16'hFFFF),
                   .DIP(2'b11),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                 );

// Log_Varx2
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000000000000edbced94),
.INIT_02(256'hed94ed94ed94eec7ef1cf10df23af055f08beff4ee5ced94ef02eee9ed94ed94),
.INIT_01(256'hed94ee73edbeedb6ed94ed94ed94ed94ed94ed94ed94ed94f826f9cefcbff252),
.INIT_00(256'hf9c9fe63fe71087904660818feb3fa9df1acfaf9f4f2f7c0f7d3ed94ed94ed94)
 )RAM16_Log_Varx2(  .DO(Log_Varx2),
                   .DOP(),
                   .ADDR(Addr_Out),
                   .CLK(CLK_SYS),
                   .DI(16'hFFFF),
                   .DIP(2'b11),
                   .EN(EN),
```

```
                  .SSR(1'b0),
                  .WE(1'b0)
                );

// Log_Varx3
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000ed94ed94),
.INIT_02(256'hed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94),
.INIT_01(256'hed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94),
.INIT_00(256'hed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94f9f400f0028b)
)RAM16_Log_Varx3(  .DO(Log_Varx3),
                   .DOP(),
                   .ADDR(Addr_Out),
                   .CLK(CLK_SYS),
                   .DI(16'hFFFF),
                   .DIP(2'b11),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                );

// Log_Varx4
RAMB16_S18 #(

 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000ed94ed94),
.INIT_02(256'hed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94),
.INIT_01(256'hed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94),
.INIT_00(256'hed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94efb8f99604550c8b04dc)
)RAM16_Log_Varx4(  .DO(Log_Varx4),
                   .DOP(),
                   .ADDR(Addr_Out),
                   .CLK(CLK_SYS),
                   .DI(16'hFFFF),
                   .DIP(2'b11),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                );

// Log_Varx5
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000ed94ed94),
.INIT_02(256'hed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94),
.INIT_01(256'hed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94),
```

```
.INIT_00(256'hed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94)
)RAM16_Log_Varx5(  .DO(Log_Varx5),
                   .DOP(),
                   .ADDR(Addr_Out),
                   .CLK(CLK_SYS),
                   .DI(16'hFFFF),
                   .DIP(2'b11),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                 );

// Log_Varx6
RAMB16_S18 #(

 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000ed94ed94),
.INIT_02(256'hed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94),
.INIT_01(256'hed94ed94ed94eea7eebeed94ef5bed94f2abf19ced94f465ed94f835f752f51f),
.INIT_00(256'hf934f5fdfca9fdbffe55ffc6011e058806a90659098f0bcf0e77106d1f0809e6)
)RAM16_Log_Varx6(  .DO(Log_Varx6),
                   .DOP(),
                   .ADDR(Addr_Out),
                   .CLK(CLK_SYS),
                   .DI(16'hFFFF),
                   .DIP(2'b11),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                 );

// Log_Varx7
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000ed94ed94),
.INIT_02(256'hed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94ed94),
.INIT_01(256'hed94ed94ed94ed94ed94ed94ed94ed94f20bed94f013ee5fee1af573ee82f4c5),
.INIT_00(256'hef2df262f8a5f5cffa0ef68dfac5ff9c0084ffc60445067a040711bf20730b24)
)RAM16_Log_Varx7(  .DO(Log_Varx7),
                   .DOP(),
                   .ADDR(Addr_Out),
                   .CLK(CLK_SYS),
                   .DI(16'hFFFF),
                   .DIP(2'b11),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                 );
```

```
// Log_Varx8
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000f131ee1d),
.INIT_02(256'hedbbef57f274f308f090f0cff294f609f729f50af39af538f8b4f9a8f80ef7e4),
.INIT_01(256'hf9c8fc0ffcf7fce3fb8ef981fa52fa8bf822f8c0f825f8eef732f147ed94ed94),
.INIT_00(256'hedf0f04aefe8ee8eedc5eeb6f10def4ded94ed99ed94ee6fed9407631bb70b56)
)RAM16_Log_Varx8(  .DO(Log_Varx8),
                   .DOP(),
                   .ADDR(Addr_Out),
                   .CLK(CLK_SYS),
                   .DI(16'hFFFF),
                   .DIP(2'b11),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                 );

// Log_Varx9
RAMB16_S18 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_03(256'h0000000000000000000000000000000000000000000000000000000ed94ed94),
.INIT_02(256'hed94f3c1f227ed94ed94ed94f373f464f1b6eee6ef2bf4b4f646f2a8f4c4f841),
.INIT_01(256'hf64ef3fff256f448f0ddf22af2d4f32eeeb9ed94ed94ed94ed94ed94ed94ed94),
.INIT_00(256'hed94ed94ed94ed94ed94ee64ed94ed94ed94ed94f0fef0c0ed94ed941abd0edc)
)RAM16_Log_Varx9(  .DO(Log_Varx9),
                   .DOP(),
                   .ADDR(Addr_Out),
                   .CLK(CLK_SYS),
                   .DI(16'hFFFF),
                   .DIP(2'b11),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                 );


// ------------------------------------------------------------
// (4) Inv_Varx              U5/13
//  ------------------------------------------------------------

// Inv_Varx0

wire [31:0] pInv_Varx0; assign  Inv_Varx0 = pInv_Varx0[17:0];

RAMB16_S36 #(

 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
```

```
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000001400000014000),
.INIT_05(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_04(256'h0001102c0001400000014000000140000001400000014000000140000001400000),
.INIT_03(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_02(256'h0001400000014000000140000000fb3c00004cf800003c4f000027b10000c328),
.INIT_01(256'h000068e100003e0f00002adc00001d3b00000e8d000019b900001a0c0000ca42),
.INIT_00(256'h0000b6f300001b110000263a000032a8000061cd0001400000014000000014000)
)RAM16_Inv_Varx0(  .DO(pInv_Varx0),
                   .DOP(),
                   .ADDR(Addr_Out[8:0]),
                   .CLK(CLK_SYS),
                   .DI(32'hFFFF_FFFF),
                   .DIP(4'b1111),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                );

// Inv_Varx1

wire [31:0] pInv_Varx1; assign       Inv_Varx1 = pInv_Varx1[17:0];

RAMB16_S36 #(

 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000001400000014000),
.INIT_05(256'h0001400000014000000140000001400000102e60000d23c0001400000014000),
.INIT_04(256'h00010f1a0000d63700011a5f00014000000112fb000110c30000c0840000bde0),
.INIT_03(256'h0001163100010d630000cf0b0000fbac0001400000014000000140000014000),
.INIT_02(256'h0001400000014000000140000000af50000071bd000018b6000016cc00014000),
.INIT_01(256'h00007fd600001013000009c00000140500004af10000127500002 7ba00005862),
.INIT_00(256'h0001063100002d6300002edd000140000000ff1900014000000140000014000)
)RAM16_Inv_Varx1(  .DO(pInv_Varx1),
                   .DOP(),
                   .ADDR(Addr_Out[8:0]),
                   .CLK(CLK_SYS),
                   .DI(32'hFFFF_FFFF),
                   .DIP(4'b1111),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                );

// Inv_Varx2

wire [31:0] pInv_Varx2; assign       Inv_Varx2 = pInv_Varx2[17:0];

RAMB16_S36 #(
```

```
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000000139dd00014000),
.INIT_05(256'h0001400000014000000140000001137a0001084c0000cf630000b2f70000e2d8),
.INIT_04(256'h0000dce70000edd50001223e0001400000010ba800010eec0001400000014000),
.INIT_03(256'h0001400000011ef70001398300013ac700014000000140000001400000014000),
.INIT_02(256'h0001400000014000000140000001400000055630000456a000030110000b0e2),
.INIT_01(256'h000045990000272600026e400000b180000127700000ba3000025a700003ebf),
.INIT_00(256'h0000bfde00003bfe00007f6f000059ba000058e6000140000001400000014000)
)RAM16_Inv_Varx2(  .DO(pInv_Varx2),
                   .DOP(),
                   .ADDR(Addr_Out[8:0]),
                   .CLK(CLK_SYS),
                   .DI(32'hFFFF_FFFF),
                   .DIP(4'b1111),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
               );

// Inv_Varx3
wire [31:0] pInv_Varx3; assign       Inv_Varx3 = pInv_Varx3[17:0];

RAMB16_S36 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000001400000014000),
.INIT_05(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_04(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_03(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_02(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_01(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_00(256'h0001400000014000000140000001400000014000000004423000001c7600001749)
)RAM16_Inv_Varx3(  .DO(pInv_Varx3),
                   .DOP(),
                   .ADDR(Addr_Out[8:0]),
                   .CLK(CLK_SYS),
                   .DI(32'hFFFF_FFFF),
                   .DIP(4'b1111),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
               );

// Inv_Varx4

wire [31:0] pInv_Varx4; assign       Inv_Varx4 = pInv_Varx4[17:0];

RAMB16_S36 #(
```

```
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000001400000014000),
.INIT_05(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_04(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_03(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_02(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_01(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_00(256'h000140000001400000014000000f4e300004754000012 9e000006ac0000116f)
)RAM16_Inv_Varx4(  .DO(pInv_Varx4),
                   .DOP(),
                   .ADDR(Addr_Out[8:0]),
                   .CLK(CLK_SYS),
                   .DI(32'hFFFF_FFFF),
                   .DIP(4'b1111),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                );

// Inv_Varx5

wire [31:0] pInv_Varx5; assign       Inv_Varx5 = pInv_Varx5[17:0];

RAMB16_S36 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000001400000014000),
.INIT_05(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_04(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_03(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_02(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_01(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_00(256'h000140000001400000014000000140000001400000014000000140000014000)
)RAM16_Inv_Varx5(  .DO(pInv_Varx5),
                   .DOP(),
                   .ADDR(Addr_Out[8:0]),
                   .CLK(CLK_SYS),
                   .DI(32'hFFFF_FFFF),
                   .DIP(4'b1111),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                );

// Inv_Varx6

wire [31:0] pInv_Varx6; assign       Inv_Varx6 = pInv_Varx6[17:0];
```

```
RAMB16_S36 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000001400000014000),
.INIT_05(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_04(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_03(256'h000140000001400000014000000117c8000114bd000140000001004d00014000),
.INIT_02(256'h0000a96f0000c15d000140000000888000014000000054bf00005eb800007ca8),
.INIT_01(256'h00004ad300006fdd0000309500002a690000276d000020eb00001bd400001008),
.INIT_00(256'h00000deb00000e78000009b0000007500000053f0000041b000000a900000949)
)RAM16_Inv_Varx6(  .DO(pInv_Varx6),
                   .DOP(),
                   .ADDR(Addr_Out[8:0]),
                   .CLK(CLK_SYS),
                   .DI(32'hFFFF_FFFF),
                   .DIP(4'b1111),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                );

// Inv_Varx7

wire [31:0] pInv_Varx7; assign       Inv_Varx7 = pInv_Varx7[17:0];

RAMB16_S36 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000001400000014000),
.INIT_05(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_04(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_03(256'h0001400000014000000140000001400000014000000140000001400000014000),
.INIT_02(256'h0000b722000140000000ea45000121dc00012bd7000077a500011ce00000823c),
.INIT_01(256'h0001061c0000af8b000050410000726700000434a0000684100003d8c0000219a),
.INIT_00(256'h00001e00000020ec000012c500000e3d000013580000037b0000008e000007f3)
)RAM16_Inv_Varx7(  .DO(pInv_Varx7),
                   .DOP(),
                   .ADDR(Addr_Out[8:0]),
                   .CLK(CLK_SYS),
                   .DI(32'hFFFF_FFFF),
                   .DIP(4'b1111),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                );

// Inv_Varx8

wire [31:0] pInv_Varx8; assign       Inv_Varx8 = pInv_Varx8[17:0];
```

```
RAMB16_S36 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000cbc400012b4a),
.INIT_05(256'h000139fc000100bb0000ae070000a1db0000dc630000d5b90000ab5900006f3a),
.INIT_04(256'h0000609c00007ded000096b900007b2100004fa7000046b4000056630000582e),
.INIT_03(256'h0000459d0000346200002ec500002f38000037c70000481800004114000003f4d),
.INIT_02(256'h0000558e00004f370000556f00004d72000060350000c98f0001400000014000),
.INIT_01(256'h000132070000e41d0000ef3300011b4000013878000115c00000cf5200010202),
.INIT_00(256'h0001400000013f490001400000011f9e0001400000000cb50000100000007c2)
)RAM16_Inv_Varx8(  .DO(pInv_Varx8),
                   .DOP(),
                   .ADDR(Addr_Out[8:0]),
                   .CLK(CLK_SYS),
                   .DI(32'hFFFF_FFFF),
                   .DIP(4'b1111),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                );

// Inv_Varx9

wire [31:0] pInv_Varx9; assign
 Inv_Varx9 = pInv_Varx9[17:0];

RAMB16_S36 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000001400000014000),
.INIT_05(256'h00014000000093e30000b4af0001400000014000000140000000999a00008894),
.INIT_04(256'h0000bee100010f61000106640000835d00006bf10000a9a60000825300005442),
.INIT_03(256'h00006b8100008f740000b09600008a760000d4480000b4640000a60f00009edf),
.INIT_02(256'h000115630001400000014000000140000001400000014000000140000014000),
.INIT_01(256'h0001400000014000000140000001400000014000000121160001400000014000),
.INIT_00(256'h000140000001400000000d0d30000d74b0001400000014000000000122000004ff)
)RAM16_Inv_Varx9(  .DO(pInv_Varx9),
                   .DOP(),
                   .ADDR(Addr_Out[8:0]),
                   .CLK(CLK_SYS),
                   .DI(32'hFFFF_FFFF),
                   .DIP(4'b1111),
                   .EN(EN),
                   .SSR(1'b0),
                   .WE(1'b0)
                );
```

```
// ------------------------------------------------------
// (5) Miux_Beta_K2D            U7/11
//  ------------------------------------------------------

// Miux_Beta_K2D0

wire [31:0] pMiux_Beta_K2D0;
 assign       Miux_Beta_K2D0 = pMiux_Beta_K2D0[17:0];

RAMB16_S36 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h00000000000000000000000000000000000000000000000000000000000000ac),
.INIT_05(256'h00000155000001ff000002a900000355000004020000004af0000055f0000060f),
.INIT_04(256'h000006bd0000076a00000817000008c40000097200000a2100000ad100000b80),
.INIT_03(256'h00000c2d00000cd800000d8200000e2c00000ed400000f790000101e000010c5),
.INIT_02(256'h0000116c00001214000012bc000136700001412000014bd0000156c0000161a),
.INIT_01(256'h000016c800001774000021821000018cf0000197d00001a2b00001ad900001b88),
.INIT_00(256'h00001c3500001ce000001d8700001e2f00001edc00001f8900002036000020e3)
)RAM16_Miux_Beta_K2D0(.DO(pMiux_Beta_K2D0),
                      .DOP(),
                      .ADDR(Addr_Out[8:0]),
                      .CLK(CLK_SYS),
                      .DI(32'hFFFF_FFFF),
                      .DIP(4'b1111),
                      .EN(EN),
                      .SSR(1'b0),
                      .WE(1'b0)
                     );

// Miux_Beta_K2D1

wire [31:0] pMiux_Beta_K2D1;
 assign       Miux_Beta_K2D1 = pMiux_Beta_K2D1[17:0];

RAMB16_S36 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h00000000000000000000000000000000000000000000000000000000000000b2),
.INIT_05(256'h000001620000020d000002b5000003610000041100000 4c70000057b0000062b),
.INIT_04(256'h000006dd000007950000085100000 90d000009c700000a8100000b3700000bee),
.INIT_03(256'h00000ca500000d5600000e0800000ec100000f800000103e000010f8000011af),
.INIT_02(256'h0000125d00001304000013ad000014570000150200001 5ae0000165900001704),
.INIT_01(256'h000017b10000185c00001903000019a800001a4d00001afc00001bab00001c5b),
.INIT_00(256'h00001d0600001db000001e5500001efc00001fa70000205400002100000021ac)
)RAM16_Miux_Beta_K2D1(.DO(pMiux_Beta_K2D1),
                      .DOP(),
                      .ADDR(Addr_Out[8:0]),
                      .CLK(CLK_SYS),
```

```
                    .DI(32'hFFFF_FFFF),
                    .DIP(4'b1111),
                    .EN(EN),
                    .SSR(1'b0),
                    .WE(1'b0)
                   );

// Miux_Beta_K2D2

wire [31:0] pMiux_Beta_K2D2; assign
 Miux_Beta_K2D2 = pMiux_Beta_K2D2[17:0];

RAMB16_S36 #(
 .INIT(36'h000000000),
 .SRVAL(36'h000000000),
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000000000000b0),
.INIT_05(256'h000001620000021500000 2cb000003810000043400000 4e3000059200000642),
.INIT_04(256'h000006f5000007ab000008610000091300000 9c500000a7800000b2d00000be2),
.INIT_03(256'h00000c9400000d4700000df800000ea700000f5200000ffb000010a40000114f),
.INIT_02(256'h000011f80000129b0000133b000013e40000148d00001534000015dd00001687),
.INIT_01(256'h00001736000017e5000018910000194100001 9f700001aaa00001b5900001c0d),
.INIT_00(256'h00001cc400001d7400001e2500001ed700001f8b0000203f000020f1000021a3)
)RAM16_Miux_Beta_K2D2(.DO(pMiux_Beta_K2D2),
                    .DOP(),
                    .ADDR(Addr_Out[8:0]),
                    .CLK(CLK_SYS),
                    .DI(32'hFFFF_FFFF),
                    .DIP(4'b1111),
                    .EN(EN),
                    .SSR(1'b0),
                    .WE(1'b0)
                   );

// Miux_Beta_K2D3

wire [31:0] pMiux_Beta_K2D3;
 assign       Miux_Beta_K2D3 = pMiux_Beta_K2D3[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
 .INIT(36'h000000000),       // Initial values on A output port
 .SRVAL(36'h000000000),      // Set/Reset value for A port output
 .WRITE_MODE("WRITE_FIRST"),
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h000000000000000000000000000000000000000000000000000000000028a),
.INIT_05(256'h000005130000079d00000a2700000cb000000f3a000011c30000144d000016d6),
.INIT_04(256'h0000196000001bea00001e73000020fd000023860000260e0000289800002b21),
.INIT_03(256'h00002dab00003034000032be00003548000037d100003a5b00003ce400003f6e),
.INIT_02(256'h000041f700004480000047 0a0000499400004c1d00004ea700005131000053ba),
.INIT_01(256'h00005643000058cd00005b5600005de00000606a000062f40000657e00006808),
.INIT_00(256'h00006a9200006d1c00006fa600007231000074bd00007747000079d200007c57)
```

```
)

RAM16_Miux_Beta_K2D3(.DO(pMiux_Beta_K2D3),
                     .DOP(),
                     .ADDR(Addr_Out[8:0]),
                     .CLK(CLK_SYS),
                     .DI(32'hFFFF_FFFF),
                     .DIP(4'b1111),
                     .EN(EN),                // 1-bit A port enable input
                     .SSR(1'b0),             // 1-bit A port set/reset input
                     .WE(1'b0)               // 1-bit A port write enable input
                    );

// Miux_Beta_K2D4

wire [31:0] pMiux_Beta_K2D4; assign
 Miux_Beta_K2D4 = pMiux_Beta_K2D4[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
// The following generic INIT_xx declarations specify the initial contents of
// the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h000000000000000000000000000000000000000000000000000000000000028b),
.INIT_05(256'h00000516000007a100000a2c00000cb700000f41000011cd00001458000016e3),
.INIT_04(256'h0000196d00001bf800001e830000210e000023990000262400002 8ae00002b3a),
.INIT_03(256'h00002dc500003050000032da00003565000037f000003a7c00003d0700003f91),
.INIT_02(256'h0000421c000044a700004732000049bd00004c4800004ed20000515e000053e9),
.INIT_01(256'h000056750000590000005b8a00005e15000060a00000632b000065b600006840),
.INIT_00(256'h00006aca00006d5500006fe000007269000074ef00007774000079f700007c6b)
)

RAM16_Miux_Beta_K2D4(.DO(pMiux_Beta_K2D4),
                     .DOP(),                 // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]),   // 10-bit A port address input
                     .CLK(CLK_SYS),          // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),     // 16-bit A port data input
                     .DIP(4'b1111),          // 2-bit A port parity data input
                     .EN(EN),                // 1-bit A port enable input
                     .SSR(1'b0),             // 1-bit A port set/reset input
                     .WE(1'b0)               // 1-bit A port write enable input
                    );

// Miux_Beta_K2D5

wire [31:0] pMiux_Beta_K2D5; assign
 Miux_Beta_K2D5 = pMiux_Beta_K2D5[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
```

```
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
// The following generic INIT_xx declarations specify the initial contents of
// the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000000000002b0),
.INIT_05(256'h0000056000000081100000ac200000d7300001024000012d30000158400001835),
.INIT_04(256'h00001ae600001d9500002044000022f4000025a50000285600002b0600002db5),
.INIT_03(256'h000030660000331700035c80000387800003b2600003dd80000408800004339),
.INIT_02(256'h000045e70000489700004b4800004df9000050aa000053580000560600058b7),
.INIT_01(256'h00005b6800005e19000060c900006378000066290000068d900006b8a00006e39),
.INIT_00(256'h000070e90000739a0000764a000078fa00007ba800007e5e00008110000083c7)
)

RAM16_Miux_Beta_K2D5(.DO(pMiux_Beta_K2D5),
                     .DOP(),                // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]),  // 10-bit A port address input
                     .CLK(CLK_SYS),         // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),    // 16-bit A port data input
                     .DIP(4'b1111),         // 2-bit A port parity data input
                     .EN(EN),               // 1-bit A port enable input
                     .SSR(1'b0),            // 1-bit A port set/reset input
                     .WE(1'b0)              // 1-bit A port write enable input
                    );

// Miux_Beta_K2D6

wire [31:0] pMiux_Beta_K2D6; assign
 Miux_Beta_K2D6 = pMiux_Beta_K2D6[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h00000000000000000000000000000000000000000000000000000000000002b8),
.INIT_05(256'h0000056e0000082400000ada00000d900000104a000012fe000015b50000186a),
.INIT_04(256'h00001b2000001dd80000208b00002342000025f7000028ae00002b6700002e1c),
.INIT_03(256'h000030d200003387000036 3c000038f300003ba600003e5b0000410e000043c1),
.INIT_02(256'h000046790000492c00004be200004e950000514a000053ff000056b100005963),
.INIT_01(256'h00005c1400005ec5000061770000642800006 6db0000698a00006c3800006ee7),
.INIT_00(256'h000071960000744200007 6ed0000799a00007c4600007ef30000819e00008436)
)

RAM16_Miux_Beta_K2D6(.DO(pMiux_Beta_K2D6),
                     .DOP(),               // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]), // 10-bit A port address input
                     .CLK(CLK_SYS),        // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),   // 16-bit A port data input
```

```
                     .DIP(4'b1111),        // 2-bit A port parity data input
                     .EN(EN),              // 1-bit A port enable input
                     .SSR(1'b0),           // 1-bit A port set/reset input
                     .WE(1'b0)             // 1-bit A port write enable input
                    );

// Miux_Beta_K2D7

wire [31:0] pMiux_Beta_K2D7; assign
 Miux_Beta_K2D7 = pMiux_Beta_K2D7[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h00000000000000000000000000000000000000000000000000000000000002b6),
.INIT_05(256'h0000056b0000082000000ad600000d890000103f000012f2000015a70000185d),
.INIT_04(256'h00001b0f00001dc50000207a0000232f000025e40000289700002b4c00002e00),
.INIT_03(256'h000030b4000033690000361c000038d200003b8600003e3a000040f0000043a4),
.INIT_02(256'h0000465a0000490f00004bc400004e790000512d000053e2000056960000594b),
.INIT_01(256'h00005c0000005eb3000061670000641c000066d40000698a00006c4100006ef3),
.INIT_00(256'h000071a80000745b0000770e000079c000007c7700007f2c000081e200008486)
)

RAM16_Miux_Beta_K2D7(.DO(pMiux_Beta_K2D7),
                     .DOP(),               // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]), // 10-bit A port address input
                     .CLK(CLK_SYS),        // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),   // 16-bit A port data input
                     .DIP(4'b1111),        // 2-bit A port parity data input
                     .EN(EN),              // 1-bit A port enable input
                     .SSR(1'b0),           // 1-bit A port set/reset input
                     .WE(1'b0)             // 1-bit A port write enable input
                    );

// Miux_Beta_K2D8

wire [31:0] pMiux_Beta_K2D8; assign
  Miux_Beta_K2D8 = pMiux_Beta_K2D8[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000000000000926),
```

```
.INIT_05(256'h0000124e00001b77000024a000002dc6000036ee000040130000493a0000525d),
.INIT_04(256'h00005b80000064a300006dcc000076f3000080130000893800000925c00009b80),
.INIT_03(256'h0000a4a20000adc30000b6e40000c0070000c9280000d2480000db6b0000e490),
.INIT_02(256'h0000edb30000f6d30000fffb000109210001124800011b6f0001249800012dc6),
.INIT_01(256'h000136f20001401b000149480001527500015ba1000164cb00016df600017721),
.INIT_00(256'h0001804d000189770001929f00019bce0001a4f60001ae1f0001b7490001c033)
)

RAM16_Miux_Beta_K2D8(.DO(pMiux_Beta_K2D8),
                     .DOP(),                // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]),  // 10-bit A port address input
                     .CLK(CLK_SYS),         // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),    // 16-bit A port data input
                     .DIP(4'b1111),         // 2-bit A port parity data input
                     .EN(EN),               // 1-bit A port enable input
                     .SSR(1'b0),            // 1-bit A port set/reset input
                     .WE(1'b0)              // 1-bit A port write enable input
                    );

// Miux_Beta_K2D9

wire [31:0] pMiux_Beta_K2D9; assign       Miux_Beta_K2D9 =
pMiux_Beta_K2D9[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h000000000000000000000000000000000000000000000000000000000000092d),
.INIT_05(256'h0000126300001b97000024c600002df60000372800004061000049990000052ca),
.INIT_04(256'h00005bf90000652b00006e6200007799000080cb000089fd0000933300009c69),
.INIT_03(256'h0000a5990000aecb0000b7fe0000c1360000ca690000d39d0000dcd00000e605),
.INIT_02(256'h0000ef390000f870000101a900010adf0001141200011d420001267600012fac),
.INIT_01(256'h000138dd0001420e00014b3e0001547500015da6000166d50001700400017932),
.INIT_00(256'h0001826700018b9e000194d000019dff0001a7290001b05a0001b98e0001c227)
)

RAM16_Miux_Beta_K2D9(.DO(pMiux_Beta_K2D9),
                     .DOP(),                 // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]),   // 10-bit A port address input
                     .CLK(CLK_SYS),          // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),     // 16-bit A port data input
                     .DIP(4'b1111),          // 2-bit A port parity data input
                     .EN(EN),                // 1-bit A port enable input
                     .SSR(1'b0),             // 1-bit A port set/reset input
                     .WE(1'b0)               // 1-bit A port write enable input
                    );
```

```
// ------------------------------------------------------------
// (6) Miux_Inv_Varx            U12/6
//  ------------------------------------------------------------

// Miux_Inv_Varx0

wire [31:0] pMiux_Inv_Varx0; assign
 Miux_Inv_Varx0 = pMiux_Inv_Varx0[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000000a5700000a1d),
.INIT_05(256'h000009cf00000999000009ae000009ee00000a2a00000a5800000a8800000a73),
.INIT_04(256'h000008c100000a1300000a1500000a5300000a6e00000a7b00000a63000009da),
.INIT_03(256'h000008e70000079a0000065b0000056f000004e6000004a8000004a600000513),
.INIT_02(256'h00000646000008fd00000c4f00000cb800000675000007f200000886000035ed),
.INIT_01(256'h00001e810000121b00000bf5000007580000032f000004b5000003b5000017ac),
.INIT_00(256'h00001612000003f10000069e000008fa00000edf000025490000188a00000b1a)
)

RAM16_Miux_Inv_Varx0(.DO(pMiux_Inv_Varx0),
                     .DOP(),                 // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]),   // 10-bit A port address input
                     .CLK(CLK_SYS),          // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),     // 16-bit A port data input
                     .DIP(4'b1111),          // 2-bit A port parity data input
                     .EN(EN),                // 1-bit A port enable input
                     .SSR(1'b0),             // 1-bit A port set/reset input
                     .WE(1'b0)               // 1-bit A port write enable input
                    );

// Miux_Inv_Varx1

wire [31:0] pMiux_Inv_Varx1; assign
 Miux_Inv_Varx1 = pMiux_Inv_Varx1[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
 .INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000000a8700000a3f),
```

```
.INIT_05(256'h00000a4900000a8300000a7f00000a670000083c000006a500000a2400000a22),
.INIT_04(256'h000008b100000711000009a000000b0c000009a5000009b40000067500000593),
.INIT_03(256'h000006fe000005d8000003f00000045300000519000004d3000004d700000587),
.INIT_02(256'h0000073c00000a6f00000d8e00000a8300000b07000003b400000554000059ad),
.INIT_01(256'h000024cc00004920000028e000004b800000f850000033700000563000000a37),
.INIT_00(256'h00001ffa000006c00000081e0000385c0000260c00002467000017fc00000abf))

RAM16_Miux_Inv_Varx1(.DO(pMiux_Inv_Varx1),
                     .DOP(),                 // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]),   // 10-bit A port address input
                     .CLK(CLK_SYS),          // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),     // 16-bit A port data input
                     .DIP(4'b1111),          // 2-bit A port parity data input
                     .EN(EN),                // 1-bit A port enable input
                     .SSR(1'b0),             // 1-bit A port set/reset input
                     .WE(1'b0)               // 1-bit A port write enable input
                    );

// Miux_Inv_Varx2

wire [31:0] pMiux_Inv_Varx2; assign
 Miux_Inv_Varx2 = pMiux_Inv_Varx2[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000a3f00000ab3),
.INIT_05(256'h00000ae800000afc00000aab000008be00000842000006 8c000005b00000074b),
.INIT_04(256'h000007420000080b000009e600000acd000009000000091b00000aa400000a44),
.INIT_03(256'h000009790000076e000006c60000059800000 4e3000004940000048e000004dd),
.INIT_02(256'h000005c70000082100000bb200000f12000006570000081400000916000002ecf),
.INIT_01(256'h000013ed00000b5100000ad6000002d600000430000002 2e0000058d000007c8),
.INIT_00(256'h000017ca00009450000177c0000108400000db3000025340000189200000b3c)
)

RAM16_Miux_Inv_Varx2(.DO(pMiux_Inv_Varx2),
                     .DOP(),                 // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]),   // 10-bit A port address input
                     .CLK(CLK_SYS),          // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),     // 16-bit A port data input
                     .DIP(4'b1111),          // 2-bit A port parity data input
                     .EN(EN),                // 1-bit A port enable input
                     .SSR(1'b0),             // 1-bit A port set/reset input
                     .WE(1'b0)               // 1-bit A port write enable input
                    );

// Miux_Inv_Varx3
```

```
wire [31:0] pMiux_Inv_Varx3; assign
 Miux_Inv_Varx3 = pMiux_Inv_Varx3[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000279c00002795),
.INIT_05(256'h000027a0000027b3000027a60000279e0000279700027a1000027b5000027a8),
.INIT_04(256'h000027a00000279900027a3000027b6000027ab000027a30000279d000027a8),
.INIT_03(256'h000027bb000027b1000027ab000027a7000027b4000027c9000027c3000027c3),
.INIT_02(256'h000027c4000027d8000027f6000027fd0000280c0000281d0000284900002887),
.INIT_01(256'h000028b7000028f000002937000029a400002a4e00002b0e00002bde00002cde),
.INIT_00(256'h00002e47000030ca00003497000039f50000418d000010b1000006ca0000013f)
)

RAM16_Miux_Inv_Varx3(.DO(pMiux_Inv_Varx3),
                     .DOP(),                  // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]),    // 10-bit A port address input
                     .CLK(CLK_SYS),           // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),      // 16-bit A port data input
                     .DIP(4'b1111),           // 2-bit A port parity data input
                     .EN(EN),                 // 1-bit A port enable input
                     .SSR(1'b0),              // 1-bit A port set/reset input
                     .WE(1'b0)                // 1-bit A port write enable input
                    );

// Miux_Inv_Varx4

wire [31:0] pMiux_Inv_Varx4; assign
 Miux_Inv_Varx4 = pMiux_Inv_Varx4[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000279600002797),
.INIT_05(256'h0000279c000027a1000027a3000027980000279900027e000027a2000027a4),
.INIT_04(256'h0000279a0000279b000027a0000027a5000027a70000279d0000279f000027a4),
.INIT_03(256'h000027aa000027ae000027a5000027a9000027af000027b8000027c0000027bc),
.INIT_02(256'h000027c5000027d3000027e5000027f800002803000028 1e0000284300002873),
.INIT_01(256'h000028ac000028e400002935000029 9e00002a2c00002aea00002bbe00002cd3),
.INIT_00(256'h00002e470000307c000033f000002b8200000e62000004860000017f000000e8)
)
```

```
RAM16_Miux_Inv_Varx4(.DO(pMiux_Inv_Varx4),
                     .DOP(),                // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]),  // 10-bit A port address input
                     .CLK(CLK_SYS),         // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),    // 16-bit A port data input
                     .DIP(4'b1111),         // 2-bit A port parity data input
                     .EN(EN),               // 1-bit A port enable input
                     .SSR(1'b0),            // 1-bit A port set/reset input
                     .WE(1'b0)              // 1-bit A port write enable input
                    );

// Miux_Inv_Varx5

wire [31:0] pMiux_Inv_Varx5; assign
 Miux_Inv_Varx5 = pMiux_Inv_Varx5[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000034aa000034d8),
.INIT_05(256'h00003617000036a70000378f000037340000372b0000388a0000398d00003a7c),
.INIT_04(256'h000039fa00003a4700003c0e00003d4900003eb200003e2900003ec5000040bc),
.INIT_03(256'h000042500000447600004425000044c900004784000049e400004ced00004cb1),
.INIT_02(256'h00004e1a000051b8000054e30000592e000059c600005c4100061950000671b),
.INIT_01(256'h00006ca100006f8c0000739c00007c110000857800008e000000954000009d29),
.INIT_00(256'h0000ad780000bb2a0000c9fa0000d9b40000f2030000fea10000b0bf00001518)
)

RAM16_Miux_Inv_Varx5(.DO(pMiux_Inv_Varx5),
                     .DOP(),                 // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]),   // 10-bit A port address input
                     .CLK(CLK_SYS),          // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),     // 16-bit A port data input
                     .DIP(4'b1111),          // 2-bit A port parity data input
                     .EN(EN),                // 1-bit A port enable input
                     .SSR(1'b0),             // 1-bit A port set/reset input
                     .WE(1'b0)               // 1-bit A port write enable input
                    );

// Miux_Inv_Varx6

wire [31:0] pMiux_Inv_Varx6; assign
 Miux_Inv_Varx6 = pMiux_Inv_Varx6[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
```

```
.WRITE_MODE("WRITE_FIRST"),
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000034b8000034c8),
.INIT_05(256'h0000354b0000361b000036cc000036fa00003720000037b7000038c2000039ab),
.INIT_04(256'h000039f800003a2f00003af600003c4300003d8900003e0700003e6e00003f67),
.INIT_03(256'h00004116000042fb000043c900003bca00003c590000482000003be000004c10),
.INIT_02(256'h000028d700002fdb0000526f000024bb00005866000017f300001bc40000269d),
.INIT_01(256'h00001868000025e40000110100000f9800000f9b00000de200000c6600000783),
.INIT_00(256'h0000070a000007f4000005be000004aa000003a2000003020000000520000008a)
)

RAM16_Miux_Inv_Varx6(.DO(pMiux_Inv_Varx6),
                     .DOP(),                 // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]),   // 10-bit A port address input
                     .CLK(CLK_SYS),          // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),     // 16-bit A port data input
                     .DIP(4'b1111),          // 2-bit A port parity data input
                     .EN(EN),                // 1-bit A port enable input
                     .SSR(1'b0),             // 1-bit A port set/reset input
                     .WE(1'b0)               // 1-bit A port write enable input
                    );

// Miux_Inv_Varx7

wire [31:0] pMiux_Inv_Varx7; assign       Miux_Inv_Varx7 =
pMiux_Inv_Varx7[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"), // "WRITE_FIRST", "READ_FIRST", or
"NO_CHANGE"
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000353700003571),
.INIT_05(256'h000035c7000036bb000037440000379400037e20000385a0000399a00003a54),
.INIT_04(256'h00003ac700003b3300003bd300003d6500003e7300003f1600003fb9000040a5),
.INIT_03(256'h000042ae00044530000454e00046550000479e00004a5e00004cbf00004e5d),
.INIT_02(256'h00002dc0000051f000003ed5000050f70000565c00002381000057be00002a8b),
.INIT_01(256'h00005a2500003ee200001df300002d4400001cab00002f7f00001dbd00001137),
.INIT_00(256'h000010b5000013fd00000c6000000a2c00000f05000002bf0000004900000081)
)

RAM16_Miux_Inv_Varx7(.DO(pMiux_Inv_Varx7),
                     .DOP(),                 // 2-bit A port parity data output
                     .ADDR(Addr_Out[8:0]),   // 10-bit A port address input
                     .CLK(CLK_SYS),          // 1-bit A port clock input
                     .DI(32'hFFFF_FFFF),     // 16-bit A port data input
                     .DIP(4'b1111),          // 2-bit A port parity data input
```

```
                    .EN(EN),                // 1-bit A port enable input
                    .SSR(1'b0),             // 1-bit A port set/reset input
                    .WE(1'b0)               // 1-bit A port write enable input
                   );

// Miux_Inv_Varx8

wire [31:0] pMiux_Inv_Varx8; assign       Miux_Inv_Varx8 =
pMiux_Inv_Varx8[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"), // "WRITE_FIRST", "READ_FIRST", or
"NO_CHANGE"
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_06(256'h0000000000000000000000000000000000000000000000000000075910000addc),
.INIT_05(256'h0000b84700009848000067ef00061330000856d000082db00006a2d00004584),
.INIT_04(256'h00003ccc00005000000060ff000050550000348500002f0b00003a2b00003c52),
.INIT_03(256'h00003050000024c90000213a00002217000028f4000035a4000030ef00003011),
.INIT_02(256'h000041bd00003d840000431800003d6c00004cf20000a247000102ff000104c2),
.INIT_01(256'h0000facb0000bb9b0000c5740000eaef000104f20000e94a0000aebc0000da39),
.INIT_00(256'h00010fdf000110f5000112dc0000f7840001139e00000ae40000009200000086)
)

RAM16_Miux_Inv_Varx8(.DO(pMiux_Inv_Varx8),
                     .DOP(),
                     .ADDR(Addr_Out[8:0]),
                     .CLK(CLK_SYS),
                     .DI(32'hFFFF_FFFF),
                     .DIP(4'b1111),
                     .EN(EN),
                     .SSR(1'b0),
                     .WE(1'b0)
                    );

// Miux_Inv_Varx9

wire [31:0] pMiux_Inv_Varx9; assign       Miux_Inv_Varx9 =
pMiux_Inv_Varx9[17:0];

RAMB16_S36 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"), // "WRITE_FIRST", "READ_FIRST", or
"NO_CHANGE"
// The following generic INIT_xx declarations specify the initial contents
// of the RAM
.INIT_07(256'h0000000000000000000000000000000000000000000000000000000000000000),
```

```
.INIT_06(256'h0000000000000000000000000000000000000000000000000000bf250000c0e8),
.INIT_05(256'h0000c32c00005b2a000070120000c7f80000ca360000ccf00000638900005922),
.INIT_04(256'h00007dc80000b57f0000b24800005a5d00004b090000777100005d5600003d66),
.INIT_03(256'h00004f6b00006af80000852f000069b60000a3ce00008cda000082da00007e65),
.INIT_02(256'h0000de66000102680001046100010056d0001065c00010762000108f900010ae7),
.INIT_01(256'h00010bed00010ce800010e0b00010f99000111930000f812000113870001147b),
.INIT_00(256'h000115e3000117ae0000b6e50000bce1000118fc000119c6000000a800000050)
)

RAM16_Miux_Inv_Varx9(.DO(pMiux_Inv_Varx9),
                     .DOP(),
                     .ADDR(Addr_Out[8:0]),
                     .CLK(CLK_SYS),
                     .DI(32'hFFFF_FFFF),
                     .DIP(4'b1111),
                     .EN(EN),
                     .SSR(1'b0),
                     .WE(1'b0)
                    );



// ---------------------------------------------------------
// (7) Log_Pcm   S6/10
//  ---------------------------------------------------------

 assign Log_Pcm0 = 16'hf887;
 assign Log_Pcm1 = 16'hf3c5;
 assign Log_Pcm2 = 16'hf532;
 assign Log_Pcm3 = 16'hf8b0;
 assign Log_Pcm4 = 16'hf92c;
 assign Log_Pcm5 = 16'hebfc;
 assign Log_Pcm6 = 16'hf4c6;
 assign Log_Pcm7 = 16'hf7f8;
 assign Log_Pcm8 = 16'hf8b2;
 assign Log_Pcm9 = 16'hf188;

// ---------------------------------------------------------
// (8) Km               U16/0
//  ---------------------------------------------------------

 assign Km0 = 16'h003c;
 assign Km1 = 16'h003c;
 assign Km2 = 16'h003c;
 assign Km3 = 16'h00c8;
 assign Km4 = 16'h00c8;
 assign Km5 = 16'h0186;
 assign Km6 = 16'h0186;
 assign Km7 = 16'h0186;
 assign Km8 = 16'h0186;
 assign Km9 = 16'h0186;

endmodule
```

```
‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
//  CWM local filter bank top level file
//
//  lower level module: SCWM_Filter (0-9)
//
//////////////////////////////////////////////////////////////////////////////////


module Filter_Bank(      input           CLK_SYS,
                         input           CLK_3_2_1,
                         input           aCLK_3_2_2,
                         input           bCLK_3_2_2,
                         input           cCLK_3_2_2,
                         input           EN,
                         input    [1:0]  Tran_Sel,
                         input    [17:0] Xt,
                         input    [15:0] Beta0,
                         input    [15:0] Beta1,
                         input    [15:0] Beta2,
                         input    [15:0] Beta3,
                         input    [15:0] Beta4,
                         input    [15:0] Beta5,
                         input    [15:0] Beta6,
                         input    [15:0] Beta7,
                         input    [15:0] Beta8,
                         input    [15:0] Beta9,
                         output   [17:0] Xt_Beta_A0,
                         output   [17:0] Xt_Beta_A1,
                         output   [17:0] Xt_Beta_A2,
                         output   [17:0] Xt_Beta_A3,
                         output   [17:0] Xt_Beta_A4,
                         output   [17:0] Xt_Beta_A5,
                         output   [17:0] Xt_Beta_A6,
                         output   [17:0] Xt_Beta_A7,
                         output   [17:0] Xt_Beta_A8,
                         output   [17:0] Xt_Beta_A9,
                         output   [17:0] Xt_Beta_B0,
                         output   [17:0] Xt_Beta_B1,
                         output   [17:0] Xt_Beta_B2,
                         output   [17:0] Xt_Beta_B3,
                         output   [17:0] Xt_Beta_B4,
                         output   [17:0] Xt_Beta_B5,
                         output   [17:0] Xt_Beta_B6,
                         output   [17:0] Xt_Beta_B7,
                         output   [17:0] Xt_Beta_B8,
                         output   [17:0] Xt_Beta_B9,
                         output   [17:0] Xt_Beta_C0,
                         output   [17:0] Xt_Beta_C1,
                         output   [17:0] Xt_Beta_C2,
```

```
                        output   [17:0] Xt_Beta_C3,
                        output   [17:0] Xt_Beta_C4,
                        output   [17:0] Xt_Beta_C5,
                        output   [17:0] Xt_Beta_C6,
                        output   [17:0] Xt_Beta_C7,
                        output   [17:0] Xt_Beta_C8,
                        output   [17:0] Xt_Beta_C9
                    );



SCWM_Filter Filter0(      .CLK_SYS(CLK_SYS),
                          .CLK_3_2_1(CLK_3_2_1),
                          .aCLK_3_2_2(aCLK_3_2_2),
                          .bCLK_3_2_2(bCLK_3_2_2),
                          .cCLK_3_2_2(cCLK_3_2_2),
                          .EN(EN),
                          .Tran_Sel(Tran_Sel),
                          .Xt(Xt),
                          .Beta(Beta0),
                          .Xt_Beta_A(Xt_Beta_A0),
                          .Xt_Beta_B(Xt_Beta_B0),
                          .Xt_Beta_C(Xt_Beta_C0)   );

SCWM_Filter Filter1(      .CLK_SYS(CLK_SYS),
                          .CLK_3_2_1(CLK_3_2_1),
                          .aCLK_3_2_2(aCLK_3_2_2),
                          .bCLK_3_2_2(bCLK_3_2_2),
                          .cCLK_3_2_2(cCLK_3_2_2),
                          .EN(EN),
                          .Tran_Sel(Tran_Sel),
                          .Xt(Xt),
                          .Beta(Beta1),
                          .Xt_Beta_A(Xt_Beta_A1),
                          .Xt_Beta_B(Xt_Beta_B1),
                          .Xt_Beta_C(Xt_Beta_C1)   );

SCWM_Filter Filter2(      .CLK_SYS(CLK_SYS),
                          .CLK_3_2_1(CLK_3_2_1),
                          .aCLK_3_2_2(aCLK_3_2_2),
                          .bCLK_3_2_2(bCLK_3_2_2),
                          .cCLK_3_2_2(cCLK_3_2_2),
                          .EN(EN),
                          .Tran_Sel(Tran_Sel),
                          .Xt(Xt),
                          .Beta(Beta2),
                          .Xt_Beta_A(Xt_Beta_A2),
                          .Xt_Beta_B(Xt_Beta_B2),
                          .Xt_Beta_C(Xt_Beta_C2)   );

SCWM_Filter Filter3(      .CLK_SYS(CLK_SYS),
                          .CLK_3_2_1(CLK_3_2_1),
```

```
                        .aCLK_3_2_2(aCLK_3_2_2),
                        .bCLK_3_2_2(bCLK_3_2_2),
                        .cCLK_3_2_2(cCLK_3_2_2),
                        .EN(EN),
                        .Tran_Sel(Tran_Sel),
                        .Xt(Xt),
                        .Beta(Beta3),
                        .Xt_Beta_A(Xt_Beta_A3),
                        .Xt_Beta_B(Xt_Beta_B3),
                        .Xt_Beta_C(Xt_Beta_C3)   );

SCWM_Filter Filter4(    .CLK_SYS(CLK_SYS),
                        .CLK_3_2_1(CLK_3_2_1),
                        .aCLK_3_2_2(aCLK_3_2_2),
                        .bCLK_3_2_2(bCLK_3_2_2),
                        .cCLK_3_2_2(cCLK_3_2_2),
                        .EN(EN),
                        .Tran_Sel(Tran_Sel),
                        .Xt(Xt),
                        .Beta(Beta4),
                        .Xt_Beta_A(Xt_Beta_A4),
                        .Xt_Beta_B(Xt_Beta_B4),
                        .Xt_Beta_C(Xt_Beta_C4)   );

SCWM_Filter Filter5(    .CLK_SYS(CLK_SYS),
                        .CLK_3_2_1(CLK_3_2_1),
                        .aCLK_3_2_2(aCLK_3_2_2),
                        .bCLK_3_2_2(bCLK_3_2_2),
                        .cCLK_3_2_2(cCLK_3_2_2),
                        .EN(EN),
                        .Tran_Sel(Tran_Sel),
                        .Xt(Xt),
                        .Beta(Beta5),
                        .Xt_Beta_A(Xt_Beta_A5),
                        .Xt_Beta_B(Xt_Beta_B5),
                        .Xt_Beta_C(Xt_Beta_C5)   );

SCWM_Filter Filter6(    .CLK_SYS(CLK_SYS),
                        .CLK_3_2_1(CLK_3_2_1),
                        .aCLK_3_2_2(aCLK_3_2_2),
                        .bCLK_3_2_2(bCLK_3_2_2),
                        .cCLK_3_2_2(cCLK_3_2_2),
                        .EN(EN),
                        .Tran_Sel(Tran_Sel),
                        .Xt(Xt),
                        .Beta(Beta6),
                        .Xt_Beta_A(Xt_Beta_A6),
                        .Xt_Beta_B(Xt_Beta_B6),
                        .Xt_Beta_C(Xt_Beta_C6)   );

SCWM_Filter Filter7(    .CLK_SYS(CLK_SYS),
                        .CLK_3_2_1(CLK_3_2_1),
                        .aCLK_3_2_2(aCLK_3_2_2),
```

```
                        .bCLK_3_2_2(bCLK_3_2_2),
                        .cCLK_3_2_2(cCLK_3_2_2),
                        .EN(EN),
                        .Tran_Sel(Tran_Sel),
                        .Xt(Xt),
                        .Beta(Beta7),
                        .Xt_Beta_A(Xt_Beta_A7),
                        .Xt_Beta_B(Xt_Beta_B7),
                        .Xt_Beta_C(Xt_Beta_C7)   );

SCWM_Filter Filter8(    .CLK_SYS(CLK_SYS),
                        .CLK_3_2_1(CLK_3_2_1),
                        .aCLK_3_2_2(aCLK_3_2_2),
                        .bCLK_3_2_2(bCLK_3_2_2),
                        .cCLK_3_2_2(cCLK_3_2_2),
                        .EN(EN),
                        .Tran_Sel(Tran_Sel),
                        .Xt(Xt),
                        .Beta(Beta8),
                        .Xt_Beta_A(Xt_Beta_A8),
                        .Xt_Beta_B(Xt_Beta_B8),
                        .Xt_Beta_C(Xt_Beta_C8)   );

SCWM_Filter Filter9(    .CLK_SYS(CLK_SYS),
                        .CLK_3_2_1(CLK_3_2_1),
                        .aCLK_3_2_2(aCLK_3_2_2),
                        .bCLK_3_2_2(bCLK_3_2_2),
                        .cCLK_3_2_2(cCLK_3_2_2),
                        .EN(EN),
                        .Tran_Sel(Tran_Sel),
                        .Xt(Xt),
                        .Beta(Beta9),
                        .Xt_Beta_A(Xt_Beta_A9),
                        .Xt_Beta_B(Xt_Beta_B9),
                        .Xt_Beta_C(Xt_Beta_C9)   );

endmodule


`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
//   SCWM local filter, recursively implement : sum(xt(k)*betam(k)),
//   the contribution of the part of tail prediction miux(k)*betam(k) can be
//   pre-calculated and stored in MEM for directly use
//
////////////////////////////////////////////////////////////////////////////////


module SCWM_Filter(     input           CLK_SYS,
                        input           CLK_3_2_1,
                        input           aCLK_3_2_2,
                        input           bCLK_3_2_2,
```

```
                        input             cCLK_3_2_2,
                        input             EN,
                        input      [1:0]  Tran_Sel,
                        input      [17:0] Xt,
                        input      [15:0] Beta,
                        output     [17:0] Xt_Beta_A,
                        output     [17:0] Xt_Beta_B,
                        output     [17:0] Xt_Beta_C
                        );

//*******************************************************
//                 internal signals
//*******************************************************

 wire EN_Bar;
 assign EN_Bar = ~EN;
 reg  [17:0] A = 18'h00000;
 reg  [17:0] B = 18'h00000;
 reg  [47:0] P1 = 48'h0000_0000_0000;
 reg  [47:0] P2 = 48'h0000_0000_0000;
 reg  [47:0] P3 = 48'h0000_0000_0000;
 wire [47:0] C, P;

//*********************************************************
// Pipe in Xt & Beta and construct [17:0]   A & B
// with EN_Bar ASYNCHRONOUS reset
//*********************************************************

 always @ (posedge CLK_3_2_1 or posedge EN_Bar)
  begin
     if (EN_Bar) begin A <= 18'h00000; B <= 18'h00000; end
      else        begin A <= {2'b00, Beta}; B <= Xt; end
  end

//*********************************************************
// Pipe out 3 parallel outputs Xt_Beta_A,B,C
// with EN_Bar ASYNCHRONOUS reset
//*********************************************************

 always @ (posedge aCLK_3_2_2 or posedge EN_Bar)
   if (EN_Bar) P1 <= 48'h0000_0000_0000;
   else       P1 <= P;
 always @ (posedge bCLK_3_2_2 or posedge EN_Bar)
   if (EN_Bar) P2 <= 48'h0000_0000_0000;
   else        P2 <= P;
 always @ (posedge cCLK_3_2_2 or posedge EN_Bar)
   if (EN_Bar) P3 <= 48'h0000_0000_0000;
   else        P3 <= P;

//*********************************************************
//   output trimming from U24/24 to U10/8
//*********************************************************
```

```
assign  Xt_Beta_A = P1[33:16],
        Xt_Beta_B = P2[33:16],
        Xt_Beta_C = P3[33:16];

//*******************************************************
// Multiplex between Xt_Beta_A, Xt_Beta_B, and Xt_Beta_C
//*******************************************************

Mux_4_1   Xt_Beta_Mux [47:0] (.D0(48'h0000_0000_0000),
                              .D1(P1),
                              .D2(P2),
                              .D3(P3),
                              .S0(Tran_Sel[0]),
                              .S1(Tran_Sel[1]),
                              .O(C)
                             );




//*****************************************************************************
//
// DSP48 ( Virtex-4 ) instantiation for function of "P = A x B + C"
//
//*****************************************************************************

DSP48 DSP48_inst ( .BCOUT(),                          // 18-bit B
                                                      // cascade output
                   .P(P),                             // 48-bit product output
                   .PCOUT(),                          // 38-bit cascade output
                   .A(A),                             // 18-bit A data input
                   .B(B),                             // 18-bit B data input
                   .BCIN(18'h00000),                  // 18-bit B cascade input
                   .C(C),                             // 48-bit cascade input
                   .CARRYIN(1'b0),                    // Carry input signal
                   .CARRYINSEL(2'b00),                // 2-bit carry input
                                                      // select
                   .CEA(1'b1),                        // A data clock enable
                                                      // input
                   .CEB(1'b1),                        // B data clock enable
                   .CEC(1'b1),                        // C data clock enable
                   .CECARRYIN(1'b0),                  // CARRYIN clock enable
                   .CECINSUB(1'b0),                   // CINSUB clock enable
                   .CECTRL(1'b0),                     // Clock Enable input
                                                      // for CTRL registers
                   .CEM(1'b1),                        // Clock Enable input
                                                      // for multiplier
                                                      // registers
                   .CEP(1'b1),                        // Clock Enable input for
                                                      // P registers
                   .CLK(CLK_SYS),                     // Clock input
                   .OPMODE(7'b011_01_01),             // 7-bit operation mode
                                                      // input
```

```
                    .PCIN(48'h0000_0000_0000),          // 48-bit PCIN input
                    .RSTA(EN_Bar),                      // Reset input for A
                                                        // pipeline registers
                    .RSTB(EN_Bar),                      // Reset input for B
                                                        // pipeline registers
                    .RSTC(EN_Bar),                      // Reset input for C
                                                        // pipeline registers
                    .RSTCARRYIN(1'b0),                  // Reset input for
                                                        // CARRYIN registers
                    .RSTCTRL(1'b0),                     // Reset input for CTRL
                                                        // registers
                    .RSTM(EN_Bar),                      // Reset input for
                                                        // multiplier registers
                    .RSTP(EN_Bar),                      // Reset input for P
                                                        // pipeline registers
                    .SUBTRACT(1'b0)                     // SUBTRACT input
                   );

// The following defparams specify the behavior of the DSP48 slice. If the
// instance name to the DSP48 is changed,
// that change needs to be reflected in the defparam statements.

 defparam DSP48_inst.AREG = 1;
 defparam DSP48_inst.BREG = 1;
 defparam DSP48_inst.B_INPUT = "DIRECT";
 defparam DSP48_inst.CARRYINREG = 0;
 defparam DSP48_inst.CARRYINSELREG = 0;
 defparam DSP48_inst.CREG = 1;
 defparam DSP48_inst.LEGACY_MODE = "MULT18X18S";
 defparam DSP48_inst.MREG = 1;
 defparam DSP48_inst.OPMODEREG = 0;
 defparam DSP48_inst.PREG = 1;
 defparam DSP48_inst.SUBTRACTREG = 0;

// End of DSP48_inst instantiation

endmodule



//////////////////////////////////////////////////////////////
//  scaling factors and likelihood calculation top level
//  file. 3 modules under this level:
//  1. Am1: calculate the numerator and denominator of scaling
//          factor
//  2. Am2: shift denominator to LUT range and multiply with
//          numeritor
//  3. Likelihood: calculating the likelihood between clusters
//          and input transients
//  10 parallel path, corresponding 10 clusters
//
//////////////////////////////////////////////////////////////
```

```
module Am_and_Likelihood(   input               CLK_SYS,
                            input               CLK_3_2_1,
                            input               aCLK_3_2_2,
                            input               bCLK_3_2_2,
                            input               cCLK_3_2_2,
                            input               aCLK_3_2_3,
                            input               bCLK_3_2_3,
                            input               cCLK_3_2_3,
                            input               CLK_3_2_4,
                            input               CLK_3_2_5,
                            input               CLK_3_2_6,
                            input               aCLK_3_2_6,
                            input               bCLK_3_2_6,
                            input               cCLK_3_2_6,
                            input               CLK_3_2_7,
                            input               CLK_3_2_8,
                            input               CLK_3_2_9,
                            input               aCLK_3_2_10,
                            input               bCLK_3_2_10,
                            input               cCLK_3_2_10,
                            input               Am_Multi_Sel,
                            input      [1:0]    Tran_Sel,
                            input               EN,
                            input               Aden_LS,
                            input               Anum_LS,
                            input      [17:0]   Xt2,
                            input      [17:0]   Xt,
                            input      [17:0]   Inv_Varx0,
                            input      [17:0]   Inv_Varx1,
                            input      [17:0]   Inv_Varx2,
                            input      [17:0]   Inv_Varx3,
                            input      [17:0]   Inv_Varx4,
                            input      [17:0]   Inv_Varx5,
                            input      [17:0]   Inv_Varx6,
                            input      [17:0]   Inv_Varx7,
                            input      [17:0]   Inv_Varx8,
                            input      [17:0]   Inv_Varx9,
                            input      [17:0]   Miux_Inv_Varx0,
                            input      [17:0]   Miux_Inv_Varx1,
                            input      [17:0]   Miux_Inv_Varx2,
                            input      [17:0]   Miux_Inv_Varx3,
                            input      [17:0]   Miux_Inv_Varx4,
                            input      [17:0]   Miux_Inv_Varx5,
                            input      [17:0]   Miux_Inv_Varx6,
                            input      [17:0]   Miux_Inv_Varx7,
                            input      [17:0]   Miux_Inv_Varx8,
                            input      [17:0]   Miux_Inv_Varx9,
                            input      [15:0]   Miux0,
                            input      [15:0]   Miux1,
                            input      [15:0]   Miux2,
                            input      [15:0]   Miux3,
                            input      [15:0]   Miux4,
                            input      [15:0]   Miux5,
```

```
input      [15:0]   Miux6,
input      [15:0]   Miux7,
input      [15:0]   Miux8,
input      [15:0]   Miux9,
input signed [15:0] Log_Varx0,
input signed [15:0] Log_Varx1,
input signed [15:0] Log_Varx2,
input signed [15:0] Log_Varx3,
input signed [15:0] Log_Varx4,
input signed [15:0] Log_Varx5,
input signed [15:0] Log_Varx6,
input signed [15:0] Log_Varx7,
input signed [15:0] Log_Varx8,
input signed [15:0] Log_Varx9,
output       [9:0]  Am1a,
output       [9:0]  Am2a,
output       [9:0]  Am3a,
output       [9:0]  Am4a,
output       [9:0]  Am5a,
output       [9:0]  Am6a,
output       [9:0]  Am7a,
output       [9:0]  Am8a,
output       [9:0]  Am9a,
output       [9:0]  Am10a,
output       [9:0]  Am1b,
output       [9:0]  Am2b,
output       [9:0]  Am3b,
output       [9:0]  Am4b,
output       [9:0]  Am5b,
output       [9:0]  Am6b,
output       [9:0]  Am7b,
output       [9:0]  Am8b,
output       [9:0]  Am9b,
output       [9:0]  Am10b,
output       [9:0]  Am1c,
output       [9:0]  Am2c,
output       [9:0]  Am3c,
output       [9:0]  Am4c,
output       [9:0]  Am5c,
output       [9:0]  Am6c,
output       [9:0]  Am7c,
output       [9:0]  Am8c,
output       [9:0]  Am9c,
output       [9:0]  Am10c,
output       [47:0] Likelihood1a,
output       [47:0] Likelihood2a,
output       [47:0] Likelihood3a,
output       [47:0] Likelihood4a,
output       [47:0] Likelihood5a,
output       [47:0] Likelihood6a,
output       [47:0] Likelihood7a,
output       [47:0] Likelihood8a,
output       [47:0] Likelihood9a,
```

```
                            output       [47:0] Likelihood10a,
                            output       [47:0] Likelihood1b,
                            output       [47:0] Likelihood2b,
                            output       [47:0] Likelihood3b,
                            output       [47:0] Likelihood4b,
                            output       [47:0] Likelihood5b,
                            output       [47:0] Likelihood6b,
                            output       [47:0] Likelihood7b,
                            output       [47:0] Likelihood8b,
                            output       [47:0] Likelihood9b,
                            output       [47:0] Likelihood10b,
                            output       [47:0] Likelihood1c,
                            output       [47:0] Likelihood2c,
                            output       [47:0] Likelihood3c,
                            output       [47:0] Likelihood4c,
                            output       [47:0] Likelihood5c,
                            output       [47:0] Likelihood6c,
                            output       [47:0] Likelihood7c,
                            output       [47:0] Likelihood8c,
                            output       [47:0] Likelihood9c,
                            output       [47:0] Likelihood10c
                          );


//************************************************
//              internal signals
//************************************************

 wire  [47:0]
 Aden0, Aden1, Aden2,  Aden3,  Aden4,  Aden5,
 Aden6,  Aden7,  Aden8,  Aden9;

 wire  [47:0]
 Anum0, Anum1, Anum2, Anum3,  Anum4,  Anum5,  Anum6,
 Anum7,  Anum8,  Anum9;

 wire  [9:0]
 Am0,   Am1,   Am2,    Am3, Am4,    Am5,    Am6,
 Am7,    Am8, Am9;



//************************************************
//  Am1: caculation numeritor and denominator
//       of scaling factor, Am1_0 -- Am1_9
//************************************************

Am1  Am1_0(   .CLK_SYS(CLK_SYS),
              .CLK_3_2_1(CLK_3_2_1),
              .aCLK_3_2_2(aCLK_3_2_2),
              .bCLK_3_2_2(bCLK_3_2_2),
              .cCLK_3_2_2(cCLK_3_2_2),
              .aCLK_3_2_3(aCLK_3_2_3),
```

```
                .bCLK_3_2_3(bCLK_3_2_3),
                .cCLK_3_2_3(cCLK_3_2_3),
                .EN(EN),
                .Am_Multi_Sel(Am_Multi_Sel),
                .Tran_Sel(Tran_Sel),
                .Xt2(Xt2),
                .Xt(Xt),
                .Inv_Varx(Inv_Varx0),
                .Miux_Inv_Varx(Miux_Inv_Varx0),
                .Aden(Aden0),
                .Anum(Anum0)  );

Am1  Am1_1(   .CLK_SYS(CLK_SYS),
                .CLK_3_2_1(CLK_3_2_1),
                .aCLK_3_2_2(aCLK_3_2_2),
                .bCLK_3_2_2(bCLK_3_2_2),
                .cCLK_3_2_2(cCLK_3_2_2),
                .aCLK_3_2_3(aCLK_3_2_3),
                .bCLK_3_2_3(bCLK_3_2_3),
                .cCLK_3_2_3(cCLK_3_2_3),
                .EN(EN),
                .Am_Multi_Sel(Am_Multi_Sel),
                .Tran_Sel(Tran_Sel),
                .Xt2(Xt2),
                .Xt(Xt),
                .Inv_Varx(Inv_Varx1),
                .Miux_Inv_Varx(Miux_Inv_Varx1),
                .Aden(Aden1),
                .Anum(Anum1)  );

Am1  Am1_2(   .CLK_SYS(CLK_SYS),
                .CLK_3_2_1(CLK_3_2_1),
                .aCLK_3_2_2(aCLK_3_2_2),
                .bCLK_3_2_2(bCLK_3_2_2),
                .cCLK_3_2_2(cCLK_3_2_2),
                .aCLK_3_2_3(aCLK_3_2_3),
                .bCLK_3_2_3(bCLK_3_2_3),
                .cCLK_3_2_3(cCLK_3_2_3),
                .EN(EN),
                .Am_Multi_Sel(Am_Multi_Sel),
                .Tran_Sel(Tran_Sel),
                .Xt2(Xt2),
                .Xt(Xt),
                .Inv_Varx(Inv_Varx2),
                .Miux_Inv_Varx(Miux_Inv_Varx2),
                .Aden(Aden2),
                .Anum(Anum2)  );


Am1  Am1_3(   .CLK_SYS(CLK_SYS),
                .CLK_3_2_1(CLK_3_2_1),
                .aCLK_3_2_2(aCLK_3_2_2),
                .bCLK_3_2_2(bCLK_3_2_2),
```

```
                .cCLK_3_2_2(cCLK_3_2_2),
                .aCLK_3_2_3(aCLK_3_2_3),
                .bCLK_3_2_3(bCLK_3_2_3),
                .cCLK_3_2_3(cCLK_3_2_3),
                .EN(EN),
                .Am_Multi_Sel(Am_Multi_Sel),
                .Tran_Sel(Tran_Sel),
                .Xt2(Xt2),
                .Xt(Xt),
                .Inv_Varx(Inv_Varx3),
                .Miux_Inv_Varx(Miux_Inv_Varx3),
                .Aden(Aden3),
                .Anum(Anum3)  );

Am1  Am1_4(   .CLK_SYS(CLK_SYS),
                .CLK_3_2_1(CLK_3_2_1),
                .aCLK_3_2_2(aCLK_3_2_2),
                .bCLK_3_2_2(bCLK_3_2_2),
                .cCLK_3_2_2(cCLK_3_2_2),
                .aCLK_3_2_3(aCLK_3_2_3),
                .bCLK_3_2_3(bCLK_3_2_3),
                .cCLK_3_2_3(cCLK_3_2_3),
                .EN(EN),
                .Am_Multi_Sel(Am_Multi_Sel),
                .Tran_Sel(Tran_Sel),
                .Xt2(Xt2),
                .Xt(Xt),
                .Inv_Varx(Inv_Varx4),
                .Miux_Inv_Varx(Miux_Inv_Varx4),
                .Aden(Aden4),
                .Anum(Anum4)  );

Am1  Am1_5(   .CLK_SYS(CLK_SYS),
                .CLK_3_2_1(CLK_3_2_1),
                .aCLK_3_2_2(aCLK_3_2_2),
                .bCLK_3_2_2(bCLK_3_2_2),
                .cCLK_3_2_2(cCLK_3_2_2),
                .aCLK_3_2_3(aCLK_3_2_3),
                .bCLK_3_2_3(bCLK_3_2_3),
                .cCLK_3_2_3(cCLK_3_2_3),
                .EN(EN),
                .Am_Multi_Sel(Am_Multi_Sel),
                .Tran_Sel(Tran_Sel),
                .Xt2(Xt2),
                .Xt(Xt),
                .Inv_Varx(Inv_Varx5),
                .Miux_Inv_Varx(Miux_Inv_Varx5),
                .Aden(Aden5),
                .Anum(Anum5)  );

Am1  Am1_6(   .CLK_SYS(CLK_SYS),
                .CLK_3_2_1(CLK_3_2_1),
                .aCLK_3_2_2(aCLK_3_2_2),
```

```
                .bCLK_3_2_2(bCLK_3_2_2),
                .cCLK_3_2_2(cCLK_3_2_2),
                .aCLK_3_2_3(aCLK_3_2_3),
                .bCLK_3_2_3(bCLK_3_2_3),
                .cCLK_3_2_3(cCLK_3_2_3),
                .EN(EN),
                .Am_Multi_Sel(Am_Multi_Sel),
                .Tran_Sel(Tran_Sel),
                .Xt2(Xt2),
                .Xt(Xt),
                .Inv_Varx(Inv_Varx6),
                .Miux_Inv_Varx(Miux_Inv_Varx6),
                .Aden(Aden6),
                .Anum(Anum6)  );

Am1  Am1_7(     .CLK_SYS(CLK_SYS),
                .CLK_3_2_1(CLK_3_2_1),
                .aCLK_3_2_2(aCLK_3_2_2),
                .bCLK_3_2_2(bCLK_3_2_2),
                .cCLK_3_2_2(cCLK_3_2_2),
                .aCLK_3_2_3(aCLK_3_2_3),
                .bCLK_3_2_3(bCLK_3_2_3),
                .cCLK_3_2_3(cCLK_3_2_3),
                .EN(EN),
                .Am_Multi_Sel(Am_Multi_Sel),
                .Tran_Sel(Tran_Sel),
                .Xt2(Xt2),
                .Xt(Xt),
                .Inv_Varx(Inv_Varx7),
                .Miux_Inv_Varx(Miux_Inv_Varx7),
                .Aden(Aden7),
                .Anum(Anum7)  );

Am1  Am1_8(     .CLK_SYS(CLK_SYS),
                .CLK_3_2_1(CLK_3_2_1),
                .aCLK_3_2_2(aCLK_3_2_2),
                .bCLK_3_2_2(bCLK_3_2_2),
                .cCLK_3_2_2(cCLK_3_2_2),
                .aCLK_3_2_3(aCLK_3_2_3),
                .bCLK_3_2_3(bCLK_3_2_3),
                .cCLK_3_2_3(cCLK_3_2_3),
                .EN(EN),
                .Am_Multi_Sel(Am_Multi_Sel),
                .Tran_Sel(Tran_Sel),
                .Xt2(Xt2),
                .Xt(Xt),
                .Inv_Varx(Inv_Varx8),
                .Miux_Inv_Varx(Miux_Inv_Varx8),
                .Aden(Aden8),
                .Anum(Anum8)  );

Am1  Am1_9(     .CLK_SYS(CLK_SYS),
                .CLK_3_2_1(CLK_3_2_1),
```

```
              .aCLK_3_2_2(aCLK_3_2_2),
              .bCLK_3_2_2(bCLK_3_2_2),
              .cCLK_3_2_2(cCLK_3_2_2),
              .aCLK_3_2_3(aCLK_3_2_3),
              .bCLK_3_2_3(bCLK_3_2_3),
              .cCLK_3_2_3(cCLK_3_2_3),
              .EN(EN),
              .Am_Multi_Sel(Am_Multi_Sel),
              .Tran_Sel(Tran_Sel),
              .Xt2(Xt2),
              .Xt(Xt),
              .Inv_Varx(Inv_Varx9),
              .Miux_Inv_Varx(Miux_Inv_Varx9),
              .Aden(Aden9),
              .Anum(Anum9)  );



//*************************************************
//  Am2  Shift Denominator to the range of LUT and
//       Multiply numeritor and denominator,
//       Am2_0 -- Am2_9
//*************************************************

Am2  Am2_0(   .CLK_SYS(CLK_SYS),
              .CLK_3_2_4(CLK_3_2_4),
              .CLK_3_2_5(CLK_3_2_5),
              .EN(EN),
              .Aden_LS(Aden_LS),
              .Anum_LS(Anum_LS),
              .Aden(Aden0),
              .Anum(Anum0),
             .Am(Am0)
              );

Am2  Am2_1(   .CLK_SYS(CLK_SYS),
              .CLK_3_2_4(CLK_3_2_4),
              .CLK_3_2_5(CLK_3_2_5),
              .EN(EN),
              .Aden_LS(Aden_LS),
              .Anum_LS(Anum_LS),
              .Aden(Aden1),
              .Anum(Anum1),
             .Am(Am1)
              );

Am2  Am2_2(   .CLK_SYS(CLK_SYS),
              .CLK_3_2_4(CLK_3_2_4),
              .CLK_3_2_5(CLK_3_2_5),
              .EN(EN),
              .Aden_LS(Aden_LS),
              .Anum_LS(Anum_LS),
              .Aden(Aden2),
```

```
                .Anum(Anum2),
                .Am(Am2)
                );

Am2  Am2_3(     .CLK_SYS(CLK_SYS),
                .CLK_3_2_4(CLK_3_2_4),
                .CLK_3_2_5(CLK_3_2_5),
                .EN(EN),
                .Aden_LS(Aden_LS),
                .Anum_LS(Anum_LS),
                .Aden(Aden3),
                .Anum(Anum3),
               .Am(Am3)
                );

Am2  Am2_4(     .CLK_SYS(CLK_SYS),
                .CLK_3_2_4(CLK_3_2_4),
                .CLK_3_2_5(CLK_3_2_5),
                .EN(EN),
                .Aden_LS(Aden_LS),
                .Anum_LS(Anum_LS),
                .Aden(Aden4),
                .Anum(Anum4),
                .Am(Am4)
                );

Am2  Am2_5(     .CLK_SYS(CLK_SYS),
                .CLK_3_2_4(CLK_3_2_4),
                .CLK_3_2_5(CLK_3_2_5),
                .EN(EN),
                .Aden_LS(Aden_LS),
                .Anum_LS(Anum_LS),
                .Aden(Aden5),
                .Anum(Anum5),
                .Am(Am5)
                );

Am2  Am2_6(     .CLK_SYS(CLK_SYS),
                .CLK_3_2_4(CLK_3_2_4),
                .CLK_3_2_5(CLK_3_2_5),
                .EN(EN),
                .Aden_LS(Aden_LS),
                .Anum_LS(Anum_LS),
                .Aden(Aden6),
                .Anum(Anum6),
                .Am(Am6)
                );

Am2  Am2_7(     .CLK_SYS(CLK_SYS),
                .CLK_3_2_4(CLK_3_2_4),
                .CLK_3_2_5(CLK_3_2_5),
                .EN(EN),
                .Aden_LS(Aden_LS),
```

```
                .Anum_LS(Anum_LS),
                .Aden(Aden7),
                .Anum(Anum7),
                .Am(Am7)
                );

Am2  Am2_8(     .CLK_SYS(CLK_SYS),
                .CLK_3_2_4(CLK_3_2_4),
                .CLK_3_2_5(CLK_3_2_5),
                .EN(EN),
                .Aden_LS(Aden_LS),
                .Anum_LS(Anum_LS),
                .Aden(Aden8),
                .Anum(Anum8),
                .Am(Am8)
                );

Am2  Am2_9(     .CLK_SYS(CLK_SYS),
                .CLK_3_2_4(CLK_3_2_4),
                .CLK_3_2_5(CLK_3_2_5),
                .EN(EN),
                .Aden_LS(Aden_LS),
                .Anum_LS(Anum_LS),
                .Aden(Aden9),
                .Anum(Anum9),
                .Am(Am9)
                );

//*****************************************************
//        Likelihood calculation:  0 -- 9
//*****************************************************

Likelihood  Likelihood_0 (      .EN(EN),
                                .CLK_SYS(CLK_SYS),
                                .CLK_3_2_6(CLK_3_2_6),
                                .aCLK_3_2_6(aCLK_3_2_6),
                                .bCLK_3_2_6(bCLK_3_2_6),
                                .cCLK_3_2_6(cCLK_3_2_6),
                                .CLK_3_2_7(CLK_3_2_7),
                                .CLK_3_2_8(CLK_3_2_8),
                                .CLK_3_2_9(CLK_3_2_9),
                                .aCLK_3_2_10(aCLK_3_2_10),
                                .bCLK_3_2_10(bCLK_3_2_10),
                                .cCLK_3_2_10(cCLK_3_2_10),
                                .Am_Multi_Sel(Am_Multi_Sel),
                                .Tran_Sel(Tran_Sel),
                                .Xt(Xt),
                                .Miux(Miux0),
                                .Inv_Varx(Inv_Varx0),
                                .Log_Varx(Log_Varx0),
                                .Am(Am0),
                                .Ama(Am1a),
                                .Amb(Am1b),
```

```
                                .Amc(Am1c),
                                .Likelihooda(Likelihood1a),
                                .Likelihoodb(Likelihood1b),
                                .Likelihoodc(Likelihood1c)
                               );

Likelihood  Likelihood_1 (      .EN(EN),
                                .CLK_SYS(CLK_SYS),
                                .CLK_3_2_6(CLK_3_2_6),
                                .aCLK_3_2_6(aCLK_3_2_6),
                                .bCLK_3_2_6(bCLK_3_2_6),
                                .cCLK_3_2_6(cCLK_3_2_6),
                                .CLK_3_2_7(CLK_3_2_7),
                                .CLK_3_2_8(CLK_3_2_8),
                                .CLK_3_2_9(CLK_3_2_9),
                                .aCLK_3_2_10(aCLK_3_2_10),
                                .bCLK_3_2_10(bCLK_3_2_10),
                                .cCLK_3_2_10(cCLK_3_2_10),
                                .Am_Multi_Sel(Am_Multi_Sel),
                                .Tran_Sel(Tran_Sel),
                                .Xt(Xt),
                                .Miux(Miux1),
                                .Inv_Varx(Inv_Varx1),
                                .Log_Varx(Log_Varx1),
                                .Am(Am1),
                                .Ama(Am2a),
                                .Amb(Am2b),
                                .Amc(Am2c),
                                .Likelihooda(Likelihood2a),
                                .Likelihoodb(Likelihood2b),
                                .Likelihoodc(Likelihood2c)
                               );

Likelihood  Likelihood_2 (      .EN(EN),
                                .CLK_SYS(CLK_SYS),
                                .CLK_3_2_6(CLK_3_2_6),
                                .aCLK_3_2_6(aCLK_3_2_6),
                                .bCLK_3_2_6(bCLK_3_2_6),
                                .cCLK_3_2_6(cCLK_3_2_6),
                                .CLK_3_2_7(CLK_3_2_7),
                                .CLK_3_2_8(CLK_3_2_8),
                                .CLK_3_2_9(CLK_3_2_9),
                                .aCLK_3_2_10(aCLK_3_2_10),
                                .bCLK_3_2_10(bCLK_3_2_10),
                                .cCLK_3_2_10(cCLK_3_2_10),
                                .Am_Multi_Sel(Am_Multi_Sel),
                                .Tran_Sel(Tran_Sel),
                                .Xt(Xt),
                                .Miux(Miux2),
                                .Inv_Varx(Inv_Varx2),
                                .Log_Varx(Log_Varx2),
                                .Am(Am2),
                                .Ama(Am3a),
```

```
                                .Amb(Am3b),
                                .Amc(Am3c),
                                .Likelihooda(Likelihood3a),
                                .Likelihoodb(Likelihood3b),
                                .Likelihoodc(Likelihood3c)
                               );

Likelihood  Likelihood_3 (      .EN(EN),
                                .CLK_SYS(CLK_SYS),
                                .CLK_3_2_6(CLK_3_2_6),
                                .aCLK_3_2_6(aCLK_3_2_6),
                                .bCLK_3_2_6(bCLK_3_2_6),
                                .cCLK_3_2_6(cCLK_3_2_6),
                                .CLK_3_2_7(CLK_3_2_7),
                                .CLK_3_2_8(CLK_3_2_8),
                                .CLK_3_2_9(CLK_3_2_9),
                                .aCLK_3_2_10(aCLK_3_2_10),
                                .bCLK_3_2_10(bCLK_3_2_10),
                                .cCLK_3_2_10(cCLK_3_2_10),
                                .Am_Multi_Sel(Am_Multi_Sel),
                                .Tran_Sel(Tran_Sel),
                                .Xt(Xt),
                                .Miux(Miux3),
                                .Inv_Varx(Inv_Varx3),
                                .Log_Varx(Log_Varx3),
                                .Am(Am3),
                                .Ama(Am4a),
                                .Amb(Am4b),
                                .Amc(Am4c),
                                .Likelihooda(Likelihood4a),
                                .Likelihoodb(Likelihood4b),
                                .Likelihoodc(Likelihood4c)
                               );

Likelihood  Likelihood_4 (      .EN(EN),
                                .CLK_SYS(CLK_SYS),
                                .CLK_3_2_6(CLK_3_2_6),
                                .aCLK_3_2_6(aCLK_3_2_6),
                                .bCLK_3_2_6(bCLK_3_2_6),
                                .cCLK_3_2_6(cCLK_3_2_6),
                                .CLK_3_2_7(CLK_3_2_7),
                                .CLK_3_2_8(CLK_3_2_8),
                                .CLK_3_2_9(CLK_3_2_9),
                                .aCLK_3_2_10(aCLK_3_2_10),
                                .bCLK_3_2_10(bCLK_3_2_10),
                                .cCLK_3_2_10(cCLK_3_2_10),
                                .Am_Multi_Sel(Am_Multi_Sel),
                                .Tran_Sel(Tran_Sel),
                                .Xt(Xt),
                                .Miux(Miux4),
                                .Inv_Varx(Inv_Varx4),
                                .Log_Varx(Log_Varx4),
                                .Am(Am4),
```

```
                                .Ama(Am5a),
                                .Amb(Am5b),
                                .Amc(Am5c),
                                .Likelihooda(Likelihood5a),
                                .Likelihoodb(Likelihood5b),
                                .Likelihoodc(Likelihood5c)
                               );

Likelihood  Likelihood_5 (      .EN(EN),
                                .CLK_SYS(CLK_SYS),
                                .CLK_3_2_6(CLK_3_2_6),
                                .aCLK_3_2_6(aCLK_3_2_6),
                                .bCLK_3_2_6(bCLK_3_2_6),
                                .cCLK_3_2_6(cCLK_3_2_6),
                                .CLK_3_2_7(CLK_3_2_7),
                                .CLK_3_2_8(CLK_3_2_8),
                                .CLK_3_2_9(CLK_3_2_9),
                                .aCLK_3_2_10(aCLK_3_2_10),
                                .bCLK_3_2_10(bCLK_3_2_10),
                                .cCLK_3_2_10(cCLK_3_2_10),
                                .Am_Multi_Sel(Am_Multi_Sel),
                                .Tran_Sel(Tran_Sel),
                                .Xt(Xt),
                                .Miux(Miux5),
                                .Inv_Varx(Inv_Varx5),
                                .Log_Varx(Log_Varx5),
                                .Am(Am5),
                                .Ama(Am6a),
                                .Amb(Am6b),
                                .Amc(Am6c),
                                .Likelihooda(Likelihood6a),
                                .Likelihoodb(Likelihood6b),
                                .Likelihoodc(Likelihood6c)
                               );

Likelihood  Likelihood_6 (      .EN(EN),
                                .CLK_SYS(CLK_SYS),
                                .CLK_3_2_6(CLK_3_2_6),
                                .aCLK_3_2_6(aCLK_3_2_6),
                                .bCLK_3_2_6(bCLK_3_2_6),
                                .cCLK_3_2_6(cCLK_3_2_6),
                                .CLK_3_2_7(CLK_3_2_7),
                                .CLK_3_2_8(CLK_3_2_8),
                                .CLK_3_2_9(CLK_3_2_9),
                                .aCLK_3_2_10(aCLK_3_2_10),
                                .bCLK_3_2_10(bCLK_3_2_10),
                                .cCLK_3_2_10(cCLK_3_2_10),
                                .Am_Multi_Sel(Am_Multi_Sel),
                                .Tran_Sel(Tran_Sel),
                                .Xt(Xt),
                                .Miux(Miux6),
                                .Inv_Varx(Inv_Varx6),
                                .Log_Varx(Log_Varx6),
```

```
                                .Am(Am6),
                                .Ama(Am7a),
                                .Amb(Am7b),
                                .Amc(Am7c),
                                .Likelihooda(Likelihood7a),
                                .Likelihoodb(Likelihood7b),
                                .Likelihoodc(Likelihood7c)
                               );

Likelihood  Likelihood_7 (      .EN(EN),
                                .CLK_SYS(CLK_SYS),
                                .CLK_3_2_6(CLK_3_2_6),
                                .aCLK_3_2_6(aCLK_3_2_6),
                                .bCLK_3_2_6(bCLK_3_2_6),
                                .cCLK_3_2_6(cCLK_3_2_6),
                                .CLK_3_2_7(CLK_3_2_7),
                                .CLK_3_2_8(CLK_3_2_8),
                                .CLK_3_2_9(CLK_3_2_9),
                                .aCLK_3_2_10(aCLK_3_2_10),
                                .bCLK_3_2_10(bCLK_3_2_10),
                                .cCLK_3_2_10(cCLK_3_2_10),
                                .Am_Multi_Sel(Am_Multi_Sel),
                                .Tran_Sel(Tran_Sel),
                                .Xt(Xt),
                                .Miux(Miux7),
                                .Inv_Varx(Inv_Varx7),
                                .Log_Varx(Log_Varx7),
                                .Am(Am7),
                                .Ama(Am8a),
                                .Amb(Am8b),
                                .Amc(Am8c),
                                .Likelihooda(Likelihood8a),
                                .Likelihoodb(Likelihood8b),
                                .Likelihoodc(Likelihood8c)
                               );

Likelihood  Likelihood_8 (      .EN(EN),
                                .CLK_SYS(CLK_SYS),
                                .CLK_3_2_6(CLK_3_2_6),
                                .aCLK_3_2_6(aCLK_3_2_6),
                                .bCLK_3_2_6(bCLK_3_2_6),
                                .cCLK_3_2_6(cCLK_3_2_6),
                                .CLK_3_2_7(CLK_3_2_7),
                                .CLK_3_2_8(CLK_3_2_8),
                                .CLK_3_2_9(CLK_3_2_9),
                                .aCLK_3_2_10(aCLK_3_2_10),
                                .bCLK_3_2_10(bCLK_3_2_10),
                                .cCLK_3_2_10(cCLK_3_2_10),
                                .Am_Multi_Sel(Am_Multi_Sel),
                                .Tran_Sel(Tran_Sel),
                                .Xt(Xt),
                                .Miux(Miux8),
                                .Inv_Varx(Inv_Varx8),
```

```
                                .Log_Varx(Log_Varx8),
                                .Am(Am8),
                                .Ama(Am9a),
                                .Amb(Am9b),
                                .Amc(Am9c),
                                .Likelihooda(Likelihood9a),
                                .Likelihoodb(Likelihood9b),
                                .Likelihoodc(Likelihood9c)
                               );

Likelihood  Likelihood_9 (      .EN(EN),
                                .CLK_SYS(CLK_SYS),
                                .CLK_3_2_6(CLK_3_2_6),
                                .aCLK_3_2_6(aCLK_3_2_6),
                                .bCLK_3_2_6(bCLK_3_2_6),
                                .cCLK_3_2_6(cCLK_3_2_6),
                                .CLK_3_2_7(CLK_3_2_7),
                                .CLK_3_2_8(CLK_3_2_8),
                                .CLK_3_2_9(CLK_3_2_9),
                                .aCLK_3_2_10(aCLK_3_2_10),
                                .bCLK_3_2_10(bCLK_3_2_10),
                                .cCLK_3_2_10(cCLK_3_2_10),
                                .Am_Multi_Sel(Am_Multi_Sel),
                                .Tran_Sel(Tran_Sel),
                                .Xt(Xt),
                                .Miux(Miux9),
                                .Inv_Varx(Inv_Varx9),
                                .Log_Varx(Log_Varx9),
                                .Am(Am9),
                                .Ama(Am10a),
                                .Amb(Am10b),
                                .Amc(Am10c),
                                .Likelihooda(Likelihood10a),
                                .Likelihoodb(Likelihood10b),
                                .Likelihoodc(Likelihood10c)
                               );

endmodule


////////////////////////////////////////////////////////////
//
// Am1: Recursively calculate   the numeritor and denominator
//        of scaling factor between input and cluster center
//
////////////////////////////////////////////////////////////


module Am1(   input                 CLK_SYS,
              input                 CLK_3_2_1,
              input                 aCLK_3_2_2,
              input                 bCLK_3_2_2,
              input                 cCLK_3_2_2,
```

```
              input                   aCLK_3_2_3,
              input                   bCLK_3_2_3,
              input                   cCLK_3_2_3,
              input                   EN,
              input                   Am_Multi_Sel,
              input      [1:0]        Tran_Sel,
              input      [17:0]       Xt2,
              input      [17:0]       Xt,
              input      [17:0]       Inv_Varx,
              input      [17:0]       Miux_Inv_Varx,
              output     [47:0]       Aden,
              output     [47:0]       Anum  );

//*************************************************
// internal signals declaration
//*************************************************

 wire            EN_Bar;
 assign          EN_Bar = ~EN;

 reg    [17:0]   Xt2_Pipe      = 18'h00000;
 reg    [17:0]   Xt_Pipe = 18'h00000;
 reg    [17:0]   Inv_Varx_Pipe = 18'h00000;
 reg    [17:0]   Miux_Inv_Varx_Pipe = 18'h00000;
 reg    [47:0]   Aden1 = 48'h0000_0000_0000;
 reg    [47:0]   Aden2 = 48'h0000_0000_0000;
 reg    [47:0]   Aden3 = 48'h0000_0000_0000;
 reg    [47:0]   Anum1 = 48'h0000_0000_0000;
 reg    [47:0]   Anum2 = 48'h0000_0000_0000;
 reg    [47:0]   Anum3 = 48'h0000_0000_0000;
 wire   [47:0]   Aden_In;
 wire   [47:0]   Anum_In;
 wire   [47:0]   Anum_tmp;
 wire   [17:0]   A;
 wire   [17:0]   B;
 wire   [47:0]   C;
 wire   [47:0]   P;


//*************************************************
//              pipeline operations
//*************************************************

 always @ (posedge CLK_3_2_1)
  begin
    Xt2_Pipe           <= Xt2;
    Xt_Pipe            <= Xt;
    Inv_Varx_Pipe      <= Inv_Varx;
    Miux_Inv_Varx_Pipe <= Miux_Inv_Varx;
  end

 always @ (posedge aCLK_3_2_2 or posedge EN_Bar)
  if (EN_Bar)  Aden1 <= 48'h0000_0000_0000;
```

```
  else         Aden1 <= P;
 always @ (posedge bCLK_3_2_2 or posedge EN_Bar)
  if (EN_Bar)  Aden2 <= 48'h0000_0000_0000;
  else         Aden2 <= P;
 always @ (posedge cCLK_3_2_2 or posedge EN_Bar)
  if (EN_Bar)  Aden3 <= 48'h0000_0000_0000;
  else         Aden3 <= P;
 always @ (posedge aCLK_3_2_3 or posedge EN_Bar)
  if (EN_Bar)  Anum1 <= 48'h0000_0000_0000;
  else         Anum1 <= P;
 always @ (posedge bCLK_3_2_3 or posedge EN_Bar)
  if (EN_Bar)  Anum2 <= 48'h0000_0000_0000;
  else         Anum2 <= P;
 always @ (posedge cCLK_3_2_3 or posedge EN_Bar)
  if (EN_Bar)  Anum3 <= 48'h0000_0000_0000;
  else         Anum3 <= P;


//************************************************
//               Data path MUX
//************************************************

MUX_2_1 A_SW [17:0] (.D0(Xt2_Pipe),
                     .D1(Xt_Pipe),
                     .S0(Am_Multi_Sel),
                     .O(A)  );

MUX_2_1 B_SW [17:0] (.D0(Inv_Varx_Pipe),
                     .D1(Miux_Inv_Varx_Pipe),
                     .S0(Am_Multi_Sel),
                     .O(B)  );

MUX_2_1 C_SW [47:0] (.D0(Aden_In),
                     .D1(Anum_In),
                     .S0(Am_Multi_Sel),
                     .O(C)  );

Mux_4_1   Aden_Mux [47:0] (.D0(48'h0000_0000_0000),
                           .D1(Aden1),
                           .D2(Aden2),
                           .D3(Aden3),
                           .S0(Tran_Sel[0]),
                           .S1(Tran_Sel[1]),
                           .O(Aden)
                          );

Mux_4_1   Anum_Mux [47:0] (.D0(48'h0000_0000_0000),
                           .D1(Anum1),
                           .D2(Anum2),
                           .D3(Anum3),
                           .S0(Tran_Sel[0]),
                           .S1(Tran_Sel[1]),
                           .O(Anum_tmp)
```

```
                        );

 assign Aden_In = Aden;
 assign Anum_In = Anum_tmp;
 assign Anum = {Anum_tmp[45:0], 2'b00};


//*****************************************************************************
//
// DSP48 ( Virtex-4 ) instantiation for function of "P = A x B + C"
//
//*****************************************************************************

DSP48 DSP48_inst ( .BCOUT(),                            // 18-bit B
cascade output
                   .P(P),                               // 48-bit product output
                   .PCOUT(),                            // 38-bit cascade output
                   .A(A),                               // 18-bit A data input
                   .B(B),                               // 18-bit B data input
                   .BCIN(18'h00000),                    // 18-bit B cascade input
                   .C(C),                               // 48-bit cascade input
                   .CARRYIN(1'b0),                      // Carry input signal
                   .CARRYINSEL(2'b00),                  // 2-bit carry input
                                                        // select
                   .CEA(1'b1),                          // A data clock enable
                   .CEB(1'b1),                          // B data clock enable
                   .CEC(1'b1),                          // C data clock enable
                   .CECARRYIN(1'b0),                    // CARRYIN clock enable
                   .CECINSUB(1'b0),                     // CINSUB clock enable
                   .CECTRL(1'b0),                       // Clock Enable input for
                                                        // CTRL registers
                   .CEM(1'b1),                          // Clock Enable input for
                                                        // multiplier registers
                   .CEP(1'b1),                          // Clock Enable input for
                                                        // P registers
                   .CLK(CLK_SYS),                       // Clock input
                   .OPMODE(7'b011_01_01),               // 7-bit operation mode
                                                        // input
                   .PCIN(48'h0000_0000_0000),           // 48-bit PCIN input
                   .RSTA(EN_Bar),                       // Reset input for A
                                                        // pipeline registers
                   .RSTB(EN_Bar),                       // Reset input for B
                                                        // pipeline registers
                   .RSTC(EN_Bar),                       // Reset input for C
                                                        // pipeline registers
                   .RSTCARRYIN(1'b0),                   // Reset input for
                                                        // CARRYIN registers
                   .RSTCTRL(1'b0),                      // Reset input for CTRL
                                                        // registers
                   .RSTM(EN_Bar),                       // Reset input for
                                                        // multiplier registers
                   .RSTP(EN_Bar),                       // Reset input for P
                                                        // pipeline registers
```

```
                  .SUBTRACT(1'b0)                    // SUBTRACT input
                 );
 defparam DSP48_inst.AREG = 1;
 defparam DSP48_inst.BREG = 1;
 defparam DSP48_inst.B_INPUT = "DIRECT";
 defparam DSP48_inst.CARRYINREG = 0;
 defparam DSP48_inst.CARRYINSELREG = 0;
 defparam DSP48_inst.CREG = 1;
 defparam DSP48_inst.LEGACY_MODE = "MULT18X18S";
 defparam DSP48_inst.MREG = 1;
 defparam DSP48_inst.OPMODEREG = 0;
 defparam DSP48_inst.PREG = 1;
 defparam DSP48_inst.SUBTRACTREG = 0;
// End of DSP48_inst instantiation

endmodule



//////////////////////////////////////////////////////////
//
//  Am2:  Shift the denominator to reciprocal LUT range,
//       and accordingly shift numeritor
//       calculate the reciprocal of denominator
//       and multiply with numeritor
//
// Two block under this level:
//       1.  Parallel_Shift_Reg (Parallel load and
//                               parallel output shift reg)
//       2.  Syn_Counter_4bit_Asyn_Rst (Synch Counter with
//                               asynch clear
//
//////////////////////////////////////////////////////////

module Am2(   input               CLK_SYS,
              input               CLK_3_2_4,
              input               CLK_3_2_5,
              input               EN,
              input               Aden_LS,
              input               Anum_LS,
              input       [47:0]  Aden,
              input       [47:0]  Anum,
              output      [9:0]   Am
          );

//*****************************************************
// internal signals
//*****************************************************

 wire        Full1,  Full2,  Q47_Bar,   CLK_Count1, CLK_Count2;
 wire        CLK_Count1BR, CLK_Count2BR;
 wire [4:0]  Count2, Count1;
 wire [47:0] Rescaled_Aden;
```

```
 wire [47:0] Rescaled_Anum;
 wire [15:0] Inv_Aden;
 wire [17:0] A;
 wire [47:0] A_Full;
 reg  [4:0]  Threshold = 5'b00000;
 reg  [9:0]  Aden_LUT_Addr = 10'h000;
 reg  [17:0] M1 = 18'h000;
 reg  [17:0] M2 = 18'h000;

//***************************
// Pipeline Reg operations
//***************************

 always @ (posedge CLK_3_2_4)
  begin
    Aden_LUT_Addr <= Rescaled_Aden[47:38];
     Threshold     <= 5'b1_1010 - Count1;
  end
 always @ (posedge CLK_3_2_5)
  begin
      M1     <= {2'b00,Inv_Aden};
      M2[17] <= Rescaled_Anum[20];
      M2[16] <= Rescaled_Anum[21];
      M2[15] <= Rescaled_Anum[22];
      M2[14] <= Rescaled_Anum[23];
      M2[13] <= Rescaled_Anum[24];
      M2[12] <= Rescaled_Anum[25];
      M2[11] <= Rescaled_Anum[26];
      M2[10] <= Rescaled_Anum[27];
      M2[9]  <= Rescaled_Anum[28];
      M2[8]  <= Rescaled_Anum[29];
      M2[7]  <= Rescaled_Anum[30];
      M2[6]  <= Rescaled_Anum[31];
      M2[5]  <= Rescaled_Anum[32];
      M2[4]  <= Rescaled_Anum[33];
      M2[3]  <= Rescaled_Anum[34];
      M2[2]  <= Rescaled_Anum[35];
      M2[1]  <= Rescaled_Anum[36];
      M2[0]  <= Rescaled_Anum[37];
  end

//************************************************
// state machine1 for shift Aden (denominator)
//************************************************

 assign CLK_Count1 = EN & CLK_SYS & Full1 & (Q47_Bar | Aden_LS);
 assign CLK_Count1BR = CLK_Count1;

//BUFG CLK_Count1BUFR (
//.O(CLK_Count1BR),
//.CE(1'b1),
//.CLR(1'b0),
//.I(CLK_Count1)
```

```
//);
// Declaring constraints in Verilog
//synthesis attribute BUFR_DIVIDE of U_BUFR is BYPASS;
// synthesis attribute LOC of U_BUFR is "BUFR_X#Y#";
// where # is valid integer locations of BUFR

 assign Q47_Bar = ~Rescaled_Aden[47];
 assign Full1 = (Count1 < 5'b11010) ? 1 : 0;

//-------------------------------------------------
//  4bit Asynchro. counter to recorder shift #
//-------------------------------------------------

Syn_Counter_4bit_Asyn_RST Counter1(.CLK(CLK_Count1BR),
                                   .RST(Aden_LS)
                                   .Q(Count1) );

//-------------------------------------------------
//  48bits parallel load parallel output shift reg
//-------------------------------------------------

Parallel_Shift_Reg        Shift1(.D(Aden),
                                 .CLK_SYS(CLK_Count1BR),
                                 .L_S(Aden_LS),
                                 .Q(Rescaled_Aden)  );


//*************************************************
// state machine2 for shifting Anum (numeritor)
//*************************************************

 assign CLK_Count2 = CLK_SYS & EN & Full2;
 assign CLK_Count2BR = CLK_Count2;


//BUFG CLK_Count2BUFR (
//.O(CLK_Count2BR),
//.CE(1'b1),
//.CLR(1'b0),
//.I(CLK_Count2)
//);

//-----------------------------------------
// inverse connect the Anum to shift reg input
//-----------------------------------------

wire [47:0] Anum_Inv; assign Anum_Inv[0] = Anum[47],
         Anum_Inv[1] = Anum[46],
         Anum_Inv[2] = Anum[45],
         Anum_Inv[3] = Anum[44],
         Anum_Inv[4] = Anum[43],
         Anum_Inv[5] = Anum[42],
         Anum_Inv[6] = Anum[41],
```

```
          Anum_Inv[7] = Anum[40],
          Anum_Inv[8] = Anum[39],
          Anum_Inv[9] = Anum[38],
          Anum_Inv[10] = Anum[37],
          Anum_Inv[11] = Anum[36],
          Anum_Inv[12] = Anum[35],
          Anum_Inv[13] = Anum[34],
          Anum_Inv[14] = Anum[33],
          Anum_Inv[15] = Anum[32],
          Anum_Inv[16] = Anum[31],
          Anum_Inv[17] = Anum[30],
          Anum_Inv[18] = Anum[29],
          Anum_Inv[19] = Anum[28],
          Anum_Inv[20] = Anum[27],
          Anum_Inv[21] = Anum[26],
          Anum_Inv[22] = Anum[25],
          Anum_Inv[23] = Anum[24],
          Anum_Inv[24] = Anum[23],
          Anum_Inv[25] = Anum[22],
          Anum_Inv[26] = Anum[21],
          Anum_Inv[27] = Anum[20],
          Anum_Inv[28] = Anum[19],
          Anum_Inv[29] = Anum[18],
          Anum_Inv[30] = Anum[17],
          Anum_Inv[31] = Anum[16],
          Anum_Inv[32] = Anum[15],
          Anum_Inv[33] = Anum[14],
          Anum_Inv[34] = Anum[13],
          Anum_Inv[35] = Anum[12],
          Anum_Inv[36] = Anum[11],
          Anum_Inv[37] = Anum[10],
          Anum_Inv[38] = Anum[9],
          Anum_Inv[39] = Anum[8],
          Anum_Inv[40] = Anum[7],
          Anum_Inv[41] = Anum[6],
          Anum_Inv[42] = Anum[5],
          Anum_Inv[43] = Anum[4],
          Anum_Inv[44] = Anum[3],
          Anum_Inv[45] = Anum[2],
          Anum_Inv[46] = Anum[1],
          Anum_Inv[47] = Anum[0];

 assign Full2 = (Count2 < Threshold) ? 1 : 0;

//-------------------------------------------------
//  4bit Asynchro. counter to recorder shift #
//-------------------------------------------------

Syn_Counter_4bit_Asyn_RST Counter2(.CLK(CLK_Count2BR),
                                   .RST(Anum_LS),
                                   .Q(Count2) );
//-------------------------------------------------
//  48bits parallel load parallel output shift reg
```

```
//-------------------------------------------------

Parallel_Shift_Reg        Shift2(.D(Anum_Inv[47:0]),
                                 .CLK_SYS(CLK_Count2BR),
                                 .L_S(Anum_LS),
                                 .Q(Rescaled_Anum)  );
//*****************************************************
// Am = Anum * (1/Aden) limit range of Am to [0.5 2]
//*****************************************************

 assign  A_Full = M1 * M2;
 assign  A = A_Full[25:8];
 assign  Am = (A > 18'h003FF) ? (10'b11_1111_1111) :
 ((A < 18'h00100) ? (10'b01_0000_0000) : (A[9:0]));

//*****************************************************************************
//   LUT: Input: [2^(-4) 64] Ouput: 1/Aden ,  resolution 2^(-4), one 18Kbit RAM
//*****************************************************************************

RAMB16_S18 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
.INIT_3F(256'h0020002000200020002000200020002000200020002000200020002000200020),
.INIT_3E(256'h0021002100210021002100210021002100210021002100210021002100210021),
.INIT_3D(256'h0021002100210021002100210021002100210021002100210021002100220022),
.INIT_3C(256'h0022002200220022002200220022002200220022002200220022002200220022),
.INIT_3B(256'h0022002200220022002200220022002200220022002200230023002300230023),
.INIT_3A(256'h0023002300230023002300230023002300230023002300230023002300230023),
.INIT_39(256'h0023002300230023002400240024002400240024002400240024002400240024),
.INIT_38(256'h0024002400240024002400240024002400240024002400240024002400240025),
.INIT_37(256'h0025002500250025002500250025002500250025002500250025002500250025),
.INIT_36(256'h0025002500250025002500250026002600260026002600260026002600260026),
.INIT_35(256'h0026002600260026002600260026002600260026002600260026002700270027),
.INIT_34(256'h0027002700270027002700270027002700270027002700270027002700270027),
.INIT_33(256'h0027002700270028002800280028002800280028002800280028002800280028),
.INIT_32(256'h0028002800280028002800280028002900290029002900290029002900290029),
.INIT_31(256'h0029002900290029002900290029002900290029002a002a002a002a002a002a),
.INIT_30(256'h002a002a002a002a002a002a002a002a002a002a002a002a002a002a002b002b),
.INIT_2F(256'h002b002b002b002b002b002b002b002b002b002b002b002b002b002b002b002c),
.INIT_2E(256'h002c002c002c002c002c002c002c002c002c002c002c002c002c002c002c002c),
.INIT_2D(256'h002d002d002d002d002d002d002d002d002d002d002d002d002d002d002d002d),
.INIT_2C(256'h002e002e002e002e002e002e002e002e002e002e002e002e002e002e002e002e),
.INIT_2B(256'h002f002f002f002f002f002f002f002f002f002f002f002f002f002f002f0030),
.INIT_2A(256'h0030003000300030003000300030003000300030003000300030003100310031),
.INIT_29(256'h0031003100310031003100310031003100310031003100320032003200320032),
.INIT_28(256'h0032003200320032003200320032003300330033003300330033003300330033),
.INIT_27(256'h0033003300330033003400340034003400340034003400340034003400340034),
.INIT_26(256'h0035003500350035003500350035003500350035003500350035003600360036),
.INIT_25(256'h0036003600360036003600360036003700370037003700370037003700370037),
.INIT_24(256'h0037003700380038003800380038003800380038003800380038003900390039),
.INIT_23(256'h0039003900390039003900390039003a003a003a003a003a003a003a003a003a),
```

```
.INIT_22(256'h003b003b003b003b003b003b003b003b003b003b003c003c003c003c003c003c),
.INIT_21(256'h003c003c003c003d003d003d003d003d003d003d003d003d003e003e003e003e),
.INIT_20(256'h003e003e003e003e003f003f003f003f003f003f003f003f0040004000400040),
.INIT_1F(256'h0040004000400040004100410041004100410041004100410041004200420042),
.INIT_1E(256'h0042004200420042004300430043004300430043004300430043004400440044),
.INIT_1D(256'h0044004400450045004500450045004500450045004600460046004600460046),
.INIT_1C(256'h0047004700470047004700470048004800480048004800480048004800490049),
.INIT_1B(256'h004900490049004a004a004a004a004a004a004b004b004b004b004b004c004c),
.INIT_1A(256'h004c004c004c004c004d004d004d004d004d004d004e004e004e004e004e004f),
.INIT_19(256'h004f004f004f004f005000500050005000500051005100510051005100520052),
.INIT_18(256'h005200520052005300530053005300530054005400540054005400550055005),
.INIT_17(256'h005500560056005600560056005700570057005700580058005800580059005),
.INIT_16(256'h00590059005a005a005a005a005b005b005b005b005c005c005c005c005d005d),
.INIT_15(256'h005d005d005e005e005e005e005f005f005f006000600060006000610061006),
.INIT_14(256'h006200620062006200630063006300640064006400650065006500650066006),
.INIT_13(256'h006600670067006700680068006800690069006a006a006a006b006b006b),
.INIT_12(256'h006c006c006d006d006d006e006e006e006f006f006f007000700071007100),
.INIT_11(256'h007200720073007300730074007400750075007500760076007700770078007),
.INIT_10(256'h007800790079007a007a007b007b007c007c007d007d007e007e007f007f0080),
.INIT_0F(256'h008000810081008200820083008300840084008500850086008600870087008),
.INIT_0E(256'h00890089008a008a008b008b008c008d008d008e008e008f0090009000910092),
.INIT_0D(256'h009200930094009400950096009600970098009800990099a009b009b009c009d),
.INIT_0C(256'h009e009e009f00a000a100a100a200a300a400a500a500a600a700a800a900aa),
.INIT_0B(256'h00ab00ac00ac00ad00ae00af00b000b100b200b300b400b500b600b700b800b9),
.INIT_0A(256'h00ba00bb00bc00bd00bf00c000c100c200c300c400c500c700c800c900ca00cc),
.INIT_09(256'h00cd00ce00cf00d100d200d300d500d600d800d900da00dc00dd00df00e000e2),
.INIT_08(256'h00e400e500e700e800ea00ec00ed00ef00f100f300f500f600f800fa00fc00fe),
.INIT_07(256'h010001020104010601080 10a010d010f011101130116011801 1a011d011f0122),
.INIT_06(256'h01250127012a012d012f013201350138013b013e014101440148014b014e0152),
.INIT_05(256'h01550159015d016001640168016c017001740179017d01820186018b01900195),
.INIT_04(256'h019a019f01a401aa01af01b501bb01c101c701ce01d401db01e201e901f001f8),
.INIT_03(256'h020002080211021902220 22b0235023f02490254025f026a02760283028f029d),
.INIT_02(256'h02ab02b902c802d802e902fa030c031f03330348035e0376038e03a803c403e1),
.INIT_01(256'h040004210444046a049204be04ec051f0555059105d10618066606bd071c0788),
.INIT_00(256'h08000889092509d90aab0ba30ccd0e39100012491555199a20002aab40008000)
)

RAM16_LUT_Inv_Aden(  .DO(Inv_Aden),
                     .DOP(),                        // 2-bit A port parity data
                                                    // output
                     .ADDR(Aden_LUT_Addr),          // 10-bit A port address
                                                    // input
                     .CLK(CLK_SYS),                 // 1-bit A port clock input
                     .DI(16'hFFFF),                 // 16-bit A port data input
                     .DIP(2'b11),                   // 2-bit A port parity data
                                                    // input
                     .EN(EN),                       // 1-bit A port enable input
                     .SSR(1'b0),                    // 1-bit A port set/reset
                                                    // input
                     .WE(1'b0)                      // 1-bit A port write enable
                                                    // input
                  );
```

```
endmodule


////////////////////////////////////////////////////////////////
//
// 48 bits parallel load parallel output shift reg, from 0 - 47
// One module under this level: Shift_reg (1bit shift reg)
//
////////////////////////////////////////////////////////////////


module Parallel_Shift_Reg(input      [47:0] D,
                          input             CLK_SYS,
                          input             L_S,
                          output     [47:0] Q );

//**********************************
//        internal  signals
//**********************************

wire SERIN = 1'b0;


Shift_Reg Parallel_Shift_0 (.CLK_SYS(CLK_SYS), // LSB
                            .SERIN(SERIN),
                            .D(D[0]),
                            .L_S(L_S),
                            .Q(Q[0])  );

Shift_Reg Parallel_Shift_1 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[0]),
                            .D(D[1]),
                            .L_S(L_S),
                            .Q(Q[1])  );

Shift_Reg Parallel_Shift_2 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[1]),
                            .D(D[2]),
                            .L_S(L_S),
                            .Q(Q[2])  );

Shift_Reg Parallel_Shift_3 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[2]),
                            .D(D[3]),
                            .L_S(L_S),
                            .Q(Q[3])  );


Shift_Reg Parallel_Shift_4 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[3]),
                            .D(D[4]),
                            .L_S(L_S),
```

```
                            .Q(Q[4])  );

Shift_Reg Parallel_Shift_5 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[4]),
                            .D(D[5]),
                            .L_S(L_S),
                            .Q(Q[5])  );


Shift_Reg Parallel_Shift_6 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[5]),
                            .D(D[6]),
                            .L_S(L_S),
                            .Q(Q[6])  );

Shift_Reg Parallel_Shift_7 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[6]),
                            .D(D[7]),
                            .L_S(L_S),
                            .Q(Q[7])  );


Shift_Reg Parallel_Shift_8 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[7]),
                            .D(D[8]),
                            .L_S(L_S),
                            .Q(Q[8])  );

Shift_Reg Parallel_Shift_9 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[8]),
                            .D(D[9]),
                            .L_S(L_S),
                            .Q(Q[9])  );
Shift_Reg Parallel_Shift_10 (.CLK_SYS(CLK_SYS),
                             .SERIN(Q[9]),
                             .D(D[10]),
                             .L_S(L_S),
                             .Q(Q[10])  );

Shift_Reg Parallel_Shift_11 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[10]),
                            .D(D[11]),
                            .L_S(L_S),
                            .Q(Q[11])  );


Shift_Reg Parallel_Shift_12 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[11]),
                            .D(D[12]),
                            .L_S(L_S),
                            .Q(Q[12])  );

Shift_Reg Parallel_Shift_13 (.CLK_SYS(CLK_SYS),
```

```
                              .SERIN(Q[12]),
                              .D(D[13]),
                              .L_S(L_S),
                              .Q(Q[13])  );

Shift_Reg Parallel_Shift_14 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[13]),
                              .D(D[14]),
                              .L_S(L_S),
                              .Q(Q[14])  );

Shift_Reg Parallel_Shift_15 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[14]),
                              .D(D[15]),
                              .L_S(L_S),
                              .Q(Q[15])  );


Shift_Reg Parallel_Shift_16 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[15]),
                              .D(D[16]),
                              .L_S(L_S),
                              .Q(Q[16])  );

Shift_Reg Parallel_Shift_17 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[16]),
                              .D(D[17]),
                              .L_S(L_S),
                              .Q(Q[17])  );

Shift_Reg Parallel_Shift_18 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[17]),
                              .D(D[18]),
                              .L_S(L_S),
                              .Q(Q[18])  );

Shift_Reg Parallel_Shift_19 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[18]),
                              .D(D[19]),
                              .L_S(L_S),
                              .Q(Q[19])  );


Shift_Reg Parallel_Shift_20 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[19]),
                              .D(D[20]),
                              .L_S(L_S),
                              .Q(Q[20])  );

Shift_Reg Parallel_Shift_21 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[20]),
                              .D(D[21]),
                              .L_S(L_S),
```

```
                            .Q(Q[21])  );

Shift_Reg Parallel_Shift_22 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[21]),
                            .D(D[22]),
                            .L_S(L_S),
                            .Q(Q[22])  );

Shift_Reg Parallel_Shift_23 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[22]),
                            .D(D[23]),
                            .L_S(L_S),
                            .Q(Q[23])  );


Shift_Reg Parallel_Shift_24 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[23]),
                            .D(D[24]),
                            .L_S(L_S),
                            .Q(Q[24])  );

Shift_Reg Parallel_Shift_25 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[24]),
                            .D(D[25]),
                            .L_S(L_S),
                            .Q(Q[25])  );

Shift_Reg Parallel_Shift_26 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[25]),
                            .D(D[26]),
                            .L_S(L_S),
                            .Q(Q[26])  );

Shift_Reg Parallel_Shift_27 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[26]),
                            .D(D[27]),
                            .L_S(L_S),
                            .Q(Q[27])  );


Shift_Reg Parallel_Shift_28 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[27]),
                            .D(D[28]),
                            .L_S(L_S),
                            .Q(Q[28])  );

Shift_Reg Parallel_Shift_29 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[28]),
                            .D(D[29]),
                            .L_S(L_S),
                            .Q(Q[29])  );

Shift_Reg Parallel_Shift_30 (.CLK_SYS(CLK_SYS),
```

```
                              .SERIN(Q[29]),
                              .D(D[30]),
                              .L_S(L_S),
                              .Q(Q[30])  );

Shift_Reg Parallel_Shift_31 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[30]),
                              .D(D[31]),
                              .L_S(L_S),
                              .Q(Q[31])  );


Shift_Reg Parallel_Shift_32 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[31]),
                              .D(D[32]),
                              .L_S(L_S),
                              .Q(Q[32])  );

Shift_Reg Parallel_Shift_33 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[32]),
                              .D(D[33]),
                              .L_S(L_S),
                              .Q(Q[33])  );

Shift_Reg Parallel_Shift_34 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[33]),
                              .D(D[34]),
                              .L_S(L_S),
                              .Q(Q[34])  );

Shift_Reg Parallel_Shift_35 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[34]),
                              .D(D[35]),
                              .L_S(L_S),
                              .Q(Q[35])  );


Shift_Reg Parallel_Shift_36 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[35]),
                              .D(D[36]),
                              .L_S(L_S),
                              .Q(Q[36])  );

Shift_Reg Parallel_Shift_37 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[36]),
                              .D(D[37]),
                              .L_S(L_S),
                              .Q(Q[37])  );

Shift_Reg Parallel_Shift_38 (.CLK_SYS(CLK_SYS),
                              .SERIN(Q[37]),
                              .D(D[38]),
                              .L_S(L_S),
```

```
                            .Q(Q[38])  );

Shift_Reg Parallel_Shift_39 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[38]),
                            .D(D[39]),
                            .L_S(L_S),
                            .Q(Q[39])  );


Shift_Reg Parallel_Shift_40 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[39]),
                            .D(D[40]),
                            .L_S(L_S),
                            .Q(Q[40])  );

Shift_Reg Parallel_Shift_41 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[40]),
                            .D(D[41]),
                            .L_S(L_S),
                            .Q(Q[41])  );

Shift_Reg Parallel_Shift_42 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[41]),
                            .D(D[42]),
                            .L_S(L_S),
                            .Q(Q[42])  );

Shift_Reg Parallel_Shift_43 (.CLK_SYS(CLK_SYS),
                           .SERIN(Q[42]),
                               .D(D[43]),
                               .L_S(L_S),
                               .Q(Q[43])  );

Shift_Reg Parallel_Shift_44 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[43]),
                            .D(D[44]),
                            .L_S(L_S),
                            .Q(Q[44])  );

Shift_Reg Parallel_Shift_45 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[44]),
                            .D(D[45]),
                            .L_S(L_S),
                            .Q(Q[45])  );

Shift_Reg Parallel_Shift_46 (.CLK_SYS(CLK_SYS),
                            .SERIN(Q[45]),
                            .D(D[46]),
                            .L_S(L_S),
                            .Q(Q[46])  );

Shift_Reg Parallel_Shift_47 (.CLK_SYS(CLK_SYS),  //MSB
                            .SERIN(Q[46]),
```

```
                              .D(D[47]),
                              .L_S(L_S),
                              .Q(Q[47])  );

endmodule



////////////////////////////////////////////////////////////////////
//
//              1 bit reg with load shift switch control
//
////////////////////////////////////////////////////////////////////


module Shift_Reg(input      CLK_SYS,
                 input      SERIN,
                 input      D,
                 input      L_S,
                 output reg Q  );

// internal signals

wire L_S_Bar, S, DI, SD;


// Shift Reg with load shift control

 assign L_S_Bar = ~L_S;
 assign S       = L_S_Bar & SERIN;
 assign DI = D & L_S;
 assign SD = S | DI;

 always @ (posedge CLK_SYS) Q <= SD;

endmodule



////////////////////////////////////////////////////////////////////////////////
//
//      5 bit synch counter with asynch clear
//
////////////////////////////////////////////////////////////////////////////////


// 5-bit Unsigned SYNCHRONOUS Up Counter with Asynchronous Clear
// Modelsim simulation shows maximum 14 ps delay between different output ports

module Syn_Counter_4bit_Asyn_RST (input         CLK,
                                  input         RST,
                                  output [4:0]  Q );
reg [4:0] tmp;
```

```
always @(posedge CLK or posedge RST)
   begin
      if (RST)
         tmp <= 4'b0000;
      else
         tmp <= tmp + 1'b1;
   end

assign Q = tmp;

endmodule


`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
//    Likelihood calculation between inputs and clusters
//    in order to easily compare likelihood and calculate the filter weight,
//    likelihhod value in this block is changed to positive number
//
////////////////////////////////////////////////////////////////////////////////



//*************************************************************
// In this block, be careful about the data extend and trim
// between signed and unsigned data
//*************************************************************

module Likelihood(      input               EN,
                        input               CLK_SYS,
                        input               CLK_3_2_6,
                        input               aCLK_3_2_6,
                        input               bCLK_3_2_6,
                        input               cCLK_3_2_6,
                        input               CLK_3_2_7,
                        input               CLK_3_2_8,
                        input               CLK_3_2_9,
                        input               aCLK_3_2_10,
                        input               bCLK_3_2_10,
                        input               cCLK_3_2_10,
                        input               Am_Multi_Sel,
                        input        [1:0]  Tran_Sel,
                        input        [17:0] Xt,
                        input        [15:0] Miux,
                        input        [17:0] Inv_Varx,
                        input signed [15:0] Log_Varx,
                        input        [9:0]  Am,
                        output reg [9:0]  Ama = 10'h000,
                        output reg [9:0]  Amb = 10'h000,
                        output reg [9:0]  Amc = 10'h000,
                        output reg [47:0] Likelihooda = 48'h0000_0000_0000,
```

```
                        output reg [47:0] Likelihoodb = 48'h0000_0000_0000,
                        output reg [47:0] Likelihoodc = 48'h0000_0000_0000
                        );

//**********************************
// internal signals
//**********************************

 wire EN_Bar;
 assign EN_Bar = ~EN;

 wire        [9:0]  Am_Mux;
 wire signed [47:0] C;
 wire signed [17:0] B;
 reg  signed [17:0] A = 18'h00000;
 wire signed [47:0] PA;
 wire signed [47:0] PA_tmp;  // PA_tmp is 2's complement of PA
 reg  signed [17:0] A1 = 18'h00000;
 reg  signed [17:0] A2 = 18'h00000;
 reg  signed [17:0] B2 = 18'h00000;
 wire signed [17:0] MB;
 wire signed [17:0] MA;
 wire signed [35:0] P;
 wire signed [17:0] P1;
 reg signed  [35:0] S1 = 36'h0_0000_0000;
 reg signed  [35:0] S2 = 36'h0_0000_0000;
 wire signed [15:0] S2_tmp;
 wire signed [35:0] Sum1;
 wire signed [35:0] Sum1_tmp;// Sum1_tmp is 2's complement of Sum1
 wire signed [35:0] Sum2;
 wire signed [47:0] Likelihood_In;
 wire signed [47:0] Likelihood_Out;

//**************************************************
// pipeline reg operations
//**************************************************

 always @ (posedge CLK_3_2_6)   A <= Xt;
 always @ (posedge CLK_3_2_8)   begin A2 <= Inv_Varx; B2 <= P1; end
 always @ (posedge aCLK_3_2_6)  Ama <= Am;
 always @ (posedge bCLK_3_2_6)  Amb <= Am;
 always @ (posedge cCLK_3_2_6)  Amc <= Am;
 always @ (posedge aCLK_3_2_10 or posedge EN_Bar)
  begin
    if (EN_Bar) Likelihooda <= 48'h0000_0000_0000;
     else       Likelihooda <= (Likelihood_Out[47] == 0) ? (Likelihood_Out) :
                               ( (~Likelihood_Out) +48'h0000_0000_0001);
  end

always @ (posedge bCLK_3_2_10 or posedge EN_Bar)
  begin
    if (EN_Bar) Likelihoodb <= 48'h0000_0000_0000;
     else       Likelihoodb <= (Likelihood_Out[47] == 0) ? (Likelihood_Out) :
```

```
                              ( (~Likelihood_Out) +48'h0000_0000_0001);
  end

always @ (posedge cCLK_3_2_10 or posedge EN_Bar)
  begin
    if (EN_Bar) Likelihoodc <= 48'h0000_0000_0000;
     else       Likelihoodc <= (Likelihood_Out[47] == 0) ? (Likelihood_Out) :
                              ( (~Likelihood_Out) +48'h0000_0000_0001);
  end

always @ (posedge CLK_3_2_7)
  begin
    if (PA[47] == 0) begin A1 <= PA[26:9];                    end
     else             begin A1 <= (~PA_tmp[26:9]) +18'h00001; end
  end

always @ (posedge CLK_3_2_9)
  begin
     //if (P[35] == 0)
      S1 <= {5'b0_0000,P[35:5]};
      //else S1 <= (~{5'b0_0000,S1_tmp[35:5]}) + 36'h0_0000_0001;

      if (Log_Varx[15] == 0) S2 <= {20'h0_0000,Log_Varx};
      else                   S2 <= (~{20'h0_0000,S2_tmp}) + 36'h0_0000_0001;
  end

//*****************************************************
// signed and unsigned data extending and trim
//*****************************************************

assign C      = (~{23'h000000,Miux,9'h000}) + 48'h0000_0000_0001;
assign PA_tmp = (~PA) + 48'h0000_0000_0001; // if PA[47] == 1,
then PA_tmp is PA's magnitude

assign S2_tmp = (~Log_Varx) + 16'h0001;   // if Log_Varx[15] == 1,
then S2_tmp is Log_Varx's magnitude

assign Sum1_tmp = (~Sum1) + 36'h0_0000_0001;

assign Sum2 = (Sum1[35] == 0) ? ({12'h000,Sum1[35:1],1'b0}) :
((~{12'h000,Sum1_tmp[35:1],1'b0}) + 48'h0000_0000_0001);

// SINCE P = Inv_Varx * (X*Am-Miu)^2 >= 0, therefore no necessary see the
// sign here
// assign S1_tmp = (~P)  + 36'h0_0000_0001;    // if P[35] == 1, then S1_tmp
// is P's magnitude

//*****************************************************
// Combinational logic connections
//*****************************************************

 assign P1 = P[30:13];     // P1 = S15/3 to prevent overflow
 assign B = {8'h00,Am_Mux};
```

```
 assign Sum1 = S1 + S2;
 assign P = MA * MB;


////////////////////////////////////////////////
//  CHANGE LIKELIHOOD SIGN TO POSITIVE HERE
//  FOR SAVING RESOURCES OF THE FOLLOWING BLOCKS
////////////////////////////////////////////////
assign Likelihood_Out = Likelihood_In + Sum2;

//*****************************************************
// MUX
//*****************************************************

MUX_2_1 A_SW [17:0] (.D0(A1),
                     .D1(A2),
                     .S0(Am_Multi_Sel),
                     .O(MA)  );


MUX_2_1 B_SW [17:0] (.D0(A1),
                     .D1(B2),
                     .S0(Am_Multi_Sel),
                     .O(MB)  );


Mux_4_1   Am_SW [9:0](.D0(10'b00_0000_0000),
                      .D1(Ama),
                      .D2(Amb),
                      .D3(Amc),
                      .S0(Tran_Sel[0]),
                      .S1(Tran_Sel[1]),
                      .O(Am_Mux)
                      );

Mux_4_1   Likelihood_Mux [47:0](.D0(48'h0000_0000_0000),
                                .D1(Likelihooda),
                                .D2(Likelihoodb),
                                .D3(Likelihoodc),
                                .S0(Tran_Sel[0]),
                                .S1(Tran_Sel[1]),
                                .O(Likelihood_In)
                                     );


//*****************************************************************
// DSP48: DSP Function Block
// Virtex-4
// Xilinx HDL Language Template version 7.1i
//*****************************************************************

DSP48 DSP48_inst ( .BCOUT(),
```

```
                    .P(PA),
                    .PCOUT(),
                    .A(A),
                    .B(B),
                    .BCIN(18'h00000),
                    .C(C),
                    .CARRYIN(1'b0),
                    .CARRYINSEL(2'b00),
                    .CEA(1'b1),
                    .CEB(1'b1),
                    .CEC(1'b1),
                    .CECARRYIN(1'b0),
                    .CECINSUB(1'b0),
                    .CECTRL(1'b0),
                    .CEM(1'b1),
                    .CEP(1'b1),
                    .CLK(CLK_SYS),
                    .OPMODE(7'b011_01_01),
                    .PCIN(48'h0000_0000_0000),
                    .RSTA(EN_Bar),
                    .RSTB(EN_Bar),
                    .RSTC(EN_Bar),
                    .RSTCARRYIN(1'b0),
                    .RSTCTRL(1'b0),
                    .RSTM(EN_Bar),
                    .RSTP(EN_Bar),
                    .SUBTRACT(1'b0)
);
 defparam DSP48_inst.AREG = 1;
 defparam DSP48_inst.BREG = 1;
 defparam DSP48_inst.B_INPUT = "DIRECT";
 defparam DSP48_inst.CARRYINREG = 0;
 defparam DSP48_inst.CARRYINSELREG = 0;
 defparam DSP48_inst.CREG = 1;
 defparam DSP48_inst.LEGACY_MODE = "MULT18X18S";
 defparam DSP48_inst.MREG = 1;
 defparam DSP48_inst.OPMODEREG = 0;
 defparam DSP48_inst.PREG = 1;
 defparam DSP48_inst.SUBTRACTREG = 0;
// End of DSP48_inst instantiation

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////
// Likelihood comparation to find champion and rival candiadte from 10 clusters
// and 3 data path, two modules under this level:
// 1. COMSW1:  compare and swtich, basic block for constructing champion finding
//    tree.
// 2  COMSW2:  similar function of COMSW1, for building rival finding tree
//
//////////////////////////////////////////////////////////////////////////////
```

```
module Likelihood_Comp(       input             EN,
                              input             CLK_SYS,
                              input             CLK_3_3_1,
                              input             CLK_3_3_2,
                              input             CLK_3_3_3,
                              input             CLK_3_3_7,
                              input      [47:0] Likelihood1a,
                              input      [47:0] Likelihood2a,
                              input      [47:0] Likelihood3a,
                              input      [47:0] Likelihood4a,
                              input      [47:0] Likelihood5a,
                              input      [47:0] Likelihood6a,
                              input      [47:0] Likelihood7a,
                              input      [47:0] Likelihood8a,
                              input      [47:0] Likelihood9a,
                              input      [47:0] Likelihood10a,
                              input      [47:0] Likelihood1b,
                              input      [47:0] Likelihood2b,
                              input      [47:0] Likelihood3b,
                              input      [47:0] Likelihood4b,
                              input      [47:0] Likelihood5b,
                              input      [47:0] Likelihood6b,
                              input      [47:0] Likelihood7b,
                              input      [47:0] Likelihood8b,
                              input      [47:0] Likelihood9b,
                              input      [47:0] Likelihood10b,
                              input      [47:0] Likelihood1c,
                              input      [47:0] Likelihood2c,
                              input      [47:0] Likelihood3c,
                              input      [47:0] Likelihood4c,
                              input      [47:0] Likelihood5c,
                              input      [47:0] Likelihood6c,
                              input      [47:0] Likelihood7c,
                              input      [47:0] Likelihood8c,
                              input      [47:0] Likelihood9c,
                              input      [47:0] Likelihood10c,
                              input      [9:0] Am1a,
                              input      [9:0] Am2a,
                              input      [9:0] Am3a,
                              input      [9:0] Am4a,
                              input      [9:0] Am5a,
                              input      [9:0] Am6a,
                              input      [9:0] Am7a,
                              input      [9:0] Am8a,
                              input      [9:0] Am9a,
                              input      [9:0] Am10a,
                              input      [9:0] Am1b,
                              input      [9:0] Am2b,
                              input      [9:0] Am3b,
                              input      [9:0] Am4b,
                              input      [9:0] Am5b,
```

```
                                input      [9:0] Am6b,
                                input      [9:0] Am7b,
                                input      [9:0] Am8b,
                                input      [9:0] Am9b,
                                input      [9:0] Am10b,
                                input      [9:0] Am1c,
                                input      [9:0] Am2c,
                                input      [9:0] Am3c,
                                input      [9:0] Am4c,
                                input      [9:0] Am5c,
                                input      [9:0] Am6c,
                                input      [9:0] Am7c,
                                input      [9:0] Am8c,
                                input      [9:0] Am9c,
                                input      [9:0] Am10c,
                                output reg [15:0] Scale = 16'h0000,
                                output reg [1:0]  M = 2'b00,
                                output reg [3:0]  Clus_Num1 = 4'b00,
                                output reg [3:0]  Clus_Num2 = 4'b00,
                                output reg [47:0] Likelihood2 =
                                                  48'h0000_0000_0000 );

//*****************************************
//   Internal Signals
//*****************************************

 reg      [47:0] MLikelihood1a, MLikelihood2a, MLikelihood3a,
MLikelihood4a, MLikelihood5a, MLikelihood6a, MLikelihood7a,
MLikelihood8a, MLikelihood9a, MLikelihood10a;

 reg [47:0] MLikelihood1b, MLikelihood2b, MLikelihood3b,
MLikelihood4b,MLikelihood5b, MLikelihood6b, MLikelihood7b,
MLikelihood8b, MLikelihood9b, MLikelihood10b;

 reg [47:0] MLikelihood1c, MLikelihood2c, MLikelihood3c,
MLikelihood4c, MLikelihood5c, MLikelihood6c, MLikelihood7c,
MLikelihood8c, MLikelihood9c, MLikelihood10c;

 wire [47:0] MLikelihood1,  MLikelihood2,  MLikelihood3,
MLikelihood4, MLikelihood5,  MLikelihood6, MLikelihood7,
MLikelihood8, MLikelihood9,  MLikelihood10;

 wire [47:0] RMLikelihood1, RMLikelihood2, RMLikelihood3,
RMLikelihood4, RMLikelihood5, RMLikelihood6, RMLikelihood7,
RMLikelihood8, RMLikelihood9, RMLikelihood10;

 wire [47:0] BRMLikelihood1, BRMLikelihood2, BRMLikelihood3,
BRMLikelihood4, BRMLikelihood5, BRMLikelihood6, BRMLikelihood7,
BRMLikelihood8,BRMLikelihood9,BRMLikelihood10;

 reg      [47:0] RBRML1, RBRML2, RBRML3, RBRML4,
                 RBRML5, RBRML6, RBRML7,RBRML8,
                 RBRML9, RBRML10;
```

```
 wire     [53:0] A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12,
                 A13, A14;
 wire     [53:0] B1, B2, B3, B4, B5, B6, B7, B8;
 wire     [53:0] C1, C2, C3, C4;
 wire     [53:0] D1, D2;
 wire     [47:0] Champion_Likelihood;
 wire     [3:0]  Champion_Clus_Num;
 wire     [1:0]  Champion_M;
 reg      [47:0] Max_Likelihood;
 wire     [51:0] RB1, RB2, RB3, RB4;
 wire     [51:0] RC1, RC2;
 wire     [51:0] RD1, RD2;
 wire     [47:0] Rival_Likelihood;
 wire     [3:0]  Rival_Clus_Num;
 reg      [9:0]  RAm1a, RAm2a, RAm3a, RAm4a, RAm5a, RAm6a, RAm7a,
                 RAm8a, RAm9a, RAm10a;
 reg      [9:0]  RAm1b, RAm2b, RAm3b, RAm4b, RAm5b, RAm6b, RAm7b,
                 RAm8b, RAm9b, RAm10b;
 reg      [9:0]  RAm1c, RAm2c, RAm3c, RAm4c, RAm5c, RAm6c, RAm7c,
                 RAm8c, RAm9c, RAm10c;
 wire     [9:0]  RAm1, RAm2,  RAm3,  RAm4, RAm5,  RAm6,  RAm7,
                 RAm8,  RAm9,  RAm10;
 wire     [9:0]  RAm;
 reg      [9:0]  Addr;
 wire     [15:0] Inv_Amin;



//*****************************************************
//      Likelihood Re-scaling
//*****************************************************

 assign  RMLikelihood1  = MLikelihood1  - Max_Likelihood;
 assign RMLikelihood2  = MLikelihood2  - Max_Likelihood;
 assign RMLikelihood3  = MLikelihood3  - Max_Likelihood;
 assign RMLikelihood4  = MLikelihood4  - Max_Likelihood;
 assign RMLikelihood5  = MLikelihood5  - Max_Likelihood;
 assign RMLikelihood6  = MLikelihood6  - Max_Likelihood;
 assign RMLikelihood7  = MLikelihood7  - Max_Likelihood;
 assign RMLikelihood8  = MLikelihood8  - Max_Likelihood;
 assign RMLikelihood9  = MLikelihood9  - Max_Likelihood;
 assign RMLikelihood10 = MLikelihood10 - Max_Likelihood;


//*************************************************************************
//   Block the Champion cluster by setting it's likelihood to all 1's
//*************************************************************************

 assign BRMLikelihood1  = (Clus_Num1 == 4'b0001) ?
 (48'h1111_1111_1111) : (RMLikelihood1);

assign BRMLikelihood2  = (Clus_Num1 == 4'b0010) ?
```

```
(48'h1111_1111_1111) : (RMLikelihood2);

assign BRMLikelihood3  = (Clus_Num1 == 4'b0011) ?
(48'h1111_1111_1111) : (RMLikelihood3);

assign BRMLikelihood4  = (Clus_Num1 == 4'b0100) ?
(48'h1111_1111_1111) : (RMLikelihood4);

assign BRMLikelihood5  = (Clus_Num1 == 4'b0101) ?
(48'h1111_1111_1111) : (RMLikelihood5);

assign BRMLikelihood6  = (Clus_Num1 == 4'b0110) ?
(48'h1111_1111_1111) : (RMLikelihood6);

assign BRMLikelihood7  = (Clus_Num1 == 4'b0111) ?
(48'h1111_1111_1111) : (RMLikelihood7);

assign BRMLikelihood8  = (Clus_Num1 == 4'b1000) ?
(48'h1111_1111_1111) : (RMLikelihood8);

assign BRMLikelihood9  = (Clus_Num1 == 4'b1001) ?
(48'h1111_1111_1111) : (RMLikelihood9);

assign BRMLikelihood10 = (Clus_Num1 == 4'b1010) ?
(48'h1111_1111_1111) : (RMLikelihood10);

//*****************************************************
//           MUX  Operations
//*****************************************************



//-------------------------------------------------------------
//         Choose Likelihoods from best data path of path A B C
//-------------------------------------------------------------

Mux_4_1   MLikelihood_Mux1 [47:0](.D0(48'h0000_0000_0000),
                                  .D1(MLikelihood1a),
                                        .D2(MLikelihood1b),
                                        .D3(MLikelihood1c),
                                        .S0(M[0]),
                                        .S1(M[1]),
                                        .O(MLikelihood1)
                                       );

Mux_4_1   MLikelihood_Mux2 [47:0](.D0(48'h0000_0000_0000),
                                  .D1(MLikelihood2a),
                                        .D2(MLikelihood2b),
                                        .D3(MLikelihood2c),
                                        .S0(M[0]),
                                        .S1(M[1]),
                                        .O(MLikelihood2)
                                       );
```

```
Mux_4_1   MLikelihood_Mux3 [47:0](.D0(48'h0000_0000_0000),
                                  .D1(MLikelihood3a),
                                        .D2(MLikelihood3b),
                                        .D3(MLikelihood3c),
                                        .S0(M[0]),
                                        .S1(M[1]),
                                        .O(MLikelihood3)
                                       );

Mux_4_1   MLikelihood_Mux4 [47:0](.D0(48'h0000_0000_0000),
                                  .D1(MLikelihood4a),
                                        .D2(MLikelihood4b),
                                        .D3(MLikelihood4c),
                                        .S0(M[0]),
                                        .S1(M[1]),
                                        .O(MLikelihood4)
                                       );

Mux_4_1   MLikelihood_Mux5 [47:0](.D0(48'h0000_0000_0000),
                                  .D1(MLikelihood5a),
                                        .D2(MLikelihood5b),
                                        .D3(MLikelihood5c),
                                        .S0(M[0]),
                                        .S1(M[1]),
                                        .O(MLikelihood5)
                                       );

Mux_4_1   MLikelihood_Mux6 [47:0](.D0(48'h0000_0000_0000),
                                  .D1(MLikelihood6a),
                                        .D2(MLikelihood6b),
                                        .D3(MLikelihood6c),
                                        .S0(M[0]),
                                        .S1(M[1]),
                                        .O(MLikelihood6)
                                       );

Mux_4_1   MLikelihood_Mux7 [47:0](.D0(48'h0000_0000_0000),
                                  .D1(MLikelihood7a),
                                        .D2(MLikelihood7b),
                                        .D3(MLikelihood7c),
                                        .S0(M[0]),
                                        .S1(M[1]),
                                        .O(MLikelihood7)
                                       );

Mux_4_1   MLikelihood_Mux8 [47:0](.D0(48'h0000_0000_0000),
                                  .D1(MLikelihood8a),
                                        .D2(MLikelihood8b),
                                        .D3(MLikelihood8c),
                                        .S0(M[0]),
                                        .S1(M[1]),
                                        .O(MLikelihood8)
```

```
                                    );

Mux_4_1   MLikelihood_Mux9 [47:0](.D0(48'h0000_0000_0000),
                                  .D1(MLikelihood9a),
                                      .D2(MLikelihood9b),
                                      .D3(MLikelihood9c),
                                      .S0(M[0]),
                                      .S1(M[1]),
                                      .O(MLikelihood9)
                                     );

Mux_4_1  MLikelihood_Mux10 [47:0](.D0(48'h0000_0000_0000),
                                  .D1(MLikelihood10a),
                                      .D2(MLikelihood10b),
                                      .D3(MLikelihood10c),
                                      .S0(M[0]),
                                      .S1(M[1]),
                                      .O(MLikelihood10)
                                     );

//---------------------------------------------------------------
//         Choose Ams from best data path of path A B C
//---------------------------------------------------------------

Mux_4_1  Am_Mux1 [9:0](.D0(10'b00_0000_0000),
                       .D1(RAm1a),
                           .D2(RAm1b),
                           .D3(RAm1c),
                           .S0(M[0]),
                           .S1(M[1]),
                           .O(RAm1)
                           );

Mux_4_1  Am_Mux2 [9:0](.D0(10'b00_0000_0000),
                       .D1(RAm2a),
                           .D2(RAm2b),
                           .D3(RAm2c),
                           .S0(M[0]),
                           .S1(M[1]),
                           .O(RAm2)
                           );


Mux_4_1  Am_Mux3 [9:0](.D0(10'b00_0000_0000),
                       .D1(RAm3a),
                           .D2(RAm3b),
                           .D3(RAm3c),
                           .S0(M[0]),
                           .S1(M[1]),
                           .O(RAm3)
                           );
```

```
Mux_4_1  Am_Mux4 [9:0](.D0(10'b00_0000_0000),
                       .D1(RAm4a),
                             .D2(RAm4b),
                             .D3(RAm4c),
                             .S0(M[0]),
                             .S1(M[1]),
                             .O(RAm4)
                             );


Mux_4_1  Am_Mux5 [9:0](.D0(10'b00_0000_0000),
                       .D1(RAm5a),
                             .D2(RAm5b),
                             .D3(RAm5c),
                             .S0(M[0]),
                             .S1(M[1]),
                             .O(RAm5)
                             );


Mux_4_1  Am_Mux6 [9:0](.D0(10'b00_0000_0000),
                       .D1(RAm6a),
                             .D2(RAm6b),
                             .D3(RAm6c),
                             .S0(M[0]),
                             .S1(M[1]),
                             .O(RAm6)
                             );


Mux_4_1  Am_Mux7 [9:0](.D0(10'b00_0000_0000),
                       .D1(RAm7a),
                             .D2(RAm7b),
                             .D3(RAm7c),
                             .S0(M[0]),
                             .S1(M[1]),
                             .O(RAm7)
                             );


Mux_4_1  Am_Mux8 [9:0](.D0(10'b00_0000_0000),
                       .D1(RAm8a),
                             .D2(RAm8b),
                             .D3(RAm8c),
                             .S0(M[0]),
                             .S1(M[1]),
                             .O(RAm8)
                             );


Mux_4_1  Am_Mux9 [9:0](.D0(10'b00_0000_0000),
                       .D1(RAm9a),
                             .D2(RAm9b),
```

```
                              .D3(RAm9c),
                              .S0(M[0]),
                              .S1(M[1]),
                              .O(RAm9)
                              );


Mux_4_1 Am_Mux10 [9:0](.D0(10'b00_0000_0000),
                       .D1(RAm10a),
                              .D2(RAm10b),
                              .D3(RAm10c),
                              .S0(M[0]),
                              .S1(M[1]),
                              .O(RAm10)
                              );

//---------------------------------------------------------------
//         Choose Best Am from Cluster 1 - 10
//---------------------------------------------------------------

Mux_16_1 Am_Mux [9:0](.D1(RAm1),
                      .D2(RAm2),
                              .D3(RAm3),
                              .D4(RAm4),
                              .D5(RAm5),
                              .D6(RAm6),
                              .D7(RAm7),
                              .D8(RAm8),
                              .D9(RAm9),
                              .D10(RAm10),
                              .S0(Clus_Num1[0]),
                              .S1(Clus_Num1[1]),
                              .S2(Clus_Num1[2]),
                              .S3(Clus_Num1[3]),
                              .O(RAm)
                              );




//*****************************************************
//                Pipeline Regs Operation
//*****************************************************

always @ (posedge CLK_3_3_1)
  begin
         MLikelihood1a  <= Likelihood1a;
         MLikelihood2a  <= Likelihood2a;
         MLikelihood3a  <= Likelihood3a;
         MLikelihood4a  <= Likelihood4a;
         MLikelihood5a  <= Likelihood5a;
```

```
MLikelihood6a  <= Likelihood6a;
MLikelihood7a  <= Likelihood7a;
MLikelihood8a  <= Likelihood8a;
MLikelihood9a  <= Likelihood9a;
MLikelihood10a <= Likelihood10a;

MLikelihood1b  <= Likelihood1b;
MLikelihood2b  <= Likelihood2b;
MLikelihood3b  <= Likelihood3b;
MLikelihood4b  <= Likelihood4b;
MLikelihood5b  <= Likelihood5b;
MLikelihood6b  <= Likelihood6b;
MLikelihood7b  <= Likelihood7b;
MLikelihood8b  <= Likelihood8b;
MLikelihood9b  <= Likelihood9b;
MLikelihood10b <= Likelihood10b;

MLikelihood1c  <= Likelihood1c;
MLikelihood2c  <= Likelihood2c;
MLikelihood3c  <= Likelihood3c;
MLikelihood4c  <= Likelihood4c;
MLikelihood5c  <= Likelihood5c;
MLikelihood6c  <= Likelihood6c;
MLikelihood7c  <= Likelihood7c;
MLikelihood8c  <= Likelihood8c;
MLikelihood9c  <= Likelihood9c;
MLikelihood10c <= Likelihood10c;

RAm1a          <= Am1a;
RAm2a          <= Am2a;
RAm3a          <= Am3a;
RAm4a          <= Am4a;
RAm5a          <= Am5a;
RAm6a          <= Am6a;
RAm7a          <= Am7a;
RAm8a          <= Am8a;
RAm9a          <= Am9a;
RAm10a         <= Am10a;

RAm1b          <= Am1b;
RAm2b          <= Am2b;
RAm3b          <= Am3b;
RAm4b          <= Am4b;
RAm5b          <= Am5b;
RAm6b          <= Am6b;
RAm7b          <= Am7b;
RAm8b          <= Am8b;
RAm9b          <= Am9b;
RAm10b         <= Am10b;

RAm1c          <= Am1c;
RAm2c          <= Am2c;
RAm3c          <= Am3c;
```

```
        RAm4c          <= Am4c;
        RAm5c          <= Am5c;
        RAm6c          <= Am6c;
        RAm7c          <= Am7c;
        RAm8c          <= Am8c;
        RAm9c          <= Am9c;
        RAm10c         <= Am10c;

  end

always @ (posedge CLK_3_3_2)
  begin
     Max_Likelihood  <=  Champion_Likelihood;
     Clus_Num1       <=  Champion_Clus_Num;
     M               <=  Champion_M;
  end

always @ (posedge CLK_3_3_3)
  begin
      RBRML1  <= BRMLikelihood1;
      RBRML2  <= BRMLikelihood2;
      RBRML3  <= BRMLikelihood3;
      RBRML4  <= BRMLikelihood4;
      RBRML5  <= BRMLikelihood5;
      RBRML6  <= BRMLikelihood6;
      RBRML7  <= BRMLikelihood7;
      RBRML8  <= BRMLikelihood8;
      RBRML9  <= BRMLikelihood9;
      RBRML10 <= BRMLikelihood10;

      Addr    <= RAm;
  end

always @ (posedge CLK_3_3_7)
  begin
      Scale       <= Inv_Amin;
      Likelihood2 <=  Rival_Likelihood;
      Clus_Num2   <=  Rival_Clus_Num;
  end


//*********************************************************
//      Champion finding tree
//*********************************************************

//-----------------------------------------------
//               1st   floor
//-----------------------------------------------

// [ M , Clus# ] = [ 1 , 1 ] , [ 1 , 2 ]

COMSW1 COMSW1a (  .MLikelihood1(MLikelihood1a),
                  .MLikelihood2(MLikelihood2a),
```

```
                    .Clus_Num1(4'b0001),
                    .Clus_Num2(4'b0010),
                    .M1(2'b01),
                    .M2(2'b01),
                    .Out1(A1[47:0]),
                    .Out2(A1[51:48]),
                    .Out3(A1[53:52]) );

// [ M , Clus# ] = [ 1 , 3 ] , [ 1 , 4 ]

COMSW1 COMSW2a (   .MLikelihood1(MLikelihood3a),
                    .MLikelihood2(MLikelihood4a),
                    .Clus_Num1(4'b0011),
                    .Clus_Num2(4'b0100),
                    .M1(2'b01),
                    .M2(2'b01),
                    .Out1(A2[47:0]),
                    .Out2(A2[51:48]),
                    .Out3(A2[53:52]) );

// [ M , Clus# ] = [ 1 , 5 ] , [ 1 , 6 ]

COMSW1 COMSW3 (.MLikelihood1(MLikelihood5a),
                .MLikelihood2(MLikelihood6a),
                .Clus_Num1(4'b0101),
                  .Clus_Num2(4'b0110),
                  .M1(2'b01),
                  .M2(2'b01),
                  .Out1(A3[47:0]),
                  .Out2(A3[51:48]),
                  .Out3(A3[53:52]) );

// [ M , Clus# ] = [ 1 , 7 ] , [ 1 , 8 ]

COMSW1 COMSW4 (.MLikelihood1(MLikelihood7a),
                .MLikelihood2(MLikelihood8a),
                .Clus_Num1(4'b0111),
                  .Clus_Num2(4'b1000),
                  .M1(2'b01),
                  .M2(2'b01),
                  .Out1(A4[47:0]),
                  .Out2(A4[51:48]),
                  .Out3(A4[53:52]) );

// [ M , Clus# ] = [ 1 , 9 ] , [ 1 , 10 ]

COMSW1 COMSW5 (.MLikelihood1(MLikelihood9a),
                .MLikelihood2(MLikelihood10a),
                .Clus_Num1(4'b1001),
                  .Clus_Num2(4'b1010),
                  .M1(2'b01),
                  .M2(2'b01),
                  .Out1(A5[47:0]),
```

```
                .Out2(A5[51:48]),
                .Out3(A5[53:52]) );

// [ M , Clus# ] = [ 2 , 1 ] , [ 2 , 2 ]

COMSW1 COMSW6 (.MLikelihood1(MLikelihood1b),
               .MLikelihood2(MLikelihood2b),
               .Clus_Num1(4'b0001),
                 .Clus_Num2(4'b0010),
                 .M1(2'b10),
                 .M2(2'b10),
                 .Out1(A6[47:0]),
                 .Out2(A6[51:48]),
                 .Out3(A6[53:52]) );

// [ M , Clus# ] = [ 2 , 3 ] , [ 2 , 4 ]

COMSW1 COMSW7 (.MLikelihood1(MLikelihood3b),
               .MLikelihood2(MLikelihood4b),
               .Clus_Num1(4'b0011),
                 .Clus_Num2(4'b0100),
                 .M1(2'b10),
                 .M2(2'b10),
                 .Out1(A7[47:0]),
                 .Out2(A7[51:48]),
                 .Out3(A7[53:52]) );

// [ M , Clus# ] = [ 2 , 5 ] , [ 2 , 6 ]

COMSW1 COMSW8 (.MLikelihood1(MLikelihood5b),
               .MLikelihood2(MLikelihood6b),
               .Clus_Num1(4'b0101),
                 .Clus_Num2(4'b0110),
                 .M1(2'b10),
                 .M2(2'b10),
                 .Out1(A8[47:0]),
                 .Out2(A8[51:48]),
                 .Out3(A8[53:52]) );

// [ M , Clus# ] = [ 2 , 7 ] , [ 2 , 8 ]

COMSW1 COMSW9 (.MLikelihood1(MLikelihood7b),
               .MLikelihood2(MLikelihood8b),
               .Clus_Num1(4'b0111),
                 .Clus_Num2(4'b1000),
                 .M1(2'b10),
                 .M2(2'b10),
                 .Out1(A9[47:0]),
                 .Out2(A9[51:48]),
                 .Out3(A9[53:52]) );

// [ M , Clus# ] = [ 2 , 9 ] , [ 2 , 10 ]
```

```
COMSW1 COMSW10 (.MLikelihood1(MLikelihood9b),
                .MLikelihood2(MLikelihood10b),
                .Clus_Num1(4'b1001),
                  .Clus_Num2(4'b1010),
                  .M1(2'b10),
                  .M2(2'b10),
                  .Out1(A10[47:0]),
                  .Out2(A10[51:48]),
                  .Out3(A10[53:52]) );

// [ M , Clus# ] = [ 3 , 1 ] , [ 3 , 2 ]

COMSW1 COMSW11 (.MLikelihood1(MLikelihood1c),
                .MLikelihood2(MLikelihood2c),
                .Clus_Num1(4'b0001),
                  .Clus_Num2(4'b0010),
                  .M1(2'b11),
                  .M2(2'b11),
                  .Out1(A11[47:0]),
                  .Out2(A11[51:48]),
                  .Out3(A11[53:52]) );

// [ M , Clus# ] = [ 3 , 3 ] , [ 3 , 4 ]

COMSW1 COMSW12 (.MLikelihood1(MLikelihood3c),
                .MLikelihood2(MLikelihood4c),
                .Clus_Num1(4'b0011),
                  .Clus_Num2(4'b0100),
                  .M1(2'b11),
                  .M2(2'b11),
                  .Out1(A12[47:0]),
                  .Out2(A12[51:48]),
                  .Out3(A12[53:52]) );

// [ M , Clus# ] = [ 3 , 5 ] , [ 3 , 6 ]

COMSW1 COMSW13 (.MLikelihood1(MLikelihood5c),
                .MLikelihood2(MLikelihood6c),
                .Clus_Num1(4'b0101),
                  .Clus_Num2(4'b0110),
                  .M1(2'b11),
                  .M2(2'b11),
                  .Out1(A13[47:0]),
                  .Out2(A13[51:48]),
                  .Out3(A13[53:52]) );

// [ M , Clus# ] = [ 3 , 7 ] , [ 3 , 8 ]

COMSW1 COMSW14 (.MLikelihood1(MLikelihood7c),
                .MLikelihood2(MLikelihood8c),
                .Clus_Num1(4'b0111),
                  .Clus_Num2(4'b1000),
                  .M1(2'b11),
```

```
                  .M2(2'b11),
                  .Out1(A14[47:0]),
                  .Out2(A14[51:48]),
                  .Out3(A14[53:52]) );

// [ M , Clus# ] = [ 3 , 9 ] , [ 3 , 10 ]

COMSW1 COMSW15 (.MLikelihood1(MLikelihood9c),
                .MLikelihood2(MLikelihood10c),
                .Clus_Num1(4'b1001),
                  .Clus_Num2(4'b1010),
                  .M1(2'b11),
                  .M2(2'b11),
                  .Out1(B8[47:0]),
                  .Out2(B8[51:48]),
                  .Out3(B8[53:52]) );

//-----------------------------------------------
//               2nd   floor
//-----------------------------------------------


COMSW1 COMSW16 (.MLikelihood1(A1[47:0]),
                .MLikelihood2(A2[47:0]),
                .Clus_Num1(A1[51:48]),
                  .Clus_Num2(A2[51:48]),
                  .M1(A1[53:52]),
                  .M2(A2[53:52]),
                  .Out1(B1[47:0]),
                  .Out2(B1[51:48]),
                  .Out3(B1[53:52]) );

COMSW1 COMSW17 (.MLikelihood1(A3[47:0]),
                .MLikelihood2(A4[47:0]),
                .Clus_Num1(A3[51:48]),
                  .Clus_Num2(A4[51:48]),
                  .M1(A3[53:52]),
                  .M2(A4[53:52]),
                  .Out1(B2[47:0]),
                  .Out2(B2[51:48]),
                  .Out3(B2[53:52]) );


COMSW1 COMSW18 (.MLikelihood1(A5[47:0]),
                .MLikelihood2(A6[47:0]),
                .Clus_Num1(A5[51:48]),
                  .Clus_Num2(A6[51:48]),
                  .M1(A5[53:52]),
                  .M2(A6[53:52]),
                  .Out1(B3[47:0]),
                  .Out2(B3[51:48]),
                  .Out3(B3[53:52]) );
```

```
COMSW1 COMSW19 (.MLikelihood1(A7[47:0]),
                .MLikelihood2(A8[47:0]),
                .Clus_Num1(A7[51:48]),
                  .Clus_Num2(A8[51:48]),
                  .M1(A7[53:52]),
                  .M2(A8[53:52]),
                  .Out1(B4[47:0]),
                  .Out2(B4[51:48]),
                  .Out3(B4[53:52]) );


COMSW1 COMSW20 (.MLikelihood1(A9[47:0]),
                .MLikelihood2(A10[47:0]),
                .Clus_Num1(A9[51:48]),
                  .Clus_Num2(A10[51:48]),
                  .M1(A9[53:52]),
                  .M2(A10[53:52]),
                  .Out1(B5[47:0]),
                  .Out2(B5[51:48]),
                  .Out3(B5[53:52]) );


COMSW1 COMSW21 (.MLikelihood1(A11[47:0]),
                .MLikelihood2(A12[47:0]),
                .Clus_Num1(A11[51:48]),
                  .Clus_Num2(A12[51:48]),
                  .M1(A11[53:52]),
                  .M2(A12[53:52]),
                  .Out1(B6[47:0]),
                  .Out2(B6[51:48]),
                  .Out3(B6[53:52]) );


COMSW1 COMSW22 (.MLikelihood1(A13[47:0]),
                .MLikelihood2(A14[47:0]),
                .Clus_Num1(A13[51:48]),
                  .Clus_Num2(A14[51:48]),
                  .M1(A13[53:52]),
                  .M2(A14[53:52]),
                  .Out1(B7[47:0]),
                  .Out2(B7[51:48]),
                  .Out3(B7[53:52]) );

//-------------------------------------------
//               3rd   floor
//-------------------------------------------


COMSW1 COMSW23 (.MLikelihood1(B1[47:0]),
                .MLikelihood2(B2[47:0]),
                .Clus_Num1(B1[51:48]),
                  .Clus_Num2(B2[51:48]),
```

```
                  .M1(B1[53:52]),
                  .M2(B2[53:52]),
                  .Out1(C1[47:0]),
                  .Out2(C1[51:48]),
                  .Out3(C1[53:52]) );

COMSW1 COMSW24 (.MLikelihood1(B3[47:0]),
                .MLikelihood2(B4[47:0]),
                .Clus_Num1(B3[51:48]),
                  .Clus_Num2(B4[51:48]),
                  .M1(B3[53:52]),
                  .M2(B4[53:52]),
                  .Out1(C2[47:0]),
                  .Out2(C2[51:48]),
                  .Out3(C2[53:52]) );


COMSW1 COMSW25 (.MLikelihood1(B5[47:0]),
                .MLikelihood2(B6[47:0]),
                .Clus_Num1(B5[51:48]),
                  .Clus_Num2(B6[51:48]),
                  .M1(B5[53:52]),
                  .M2(B6[53:52]),
                  .Out1(C3[47:0]),
                  .Out2(C3[51:48]),
                  .Out3(C3[53:52]) );


COMSW1 COMSW26 (.MLikelihood1(B7[47:0]),
                .MLikelihood2(B8[47:0]),
                .Clus_Num1(B7[51:48]),
                  .Clus_Num2(B8[51:48]),
                  .M1(B7[53:52]),
                  .M2(B8[53:52]),
                  .Out1(C4[47:0]),
                  .Out2(C4[51:48]),
                  .Out3(C4[53:52]) );

//-------------------------------------------
//               4th   floor
//-------------------------------------------


COMSW1 COMSW27 (.MLikelihood1(C1[47:0]),
                .MLikelihood2(C2[47:0]),
                .Clus_Num1(C1[51:48]),
                  .Clus_Num2(C2[51:48]),
                  .M1(C1[53:52]),
                  .M2(C2[53:52]),
                  .Out1(D1[47:0]),
                  .Out2(D1[51:48]),
                  .Out3(D1[53:52]) );
```

```
COMSW1 COMSW28 (.MLikelihood1(C3[47:0]),
                .MLikelihood2(C4[47:0]),
                .Clus_Num1(C3[51:48]),
                  .Clus_Num2(C4[51:48]),
                  .M1(C3[53:52]),
                  .M2(C4[53:52]),
                  .Out1(D2[47:0]),
                  .Out2(D2[51:48]),
                  .Out3(D2[53:52]) );

//-----------------------------------------------
//               5th   floor : Champion Output Node
//-----------------------------------------------


COMSW1 COMSW29 (.MLikelihood1(D1[47:0]),
                .MLikelihood2(D2[47:0]),
                .Clus_Num1(D1[51:48]),
                  .Clus_Num2(D2[51:48]),
                  .M1(D1[53:52]),
                  .M2(D2[53:52]),
                  .Out1(Champion_Likelihood),
                  .Out2(Champion_Clus_Num),
                  .Out3(Champion_M) );




//*********************************************************
//      Rival finding tree
//*********************************************************



//-------------------------------------------------------
//                        1st  floor
//-------------------------------------------------------

// [ M , Clus# 1,2]

COMSW2   RCOMSW1 (.RBRML1(RBRML1),
                  .RBRML2(RBRML2),
                  .Clus_Num1(4'b0001),
                  .Clus_Num2(4'b0010),
                  .Out1(RB1[47:0]),
                  .Out2(RB1[51:48])  );

// [ M , Clus# 3,4]

COMSW2   RCOMSW2 (.RBRML1(RBRML3),
                  .RBRML2(RBRML4),
                  .Clus_Num1(4'b0011),
                  .Clus_Num2(4'b0100),
```

```
                .Out1(RB2[47:0]),
                .Out2(RB2[51:48])  );

// [ M , Clus# 5,6]

COMSW2   RCOMSW3 (.RBRML1(RBRML5),
                .RBRML2(RBRML6),
                .Clus_Num1(4'b0101),
                .Clus_Num2(4'b0110),
                .Out1(RB3[47:0]),
                .Out2(RB3[51:48])  );

// [ M , Clus# 7,8]

COMSW2   RCOMSW4 (.RBRML1(RBRML7),
                .RBRML2(RBRML8),
                .Clus_Num1(4'b0111),
                .Clus_Num2(4'b1000),
                .Out1(RB4[47:0]),
                .Out2(RB4[51:48])  );

// [ M , Clus# 9,10]

COMSW2   RCOMSW5 (.RBRML1(RBRML9),
                .RBRML2(RBRML10),
                .Clus_Num1(4'b1001),
                .Clus_Num2(4'b1010),
                .Out1(RD2[47:0]),
                .Out2(RD2[51:48])  );


//-------------------------------------------------------
//                       2nd  floor
//-------------------------------------------------------


COMSW2   RCOMSW6 (.RBRML1(RB1[47:0]),
                .RBRML2(RB2[47:0]),
                .Clus_Num1(RB1[51:48]),
                .Clus_Num2(RB2[51:48]),
                .Out1(RC1[47:0]),
                .Out2(RC1[51:48])  );

COMSW2   RCOMSW7 (.RBRML1(RB3[47:0]),
                .RBRML2(RB4[47:0]),
                .Clus_Num1(RB3[51:48]),
                .Clus_Num2(RB4[51:48]),
                .Out1(RC2[47:0]),
                .Out2(RC2[51:48])  );


//-------------------------------------------------------
//                       3rd  floor
```

```
//-------------------------------------------------------


COMSW2   RCOMSW8 (.RBRML1(RC1[47:0]),
                  .RBRML2(RC2[47:0]),
                  .Clus_Num1(RC1[51:48]),
                  .Clus_Num2(RC2[51:48]),
                  .Out1(RD1[47:0]),
                  .Out2(RD1[51:48])  );

//-------------------------------------------------------
//                        4th  floor  Rival output node
//-------------------------------------------------------

COMSW2   RCOMSW9 (.RBRML1(RD1[47:0]),
                  .RBRML2(RD2[47:0]),
                  .Clus_Num1(RD1[51:48]),
                  .Clus_Num2(RD2[51:48]),
                  .Out1(Rival_Likelihood),
                  .Out2(Rival_Clus_Num)  );

//************************************************************************
//   LUT Input: [2^(-9) 2] Output: 1/Amin; Resolution: 1K,(2^(-9))
//                          World length: 16, U2/14
//************************************************************************

RAMB16_S18 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
.INIT_3F(256'h2000200820102018202020282030203820412049205120592061206920722079a),
.INIT_3E(256'h2082208a2093209b20a320ac20b420bc20c520cd20d520de20e620ef20f72100),
.INIT_3D(256'h2108211121192122212a2133213c2144214d2156215e21672170217921818218a),
.INIT_3C(256'h2193219c21a521ad21b621bf21c821d121da21e321ec21f521fe220722102219),
.INIT_3B(256'h2222222b2234223e22472250225922622262275227e22872291229a22a322ad),
.INIT_3A(256'h22b622c022c922d322dc22e622ef22f92302230c2315231f23292332233c2346),
.INIT_39(256'h234f23592363236d23772380238a2394239e23a823b223bc23c623d023da23e4),
.INIT_38(256'h23ee23f82402240c24172421242b2435243f244a2454245e24692473247d2488),
.INIT_37(256'h2492249d24a724b224bc24c724d124dc24e724f124fc25072511251c25272532),
.INIT_36(256'h253d25472552255d25682573257e25892594259f25aa25b525c025cb25d725e2),
.INIT_35(256'h25ed25f82604260f261a26262631263c26482653265f266a26762681268d2699),
.INIT_34(256'h26a426b026bc26c726d326df26eb26f72702270e271a27262732273e274a2756),
.INIT_33(256'h2762276f277b27872793279f27ac27b827c427d127dd27ea27f62803280f281c),
.INIT_32(256'h282828352841284e285b286828742881288e289b28a828b528c228cf28dc28e9),
.INIT_31(256'h28f629032910291d292a2938294529522960296d297a2988299529a329b129be),
.INIT_30(256'h29cc29d929e729f52a032a102a1e2a2c2a3a2a482a562a642a722a802a8e2a9c),
.INIT_2F(256'h2aab2ab92ac72ad62ae42af22b012b0f2b1e2b2c2b3b2b492b582b672b752b84),
.INIT_2E(256'h2b932ba22bb12bc02bcf2bde2bed2bfc2c0b2c1a2c292c392c482c572c672c76),
.INIT_2D(256'h2c862c952ca52cb42cc42cd42ce32cf32d032d132d232d322d422d532d632d73),
.INIT_2C(256'h2d832d932da32db42dc42dd42de52df52e062e162e272e382e482e592e6a2e7b),
.INIT_2B(256'h2e8c2e9d2eae2ebf2ed02ee12ef22f032f152f262f372f492f5a2f6c2f7d2f8f),
.INIT_2A(256'h2fa12fb22fc42fd62fe82ffa300c301e303030423055306730793080309e30b0),
```

```
.INIT_29(256'h30c330d630e830fb310e3121313431463159316d3180319331a631b931cd31e0),
.INIT_28(256'h31f43207321b322e32423256326a327d329132a532b932ce32e232f6330a331f),
.INIT_27(256'h33333348335c33713386339a33af33c433d933ee34033418342e34433458346e),
.INIT_26(256'h3483349934af34c434da34f03506351c35323548355e3575358b35a135b835ce),
.INIT_25(256'h35e535fc3613362936403657366f3686369d36b436cc36e336fb3713372a3742),
.INIT_24(256'h375a3772378a37a237ba37d337eb3804381c3835384d3866387f389838b138ca),
.INIT_23(256'h38e438fd3916393039493963397d399739b139cb39e539ff3a193a343a4e3a69),
.INIT_22(256'h3a843a9e3ab93ad43aef3b0b3b263b413b5d3b783b943bb03bcc3be83c043c20),
.INIT_21(256'h3c3c3c593c753c923cae3ccb3ce83d053d223d403d5d3d7a3d983db63dd43df1),
.INIT_20(256'h3e103e2e3e4c3e6a3e893ea73ec63ee53f043f233f423f623f813fa13fc03fe0),
.INIT_1F(256'h400040204040061408140a240c240e34104412541464168418941ab41cd41ee),
.INIT_1E(256'h4211423342554277429a42bd42e04303432643494364394304 3b443d843fc4420),
.INIT_1D(256'h44444469448d44b244d744fc45214547456c459245b845de4604462b46514678),
.INIT_1C(256'h469f46c646ed4715473c4764478c47b447dc4805482d4856487f48a848d148fb),
.INIT_1B(256'h4925494e497949a349cd49f84a234a4e4a794aa44ad04afc4b284b544b814bad),
.INIT_1A(256'h4bda4c074c344c624c904cbd4cec4d1a4d484d774da64dd54e054e354e644e95),
.INIT_19(256'h4ec54ef64f264f574f894fba4fec501e5050508350b650e9511c514f518351b7),
.INIT_18(256'h51ec52205255528a52bf52f5532b5361539853ce5405543d547454ac54e4551d),
.INIT_17(256'h5555558e55c85601563b567656b056eb57265762579d57da58165853589058cd),
.INIT_16(256'h590b5949598859c65a065a455a855ac55b065b475b885bca5c0c5c4e5c915cd4),
.INIT_15(256'h5d175d5b5d9f5de45e295e6f5eb55efb5f415f895fd06018606060a960f2613c),
.INIT_14(256'h618661d1621c626762b362ff634c639963e76435648464d36523657365c46615),
.INIT_13(256'h666666b9670b675e67b26807685b68b16907695d69b46a0c6a646abc6b166b70),
.INIT_12(256'h6bca6c256c816cdd6d3a6d986df66e546eb46f146f756fd67038709b70fe7162),
.INIT_11(256'h71c7722d729372fa736173ca7433749d7507757375df764c76ba772877977808),
.INIT_10(256'h787878ea795d79d07a457aba7b307ba77c1f7c987d127d8c7e087e847f027f80),
.INIT_0F(256'h800080818102818582088 28d83128399842184aa853485bf864c86d9876887f8),
.INIT_0E(256'h8889891b89ae8a438ad98b708c098ca38d3e8dda8e788f178fb8905a90fe91a3),
.INIT_0D(256'h924992f1939b944694f295a09650970197b49869991f99d79a919b4c9c0a9cc9),
.INIT_0C(256'h9d8a9e4d9f119fd8a0a1a16ba238a306a3d7a4aaa57fa656a72fa80ba8e8a9c8),
.INIT_0B(256'haaabab8fac77ad60ae4caf3bb02cb120b216b30fb40bb50ab60bb710b817b921),
.INIT_0A(256'hba2fbb3fbc52bd69be83bfa0c0c1c1e5c30cc437c566c698c7cec908ca46cb87),
.INIT_09(256'hcccdce17cf64d0b7d20dd368d4c7d62cd794d902da74dbebdd68dee9e070e1fc),
.INIT_08(256'he38ee526e6c3e866ea0febbeed73ef2ff0f1f2baf48af660f83efa23fc10fe04),
.INIT_07(256'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff),
.INIT_06(256'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff),
.INIT_05(256'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff),
.INIT_04(256'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff),
.INIT_03(256'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff),
.INIT_02(256'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff),
.INIT_01(256'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff),
.INIT_00(256'hffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff)
)

RAM16_LUT_Inv_Aden(  .DO(Inv_Amin),
                     .DOP(),
                     .ADDR(Addr),
                     .CLK(CLK_SYS),
                     .DI(16'hFFFF),
                     .DIP(2'b11),
                     .EN(EN),
                     .SSR(1'b0),
                     .WE(1'b0)
```

```
                );

endmodule


////////////////////////////////////////////////////////////////////////////////
//
//   Comapre two entires on Mlikelihood, and switch the data path to the little
//   one.
//
//
////////////////////////////////////////////////////////////////////////////////

module COMSW1(input  [47:0] MLikelihood1,
              input  [47:0] MLikelihood2,
              input  [3:0]  Clus_Num1,
              input  [3:0]  Clus_Num2,
              input  [1:0]  M1,
              input  [1:0]  M2,
              output [47:0] Out1,
              output [3:0]  Out2,
              output [1:0]  Out3 );


//*****************************************************
//               internal signals
//*****************************************************

wire    SW;  // switch control


//*****************************************************
//           Likelihood comparison
//*****************************************************

assign SW = (MLikelihood1 < MLikelihood2) ? 0 : 1;

//*****************************************************
//           data path switch
//*****************************************************

MUX_2_1 Likelihood_SW [47:0] (.D0(MLikelihood1),
                              .D1(MLikelihood2),
                              .S0(SW),
                              .O(Out1)  );

MUX_2_1 Clus_Num_SW   [3:0]  (.D0(Clus_Num1),
                              .D1(Clus_Num2),
                              .S0(SW),
                              .O(Out2)  );

MUX_2_1 M_SW          [1:0]  (.D0(M1),
                              .D1(M2),
```

```
                              .S0(SW),
                              .O(Out3)  );


endmodule


`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
//   Comapre two entires on Mlikelihood, and switch the data path to the little
//   one.
//
//
////////////////////////////////////////////////////////////////////////////////


module COMSW2(input  [47:0] RBRML1,
              input  [47:0] RBRML2,
              input  [3:0]  Clus_Num1,
              input  [3:0]  Clus_Num2,
              output [47:0] Out1,
              output [3:0]  Out2  );

//*********************************************
//           internal signals
//*********************************************

wire    SW;  // switch control

//*********************************************
//          Likelihood comparison
//*********************************************

assign SW = (RBRML1 < RBRML2) ? 0 : 1;


//*********************************************
//          data path switch
//*********************************************

MUX_2_1 RBRML_SW    [47:0] (.D0(RBRML1),
                            .D1(RBRML2),
                            .S0(SW),
                            .O(Out1)  );

MUX_2_1 Clus_Num_SW [3:0]  (.D0(Clus_Num1),
                            .D1(Clus_Num2),
                            .S0(SW),
                            .O(Out2)  );

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
//  Local filter output switch logic and weight generation
//  Weight generation use relative likelihood: Champion likelihood = 1,
//     Rival likelihood = P(rival) / P(cham)
//
//////////////////////////////////////////////////////////////////////////////////



module filter_switch_and_weight(          input CLK_SYS,
                                          input                 CLK_3_3_4,
                                          input                 CLK_3_3_5,
                                          input                 CLK_3_3_6,
                                          input                 CLK_3_3_7,
                                          input                 EN,
                                          input         [1:0]   M,
                                          input         [3:0]   Clus_Num1,
                                          input         [3:0]   Clus_Num2,
                                          input         [47:0]  Log_Likelihood2,
                                          input signed  [15:0]  Log_Pcm1,
                                          input signed  [15:0]  Log_Pcm2,
                                          input signed  [15:0]  Log_Pcm3,
                                          input signed  [15:0]  Log_Pcm4,
                                          input signed  [15:0]  Log_Pcm5,
                                          input signed  [15:0]  Log_Pcm6,
                                          input signed  [15:0]  Log_Pcm7,
                                          input signed  [15:0]  Log_Pcm8,
                                          input signed  [15:0]  Log_Pcm9,
                                          input signed  [15:0]  Log_Pcm10,
                                          input         [17:0]  Xt_Beta1a,
                                          input         [17:0]  Xt_Beta2a,
                                          input         [17:0]  Xt_Beta3a,
                                          input         [17:0]  Xt_Beta4a,
                                          input         [17:0]  Xt_Beta5a,
                                          input         [17:0]  Xt_Beta6a,
                                          input         [17:0]  Xt_Beta7a,
                                          input         [17:0]  Xt_Beta8a,
                                          input         [17:0]  Xt_Beta9a,
                                          input         [17:0]  Xt_Beta10a,
                                          input         [17:0]  Xt_Beta1b,
                                          input         [17:0]  Xt_Beta2b,
                                          input         [17:0]  Xt_Beta3b,
                                          input         [17:0]  Xt_Beta4b,
                                          input         [17:0]  Xt_Beta5b,
                                          input         [17:0]  Xt_Beta6b,
                                          input         [17:0]  Xt_Beta7b,
                                          input         [17:0]  Xt_Beta8b,
                                          input         [17:0]  Xt_Beta9b,
```

```
                                        input         [17:0] Xt_Beta10b,
                                        input         [17:0] Xt_Beta1c,
                                        input         [17:0] Xt_Beta2c,
                                        input         [17:0] Xt_Beta3c,
                                        input         [17:0] Xt_Beta4c,
                                        input         [17:0] Xt_Beta5c,
                                        input         [17:0] Xt_Beta6c,
                                        input         [17:0] Xt_Beta7c,
                                        input         [17:0] Xt_Beta8c,
                                        input         [17:0] Xt_Beta9c,
                                        input         [17:0] Xt_Beta10c,
                                        input         [17:0] Miu_Beta_K2D1,
                                        input         [17:0] Miu_Beta_K2D2,
                                        input         [17:0] Miu_Beta_K2D3,
                                        input         [17:0] Miu_Beta_K2D4,
                                        input         [17:0] Miu_Beta_K2D5,
                                        input         [17:0] Miu_Beta_K2D6,
                                        input         [17:0] Miu_Beta_K2D7,
                                        input         [17:0] Miu_Beta_K2D8,
                                        input         [17:0] Miu_Beta_K2D9,
                                        input         [17:0] Miu_Beta_K2D10,
                                        output reg    [17:0] Xt_Beta_F1 =
                                                             18'h0000,
                                        output reg    [17:0] Xt_Beta_F2 =
                                                             18'h0000,
                                        output reg    [17:0] Miu_Beta_K2D_F1 =
                                                             18'h0000,
                                        output reg    [17:0] Miu_Beta_K2D_F2 =
                                                             18'h0000,
                                        output reg    [8:0] Weight1 = 9'h000,
                                        output reg    [8:0] Weight2 = 9'h000
                                       );

//*****************************************************
//                  internal signals
//*****************************************************

//(1) for weight generation block

 wire signed [15:0] Log_PcmC, Log_PcmR;
 reg  signed [15:0] A1, A2;
 wire signed [15:0] A;
 wire signed [47:0] Log_Pcm, NLog_Likelihood2;
 wire signed [47:0] RWL;
 wire        [47:0] RWL_Abs;
 wire signed [47:0] RWL_tmp;
 reg         [9:0]  Addr_Exp;
 wire        [15:0] PxcmPcm;
 wire        [9:0]  Addr_tmp;
 reg         [9:0]  Addr_Inv_Sum;
 wire        [15:0] Inv_Sum;
 wire        [18:0] Weight2_tmp;
```

```
//(2) for filer output switch

wire        [17:0]
Xt_Beta_M1, Xt_Beta_M2, Xt_Beta_M3, Xt_Beta_M4,
Xt_Beta_M5,Xt_Beta_M6, Xt_Beta_M7, Xt_Beta_M8, Xt_Beta_M9,
Xt_Beta_M10;

wire        [17:0]
Xt_Beta_Out1, Xt_Beta_Out2;

wire [17:0] Miu_Beta_K2D_Out1, Miu_Beta_K2D_Out2;



//************************************************************
//               Filter Weight Genaration
//************************************************************


//------------------------------------------------------------
//             pipeline  regs operations
//------------------------------------------------------------

always @ (posedge CLK_3_3_4)
   begin
        A1 <= Log_PcmC;
        A2 <= Log_PcmR;
    end

always @ (posedge CLK_3_3_5)
   begin
       case (RWL_Abs[16:7] > 10'h3FF)
          1'b0 : Addr_Exp <= RWL_Abs[16:7];
          1'b1 : Addr_Exp <= 10'b00_0000_0000;
        endcase
    end

always @ (posedge CLK_3_3_6)  Addr_Inv_Sum <= Addr_tmp;

always @ (posedge CLK_3_3_7)
   begin
        Weight1 <= Inv_Sum[8:0];
        Weight2 <=  Weight2_tmp[12:4];
    end

//------------------------------------------------------------
//    Signed / Unsigned # translation and extend & trim
//------------------------------------------------------------

assign NLog_Likelihood2 = (~Log_Likelihood2) + 48'h0000_0000_0001;
```

```
// translate the magnitude of likelihood to the negative value
// likelihood2 is the relative value to the champion cluster,
// the log likelihood of champion is now 0.

// extend the S6/10 to S36/12

assign Log_Pcm  = (A[15] == 0) ? ({30'h0000_0000, A, 2'b00}) :
(~{30'h0000_0000, ((~A) + 16'h0001), 2'b00} + 48'h0000_0000_0001);


// trans RWL to the absolution value to extract the LUT address

assign RWL_Abs = (RWL_tmp[47] == 0) ? (RWL_tmp) :
                 (48'h0000_0000_0000);


//-------------------------------------------------------
//              Adders, Multipliers
//-------------------------------------------------------

 assign RWL = NLog_Likelihood2 + Log_Pcm;
 assign RWL_tmp = RWL + 48'h0000_0001_C000;
 assign A = A2 - A1;  // Pcm2 / Pcm1
 assign Addr_tmp = PxcmPcm[9:0] + 10'b00_0001_0000;
 assign Weight2_tmp = Inv_Sum[8:0] * PxcmPcm[9:0];

//-------------------------------------------------------
//            MUXs
//-------------------------------------------------------

Mux_16_1 Log_PcmC_Mux [15:0](      .D1(Log_Pcm1),
                                   .D2(Log_Pcm2),
                                   .D3(Log_Pcm3),
                                   .D4(Log_Pcm4),
                                   .D5(Log_Pcm5),
                                   .D6(Log_Pcm6),
                                   .D7(Log_Pcm7),
                                   .D8(Log_Pcm8),
                                   .D9(Log_Pcm9),
                                   .D10(Log_Pcm10),
                                   .S0(Clus_Num1[0]),
                                   .S1(Clus_Num1[1]),
                                   .S2(Clus_Num1[2]),
                                   .S3(Clus_Num1[3]),
                                   .O(Log_PcmC)
                                  );

Mux_16_1 Log_PcmR_Mux [15:0](      .D1(Log_Pcm1),
                                   .D2(Log_Pcm2),
                                   .D3(Log_Pcm3),
                                   .D4(Log_Pcm4),
                                   .D5(Log_Pcm5),
                                   .D6(Log_Pcm6),
```

```
                                    .D7(Log_Pcm7),
                                    .D8(Log_Pcm8),
                                    .D9(Log_Pcm9),
                                    .D10(Log_Pcm10),
                                    .S0(Clus_Num2[0]),
                                    .S1(Clus_Num2[1]),
                                    .S2(Clus_Num2[2]),
                                    .S3(Clus_Num2[3]),
                                    .O(Log_PcmR)
                                   );


 //-------------------------------------------------------
 //       LUT:  EXP &  1/Sum
 //-------------------------------------------------------


 //EXP:  Input: [-28 4] Output: EXP(*); Resolution: 1K,(2^(-5))
 //      Output World length: 16, U6/4
 //=====================================================================

RAMB16_S18 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),

.INIT_3F(256'h034f0335031b030302eb02d402be02a80293027f026b02580246023402230212),
.INIT_3E(256'h020201f201e201d401c501b701aa019d019001840178016c01610156014c0141),
.INIT_3D(256'h0137012e0125011c0113010a010200fa00f300eb00e400dd00d600cf00c900c3),
.INIT_3C(256'h00bd00b700b100ac00a700a2009d00980093008f008a00860082007e007a0076),
.INIT_3B(256'h0073006f006c00680065006200 5f005c005900560054005 1004f004c004a0048),
.INIT_3A(256'h004600430041003f003d003b003a00380036003400330031003 0002e002d002b),
.INIT_39(256'h002a00290028002600250024002300220021002000 1f001e001d001c001b001a),
.INIT_38(256'h001a001900180017001700160015001500140013001300120012001100110010),
.INIT_37(256'h0010000f000f000e000e000d000d000c000c000c000b000b000b000a000a000a),
.INIT_36(256'h000900090009000900080008000800080007000700070007000600600060006),
.INIT_35(256'h000600060005000500050005000500050004000400040004000400040004),
.INIT_34(256'h000300030003000300030003000300030003000300030002000200020002),
.INIT_33(256'h000200020002000200020002000200020002000200020001000100010001),
.INIT_32(256'h000100010001000100010001000100010001000100010001000100010001),
.INIT_31(256'h000100010001000100010001000100010001000100010001000100000000),
.INIT_30(256'h0000000000000000000000000000000000000000000000000000000000000000),
.INIT_2F(256'h0000000000000000000000000000000000000000000000000000000000000000)
)

RAM16_Miux1(  .DO(PxcmPcm),
              .DOP(),                       // 2-bit A port parity data output
              .ADDR(Addr_Exp),              // 10-bit A port address input
              .CLK(CLK_SYS),                // 1-bit A port clock input
              .DI(16'hFFFF),                // 16-bit A port data input
              .DIP(2'b11),                  // 2-bit A port parity data input
              .EN(EN),                      // 1-bit A port enable input
```

```
            .SSR(1'b0),                     // 1-bit A port set/reset input
            .WE(1'b0)                       // 1-bit A port write enable input
          );




// 1/Sum: Input: [2^(-4) 64] Output: 1/Sum; Resolution: 1K,(2^(-4))
//  Output World length: 16, U1/8
//================================================================

RAMB16_S18 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
.INIT_3F(256'h0004000400040004000400040004000400040004000400040004000400040004),
.INIT_3E(256'h0004000400040004000400040004000400040004000400040004000400040004),
.INIT_3D(256'h0004000400040004000400040004000400040004000400040004000400040004),
.INIT_3C(256'h0004000400040004000400040004000400040004000400040004000400040004),
.INIT_3B(256'h0004000400040004000400040004000400040004000400040004000400040004),
.INIT_3A(256'h0004000400040004000400040004000400040004000400040004000400040004),
.INIT_39(256'h0004000400040004000400040004000400040004000400040004000400040004),
.INIT_38(256'h0004000400050005000500050005000500050005000500050005000500050005),
.INIT_37(256'h0005000500050005000500050005000500050005000500050005000500050005),
.INIT_36(256'h0005000500050005000500050005000500050005000500050005000500050005),
.INIT_35(256'h0005000500050005000500050005000500050005000500050005000500050005),
.INIT_34(256'h0005000500050005000500050005000500050005000500050005000500050005),
.INIT_33(256'h0005000500050005000500050005000500050005000500050005000500050005),
.INIT_32(256'h0005000500050005000500050005000500050005000500050005000500050005),
.INIT_31(256'h0005000500050005000500050005000500050005000500050005000500050005),
.INIT_30(256'h0005000500050005000500050005000500050005000500050005000500050005),
.INIT_2F(256'h0005000500050005000500050005000500050005000500050005000500050005),
.INIT_2E(256'h0005000500050005000500050005000500060006000600060006000600060006),
.INIT_2D(256'h0006000600060006000600060006000600060006000600060006000600060006),
.INIT_2C(256'h0006000600060006000600060006000600060006000600060006000600060006),
.INIT_2B(256'h0006000600060006000600060006000600060006000600060006000600060006),
.INIT_2A(256'h0006000600060006000600060006000600060006000600060006000600060006),
.INIT_29(256'h0006000600060006000600060006000600060006000600060006000600060006),
.INIT_28(256'h0006000600060006000600060006000600060006000600060006000600060006),
.INIT_27(256'h0006000600060006000600060006000600060006000600070007000700070007),
.INIT_26(256'h0007000700070007000700070007000700070007000700070007000700070007),
.INIT_25(256'h0007000700070007000700070007000700070007000700070007000700070007),
.INIT_24(256'h0007000700070007000700070007000700070007000700070007000700070007),
.INIT_23(256'h0007000700070007000700070007000700070007000700070007000700070007),
.INIT_22(256'h0007000700070007000700070007000700070007000700070007000700080008),
.INIT_21(256'h0008000800080008000800080008000800080008000800080008000800080008),
.INIT_20(256'h0008000800080008000800080008000800080008000800080008000800080008),
.INIT_1F(256'h0008000800080008000800080008000800080008000800080008000800080008),
.INIT_1E(256'h0008000800080008000800080008000800080008000800080008000800080009),
.INIT_1D(256'h0009000900090009000900090009000900090009000900090009000900090009),
.INIT_1C(256'h0009000900090009000900090009000900090009000900090009000900090009),
```

```
.INIT_1B(256'h0009000900090009000900090009000900090009000900090009000900090009),
.INIT_1A(256'h0009000a000a000a000a000a000a000a000a000a000a000a000a000a000a000a),
.INIT_19(256'h000a000a000a000a000a000a000a000a000a000a000a000a000a000a000a000a),
.INIT_18(256'h000a000a000a000a000a000a000a000a000a000a000a000b000b000b000b000b),
.INIT_17(256'h000b000b000b000b000b000b000b000b000b000b000b000b000b000b000b000b),
.INIT_16(256'h000b000b000b000b000b000b000b000b000b000b000b000b000b000c000c000c),
.INIT_15(256'h000c000c000c000c000c000c000c000c000c000c000c000c000c000c000c000c),
.INIT_14(256'h000c000c000c000c000c000c000c000c000c000c000d000d000d000d000d000d),
.INIT_13(256'h000d000d000d000d000d000d000d000d000d000d000d000d000d000d000d000d),
.INIT_12(256'h000d000e000e000e000e000e000e000e000e000e000e000e000e000e000e000e),
.INIT_11(256'h000e000e000e000e000e000e000f000f000f000f000f000f000f000f000f000f),
.INIT_10(256'h000f000f000f000f000f000f000f000f001000100010001000100010001000010),
.INIT_0F(256'h0010001000100010001000100010001000110011001100110011001100110011),
.INIT_0E(256'h0011001100110011001100110012001200120012001200120012001200120012),
.INIT_0D(256'h0012001200120013001300130013001300130013001300130013001300140014),
.INIT_0C(256'h0014001400140014001400140014001400140015001500150015001500150015),
.INIT_0B(256'h0015001500160016001600160016001600160016001600170017001700170017),
.INIT_0A(256'h0017001700180018001800180018001800180019001900190019001900190019),
.INIT_09(256'h001a001a001a001a001a001a001b001b001b001b001b001b001c001c001c001c),
.INIT_08(256'h001c001d001d001d001d001d001e001e001e001e001f001f001f001f00200020),
.INIT_07(256'h0020002000210021002100210022002200220022002300230023002400240024),
.INIT_06(256'h0025002500250026002600260027002700270028002800290029002a002a),
.INIT_05(256'h002b002b002c002c002d002d002e002e002f002f003000300031003100320033),
.INIT_04(256'h0033003400350035003600370037003800390003a003b003b003c003d003e003f),
.INIT_03(256'h0040004100420043004400450047004800490004a004c004d004f005000520054),
.INIT_02(256'h0055005700590005b005d005f006200640066006900 6c006f007200750078007c),
.INIT_01(256'h0080008400890008d00920098009e00a400ab00b200ba00c300cd00d800e400f1),
.INIT_00(256'h010001110125013b01550174019a01c70200024902ab0333040005550800 1000)
)

RAM16_Miux0(  .DO(Inv_Sum),
              .DOP(),                          // 2-bit A port parity data output
              .ADDR(Addr_Inv_Sum),             // 10-bit A port address input
              .CLK(CLK_SYS),                   // 1-bit A port clock input
              .DI(16'hFFFF),                   // 16-bit A port data input
              .DIP(2'b11),                     // 2-bit A port parity data input
              .EN(EN),                         // 1-bit A port enable input
              .SSR(1'b0),                      // 1-bit A port set/reset input
              .WE(1'b0)                        // 1-bit A port write enable input
            );


//*********************************************************
//             Filter output swithcing
//*********************************************************


 //---------------------------------------------------------
 //           Pipeline Regs opesraitons
 //---------------------------------------------------------


always @(posedge CLK_3_3_7)
```

```
  begin
      Xt_Beta_F1      <= Xt_Beta_Out1;
      Xt_Beta_F2      <= Xt_Beta_Out2;
      Miu_Beta_K2D_F1 <= Miu_Beta_K2D_Out1;
      Miu_Beta_K2D_F2 <= Miu_Beta_K2D_Out2;
  end


//-------------------------------------------------------
//         Output Switching -- MUX
//-------------------------------------------------------


Mux_4_1 Xt_Beta_M1_Mux [17:0](.D0(18'h0_0000),
                              .D1(Xt_Beta1a),
                                  .D2(Xt_Beta1b),
                                  .D3(Xt_Beta1c),
                                  .S0(M[0]),
                                  .S1(M[1]),
                                  .O(Xt_Beta_M1)
                                 );

Mux_4_1 Xt_Beta_M2_Mux [17:0](.D0(18'h0_0000),
                              .D1(Xt_Beta2a),
                                  .D2(Xt_Beta2b),
                                  .D3(Xt_Beta2c),
                                  .S0(M[0]),
                                  .S1(M[1]),
                                  .O(Xt_Beta_M2)
                                 );

Mux_4_1 Xt_Beta_M3_Mux [17:0](.D0(18'h0_0000),
                              .D1(Xt_Beta3a),
                                  .D2(Xt_Beta3b),
                                  .D3(Xt_Beta3c),
                                  .S0(M[0]),
                                  .S1(M[1]),
                                  .O(Xt_Beta_M3)
                                 );

Mux_4_1 Xt_Beta_M4_Mux [17:0](.D0(18'h0_0000),
                              .D1(Xt_Beta4a),
                                  .D2(Xt_Beta4b),
                                  .D3(Xt_Beta4c),
                                  .S0(M[0]),
                                  .S1(M[1]),
                                  .O(Xt_Beta_M4)
                                 );

Mux_4_1 Xt_Beta_M5_Mux [17:0](.D0(18'h0_0000),
                              .D1(Xt_Beta5a),
                                  .D2(Xt_Beta5b),
                                  .D3(Xt_Beta5c),
```

```
                                    .S0(M[0]),
                                    .S1(M[1]),
                                    .O(Xt_Beta_M5)
                                   );

Mux_4_1 Xt_Beta_M6_Mux [17:0](.D0(18'h0_0000),
                              .D1(Xt_Beta6a),
                                    .D2(Xt_Beta6b),
                                    .D3(Xt_Beta6c),
                                    .S0(M[0]),
                                    .S1(M[1]),
                                    .O(Xt_Beta_M6)
                                   );

Mux_4_1 Xt_Beta_M7_Mux [17:0](.D0(18'h0_0000),
                              .D1(Xt_Beta7a),
                                    .D2(Xt_Beta7b),
                                    .D3(Xt_Beta7c),
                                    .S0(M[0]),
                                    .S1(M[1]),
                                    .O(Xt_Beta_M7)
                                   );

Mux_4_1 Xt_Beta_M8_Mux [17:0](.D0(18'h0_0000),
                              .D1(Xt_Beta8a),
                                    .D2(Xt_Beta8b),
                                    .D3(Xt_Beta8c),
                                    .S0(M[0]),
                                    .S1(M[1]),
                                    .O(Xt_Beta_M8)
                                   );

Mux_4_1 Xt_Beta_M9_Mux [17:0](.D0(18'h0_0000),
                              .D1(Xt_Beta9a),
                                    .D2(Xt_Beta9b),
                                    .D3(Xt_Beta9c),
                                    .S0(M[0]),
                                    .S1(M[1]),
                                    .O(Xt_Beta_M9)
                                   );

Mux_4_1 Xt_Beta_M10_Mux [17:0](     .D0(18'h0_0000),
                                    .D1(Xt_Beta10a),
                                    .D2(Xt_Beta10b),
                                    .D3(Xt_Beta10c),
                                    .S0(M[0]),
                                    .S1(M[1]),
                                    .O(Xt_Beta_M10)
                                   );


Mux_16_1 Xt_Beta_Out1_Mux [17:0](       .D1(Xt_Beta_M1),
                                        .D2(Xt_Beta_M2),
```

```
                                        .D3(Xt_Beta_M3),
                                        .D4(Xt_Beta_M4),
                                        .D5(Xt_Beta_M5),
                                        .D6(Xt_Beta_M6),
                                        .D7(Xt_Beta_M7),
                                        .D8(Xt_Beta_M8),
                                        .D9(Xt_Beta_M9),
                                        .D10(Xt_Beta_M10),
                                        .S0(Clus_Num1[0]),
                                        .S1(Clus_Num1[1]),
                                        .S2(Clus_Num1[2]),
                                        .S3(Clus_Num1[3]),
                                        .O(Xt_Beta_Out1)
                                       );

Mux_16_1 Xt_Beta_Out2_Mux [17:0](       .D1(Xt_Beta_M1),
                                        .D2(Xt_Beta_M2),
                                        .D3(Xt_Beta_M3),
                                        .D4(Xt_Beta_M4),
                                        .D5(Xt_Beta_M5),
                                        .D6(Xt_Beta_M6),
                                        .D7(Xt_Beta_M7),
                                        .D8(Xt_Beta_M8),
                                        .D9(Xt_Beta_M9),
                                        .D10(Xt_Beta_M10),
                                        .S0(Clus_Num2[0]),
                                        .S1(Clus_Num2[1]),
                                        .S2(Clus_Num2[2]),
                                        .S3(Clus_Num2[3]),
                                        .O(Xt_Beta_Out2)
                                       );


Mux_16_1 Miu_Beta_K2D_Out1_Mux [17:0](       .D1(Miu_Beta_K2D1),
                                             .D2(Miu_Beta_K2D2),
                                             .D3(Miu_Beta_K2D3),
                                             .D4(Miu_Beta_K2D4),
                                             .D5(Miu_Beta_K2D5),
                                             .D6(Miu_Beta_K2D6),
                                             .D7(Miu_Beta_K2D7),
                                             .D8(Miu_Beta_K2D8),
                                             .D9(Miu_Beta_K2D9),
                                             .D10(Miu_Beta_K2D10),
                                             .S0(Clus_Num1[0]),
                                             .S1(Clus_Num1[1]),
                                             .S2(Clus_Num1[2]),
                                             .S3(Clus_Num1[3]),
                                             .O(Miu_Beta_K2D_Out1)
                                            );


Mux_16_1 Miu_Beta_K2D_Out2_Mux [17:0](       .D1(Miu_Beta_K2D1),
                                             .D2(Miu_Beta_K2D2),
```

```
                                        .D3(Miu_Beta_K2D3),
                                        .D4(Miu_Beta_K2D4),
                                        .D5(Miu_Beta_K2D5),
                                        .D6(Miu_Beta_K2D6),
                                        .D7(Miu_Beta_K2D7),
                                        .D8(Miu_Beta_K2D8),
                                        .D9(Miu_Beta_K2D9),
                                        .D10(Miu_Beta_K2D10),
                                        .S0(Clus_Num2[0]),
                                        .S1(Clus_Num2[1]),
                                        .S2(Clus_Num2[2]),
                                        .S3(Clus_Num2[3]),
                                        .O(Miu_Beta_K2D_Out2)
                                       );


endmodule


‘timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
//  Post processing block top level file. This block store and update the status
//  and data of all alive transients, recover the tail prediction using the info
//  from block3, and generate tail_cur, tail_sum predictions and, and pre-offset
//  estimates.
//  Seven modules under this level:
//  1. Open_new_line: open a new transient processing line if alert is set and
//     use
//                     a pointer to point to this line.
//  2. Post_data_input: input the updated date from block3 to the line pointed
//     by
//                     pointer.
//  3. close_line_array: update the transient alive status for current time, and
//                     encode the line status.
//  4. tail_recover: recover the transient tail for one step prediction using
//     the
//                    information from block3.
//  5. tail_curr_generation: generate the tail_curr depend on the recovered tail
//                    and the encoded line status.
//  6. tail_sum_generation: generate the tail_sum depend on the recovered tail
//                    and the encoded line status.
//  7. offset_generation: generate the pre-offset depend on transient output
//                    (steady state) and the encoded line status.
//
////////////////////////////////////////////////////////////////////////////////

module Post_Proc(     input         CLK_4_1,
                      input         CLK_4_2,
                      input         CLK_4_3_1,  // CLK_4_3 = CLK_4_3_1.
                      input         CLK_4_3_2,
                      input         CLK_4_3_3,
                      input         CLK_4_3_4,
```

```
                    input         CLK_4_3_5,
                    input         CLK_4_3_6,
                    input         CLK_4_3_7,
                    input         CLK_4_3_8,
                    input         CLK_4_3_9,
                    input         CLK_4_4_1,
                    input         CLK_4_4_2,
                    input         CLK_SYS,
                    input  [2:0]  Tail_Sel,
                    input  [8:0]  Clus_Kn,
                    input  [3:0]  Clus_Num1,
                    input  [3:0]  Clus_Num2,
                    input  [8:0]  Clus_Weight1,
                    input  [8:0]  Clus_Weight2,
                    input  [15:0] Clus_Scale,
                    input  [15:0] Clus_Output,
                    output [15:0] Tail1_Output,
                    output [8:0]  Tail1_Kn,
                    output [17:0] Tail1_Start,
                    output        Tail1_Post_Alive,
                    output [15:0] Tail2_Output,
                    output [8:0]  Tail2_Kn,
                    output [17:0] Tail2_Start,
                    output        Tail2_Post_Alive,
                    output [15:0] Tail3_Output,
                    output [8:0]  Tail3_Kn,
                    output [17:0] Tail3_Start,
                    output        Tail3_Post_Alive,
                    input         EN,
                    input         Alert,
                    input  [17:0] N,
                    input  [15:0] Offset_FB,
                    output [17:0] Tail_Curr,
                    output [17:0] Tail_Sum,
                    output [15:0] Pre_Offset,

                    output [7:0]  Addr_Null
                );

//*********************************************************
// internal signal declaration
//*********************************************************

 wire        Curr1, Curr0;
 wire        Pre_Alive1, Pre_Alive2, Pre_Alive3;
 wire        Alert_Prim;
 wire [3:0]  Tail1_Num1,    Tail1_Num2;
 wire [8:0]  Tail1_Weight1, Tail1_Weight2;
 wire [15:0] Tail1_Scale;
 wire        Line1_Cur;
 wire [3:0]  Tail2_Num1, Tail2_Num2;
 wire [8:0]  Tail2_Weight1, Tail2_Weight2;
 wire [15:0] Tail2_Scale;
```

```
 wire        Line2_Cur;
 wire [3:0]  Tail3_Num1, Tail3_Num2;
 wire [8:0]  Tail3_Weight1, Tail3_Weight2;
 wire [15:0] Tail3_Scale;
 wire        Line3_Cur;
 wire [2:0]  L1S, L2S, L3S;
 wire [17:0] Tail1, Tail2, Tail3;

//***********************************************************
// 4.1 Open New Processing Line When Alert Set
//***********************************************************


Open_New_Line   Open_New_Line_Inst(.Alert(Alert),
                                   .Alive1_In(Tail1_Post_Alive),
                                   .Alive2_In(Tail2_Post_Alive),
                                   .Alive3_In(Tail3_Post_Alive),
                                   .CurrIn_1(Curr1),
                                   .CurrIn_0(Curr0),
                                   .CLK_4_1(CLK_4_1),
                                   .Alive1_Out(Pre_Alive1),
                                   .Alive2_Out(Pre_Alive2),
                                   .Alive3_Out(Pre_Alive3),
                                   .Curr1(Curr1),
                                   .Curr0(Curr0),
                                   .Alert_Prim(Alert_Prim)  );

//***********************************************************
// 4.2 Cluster data input and storage
//***********************************************************

Post_Data_Input Post_Data_Input_Inst( .EN(EN),
                                      .Curr1(Curr1),
                                      .Curr0(Curr0),
                                      .Alert_Prim(Alert_Prim),
                                      .CLK_4_2(CLK_4_2),
                                      .Clus_Num1(Clus_Num1),
                                      .Clus_Num2(Clus_Num2),
                                      .Clus_Weight1(Clus_Weight1),
                                      .Clus_Weight2(Clus_Weight2),
                                      .Clus_Scale(Clus_Scale),
                                      .Clus_Output(Clus_Output),
                                      .Clus_Kn(Clus_Kn),
                                      .N(N),
                                      .Tail1_Num1(Tail1_Num1),
                                      .Tail1_Num2(Tail1_Num2),
                                      .Tail1_Weight1(Tail1_Weight1),
                                      .Tail1_Weight2(Tail1_Weight2),
                                      .Tail1_Scale(Tail1_Scale),
                                      .Tail1_Output(Tail1_Output),
                                      .Tail1_Kn(Tail1_Kn),
                                      .Tail1_Start(Tail1_Start),
                                      .Line1_Cur(Line1_Cur),
```

```
                                    .Tail2_Num1(Tail2_Num1),
                                    .Tail2_Num2(Tail2_Num2),
                                    .Tail2_Weight1(Tail2_Weight1),
                                    .Tail2_Weight2(Tail2_Weight2),
                                    .Tail2_Scale(Tail2_Scale),
                                    .Tail2_Output(Tail2_Output),
                                    .Tail2_Kn(Tail2_Kn),
                                    .Tail2_Start(Tail2_Start),
                                    .Line2_Cur(Line2_Cur),
                                    .Tail3_Num1(Tail3_Num1),
                                    .Tail3_Num2(Tail3_Num2),
                                    .Tail3_Weight1(Tail3_Weight1),
                                    .Tail3_Weight2(Tail3_Weight2),
                                    .Tail3_Scale(Tail3_Scale),
                                    .Tail3_Output(Tail3_Output),
                                    .Tail3_Kn(Tail3_Kn),
                                    .Tail3_Start(Tail3_Start),
                                    .Line3_Cur(Line3_Cur) );

//***********************************************************
// 4.3 Processing Line status update
//***********************************************************

Close_Line_Array Close_Line_Array_Inst(.N(N),
                                       .CLK_4_3(CLK_4_3_1),
                                       .Pre_Alive1(Pre_Alive1),
                                       .Tail1_Kn(Tail1_Kn),
                                       .Tail1_Start(Tail1_Start),
                                       .Line1_Cur(Line1_Cur),
                                       .Pre_Alive2(Pre_Alive2),
                                       .Tail2_Kn(Tail2_Kn),
                                       .Tail2_Start(Tail2_Start),
                                       .Line2_Cur(Line2_Cur),
                                       .Pre_Alive3(Pre_Alive3),
                                       .Tail3_Kn(Tail3_Kn),
                                       .Tail3_Start(Tail3_Start),
                                       .Line3_Cur(Line3_Cur),
                                       .L1S(L1S),
                                       .Post_Alive1(Tail1_Post_Alive),
                                       .L2S(L2S),
                                       .Post_Alive2(Tail2_Post_Alive),
                                       .L3S(L3S),
                                       .Post_Alive3(Tail3_Post_Alive) );


//***********************************************************
// 4.3 Tail Recover from tial #, weight, offset address, and scale #.
//***********************************************************

Tail_Recover Tail_Recover_Inst(.N(N),
                               .Tail_Sel(Tail_Sel),
                               .CLK_4_3_1(CLK_4_3_1),
                               .CLK_4_3_2(CLK_4_3_2),
```

```
                                .CLK_4_3_3(CLK_4_3_3),
                                .CLK_4_3_4(CLK_4_3_4),
                                .CLK_4_3_5(CLK_4_3_5),
                                .CLK_4_3_6(CLK_4_3_6),
                                .CLK_4_3_7(CLK_4_3_7),
                                .CLK_4_3_8(CLK_4_3_8),
                                .CLK_4_3_9(CLK_4_3_9),
                                .CLK_SYS(CLK_SYS),
                                .Tail1_Num1(Tail1_Num1),
                                .Tail1_Num2(Tail1_Num2),
                                .Tail1_Weight1(Tail1_Weight1),
                                .Tail1_Weight2(Tail1_Weight2),
                                .Tail1_Scale(Tail1_Scale),
                                .Tail1_Start(Tail1_Start),
                                .Tail2_Num1(Tail2_Num1),
                                .Tail2_Num2(Tail2_Num2),
                                .Tail2_Weight1(Tail2_Weight1),
                                .Tail2_Weight2(Tail2_Weight2),
                                .Tail2_Scale(Tail2_Scale),
                                .Tail2_Start(Tail2_Start),
                                .Tail3_Num1(Tail3_Num1),
                                .Tail3_Num2(Tail3_Num2),
                                .Tail3_Weight1(Tail3_Weight1),
                                .Tail3_Weight2(Tail3_Weight2),
                                .Tail3_Scale(Tail3_Scale),
                                .Tail3_Start(Tail3_Start),
                                .Tail1(Tail1),
                                .Tail2(Tail2),
                                .Tail3(Tail3),
                                .Addr_Null(Addr_Null)
                            );

//*********************************************************
// 4.4  Generate Offset, Tail_Sum, Tail_Curr.
//*********************************************************

Offet_Generation Offet_Gen_Inst(.Tail1_Output(Tail1_Output),
                                .L1S1(L1S[1]),
                                .L1S0(L1S[0]),
                                .Tail2_Output(Tail2_Output),
                                .L2S1(L2S[1]),
                                .L2S0(L2S[0]),
                                .Tail3_Output(Tail3_Output),
                                .L3S1(L3S[1]),
                                .L3S0(L3S[0]),
                                .Offset_FB(Offset_FB),
                                .CLK_4_4_1(CLK_4_4_1),
                                .CLK_4_4_2(CLK_4_4_2),
                                .Pre_Offset(Pre_Offset) );

Tail_Sum_Generation Tail_Sum_Inst(.L1S(L1S),
                                  .L2S(L2S),
                                  .L3S(L3S),
```

```
                                    .Tail1(Tail1),
                                    .Tail2(Tail2),
                                    .Tail3(Tail3),
                                    .CLK_4_4_1(CLK_4_4_1),
                                    .CLK_4_4_2(CLK_4_4_2),
                                    .Tail_Sum(Tail_Sum)  );

Tail_Curr_Generation Tail_Curr_Inst(.Tail1(Tail1),
                                    .Tail2(Tail2),
                                    .Tail3(Tail3),
                                    .L1S(L1S),
                                    .L2S(L2S),
                                    .L3S(L3S),
                                    .Tail_Curr(Tail_Curr) );

endmodule


‘timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
//     Open mew transient processing line if alert set, and using a pointer to
//     indicate this line.
//
////////////////////////////////////////////////////////////////////////////////

module Open_New_Line(input           Alert,
                     input           Alive1_In,
                     input           Alive2_In,
                     input           Alive3_In,
                     input           CurrIn_1,
                     input           CurrIn_0,
                     input           CLK_4_1,
                     output          Alive1_Out,
                     output          Alive2_Out,
                     output          Alive3_Out,
                     output          Curr1,
                     output          Curr0,
                     output  reg     Alert_Prim  = 0 );


//*********************************************************
//  internal signals
//*********************************************************

 reg Alive1_In_Prim = 1’b0,
     Alive2_In_Prim = 1’b0,
     Alive3_In_Prim = 1’b0,
     CurrIn_1_Prim  = 1’b0,
     CurrIn_0_Prim  = 1’b0;

//*********************************************************
//      pipeline operation
```

```
//*********************************************************

always @ (posedge CLK_4_1)
   begin
      Alert_Prim     <= Alert;
      Alive1_In_Prim <= Alive1_In;
      Alive2_In_Prim <= Alive2_In;
      Alive3_In_Prim <= Alive3_In;
      CurrIn_1_Prim  <= CurrIn_1;
      CurrIn_0_Prim  <= CurrIn_0;
    end

//*********************************************************
//   finding empty line with the priority of 1 > 2 > 3
//*********************************************************

assign  Alive1_Out =   Alert_Prim | Alive1_In_Prim,
        Alive2_Out = ( Alert_Prim & Alive1_In_Prim ) | Alive2_In_Prim,
        Alive3_Out = ( Alert_Prim & Alive1_In_Prim   & Alive2_In_Prim )
                     | Alive3_In_Prim;

//*********************************************************
//    point to the new open line
//*********************************************************

assign

Curr1 =   Alert_Prim ? (Alive1_In_Prim ? (Alive2_In_Prim ?
(Alive3_In_Prim ? (CurrIn_1_Prim) : (1)) : (1)) : (0)) :
(CurrIn_1_Prim),

Curr0 =   Alert_Prim ? (Alive1_In_Prim ? (Alive2_In_Prim ?
(Alive3_In_Prim ? (CurrIn_0_Prim) : (1)) : (0)) : (1)) :
(CurrIn_0_Prim);


endmodule


‘timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// Post_data_input:  input new data from block3 to the line indicated by
// the current
// line pointer
//
////////////////////////////////////////////////////////////////////////////////

module Post_Data_Input(// Control signals
                       input              EN,
                       input              Curr1,
                       input              Curr0,
                       input              Alert_Prim,
                       input              CLK_4_2,
```

```
                    // Input data from Clus_Proc Unit
                        input       [3:0]  Clus_Num1,
                        input       [3:0]  Clus_Num2,
                        input       [8:0]  Clus_Weight1,
                        input       [8:0]  Clus_Weight2,
                        input       [15:0] Clus_Scale,
                        input       [15:0] Clus_Output,
                        input       [8:0]  Clus_Kn,
                        input       [17:0] N,
                    // Transient Processing Line1 Storage
                        output reg  [3:0]  Tail1_Num1    = 4'hF,
                        output reg  [3:0]  Tail1_Num2    = 4'hF,
                        output reg  [8:0]  Tail1_Weight1 = 9'h000,
                        output reg  [8:0]  Tail1_Weight2 = 9'h000,
                        output reg  [15:0] Tail1_Scale   = 16'h0000,
                        output reg  [15:0] Tail1_Output  = 16'h0000,
                        output reg  [8:0]  Tail1_Kn      = 9'h000,
                        output reg  [17:0] Tail1_Start   = 18'h00000,
                        output             Line1_Cur,
                    // Transient Processing Line2 Storage
                        output reg  [3:0]  Tail2_Num1    = 4'hF,
                        output reg  [3:0]  Tail2_Num2    = 4'hF,
                        output reg  [8:0]  Tail2_Weight1 = 9'h000,
                        output reg  [8:0]  Tail2_Weight2 = 9'h000,
                        output reg  [15:0] Tail2_Scale   = 16'h0000,
                        output reg  [15:0] Tail2_Output  = 16'h0000,
                        output reg  [8:0]  Tail2_Kn      = 9'h000,
                        output reg  [17:0] Tail2_Start   = 18'h00000,
                        output             Line2_Cur,
                   // Transient Processing Line3 Storage
                        output reg  [3:0]  Tail3_Num1    = 4'hF,
                        output reg  [3:0]  Tail3_Num2    = 4'hF,
                        output reg  [8:0]  Tail3_Weight1 = 9'h000,
                        output reg  [8:0]  Tail3_Weight2 = 9'h000,
                        output reg  [15:0] Tail3_Scale   = 16'h0000,
                        output reg  [15:0] Tail3_Output  = 16'h0000,
                        output reg  [8:0]  Tail3_Kn      = 9'h000,
                        output reg  [17:0] Tail3_Start   = 18'h00000,
                        output             Line3_Cur );



//*********************************************************
//       internal signals
//*********************************************************

wire Line1_CLK, Line2_CLK, Line3_CLK,
     S1_CLK,    S2_CLK,    S3_CLK;

//*********************************************************
// Combunatiobnal Logic to generate the pipeline &
// clock enables
//*********************************************************
```

```
assign Line1_Cur = ~Curr1 & Curr0,
       Line2_Cur = ~Curr0 & Curr1,
       Line3_Cur =  Curr1 & Curr0,

       Line1_CLK = Line1_Cur & CLK_4_2 & EN,
       Line2_CLK = Line2_Cur & CLK_4_2 & EN,
       Line3_CLK = Line3_Cur & CLK_4_2 & EN,

       S1_CLK    = Line1_Cur & CLK_4_2 & Alert_Prim,
       S2_CLK    = Line2_Cur & CLK_4_2 & Alert_Prim,
       S3_CLK    = Line3_Cur & CLK_4_2 & Alert_Prim;

//*********************************************************
// Data Storage in Register Files
//*********************************************************

always @ (posedge Line1_CLK)
  begin
    Tail1_Num1    <= Clus_Num1 - 4'b0001;
    Tail1_Num2    <= Clus_Num2 - 4'b0001;
    Tail1_Weight1 <= Clus_Weight1;
    Tail1_Weight2 <= Clus_Weight2;
    Tail1_Scale   <= Clus_Scale;
    Tail1_Output  <= Clus_Output;
    Tail1_Kn      <= Clus_Kn;
  end

always @ (posedge Line2_CLK)
  begin
    Tail2_Num1    <= Clus_Num1 - 4'b0001;
    Tail2_Num2    <= Clus_Num2 - 4'b0001;
    Tail2_Weight1 <= Clus_Weight1;
    Tail2_Weight2 <= Clus_Weight2;
    Tail2_Scale   <= Clus_Scale;
    Tail2_Output  <= Clus_Output;
    Tail2_Kn      <= Clus_Kn;
  end

always @ (posedge Line3_CLK)
  begin
    Tail3_Num1    <= Clus_Num1 - 4'b0001;
    Tail3_Num2    <= Clus_Num2 - 4'b0001;
    Tail3_Weight1 <= Clus_Weight1;
    Tail3_Weight2 <= Clus_Weight2;
    Tail3_Scale   <= Clus_Scale;
    Tail3_Output  <= Clus_Output;
    Tail3_Kn      <= Clus_Kn;
  end

//*********************************************************
//   recode transient start point
//*********************************************************
```

```
 always @ (posedge S1_CLK) Tail1_Start = N;
 always @ (posedge S2_CLK) Tail2_Start = N;
 always @ (posedge S3_CLK) Tail3_Start = N;

endmodule


‘timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
//  close line arry tope level,to update the line status and encode to
//  outputs that control
//  the following blocks.
//  one module under this level: close_line
//
////////////////////////////////////////////////////////////////////////////////

module Close_Line_Array(input      [17:0] N,
                        input             CLK_4_3,
                        input             Pre_Alive1,
                        input      [8:0] Tail1_Kn,
                        input      [17:0] Tail1_Start,
                        input             Line1_Cur,
                        input             Pre_Alive2,
                        input      [8:0] Tail2_Kn,
                        input      [17:0] Tail2_Start,
                        input             Line2_Cur,
                        input             Pre_Alive3,
                        input      [8:0] Tail3_Kn,
                        input      [17:0] Tail3_Start,
                        input             Line3_Cur,
                        output     [2:0]  L1S,
                        output            Post_Alive1,
                        output     [2:0]  L2S,
                        output            Post_Alive2,
                        output     [2:0]  L3S,
                        output            Post_Alive3 );


//**********************************************************
// Implementation of Line1 Closing Operation
//**********************************************************

Close_Line      Close_Line_Inst1(.CLK_4_3(CLK_4_3),
                                 .Kn(Tail1_Kn),
                                 .Start(Tail1_Start),
                                 .Line_Cur(Line1_Cur),
                                 .Alive_In(Pre_Alive1),
                                 .N(N),
                                 .Line_Status(L1S),
                                 .Alive_Out(Post_Alive1));
```

```
//*********************************************************
// Implementation of Line2 Closing Operation
//*********************************************************

Close_Line      Close_Line_Inst2(.CLK_4_3(CLK_4_3),
                                 .Kn(Tail2_Kn),
                                 .Start(Tail2_Start),
                                 .Line_Cur(Line2_Cur),
                                 .Alive_In(Pre_Alive2),
                                 .N(N),
                                 .Line_Status(L2S),
                                 .Alive_Out(Post_Alive2));

//*********************************************************
// Implementation of Line3 Closing Operation
//*********************************************************

Close_Line      Close_Line_Inst3(.CLK_4_3(CLK_4_3),
                                 .Kn(Tail3_Kn),
                                 .Start(Tail3_Start),
                                 .Line_Cur(Line3_Cur),
                                 .Alive_In(Pre_Alive3),
                                 .N(N),
                                 .Line_Status(L3S),
                                 .Alive_Out(Post_Alive3));


endmodule


‘timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
// close line used to encoding the line status
//
////////////////////////////////////////////////////////////////////////////////

module Close_Line(      input      [8:0] Kn,
                        input      [17:0] Start,
                        input      [17:0] N,
                        input             Alive_In,
                        input             Line_Cur,
                        input             CLK_4_3,
                        output     [2:0]  Line_Status,
                        output            Alive_Out   );


//*********************************************************
//  internal signals
//*********************************************************

 wire     [17:0] Lock_Out = 18’b00_0000_0000_0000_0101;
 reg      [8:0]  Kn_Reg          = 9’h000;
 reg      [17:0] Start_Reg = 18’h00000;
```

```
 reg      [17:0] N_Reg = 18'h00000;
 reg             Alive_In_Reg   = 1'b0;
 reg             Line_Cur_Reg   = 1'b0;
 wire [17:0]     Trans_Duration = 18'h00000;


//*********************************************************
//  pipeline operation
//*********************************************************

always @ (posedge CLK_4_3)
  begin
     Kn_Reg       <= Kn;
     Start_Reg    <= Start;
     N_Reg        <= N;
     Alive_In_Reg <= Alive_In;
     Line_Cur_Reg <= Line_Cur;
  end


//*********************************************************
//                  encoding tree
//*********************************************************

// Switching Tree: Priority: (Highest) Alive_In, Line_Cur,
// "N - Start > Lock_Out?",
// "N-Start = Kn?" (Lowest)
//                                      Alive_In
//                           1             |           0
//                       ------------------------------------------
//                      |                                          |
//                  Line_Cur            [0 0 0], 0 (Line_Status[2:0], Alive_Out)
//                  1    |      0
//            ------------------------------------
//           |                                    |
//   "N - Start >=Lock_Out?"                "N - Start = Kn?"
//          1    |    0                             1   |   0
//   --------------------------               -------------------------
//   |                          |             |                         |
// "N - Start = Kn?"     [0,0,0] Alive_In    [0,1,1],0              [0,1,0],1
//   1    |   0
//   -------------------
//  |                   |
//[1,1,1],0         [1,1,0],1


assign
Trans_Duration              = N_Reg - Start_Reg,
{Line_Status, Alive_Out}    = Alive_In_Reg ? (Line_Cur_Reg ?
((Trans_Duration >= Lock_Out) ? ((Trans_Duration == Kn_Reg) ?
(4'b1110) : (4'b1101)) : ({3'b000, Alive_In_Reg})) :
((Trans_Duration == Kn_Reg) ? (4'b0110) : (4'b0101))) : (4'b0000);
```

```
endmodule


‘timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
//  Tail recover for line1-3, using the the info. from block3 (filter weight,
//    scale factor, cluster numbers) of champion and rival clusters.
//  module Tail_recover_mem_block under this level
//
//////////////////////////////////////////////////////////////////////////////////


//‘include "C:\zhutaoFPGAproject\SCWM1\SCWM1\SCWM_Header.h"



module Tail_Recover(input      [17:0] N,
                    input      [2:0]  Tail_Sel,
                    input             CLK_4_3_1,
                    input             CLK_4_3_2,
                    input             CLK_4_3_3,
                    input             CLK_4_3_4,
                    input             CLK_4_3_5,
                    input             CLK_4_3_6,
                    input             CLK_4_3_7,
                    input             CLK_4_3_8,
                    input             CLK_4_3_9,
                    input             CLK_SYS,
                    input      [3:0]  Tail1_Num1,
                    input      [3:0]  Tail1_Num2,
                    input      [8:0]  Tail1_Weight1,
                    input      [8:0]  Tail1_Weight2,
                    input      [15:0] Tail1_Scale,
                    input      [17:0] Tail1_Start,
                    input      [3:0]  Tail2_Num1,
                    input      [3:0]  Tail2_Num2,
                    input      [8:0]  Tail2_Weight1,
                    input      [8:0]  Tail2_Weight2,
                    input      [15:0] Tail2_Scale,
                    input      [17:0] Tail2_Start,
                    input      [3:0]  Tail3_Num1,
                    input      [3:0]  Tail3_Num2,
                    input      [8:0]  Tail3_Weight1,
                    input      [8:0]  Tail3_Weight2,
                    input      [15:0] Tail3_Scale,
                    input      [17:0] Tail3_Start,
                    output reg [17:0] Tail1 = 18’h00000,
                    output reg [17:0] Tail2 = 18’h00000,
                    output reg [17:0] Tail3 = 18’h00000,

                    output     [7:0]  Addr_Null
                   );
```

```
//*********************************************************
// Internal variables declarations.
//*********************************************************
 wire       [3:0]  Tail_Num;
 wire       [17:0] Start;
 wire       [15:0] Scale;
 wire       [8:0]  Tail_Weight;
 wire       [17:0] Tail;
 wire       [17:0] Tail2_In, Tail3_In,Tail1_In;
 reg        [17:0] Tail1_A_In = 18'h00000, Tail1_A = 18'h00000,
                   Tail1_B = 18'h00000;
 reg        [17:0] Tail2_A_In = 18'h00000, Tail2_A = 18'h00000,
                   Tail2_B = 18'h00000;
 reg        [17:0] Tail3_A_In = 18'h00000, Tail3_A = 18'h00000,
                   Tail3_B = 18'h00000;
 wire       [9:0]  CS;
 wire       [17:0] Addr, Addr1; wire [17:0] Data;
 wire       [35:0] Tail_Weighted_Full;
 wire       [17:0] Tail_Weighted;
 wire       [35:0] Tail_Scaled_Full;
 wire       [17:0] Tail_Scaled;

//*********************************************************
// Multiplexer Implementation   : 5 MUXs for Tail_Clus#,
// Tail_Start, Tail_Weight, Tail_Scale, Tail
//*********************************************************

Mux_8_1   Tail_Num_Mux [3:0] (        .D0(Tail1_Num1),
                                      .D1(Tail1_Num2),
                                      .D2(Tail2_Num1),
                                      .D3(Tail2_Num2),
                                      .D4(Tail3_Num1),
                                      .D5(Tail3_Num2),
                                      .D6(4'b1111),
                                      .D7(4'b1111),
                                      .S0(Tail_Sel[0]),
                                      .S1(Tail_Sel[1]),
                                      .S2(Tail_Sel[2]),
                                      .O(Tail_Num)   );

Mux_8_1   Tail_Weight_Mux [8:0] (        .D0(Tail1_Weight1),
                                         .D1(Tail1_Weight2),
                                         .D2(Tail2_Weight1),
                                         .D3(Tail2_Weight2),
                                         .D4(Tail3_Weight1),
                                         .D5(Tail3_Weight2),
                                         .D6(9'b0_0000_0000),
                                         .D7(9'b0_0000_0000),
                                         .S0(Tail_Sel[0]),
                                         .S1(Tail_Sel[1]),
                                         .S2(Tail_Sel[2]),
```

```
                                        .O(Tail_Weight)   );

Mux_4_1   Start_Mux [17:0] (.D0(Tail1_Start),
                            .D1(Tail2_Start),
                                  .D2(Tail3_Start),
                                  .D3(18'b00_0000_0000_0000_0000),
                                  .S0(Tail_Sel[1]),
                                  .S1(Tail_Sel[2]),
                                  .O(Start)   );

Mux_4_1   Scale_Mux [15:0] (.D0(Tail1_Scale),
                            .D1(Tail2_Scale),
                                  .D2(Tail3_Scale),
                                  .D3(16'h0000),
                                  .S0(Tail_Sel[1]),
                                  .S1(Tail_Sel[2]),
                                  .O(Scale)   );


Mux_4_1   Tail_Mux [17:0] ( .D0(Tail1_In),
                            .D1(Tail2_In),
                                  .D2(Tail3_In),
                                  .D3(18'h00000),
                                  .S0(Tail_Sel[1]),
                                  .S1(Tail_Sel[2]),
                                  .O(Tail)   );

//***********************************************************
// Decoder for generating the EN signal for MEMs
//***********************************************************

Decoder  Decoder_Inst1(.A0(Tail_Num[0]),
                       .A1(Tail_Num[1]),
                              .A2(Tail_Num[2]),
                              .A3(Tail_Num[3]),
                              .CS0(CS[0]),
                              .CS1(CS[1]),
                              .CS2(CS[2]),
                              .CS3(CS[3]),
                              .CS4(CS[4]),
                              .CS5(CS[5]),
                              .CS6(CS[6]),
                              .CS7(CS[7]),
                              .CS8(CS[8]),
                              .CS9(CS[9]) );

//***********************************************************
// Address generation for MEM
//***********************************************************

 assign Addr1[17:0] = N[17:0] - Start[17:0];
 assign Addr[17:0]  = Addr1[17:0] + 18'b00_0000_0000_0000_0010;
```

```
// this mean nothing, just prevent XST delete the useful logic HW
assign Addr_Null = Addr[17:10];

//*********************************************************
// SCWM Parameter MEM Block
//*********************************************************

Tail_Recover_Mem_Block  Tail_Miux(.Addr(Addr[9:0]),
                                  .CLK_SYS(CLK_SYS),
                                  .CS(CS),
                                  .Data(Data) );


//*********************************************************
// Multiplier & Adder
//*********************************************************

assign Tail_Weighted_Full = Data * {9'b0_0000_0000, Tail_Weight},
       Tail_Weighted = Tail_Weighted_Full[25:8];

assign Tail_Scaled_Full = Tail * {2'b00, Scale},
       Tail_Scaled = Tail_Scaled_Full[31:14];

assign Tail1_In = Tail1_A + Tail1_B,
       Tail2_In = Tail2_A + Tail2_B,
       Tail3_In = Tail3_A + Tail3_B;

//*********************************************************
// Intermidiate State Regs
//*********************************************************

 always @ (posedge CLK_4_3_1) Tail1_A_In <= Tail_Weighted;
 always @ (posedge CLK_4_3_4) Tail2_A_In <= Tail_Weighted;
 always @ (posedge CLK_4_3_7) Tail3_A_In <= Tail_Weighted;
 always @ (posedge CLK_4_3_2) begin Tail1_A <= Tail1_A_In;
                                    Tail1_B <= Tail_Weighted; end
 always @ (posedge CLK_4_3_5) begin Tail2_A <= Tail2_A_In;
                                    Tail2_B <= Tail_Weighted; end
 always @ (posedge CLK_4_3_8) begin Tail3_A <= Tail3_A_In;
                                    Tail3_B <= Tail_Weighted; end
 always @ (posedge CLK_4_3_3) Tail1 <= Tail_Scaled;
 always @ (posedge CLK_4_3_6) Tail2 <= Tail_Scaled;
 always @ (posedge CLK_4_3_9) Tail3 <= Tail_Scaled;


endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
//         Miux (cluster mean) memory block for tail recover
//
```

```
////////////////////////////////////////////////////////////////////////////////

//`include "C:\zhutaoFPGAproject\SCWM1\SCWM1\SCWM_Header.h"

module Tail_Recover_Mem_Block(input  [9:0]   Addr,
                              input          CLK_SYS,
                              input  [9:0]   CS,
                              output [17:0]  Data );


 wire [17:0] Data0, Data1, Data2, Data3, Data4, Data5, Data6,
             Data7, Data8, Data9;
 wire [9:0]  CSBar;

// output data bus multiplexing

 not    Not_Gate [9:0]  (CSBar, CS);
 bufif0 bufif0_0 [17:0] (Data, Data0, CSBar[0]);
 bufif0 bufif0_1 [17:0] (Data, Data1, CSBar[1]);
 bufif0 bufif0_2 [17:0] (Data, Data2, CSBar[2]);
 bufif0 bufif0_3 [17:0] (Data, Data3, CSBar[3]);
 bufif0 bufif0_4 [17:0] (Data, Data4, CSBar[4]);
 bufif0 bufif0_5 [17:0] (Data, Data5, CSBar[5]);
 bufif0 bufif0_6 [17:0] (Data, Data6, CSBar[6]);
 bufif0 bufif0_7 [17:0] (Data, Data7, CSBar[7]);
 bufif0 bufif0_8 [17:0] (Data, Data8, CSBar[8]);
 bufif0 bufif0_9 [17:0] (Data, Data9, CSBar[9]);



//Miux0
RAMB16_S18 #(
// The following defparam declarations specify the behavior of the RAM.
.INIT(36'h000000000),       // Initial values on A output port
.SRVAL(36'h000000000),      // Set/Reset value for A port output
.WRITE_MODE("WRITE_FIRST"),
.INIT_18(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_17(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_16(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_15(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_14(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_13(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_12(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_11(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_10(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_0F(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_0E(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_0D(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_0C(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_0B(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_0A(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_09(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_08(256'h0429042904290429042904290429042904290429042904290429042904290429),
```

```
.INIT_07(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_06(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_05(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_04(256'h0429042904290429042904290429042904290429042904290429042904290429),
.INIT_03(256'h042904290429042904290429042c042d0428041e04240438043c043d043a0423040c),
.INIT_02(256'h03ec03d703df03f9041104230436042e041e040804090421042c0431042803f1),
.INIT_01(256'h0390030a028b022c01f601dd01dc02080282039804ec067b0abd10de1b7c235e),
.INIT_00(256'h253a255823b520271c00176e12370efb0f7112a5162a16ae13760eea09d10471)
)

RAMB16_Tail_Miux0(  .DO(Data0[15:0]),
                    .DOP(Data0[17:16]),
                    .ADDR(Addr),
                    .CLK(CLK_SYS),
                    .DI(16'hFFFF),
                    .DIP(2'b11),
                    .EN(CS[0]),
                    .SSR(1'b0),
                    .WE(1'b0)
                  );




//Miux1
RAMB16_S18 #(
 // The following defparam declarations specify the behavior of the RAM.
 .INIT(36'h000000000),       // Initial values on A output port
 .SRVAL(36'h000000000),      // Set/Reset value for A port output
 .WRITE_MODE("WRITE_FIRST"),
 .INIT_18(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_17(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_16(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_15(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_14(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_13(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_12(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_11(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_10(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_0F(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_0E(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_0D(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_0C(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_0B(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_0A(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_09(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_08(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_07(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_06(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_05(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_04(256'h0440044004400440044004400440044004400440044004400440044004400440),
 .INIT_03(256'h0440044004400440044004400043c043004250413043104 5e046c044d043904360419),
 .INIT_02(256'h041d043404330429041 2040c040e040e041a0439045d046b047d048e044b03c2),
```

```
 .INIT_01(256'h033802c7026f0233020a01ee01f0023602e5042c056c07ad0c69132d1dea23df),
 .INIT_00(256'h24d8246321911e2e1a81164f115a0ecb0f9c1309162d168b13170e900998044c)
   )

RAMB16_Tail_Miux1(  .DO(Data1[15:0]),
                    .DOP(Data1[17:16]),
                    .ADDR(Addr),
                    .CLK(CLK_SYS),
                    .DI(16'hFFFF),
                    .DIP(2'b11),
                    .EN(CS[1]),
                    .SSR(1'b0),
                    .WE(1'b0)
                  );


//Miux2
RAMB16_S18 #(
 // The following defparam declarations specify the behavior of the RAM.
 .INIT(36'h000000000),       // Initial values on A output port
 .SRVAL(36'h000000000),      // Set/Reset value for A port output
 .WRITE_MODE("WRITE_FIRST"),
 .INIT_18(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_17(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_16(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_15(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_14(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_13(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_12(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_11(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_10(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_0F(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_0E(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_0D(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_0C(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_0B(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_0A(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_09(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_08(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_07(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_06(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_05(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_04(256'h0424042404240424042404240424042404240424042404240424042404240424),
 .INIT_03(256'h0424042404240424042404260440046504620440c04440432041d0418042e0448),
 .INIT_02(256'h045d04650444041004004a0411041e04340454045e0452044e044d0442041b),
 .INIT_01(256'h03ca035002c4024601f401d501d201f20250034004ad060709820ee4183121df),
 .INIT_00(256'h24a6250023aa20ba1d0317fd12df0fdf0fdf13c71796179013ba0ee209d4047e)
   )

RAMB16_Tail_Miux2(  .EN(CS[2]),
                    .DO(Data2[15:0]),
                    .DOP(Data2[17:16]),
                    .ADDR(Addr),
```

```
                    .CLK(CLK_SYS),
                    .DI(16'hFFFF),
                    .DIP(2'b11),
                    .SSR(1'b0),
                    .WE(1'b0)
                  );


//Miux3
RAMB16_S18 #(
 // The following defparam declarations specify the behavior of the RAM.
 .INIT(36'h000000000),       // Initial values on A output port
 .SRVAL(36'h000000000),      // Set/Reset value for A port output
 .WRITE_MODE("WRITE_FIRST"),
 .INIT_18(256'h0fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd3),
 .INIT_17(256'h0fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd3),
 .INIT_16(256'h0fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd3),
 .INIT_15(256'h0fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd3),
 .INIT_14(256'h0fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd3),
 .INIT_13(256'h0fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd3),
 .INIT_12(256'h0fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd3),
 .INIT_11(256'h0fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd3),
 .INIT_10(256'h0fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd3),
 .INIT_0F(256'h0fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd3),
 .INIT_0E(256'h0fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd3),
 .INIT_0D(256'h0fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd30fd3),
 .INIT_0C(256'h0fd30fd30fd30fd30fd30fd30fd30fd30fd00fd40fdc0fd60fd30fd00fd4),
 .INIT_0B(256'h0fdc0fd70fd30fd00fd50fdc0fd70fd30fd00fd50fdd0fd70fd30fd00fd50fdd),
 .INIT_0A(256'h0fd70fd30fd00fd50fdd0fd70fd30fd10fd50fdd0fd70fd40fd10fd50fdd0fd7),
 .INIT_09(256'h0fd40fd10fd50fdd0fd80fd40fd10fd60fdd0fd80fd40fd10fd60fdd0fd80fd4),
 .INIT_08(256'h0fd10fd60fdd0fd80fd40fd20fd60fde0fd80fd40fd20fd60fde0fd80fd40fd2),
 .INIT_07(256'h0fd60fde0fd80fd50fd20fd60fde0fd90fd50fd20fd60fde0fd90fd50fd20fd7),
 .INIT_06(256'h0fde0fd90fd50fd20fd70fde0fd90fd50fd30fd70fdf0fd90fd50fd30fd70fdf),
 .INIT_05(256'h0fd90fd60fd30fd70fdf0fda0fd60fd30fd70fdf0fda0fd60fd30fd70fdf0fda),
 .INIT_04(256'h0fd60fd30fd70fdf0fda0fd60fd40fd80fe00fdb0fd70fd40fd80fe00fdb0fd7),
 .INIT_03(256'h0fd40fd80fe00fdb0fd70fd40fd80fe10fdb0fd80fd50fd90fe10fdb0fd80fd5),
 .INIT_02(256'h0fda0fe10fdc0fd90fd60fda0fe20fdd0fda0fd70fdb0fe30fde0fdb0fd80fdd),
 .INIT_01(256'h0fe50fe00fde0fdc0fe20fea0fe80fe80fe80ff00ffc0fff1005100c101d1036),
 .INIT_00(256'h10491060107d10a810ec1139118c11f3128313841509172f1a381f5b1e8706db)
  )

RAMB16_Tail_Miux3(  .DO(Data3[15:0]),
                    .DOP(Data3[17:16]),
                    .ADDR(Addr),
                    .CLK(CLK_SYS),
                    .DI(16'hFFFF),
                    .DIP(2'b11),
                    .EN(CS[3]),
                    .SSR(1'b0),
                    .WE(1'b0)
                  );
```

```
//Miux4
RAMB16_S18 #(
 // The following defparam declarations specify the behavior of the RAM.
 .INIT(36'h000000000),       // Initial values on A output port
 .SRVAL(36'h000000000),      // Set/Reset value for A port output
 .WRITE_MODE("WRITE_FIRST"),
 .INIT_18(256'h0fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd1),
 .INIT_17(256'h0fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd1),
 .INIT_16(256'h0fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd1),
 .INIT_15(256'h0fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd1),
 .INIT_14(256'h0fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd1),
 .INIT_13(256'h0fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd1),
 .INIT_12(256'h0fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd1),
 .INIT_11(256'h0fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd1),
 .INIT_10(256'h0fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd1),
 .INIT_0F(256'h0fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd1),
 .INIT_0E(256'h0fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd1),
 .INIT_0D(256'h0fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd1),
 .INIT_0C(256'h0fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd10fd30fd60fd60fd10fd10fd3),
 .INIT_0B(256'h0fd50fd60fd10fd10fd30fd60fd70fd10fd10fd30fd60fd70fd10fd10fd30fd6),
 .INIT_0A(256'h0fd70fd10fd10fd30fd60fd70fd10fd10fd30fd60fd70fd20fd10fd30fd60fd7),
 .INIT_09(256'h0fd20fd10fd40fd60fd70fd20fd20fd40fd60fd70fd20fd20fd40fd70fd70fd2),
 .INIT_08(256'h0fd20fd40fd70fd70fd20fd20fd40fd70fd70fd20fd20fd40fd70fd80fd20fd2),
 .INIT_07(256'h0fd40fd70fd80fd30fd20fd50fd70fd80fd30fd30fd50fd70fd80fd30fd30fd5),
 .INIT_06(256'h0fd70fd80fd30fd30fd50fd70fd80fd30fd30fd50fd80fd80fd30fd30fd50fd8),
 .INIT_05(256'h0fd80fd30fd40fd50fd80fd80fd30fd40fd60fd80fd90fd40fd40fd60fd80fd9),
 .INIT_04(256'h0fd40fd40fd60fd80fd90fd40fd40fd60fd80fd90fd40fd50fd70fd90fd90fd5),
 .INIT_03(256'h0fd50fd70fd90fda0fd50fd50fd70fd90fda0fd50fd60fd70fd90fda0fd60fd6),
 .INIT_02(256'h0fd80fda0fdb0fd60fd70fd90fda0fdb0fd70fd80fd90fdc0fdc0fd80fd90fdb),
 .INIT_01(256'h0fde0fdf0fdc0fdd0fe00fe30fe60fe50fe90fee0ff50ffd1001100c101b102e),
 .INIT_00(256'h1045105b107c10a610de112a117f11ee1283136514c616be19cf1f181cbd06a7)
  )

RAMB16_Tail_Miux4(  .EN(CS[4]),
                    .DO(Data4[15:0]),
                    .DOP(Data4[17:16]),
                    .ADDR(Addr),
                    .CLK(CLK_SYS),
                    .DI(16'hFFFF),
                    .DIP(2'b11),
                    .SSR(1'b0),
                    .WE(1'b0)
                  );



//Miux5
RAMB16_S18 #(
 // The following defparam declarations specify the behavior of the RAM.
 .INIT(36'h000000000),       // Initial values on A output port
 .SRVAL(36'h000000000),      // Set/Reset value for A port output
 .WRITE_MODE("WRITE_FIRST"), // "WRITE_FIRST", "READ_FIRST", or "NO_CHANGE"
```

```
 // The following generic INIT_xx declarations specify the initial contents of
 // the RAM
 .INIT_18(256'h10ca10b810d410cd10cc10cd10bb10d610d410d810da10be10cc10c810d210d7),
 .INIT_17(256'h10be10d110d210d510ce10b510cf10d010d210cb10b910d410d110c910c710b8),
 .INIT_16(256'h10da10d910d110cd10bd10d810cd10c810c910b810d810d410d210cb10b810cd),
 .INIT_15(256'h10c710d210ce10bb10d710d310d510ce10ba10d210d310d710d210be10d810d5),
 .INIT_14(256'h10cf10c910b810db10d810d410d210c410df10d310ce10ca10bf10dd10d910dc),
 .INIT_13(256'h10d610be10d010cf10d910d910c710de10d910dd10d610be10d710d310d910d6),
 .INIT_12(256'h10c310dd10d810d010c510ba10dd10de10df10d210c210e210db10d910d310c7),
 .INIT_11(256'h10e510dc10d810d210c410da10d110d710d610c810e310df10e610db10c310d7),
 .INIT_10(256'h10d510df10d910c810e310e310dd10ce10bb10d810de10e310dd10cc10e510e0),
 .INIT_0F(256'h10d910cc10bf10df10e410e710dd10cf10ea10e010db10d210c910e910e710ec),
 .INIT_0E(256'h10e510da10f110e010e210db10cf10ec10e810ec10e210d810f110e310ea10e8),
 .INIT_0D(256'h10d810f210ec10ef10e510d210e610dc10e410e210d710ee10ea10f310ea10d8),
 .INIT_0C(256'h10ec10e210ec10e710da10f410ec10f510ea10da10f010e910f210ed10e210fe),
 .INIT_0B(256'h10f510f910f310e310ff10f610fa10f110e9110710fb10fd10f310ec11081100),
 .INIT_0A(256'h110210ef10e9110a1107110b110010ef110b110e110f10fd10eb110e11131122),
 .INIT_09(256'h111110ff111e111c11221115110f112e11221125111011091133113411381123),
 .INIT_08(256'h111a113d1138114211301125114a113f1145112f1121114b11471152113f1138),
 .INIT_07(256'h1164116011671153114a1170116c1176115e1150117b117c118a117111661194),
 .INIT_06(256'h119311a01187117c11ac11ae11bd119e119211c511c611d811be11b811f311f0),
 .INIT_05(256'h11fd11d911d11205120b1222120b1202123b12461255123412301260a126e1292),
 .INIT_04(256'h1273126812b112be12cc12a512ac12f9130d132612f312f8134f136b1385135e),
 .INIT_03(256'h136613ba13d513ff13d113c9143114611492146c146d14e0150b154815111523),
 .INIT_02(256'h15a315dc163916151611169e17051765173117501806188419141 8dd191c19e5),
 .INIT_01(256'h1a871b621b421b841c9b1d8e1ec51ead1f3e20b021f423ac23e924e72708293e),
 .INIT_00(256'h2b742c9e2e3f31a1356338cd3bb33edd45634ade50cb571560ce65da46b30870)
   )

RAMB16_Tail_Miux5(  .EN(CS[5]),
                    .DO(Data5[15:0]),
                    .DOP(Data5[17:16]),
                    .ADDR(Addr),
                    .CLK(CLK_SYS),
                    .DI(16'hFFFF),
                    .DIP(2'b11),
                    .SSR(1'b0),
                    .WE(1'b0)
                  );



//Miux6
RAMB16_S18 #(
 // The following defparam declarations specify the behavior of the RAM.
 .INIT(36'h000000000),       // Initial values on A output port
 .SRVAL(36'h000000000),      // Set/Reset value for A port output
 .WRITE_MODE("WRITE_FIRST"), // "WRITE_FIRST", "READ_FIRST", or "NO_CHANGE"
 // The following generic INIT_xx declarations specify the initial contents
 // of the RAM
 .INIT_18(256'h10c810c510cc10d110cf10cc10c710cd10d110cf10cd10c910cd10d310d310d2),
 .INIT_17(256'h10cb10cc10d110cf10cd10c610cb10d110cf10cf10c910cc10d110cf10cf10cb),
 .INIT_16(256'h10cf10d410d310d010ca10ce10d310d310d110cc10d010d510d310d010ca10ce),
```

```
 .INIT_15(256'h10d410d210d110cc10d110d610d510d110c910ce10d310d510d510ce10d210d9),
 .INIT_14(256'h10d810d410cc10ce10d510d510d410cf10d310d710d610d210cb10ce10d610d6),
 .INIT_13(256'h10d410cf10d410d910d510d310cc10cf10d710d610d410cf10d410d810d510d3),
 .INIT_12(256'h10cf10d510da10d810d510cf10d510dc10da10d510d210d810dc10d810d510d0),
 .INIT_11(256'h10d510dd10db10d910d410d610da10d810d410ce10d510de10e010dc10d510d8),
 .INIT_10(256'h10dd10da10d610d110d810de10de10dd10d810d910db10d810d710d310d910de),
 .INIT_0F(256'h10dd10db10d410d910dc10db10da10d710df10e510e410de10d610d910df10df),
 .INIT_0E(256'h10da10d610de10e510e410de10d310d710e210e610e310da10df10e510e510e1),
 .INIT_0D(256'h10d710dc10e310e510e410de10e210e910e710e310dc10e110e910ea10e610e0),
 .INIT_0C(256'h10e910f110ef10e810e110e610ee10f110ef10e810ec10f410f510f110ea10ee),
 .INIT_0B(256'h10f410f310ef10eb10f310fb10fa10f310ed10f7110010ff10fb10f310f91104),
 .INIT_0A(256'h1105110310fa10fc11041104110110fc1103110a110b11091103110911131114),
 .INIT_09(256'h110f1108110e1118111a1112110d111411201122111e1115111e112811261122),
 .INIT_08(256'h111a112411311133112e11281133114011431140113611 3d114811481142113a),
 .INIT_07(256'h1144115311591156114f115a116b1170116b1161116c117e118411801176117f),
 .INIT_06(256'h11911196118f1188119611a811ae11a911a111af11c311cb11c811c011cd11e2),
 .INIT_05(256'h11e911e711e111f1120b1213120c120112141236124412401237124d126e1278),
 .INIT_04(256'h12751274128912a912bf12bb12b212cd12f5130d130b13061321134d13691367),
 .INIT_03(256'h1364138613b813d913dc13d913fb1437146214681469149114d5150e1516151d),
 .INIT_02(256'h155115a515eb15fe160d164916b41711173017461796181b189d18d018f9195c),
 .INIT_01(256'h1a091acb1b1d1b5a1bea1cd91de71e6d1edb1fae20f92272235c242c258627a6),
 .INIT_00(256'h29c12b5b2cce2f1232a835fb39073bf740bb46594bdd51a958a05dce3e120770)
  )

RAMB16_Tail_Miux6(  .EN(CS[6]),
                    .DO(Data6[15:0]),
                    .DOP(Data6[17:16]),
                    .ADDR(Addr),
                    .CLK(CLK_SYS),
                    .DI(16'hFFFF),
                    .DIP(2'b11),
                    .SSR(1'b0),
                    .WE(1'b0)
                  );



//Miux7
RAMB16_S18 #(
 // The following defparam declarations specify the behavior of the RAM.
 .INIT(36'h000000000),       // Initial values on A output port
 .SRVAL(36'h000000000),      // Set/Reset value for A port output
 .WRITE_MODE("WRITE_FIRST"), // "WRITE_FIRST", "READ_FIRST", or "NO_CHANGE"
 // The following generic INIT_xx declarations specify the initial contents
 // of the RAM
 .INIT_18(256'h10d510d410d510e510dc10d610d510d410e010d710d310d610d610e210d910d4),
 .INIT_17(256'h10d410d310de10d810d710d710d610e110da10d610d710d510df10da10d710d7),
 .INIT_16(256'h10d710e310dc10d710d610d510e110db10d710d810d710e510dd10d610d610d7),
 .INIT_15(256'h10e410dd10d810d910d910e510dc10d510d510d610e410db10d810d910da10e7),
 .INIT_14(256'h10dd10d810d810d910e710e010db10db10da10e610de10d910d910d910e810e1),
 .INIT_13(256'h10db10d910d810e710e210dc10dc10dc10e910e310dc10da10d910e910e310dd),
 .INIT_12(256'h10dd10de10ea10e210dc10dd10de10ec10e510df10de10de10e910e210dd10de),
 .INIT_11(256'h10df10ed10e410de10e010df10eb10e210e010e210e310ee10e610e110e010de),
```

```
 .INIT_10(256'h10ea10e410e210e410e310ee10e810e410e210df10eb10e610e410e410e310ef),
 .INIT_0F(256'h10ea10e610e410e110ec10e810e510e610e710f210eb10e410e210e310f010ec),
 .INIT_0E(256'h10e810e810e810f710f010e810e610e510f410f010eb10eb10eb10f810f310ed),
 .INIT_0D(256'h10eb10eb10f810f210ef10ee10ee10fd10f510ec10eb10ee10fd10f610f110f0),
 .INIT_0C(256'h10f2110010fa10f410f110f010ff10fa10f610f610f51103110010fa10f810f8),
 .INIT_0B(256'h1106110010fb10fb10fc110b11051101110111011101110d11071103110311041112),
 .INIT_0A(256'h110d11081107110911171110110b110c110d111d11171112111211131123111e),
 .INIT_09(256'h111a11191119112811211111e111e11201130112b11271128112811391136113),
 .INIT_08(256'h113111311141113b113711371139114e114b114811481149115c11571152115),
 .INIT_07(256'h11531165116311621162116411791176117411741176118d118c11891189118c),
 .INIT_06(256'h11a411a2119f119f11a411bd11bb11b911b811bb11d511d411d311d411da11f8),
 .INIT_05(256'h11f811f711f911fd121e122112201223122a124e125012511255125c12841289),
 .INIT_04(256'h128c1291129a12c512cb12d012d312e01312131d132113271336136d137c1384),
 .INIT_03(256'h138b139d13dc13ef13f81405141b1461147e148e149b14b7150d153315491560),
 .INIT_02(256'h158315e5161b163b165a168a170a1755178317ae17ee188f18fb193c197d19dc),
 .INIT_01(256'h1aac1b541bb91c221ca61dbf1eb31f581ffa20c6225523c124de25fc276c29d0),
 .INIT_00(256'h2c062ddb2fc532a536873a513dd9419547484db654645b72636464fd420d081a)
 )

RAMB16_Tail_Miux7(  .EN(CS[7]),
                    .DO(Data7[15:0]),    // 16-bit A port data output
                    .DOP(Data7[17:16]),  // 2-bit A port parity data output
                    .ADDR(Addr),         // 10-bit A port address input
                    .CLK(CLK_SYS),       // 1-bit A port clock input
                    .DI(16'hFFFF),       // 16-bit A port data input
                    .DIP(2'b11),         // 2-bit A port parity data input
                    .SSR(1'b0),          // 1-bit A port set/reset input
                    .WE(1'b0)            // 1-bit A port write enable input
                  );


//Miux8
RAMB16_S18 #(
 // The following defparam declarations specify the behavior of the RAM.
 .INIT(36'h000000000),       // Initial values on A output port
 .SRVAL(36'h000000000),      // Set/Reset value for A port output
 .WRITE_MODE("WRITE_FIRST"), // "WRITE_FIRST", "READ_FIRST", or "NO_CHANGE"
 // The following generic INIT_xx declarations specify the initial contents of
 // the RAM
 .INIT_18(256'h38dc38c339013943392438d638c93904393d392338d838bd38fc393f391f38d2),
 .INIT_17(256'h38c33900393e392838dd38bf39003943392438d638c839073943392838de38c0),
 .INIT_16(256'h38fc393e392138d538c43903393e392538dd38c339023946392938db38c63908),
 .INIT_15(256'h3942392438d738bf38fc393c391e38d338c139053943392738dc38c439023944),
 .INIT_14(256'h392738d838c639093944392838dc38c139003943392838dc38c2390539413926),
 .INIT_13(256'h38dd38c739043941392738da38bd38ff393e392038d738c43900393f392638da),
 .INIT_12(256'h38c439073940392038d938be38fa393a392038d338be39053946392938e138cb),
 .INIT_11(256'h39063945392a38db38bd38fe393c391e38d638c038ff393f392938de38c23904),
 .INIT_10(256'h3940391d38d238c038fd393d392838dd38c339053941391f38d438be38fc3937),
 .INIT_0F(256'h391d38d238b838f93938391b38d338c039013940392b38e538c9390b39473925),
 .INIT_0E(256'h38d838bd38f73931391838d038b738fd3941392738e038cc390a3944392938de),
 .INIT_0D(256'h38bf38fd393b391b38d038bb38fe393b392338de38c639043941392338d538bc),
 .INIT_0C(256'h3900393c392338e138c839083948392c38e338cb390b3947392a38e838d33914),
 .INIT_0B(256'h3956393b38f138db391e3958393b38fb38e43923396739513910390039 3f397a),
```

```
 .INIT_0A(256'h395e391f390f394f3992397b393e3935397439b1399a39653962399f39e039cc),
 .INIT_09(256'h399a399b39d63a163a0239d839e13a193a5b3a493a243a303a663aa33a923a74),
 .INIT_08(256'h3a853ab33aed3adc3ab73ac33af23b223b153af93af93b203b523b493b263b28),
 .INIT_07(256'h3b543b843b7c3b5d3b613b8e3bbc3bb43b993b9d3bce3c013bfb3bdf3bea3c19),
 .INIT_06(256'h3c4e3c493c323c403c713ca83ca73c943ca43cda3d143d133d013d163d543d90),
 .INIT_05(256'h3d983d873d9b3dde3e243e303e273e3f3e813ece3ee23edf3efe3f4c3fa13fbb),
 .INIT_04(256'h3fbc3fe4403b409640b340bd40e8414641af41de41f2422c4298430f43474367),
 .INIT_03(256'h43b3442e44b044f745274584461846b3470b474e47bb48614916498649da4a5b),
 .INIT_02(256'h4b204bec4c724cde4d7f4e5f4f515000508e5152526053825465552b56315790),
 .INIT_01(256'h58d559e45af05c6a5dfc5f3d603f6132625a6368648665846660670e6799684d),
 .INIT_00(256'h68e6694569a96a2a6ae56b826be16c436cbf6d6d6df26e276e3f6db348c008aa)
  )

RAMB16_Tail_Miux8(  .EN(CS[8]),
                    .DO(Data8[15:0]),
                    .DOP(Data8[17:16]),
                    .ADDR(Addr),
                    .CLK(CLK_SYS),
                    .DI(16'hFFFF),
                    .DIP(2'b11),
                    .SSR(1'b0),
                    .WE(1'b0)
                  );


//Miux9
RAMB16_S18 #(
 // The following defparam declarations specify the behavior of the RAM.
 .INIT(36'h000000000),       // Initial values on A output port
 .SRVAL(36'h000000000),      // Set/Reset value for A port output
 .WRITE_MODE("WRITE_FIRST"), // "WRITE_FIRST", "READ_FIRST", or "NO_CHANGE"
 // The following generic INIT_xx declarations specify the initial contents of
 // the RAM
 .INIT_18(256'h38cd38c53937391c38f438cb38c239353928390838db38cf3943392e390838db),
 .INIT_17(256'h38d539403931390f38da38cc39473936390e38e338db39483933390e38dd38c6),
 .INIT_16(256'h393a392d390438d838d3393d392f391238e738d9394f393d391538e038dd394f),
 .INIT_15(256'h393b391638e938d33946393f391b38e138e1394c3930390838dc38c239353931),
 .INIT_14(256'h390638cf38d439463932390f38e838d03940393b391938dc38de395239353918),
 .INIT_13(256'h38f338d7394c3949391a38d538dc39453927390b38e038cd39393932391338dd),
 .INIT_12(256'h38d7394a392a390438da38c1392e392c390938cc38ca393f392638ff38da38ce),
 .INIT_11(256'h39343931390d38ca38c539393928390938de38d93943393b392038de38d6394d),
 .INIT_10(256'h3934390c38df38de39483943392938e638d239453935390e38dc38d5393b392f),
 .INIT_0F(256'h390e38d338d239453932390b38d638ca392f3926390a38cd38c4393c39333910),
 .INIT_0E(256'h38d838da39413936391938d538c8393b3924390038d038ca3934392b390c38c8),
 .INIT_0D(256'h38c53934392138fd38c738bd392b392e391138d238da3948392e391238e338d4),
 .INIT_0C(256'h39413940392138dc38eb3955393d391e38e338c739353939391a38d338df394f),
 .INIT_0B(256'h393a392238fa38e439503955393638f038fa3967394c3934390f3907396b396f),
 .INIT_0A(256'h39593917392139923978396939 4e394739a839b539ae3970397b39e939ca39c5),
 .INIT_09(256'h39b439a63a113a1e3a0b39df39ee3a583a4d3a513a4a3a303a883a943a843a6e),
 .INIT_08(256'h3a883ad93acf3ac33abc3ac33b103b1d3b0f3b053b1d3b623b803b663b573b74),
 .INIT_07(256'h3baa3bc73ba33b993bb63bef3c0f3bee3bda3bff3c403c593c363c293c513c89),
 .INIT_06(256'h3ca83c923c843ca43cec3d1f3d0b3cfc3d223d6d3d8a3d6c3d6c3da43deb3e17),
 .INIT_05(256'h3e023e053e383e8a3ec53eb63ebb3ef03f403f823f7c3f8c3fc84025406c406e),
```

```
 .INIT_04(256'h408440cc4138418b419141b5420e428442e1430243343934417447d449f44dd),
 .INIT_03(256'h454c45eb466e46a446fa4780484048d2491749824a174ae14b954bf54c754d2a),
 .INIT_02(256'h4e124ee84f654ffd50e251fa52f253895458559b56f8580d58fb5a1e5bac5d46),
 .INIT_01(256'h5e8f5f72608a61b962c563f264dd65d666a0675d6827689268f2695a69fd6ac3),
 .INIT_00(256'h6b2b6b906c056ca46d6e6dd76e366e986f276fdf701b704c706570b64a75080b)
  )

RAMB16_Tail_Miux9(  .EN(CS[9]),
                    .DO(Data9[15:0]),
                    .DOP(Data9[17:16]),
                    .ADDR(Addr),
                    .CLK(CLK_SYS),
                    .DI(16'hFFFF),
                    .DIP(2'b11),
                    .SSR(1'b0),
                    .WE(1'b0)
                  );
endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
//       Generate the tail_curr depend on line encoded status and trans outputs
//
//////////////////////////////////////////////////////////////////////////////////

module Tail_Curr_Generation(input  [17:0] Tail1,
                                    input  [17:0] Tail2,
                                    input  [17:0] Tail3,
                                    input  [2:0]  L1S,
                                    input  [2:0]  L2S,
                                    input  [2:0]  L3S,

                                    output [17:0] Tail_Curr );


// internal signals declaration

wire L1S_In, L2S_In, L3S_In;

// generate 3-sate gate control

assign L1S_In = L1S[2] & L1S[1] &(~L1S[0]),
       L2S_In = L2S[2] & L2S[1] &(~L2S[0]),
       L3S_In = L3S[2] & L3S[1] &(~L3S[0]);

//  3-state data path sharing

 bufif0 bufif0_1 [17:0] (Tail_Curr, Tail1,     (~L1S_In));
 bufif0 bufif0_2 [17:0] (Tail_Curr, Tail2,     (~L2S_In));
 bufif0 bufif0_3 [17:0] (Tail_Curr, Tail3,     (~L3S_In));
 bufif0 bufif0_4 [17:0] (Tail_Curr, 18'h00000, (L1S_In | L2S_In |
                         L3S_In));
```

```
endmodule


‘timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//Generate the tail_curr depend on line encoded status and trans outputs
//
////////////////////////////////////////////////////////////////////////////////


module Tail_Sum_Generation(input  [2:0]  L1S,
                           input  [2:0]  L2S,
                           input  [2:0]  L3S,
                           input  [17:0] Tail1,
                           input  [17:0] Tail2,
                           input  [17:0] Tail3,
                           input         CLK_4_4_1,
                           input         CLK_4_4_2,
                           output [17:0] Tail_Sum    );

//******************************************************
// internal signals declaratin
//******************************************************

 wire       L3S_In;
 reg        L1S_Pipe   = 3'b000,
            L2S_Pipe   = 3'b000;
 reg [17:0] Tail1_Pipe = 18'h00000,
            Tail2_Pipe = 18'h00000;
 reg [17:0] Sum1 = 18'h00000,
            Sum2       = 18'h00000;


//******************************************************
//                Pipeline control
//******************************************************

always @ ( posedge CLK_4_4_1 )
  begin
      L1S_Pipe <= (~L1S[2]) & L1S[1] & (~L1S[0]);
      L2S_Pipe <= (~L2S[2]) & L2S[1] & (~L2S[0]);
      Tail1_Pipe <= Tail1;
      Tail2_Pipe <= Tail2;
  end

always @ ( posedge CLK_4_4_2 )
  begin
     Sum1 <= (Tail1_Pipe & {18{L1S_Pipe}}) + (Tail2_Pipe & {18{L2S_Pipe}});
     Sum2 <= Tail3 & {18{L3S_In}};
  end
```

```
//*********************************************************
//        generate tail_sum
//*********************************************************

assign L3S_In = (~L3S[2]) & L3S[1] & (~L3S[0]),
       Tail_Sum = Sum1 + Sum2;

endmodule


`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
//    Generate pre offset estimate depend on steady status output of transient
//    ,encoded line status, and the offset feed back from block5 (accummulation)
//
////////////////////////////////////////////////////////////////////////////////

module Offet_Generation(input [15:0] Tail1_Output,
                        input        L1S1,
                        input        L1S0,
                        input [15:0] Tail2_Output,
                        input        L2S1,
                        input        L2S0,
                        input [15:0] Tail3_Output,
                        input        L3S1,
                        input        L3S0,
                        input [15:0] Offset_FB,
                        input        CLK_4_4_1,
                        input        CLK_4_4_2,
                        output[15:0] Pre_Offset
                                );

//*********************************************************
// Internal signals declareation
//*********************************************************

 reg [15:0] Tail1_Output_Pipe = 16'h0000;
 reg [15:0] Tail2_Output_Pipe = 16'h0000;
 reg [15:0] Tail3_Output_Pipe = 16'h0000;
 reg [15:0] Sum1_Pipe = 16'h0000;
 reg [15:0] Sum2_Pipe         = 16'h0000;
 reg        L1S0_Pipe = 1'b0;
 reg        L1S1_Pipe         = 1'b0;
 reg        L2S0_Pipe = 1'b0;
 reg        L2S1_Pipe = 1'b0;
 reg        L3S0_Pipe = 1'b0;
 reg        L3S1_Pipe = 1'b0;
 wire[15:0] O1, O2, O3, Sum1, Sum2;
 reg [15:0] Offset_FB_Pipe = 16'h0000;

//*********************************************************
//    pipe line operations
```

```
//*********************************************************

always @ (posedge CLK_4_4_1)
  begin
      Tail1_Output_Pipe <= Tail1_Output;
      Tail2_Output_Pipe <= Tail2_Output;
      Tail3_Output_Pipe <= Tail3_Output;
      L1S0_Pipe         <= L1S0;
      L1S1_Pipe         <= L1S1;
      L2S0_Pipe         <= L2S0;
      L2S1_Pipe         <= L2S1;
      L3S0_Pipe         <= L3S0;
      L3S1_Pipe         <= L3S1;
      Offset_FB_Pipe    <= Offset_FB;
  end

always @ (posedge CLK_4_4_2)
  begin
    Sum1_Pipe <= Sum1;
    Sum2_Pipe <= Sum2;
  end

//*********************************************************
//   accumulate the offset
//*********************************************************

assign O1  =  Tail1_Output_Pipe & {16{L1S0_Pipe}} &
              {16{L1S1_Pipe}},
       O2  =  Tail2_Output_Pipe & {16{L2S0_Pipe}} & {16{L2S1_Pipe}},
       O3  =  Tail3_Output_Pipe & {16{L3S0_Pipe}} & {16{L3S1_Pipe}},
     Sum1  =  O1 + Offset_FB_Pipe,
     Sum2  =  O2 + O3,
     Pre_Offset =  Sum1_Pipe + Sum2_Pipe;

endmodule


`timescale 1ns / 1ps
////////////////////////////////////////////////////////////////////////////////
//
//  Generate pipeline control clks and data path and resource multiplexing
//  controls
//  Time resolution:  one pipeline control clk include 32 system clock
//                    256 pipeline control clks => 1 step in time increasing
//
//  Two modules under this level: 14 and 18 bit synchro, counter
//                                          syn_counter_14,
//                                          syn_counter_18
////////////////////////////////////////////////////////////////////////////////


module pipeline_generation(         input          CLK_SYS,
                                    input          Mode_Sel,
```

```
                                    input          Alert,
                                    output  [17:0] N,
                                    output         Am_Multi_Sel,
                                    output  [1:0]  Tran_Sel,
                                    output  [2:0]  Tail_Sel,
                                    output  [15:0] GCLK0,
                                    output  [15:0] GCLK1,
                                    output  [15:0] GCLK2,
                                    output  [15:0] GCLK3,
                                    output  [15:0] GCLK4,
                                    output  [15:0] GCLK5,
                                    output  [15:0] GCLK6,
                                    output  [15:0] GCLK7,
                                    output  [15:0] GCLK8,
                                    output  [15:0] GCLK9,
                                    output  [15:0] GCLK10,
                                    output  [15:0] GCLK11,
                                    output  [15:0] GCLK12,
                                    output  [15:0] GCLK13,
                                    output  [15:0] GCLK14,
                                    output  [15:0] GCLK15  );

//*****************************************************
//     internal signals
//*****************************************************

 wire   [13:0]  S;
 wire   [15:0]  SCLK;
 wire           Lock;

//*****************************************************
//               Mode Select
//*****************************************************

assign   Lock = (~Mode_Sel) | Alert;

//*****************************************************
//     14 bit synchronized counter
//*****************************************************

syn_counter_14 counter14_inst (.C(CLK_SYS),
                               .CLR(1'b0),
                               .Q(S));

//*************************************************************
//     18 bit synchronized counter   negative edge trigger
//*************************************************************

syn_counter_18 counter18_inst (.C(S[12]),
                               .CLR(1'b0),
                               .Q(N));

//*****************************************************
```

```
//       4 -- 16 Decoder
//*****************************************************

Decoder_4_16 decoder16_inst(.S(S[8:5]),
                            .SCLK(SCLK));

//*****************************************************
//              GCLK generation
//*****************************************************

 assign GCLK0 = (S[12:9] == 4'h0) ? SCLK : 16'h0000;
 assign GCLK1 = (S[12:9] == 4'h1) ? ({16{Lock}} & SCLK) : 16'h0000;
 assign GCLK2 = (S[12:9] == 4'h2) ? ({16{Lock}} & SCLK) : 16'h0000;
 assign GCLK3 = (S[12:9] == 4'h3) ? ({16{Lock}} & SCLK) : 16'h0000;
 assign GCLK4 = (S[12:9] == 4'h4) ? ({16{Lock}} & SCLK) : 16'h0000;
 assign GCLK5 = (S[12:9] == 4'h5) ? ({16{Lock}} & SCLK) : 16'h0000;
 assign GCLK6 = (S[12:9] == 4'h6) ? ({16{Lock}} & SCLK) : 16'h0000;
 assign GCLK7 = (S[12:9] == 4'h7) ? ({16{Lock}} & SCLK) : 16'h0000;
 assign GCLK8 = (S[12:9] == 4'h8) ? ({16{Lock}} & SCLK) : 16'h0000;
 assign GCLK9 = (S[12:9] == 4'h9) ? ({16{Lock}} & SCLK) : 16'h0000;
 assign GCLK10 = (S[12:9] == 4'hA) ? SCLK : 16'h0000;
 assign GCLK11 = (S[12:9] == 4'hB) ? SCLK : 16'h0000;
 assign GCLK12 = (S[12:9] == 4'hC) ? SCLK : 16'h0000;
 assign GCLK13 = (S[12:9] == 4'hD) ? SCLK : 16'h0000;
 assign GCLK14 = (S[12:9] == 4'hE) ? SCLK : 16'h0000;
 assign GCLK15 = (S[12:9] == 4'hF) ? SCLK : 16'h0000;


//*****************************************************
//                Switching signal generations
//*****************************************************


 assign Tail_Sel = S[9:7];
 assign Tran_Sel = S[11:10];
 assign Am_Multi_Sel = S[7];


endmodule


`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    03:00:24 04/13/06
// Design Name:
// Module Name:    syn_counter_14
// Project Name:
// Target Device:
// Tool versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module syn_counter_14 (C, CLR, Q);

 input C, CLR;
 output [13:0] Q;

 reg [13:0] tmp = 14'h0000;

always @(posedge C or posedge CLR)
  begin
    if (CLR)
      tmp <= 14'h0000;
    else
      tmp <= tmp + 1'b1;
    end

assign Q = tmp;

endmodule

`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    10:58:10 04/14/06
// Design Name:
// Module Name:    syn_counter_18
// Project Name:
// Target Device:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module syn_counter_18 (C, CLR, Q);

input C, CLR;
```

```
output [17:0] Q;

reg [17:0] tmp = 18'h00000;

always @(negedge C or posedge CLR)
  begin
    if (CLR)
      tmp <= 18'h00000;
    else
      tmp <= tmp + 1'b1;
    end

assign Q = tmp;

endmodule
```