ENABLING RAPID PROTOTYPING OF AUDIO SIGNAL PROCESSING SYSTEMS USING SYSTEM-ON-CHIP FIELD PROGRAMMABLE GATE ARRAYS

by

Trevor Charles Vannoy

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

of

Master of Science

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

April 2020

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Dr. Ross Snider, for all of his guidance throughout the years; I have enjoyed every project we have worked on together, and look forward to future collaborations. I'd also like to thank the rest of my committee: Dr. Rob Maher, Dr. Brock LaMeres, and Dr. Brad Whitaker. Much of my learning has been facilitated by them. I'd like to thank Dr. Wataru Nakagawa for all of his advice over the years, and for keeping me sane with conversation and free food.

I would like to thank all of my colleagues at Flat Earth Inc: Tyler Davis for being a great mentor and PI; Connor Dack for reviewing all of my mediocre circuit board designs; and Dustin Sobrero for getting me into mechanical keyboards, and for doing all of the user interface programming that I can't stand. It has been fun collaborating with all of you. I'd also like to thank all of the interns and capstone students I've worked with for letting me pawn work off on them; in particular, I'd like to thank Aaron Koenigsberg for working on code autogeneration and system admin tasks with me.

Finally, I'd like to thank all of my friends and family for their love and support, especially when times have been tough—you mean everything to me.

This work was funded by a College of Engineering Benjamin Fellowship and the National Institutes of Health under award number R44DC015443.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

LIST OF FIGURES

Figure Page

LIST OF FIGURES – CONTINUED

Figure Page

LIST OF LISTINGS

Listing Page

ABSTRACT

System-on-Chip Field Programmable Gate Arrays are excellent devices for high performance, low latency signal processing. Unfortunately, they are notoriously difficult to use, requiring significant hardware and software engineering expertise. To address these challenges, a development framework is created that utilizes graphical programming and automatic code generation; this framework reduces development time and reduces the need to be an expert in Field Programmable Gate Arrays. A sound effects processor and a real-time audio beamformer were created to showcase the development framework and serve as reference designs for other developers. The development framework, coupled with open source audio hardware, enables both experts and non-experts to rapidly prototype audio signal processing systems using System-on-Chip Field Programmable Gate Arrays.

1

INTRODUCTION

As algorithms and applications are becoming more computationally intensive, hardware and software performance demands are increasing. Historically, Moore's Law [1] and Dennard Scaling [2] have resulted in significant CPU performance increases every year. As transistors keep getting smaller, this performance scaling is slowing down for a variety of reasons, including power dissipation and thermal runaway [3, 4]. CPU performance trends can be seen in Fig. 1.1, which shows single thread performance plateauing in spite of increasing numbers of transistors. Given these trends, parallel computer architectures are a good way to increase performance.

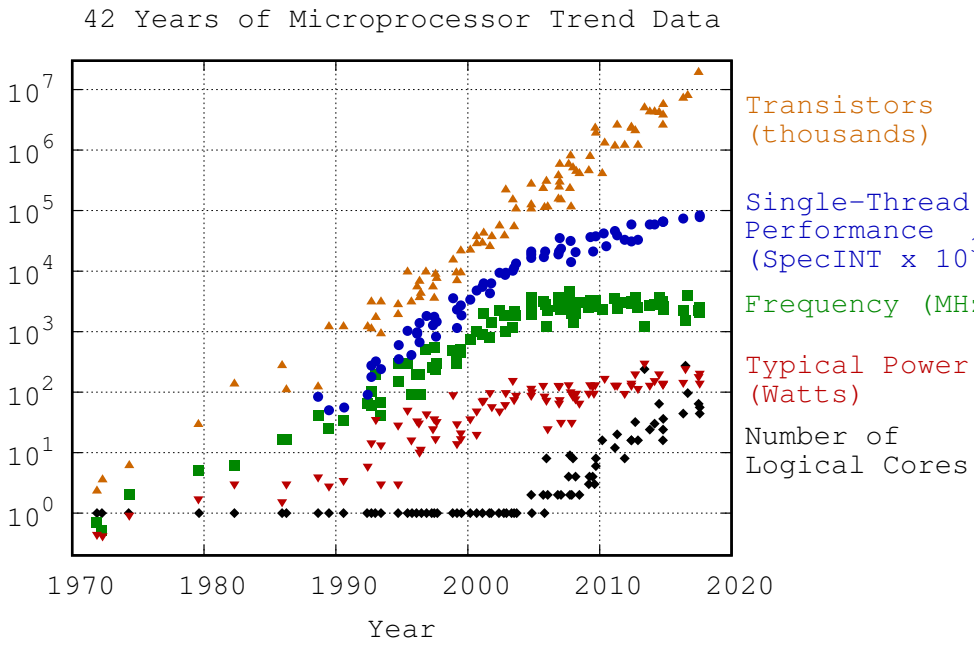


Figure 1.1: Microprocessor trends [5]. Single thread performance is plateauing in spite of increasing number of transistors. Original data up to the year 2010 collected by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten. New data collected by K. Rupp.

Due to their inherently parallel nature, System-on-Chip Field Programmable Gate Arrays (SoC FPGAs) are excellent devices for high performance, low latency audio signal processing—especially streaming audio applications. In fact, many commercial audio devices are already using FPGAs, such as the Waldorf Kyra synthesizer[1] and the Novation Peak synthesizer[2].

While SoC FPGAs are great for high performance audio and digital signal processing (DSP), they are notoriously difficult to develop with. The development process requires significant hardware and software expertise; as a result, SoC FPGA teams often have multiple hardware and software engineers, each of which are experts in different parts of the development process. For non-experts, the development process is often intractable. Unfortunately, many developers of audio processing algorithms—audio engineers, audiologists, speech scientists, etc.—are not FPGA engineers, so they usually don't use SoC FPGAs unless they are part of a larger development team.

Realizing that this inaccessibility limits the adoption of SoC FPGAs, this research seeks to increase productivity and accessibility for non-experts by creating a development framework and open source hardware. This research is part of an NIH SBIR grant [6] to develop an open source speech signal processing platform [7–9] to accelerate research and development of novel algorithms for hearing aids, cochlear implants, and consumer electronic devices.

The development framework utilizes model-based design and automatic code generation, eliminating the need to implement algorithms in low-level hardware description languages (HDLs). Additionally, device drivers and a control application are automatically created to allow real-time control of algorithm parameters. Automatic

[1] `https://waldorfmusic.com/en/kyra`
[2] `https://novationmusic.com/en/peak-explained`

code generation significantly speeds up the SoC FPGA development process.

The open source hardware comprises three main designs: a low-cost platform intended for students, a high-performance platform for general research and development, and a high-performance platform for beamforming applications. Connor Dack at Flat Earth Inc.[3] developed the first two, which will be discussed later in the Background chapter. I developed the beamforming hardware, which will be discussed during the Real-time Beamformer chapter.

My primary contributions in this thesis are

- Design of an automatic code generation framework for DSP applications using SoC FPGAs
- Creation of an SoC FPGA-based sound effects processor example design
- Design of a hardware system for real-time beamforming
- Creation of a delay and sum beamforming example design

The first two contributions were presented in part at the 147$^{\text{th}}$ Audio Engineering Society Convention [8].

The rest of this manuscript is organized as follows: chapter 2 discusses relevant background and context needed to understand this thesis; chapter 3 discusses the development framework; chapter 4 demonstrates a sound effects processor developed with the development framework and deployed on the low-cost hardware platform; chapter 5 shows a real-time beamformer application; and finally, chapter 6 discusses future work and conclusions.

[3]Flat Earth Inc. is *not* associated with modern Flat Earth societies (a.k.a. "Flat Earthers"). In fact, the CEO has a background in Geodesy, and spent years proving that the Earth is, in fact, round. The company name is largely a pun and a marketing ploy.

4

BACKGROUND

SoC FPGAs

An FPGA is an integrated circuit that contains a two-dimensional array of logic blocks that can be connected together via programmable interconnects, as shown in Fig. 2.1. The programmable interconnects are typically referred to as the FPGA "fabric". These devices can be reprogrammed at any time in the field after manufacturing. An example of a programmable interconnect architecture is shown in Fig. 2.2.

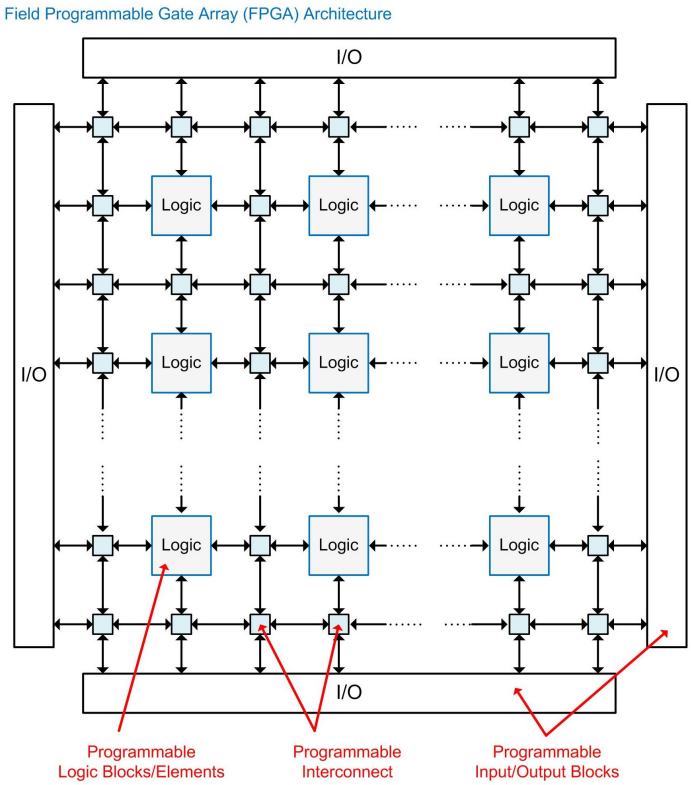


Figure 2.1: Simplified FPGA architecture, showing logic blocks, interconnect, and I/O banks. Image courtesy of [10].

Modern FPGAs often have multiple different types of logic blocks. The most

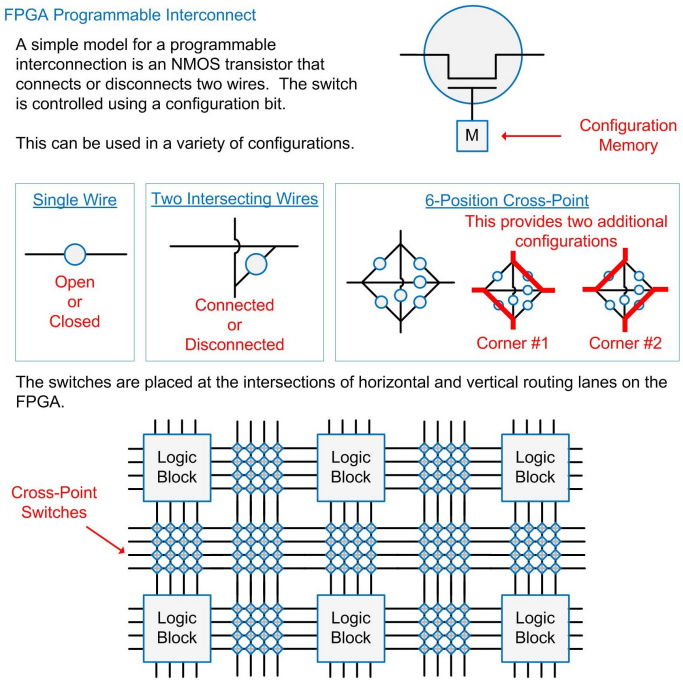


Figure 2.2: Example architecture of FPGA programmable interconnect using 6-position cross-point switches based upon NMOS transistors. Image courtesy of [10].

common logic blocks contain varying numbers of lookup tables, registers, and adders; the makeup of these logic blocks vary by manufacturer and device family. Fig. 2.3 shows a basic logic block; a more complicated logic block—the Adaptive Logic Module from an Intel Cyclone V [11]—is shown in Fig. 2.4. Other common blocks include memory blocks and DSP slices, which contain hardened multipliers and adders.

Any number of logic elements can be active at the same time, thus computation in the FPGA fabric is inherently parallel. The parallel nature of FPGAs means that more performance can be gained by simply increasing the number of logic elements.

A hardware description language (HDL), such as VHDL or Verilog, is used to describe the circuits that will be programmed on the FPGA. Although HDLs resemble traditional software languages, the key difference is that they describe hardware, and thus statements are, in general, not sequential. The logic described by HDL code is

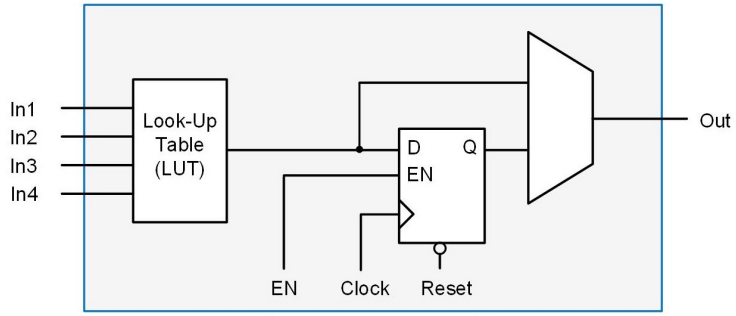


Figure 2.3: A basic logic block comprising a lookup table, D flip-flop, and multiplexer. Image courtesy of [10]

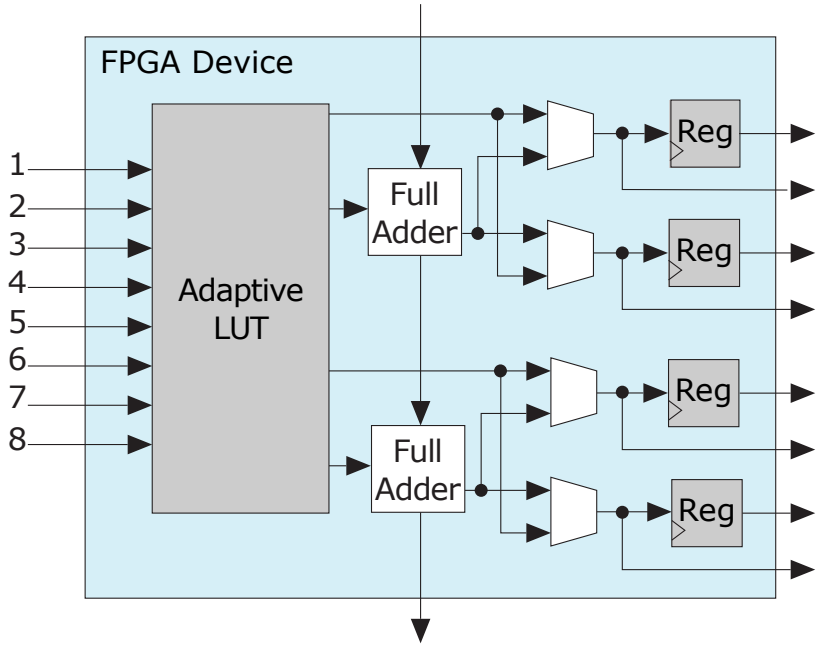


Figure 2.4: An Intel Cyclone V Adaptive Logic Module comprising an 8-input lookup table, two full adders, 4 multiplexers, and 4 registers [11]

converted into a bitstream by the FPGA vendor's tools; this bitstream file configures the FPGA fabric with the desired functionality.

System-on-Chip (SoC) FPGAs, shown in Fig. 2.5, combine the FPGA fabric with a full computer system, known as the Hard Processor System (HPS) in Intel FPGAs. The computer system contains CPUs, caches, peripherals, and a synchronous dynamic

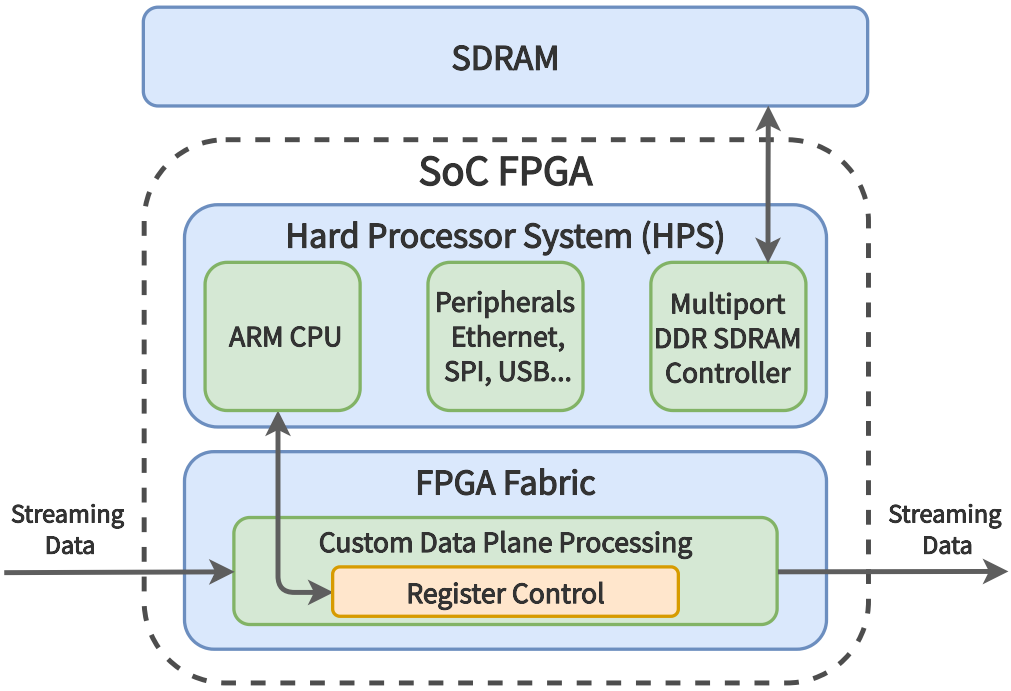


Figure 2.5: SoC FPGAs contain the FPGA fabric as well as a complete computer system.

random-access memory (SDRAM) controller, among other things. As opposed to soft core processors, which are built with logic elements in the FPGA fabric, the HPS is built, or "hardened", into the integrated circuit itself. A chip floorplan—showing the HPS, logic block types, and other hardened elements—for an Intel Cyclone V FPGA is shown in Fig. 2.6.

Linux is often the operating system of choice for the HPS. It is relatively small, modular, and open source, all of which make it a good fit for an embedded system. Higher level software—such as web servers, control and communication interfaces, and even full desktop environments—are run on Linux, while real-time algorithms are handled in the FPGA fabric.

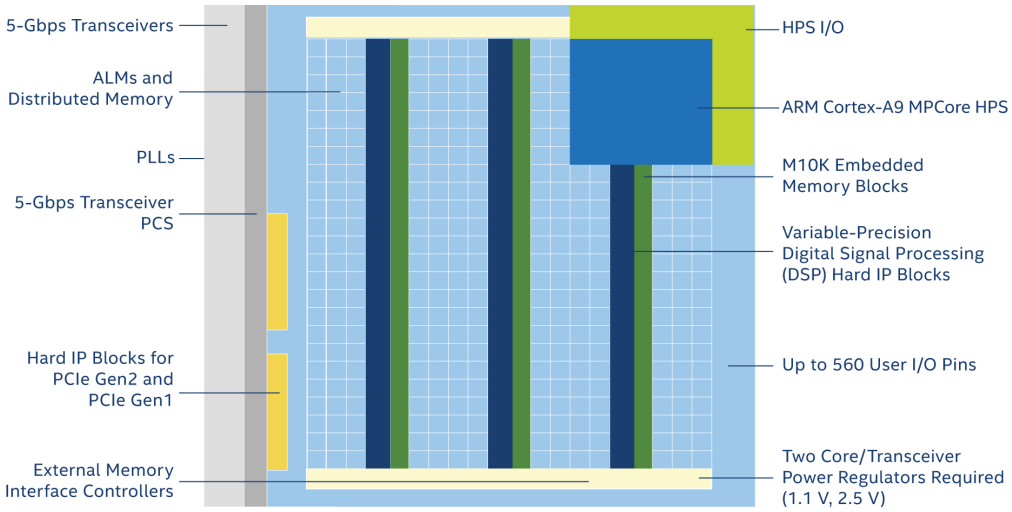


Figure 2.6: Floorplan for an Intel Cyclone V FPGA. The floorplan shows where all of the hardened elements and various logic blocks are located on the integrated circuit. Image courtesy of Intel [12].

Low Latency

Unlike traditional processors, where external data needs to get stored in memory, data interfaces can be connected directly to the FPGA fabric where the computation happens; consequently, FPGAs provide the lowest possible latency of any general purpose computational device.

FPGAs provide deterministic latency to the accuracy of a single clock cycle; the latency from point A to point B in an FPGA computation never changes. Contrast this to traditional CPUs, where cache misses, DRAM refresh cycles, and operating system scheduling all contribute to nondeterministic latency. For real-time signal processing, deterministic latency, though not always needed, can be a major advantage.

High Performance DSP

In DSP, one of the most common operations is multiply-accumulate. FPGAs can have thousands of multipliers, adders, and memory blocks running in parallel, resulting in very high performance. An Intel Agilex AGF 027 provides up to 8,736 floating-point DSP blocks [13], each of which contain one single-precision floating-point multiplier and one adder [14]. This FPGA provides 11.8 *trillion* single-precision floating-point operations per second (FLOPS). In comparison, an Intel i7-8700K CPU provides 710.4 *billion* single-precision floating-point operations per second [15].

As a more modest, and significantly cheaper example, an Intel Cyclone V 5CSXC6 contains 112 fixed-point DSP blocks [16].

Audio Signal Processing Systems

Traditionally, audio signal processing systems are implemented on Digital Signal Processors (DSP chips). These chips are microcontrollers whose architecture is specifically designed and optimized for DSP applications. For example, they often contain a multiply-accumulate instruction that can be performed in one clock cycle.

Since DSP is rarely the only thing an embedded system has to do, it is common to run a real-time operating system (RTOS) on a DSP chip. This enables single-threaded multitasking, which is not the same as running multiple tasks in parallel. In an RTOS, each task has a priority that is set by the user. The RTOS uses these priorities to help schedule which task is running on the CPU at any given time. If a signal processing task gets interrupted by a higher priority task, the latency of the signal being processed is increased.

Performing DSP with FPGAs, on the other hand, provides deterministic latencies that never change. Additionally, an arbitrary number of signal processing

chains can be run on an FPGA in parallel at the same time. Unlike microprocessors, where number of CPU cores and memory bandwidth limits the amount of achievable parallelism, the only limitation on FPGAs is the number of parallel signal processing chains that can fit in the FPGA fabric; if more signal chains are needed, one can optimize the size of the signal chains or simply buy a larger FPGA. However, this flexibility and performance comes at the cost of increased development complexity; DSP chips are typically easier to program than FPGAs.

Model-based Design

Model-based design is a design process that starts with a computational model of the system, which often encompasses multiple disciplines (mechanical, electrical, chemical, etc.). Starting at a high abstraction level allows engineers to focus on the behavior and specifications of the system, rather than the implementation details. Models are typically developed and simulated in graphical programming languages, such as Simulink.

In many situations, especially in embedded and automotive systems, automatic code generation is used to convert the model into code that can be used to deploy the system. Utilizing automatic code generation, changes to the model are automatically reflected in the deployed design, which can significantly reduce time-to-market. Using model-based design, General Motors saved an estimated 24 months of development time on a hybrid powertrain [17].

Printed Circuit Board Design

When creating custom circuits, one has three primary options: wire individual components together on a breadboard, solder components together on a perfboard,

or design a printed circuit board (PCB). The first two options work fine for projects with relatively few components, but projects with many components, high pin counts, or tight size constraints are best done with a PCB. A PCB consists of copper layers separated by insulating material. Circuit connections between components are created on the copper layers, and unwanted copper is removed.

The process of designing a PCB comprises schematic entry, component layout, routing, manufacturing, and assembly. Schematic entry is the process of creating a circuit schematic in an electrical computer-aided-design program such as Altium, Eagle, or KiCad. Once the schematic is created, the circuit components are arranged; careful layout of components makes the next step, routing, much easier. With the components laid out, connections between components are routed. After all connections are made, manufacturing files are exported and sent to a PCB manufacturer. Once the PCBs are manufactured, the boards are assembled by soldering the components onto their pads. For a general overview of PCBs, see [18]. See [19] for best practices when creating PCBs with both analog and digital circuits.

Open Speech Hardware Platforms

The open speech platform [7] has two hardware platforms: a low-cost platform for students, and a high performance platform for researchers. Both platforms comprise a commercial-off-the-shelf FPGA development board with custom daughter cards designed by Flat Earth Inc.[1]

[1] Again, Flat Earth Inc. is *not* associated with modern Flat Earth Societies. They believe the Earth is round. Refer to the footnote on page 3.

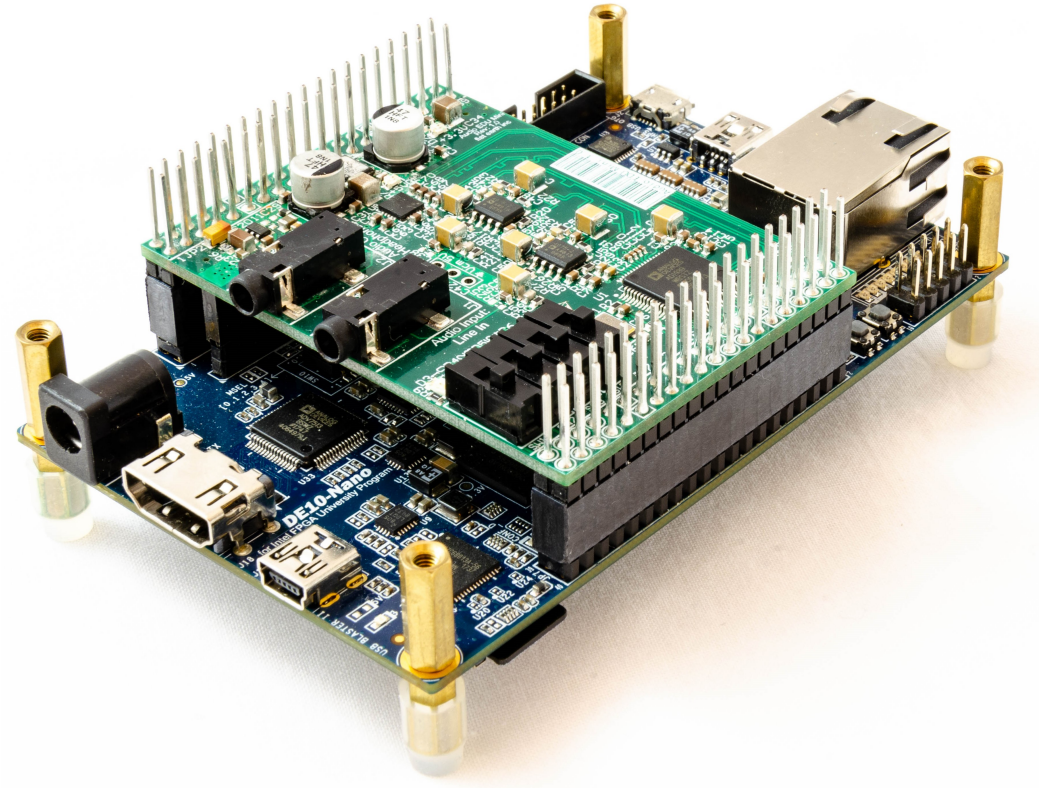


Figure 2.7: Low-cost open source hardware platform comprising a DE10-Nano development board and a Flat Earth Audio Mini daughter card (green PCB).

Low-cost Hardware

The low-cost platform is based upon a Terasic DE10-Nano FPGA development board[2], which has a Cyclone V SoC FPGA on it. A custom daughter card, shown in Fig. 2.7, has been designed by Connor Dack at Flat Earth Inc.

The architecture of the printed circuit board is shown in Fig. 2.8. Audio gets passed into the 3.5 mm line in audio jack, digitized by the AD1939 codec, then sent to the FPGA for processing. Once all processing is done, audio is sent back to the codec and converted to an analog signal before being sent out the 3.5 mm headphone jack. The board has a flat frequency response above 400 Hz, as shown in Fig. 2.9.

[2] https://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=1046

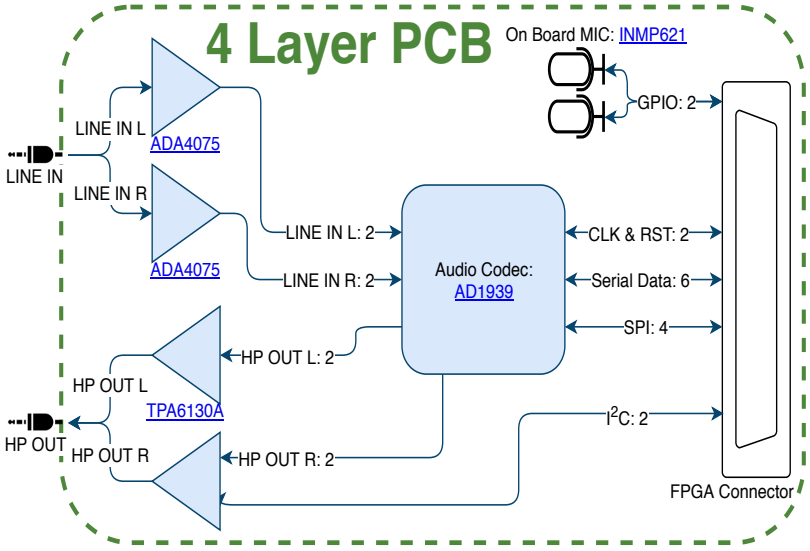


Figure 2.8: Block diagram of the Flat Earth Audio Mini.

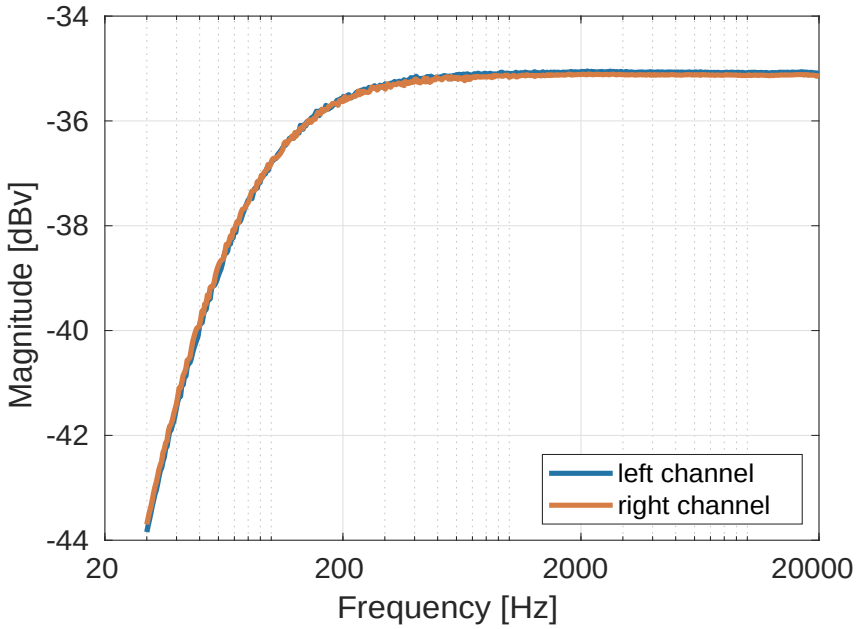


Figure 2.9: Frequency response of the Audio Mini.

Low Latency With the codec running at a 192 kHz sampling rate, a total latency of 180 μs was measured. This is around 10 times faster than the fastest traditional

desktop operating systems [20] and embedded Linux devices [21].

High Performance Hardware

A high performance platform is also being developed by Flat Earth Inc. It was originally based on a Reflex CES Achilles Arria 10 SoC SoM[3]. This Arria 10 has 1687 single-precision floating point DSP slices for use in advanced DSP algorithms. The preliminary daughter card design is shown in Fig. 2.10.

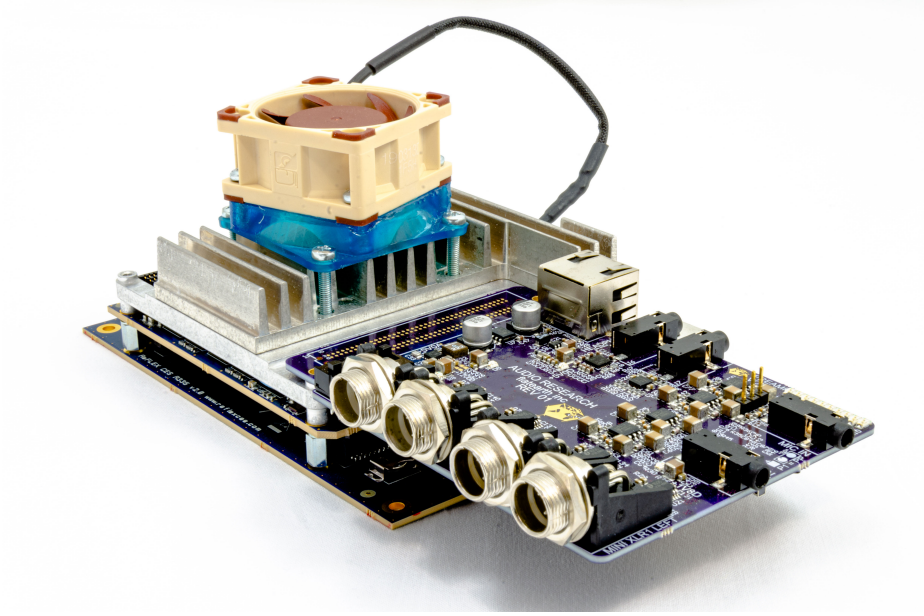


Figure 2.10: Audio Research daughter card (purple PCB) on a Reflex Arria 10 SoC SoM.

This preliminary design, along with the hardware discussed in the Real-time Beamforming System section of the Real-time Beamformer chapter, laid the groundwork for the next revision, which uses a different Arria 10 development board and is under development as of this writing.

[3] https://www.reflexces.com/intel-fpga/arria-10-intel-fpga/arria-10-soc-som

FPGA Open Speech Tools Software Platform

To communicate with and control the algorithms running on either open speech FPGA platform, a software architecture, shown in Fig. 2.11, has been developed at Flat Earth Inc. A web-enabled device, such as a tablet, smartphone, or desktop computer, runs a user interface application. This application controls algorithm parameters by sending commands to a server on the HPS. The server interfaces with device drivers that handle changing the algorithm parameters in the FPGA fabric.

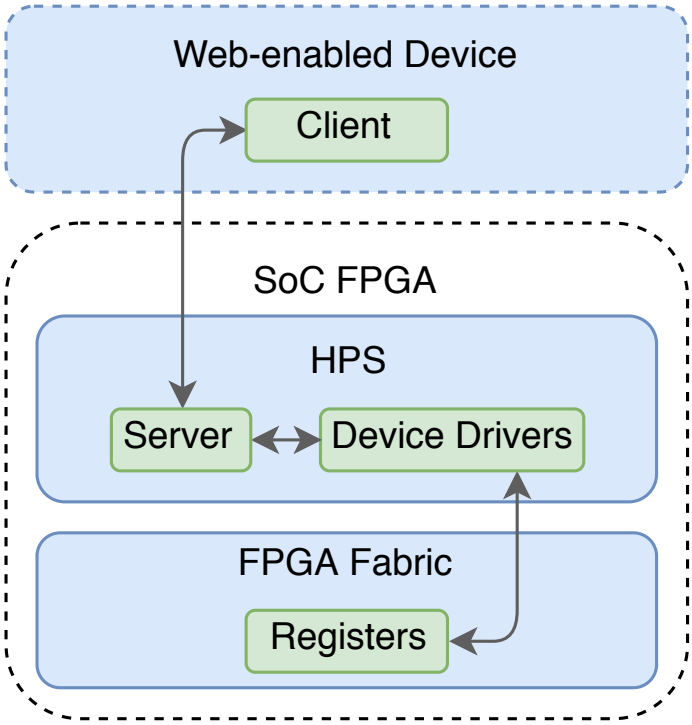


Figure 2.11: Software architecture diagram. The client application sends values to the server running on the HPS, which get written to registers via device drivers.

16

DEVELOPMENT FRAMEWORK

In spite of the many performance advantages SoC FPGAs have, developing for these devices is inaccessible to non-experts. To increase the accessibility of SoC FPGA development, a model-based hardware/software co-design development framework has been created. This framework is restricted to developing DSP applications, rather than the case of general digital system design. The development framework presented here reduces development time and complexity, enabling rapid prototyping of audio processing algorithms on SoC FPGAs. The typical development flow for embedded signal processing applications is long and complex, comprising the following steps:

1. Develop the algorithm in a high-level programming language, such as MATLAB, Simulink, Python, etc.

2. Simulate the algorithm to verify behavior and performance

3. Implement the algorithm for the target hardware using a low-level language such as C, VHDL, Verilog, or even assembly

4. Test the algorithm on the target hardware

If a problem is found at any of these steps, the developer must fix the problem at one of the previous steps, then repeat all subsequent steps again. For example, if a problem with the algorithm is found during final testing, the developer will go back to step 1 to fix the problem, then propagate that change through all subsequent steps; this can be a long process.

The development process presented here seeks to eliminate the algorithm implementation step by autogenerating the low-level implementation code. Our development framework, presented previously in [8], automatically generates VHDL

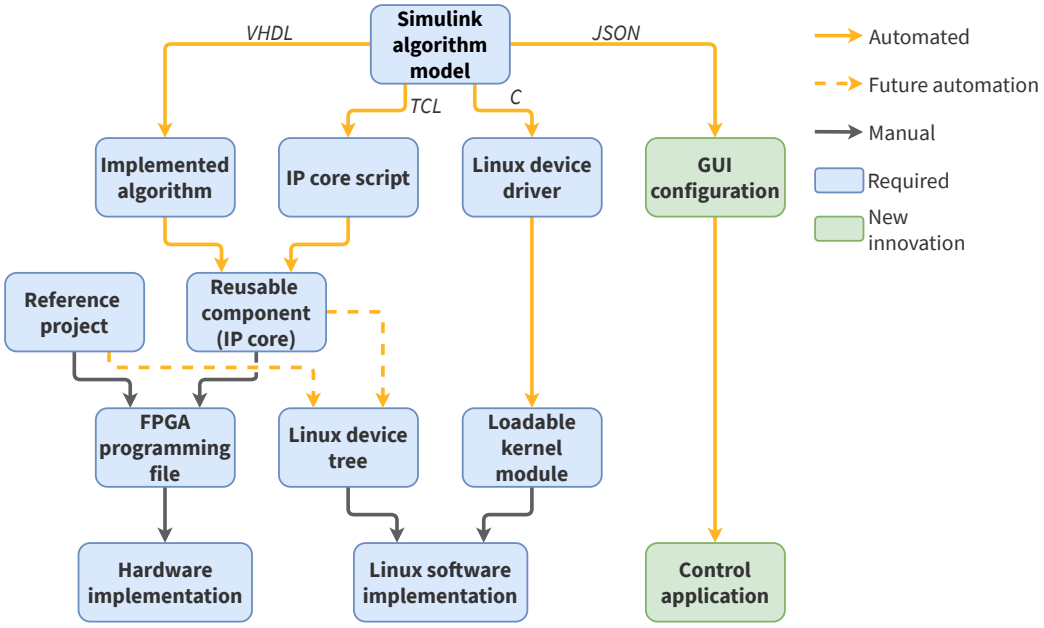


Figure 3.1: Flow diagram of the development process. The blocks in blue are required to have a functioning system; these steps are typically done manually in most development workflows. The blocks in green are not typically included in standard SoC FPGA development workflows. The arrows highlighted in yellow are automated in our workflow. The languages needed for each step are indicated next to the arrows.

code that runs the algorithm in the FPGA fabric, C device drivers that control the algorithm from Linux on the HPS, and a GUI that is used to set algorithm parameters. A graphical representation of the development flow is shown in Fig. 3.1. Utilizing automatic code generation eliminates the need to make manual code changes when the algorithm model is changed.

Concretely, the workflow presented here comprises the following steps:

1. Develop the algorithm in Simulink

2. Simulate the algorithm in Simulink

3. Automatically generate all implementation code

4. Test the algorithm on the target hardware

Algorithm Modeling and Development

Algorithm modeling and development is all done in Simulink. As shown in Fig. 3.2, each Simulink model contains four top-level subsystems:

- bus signal generation
- bus signal capture
- register control signals
- a dataplane

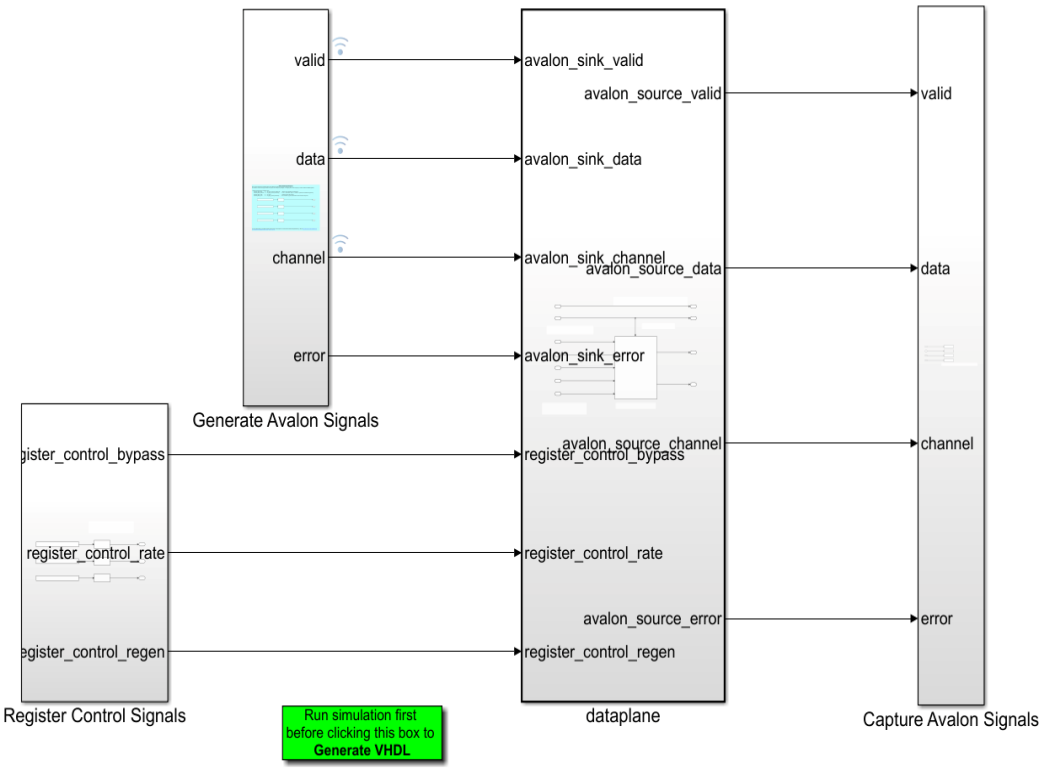


Figure 3.2: The top level of a typical Simulink model, showing the four main subsystems: bus signal generation, bus signal capture, register control signals, and the dataplane.

Each Simulink model has a handful of associated scripts that define model parameters and control registers, create simulation test signals, and verify simulation outputs, among other things. These scripts, written originally by my advisor, Ross Snider, can be found in Appendix A.

The subsystems shown in Fig. 3.2 are described in more detail below. Example Simulink models can be found on GitHub[1].

Bus Signal Subsystems

The bus signal subsystems have two purposes: during simulation, they generate the input stimuli for the algorithm and capture the outputs of the algorithm; on the SoC FPGA, they provide an interface to connect the algorithm to other parts of the system.

There are two main bus types to choose from on Intel SoC FPGAs: Intel's Avalon bus, or ARM's AXI-4 bus. Both bus specifications can handle streaming data (like audio), but the Avalon bus specification was chosen because it is simpler. AXI-4, on the other hand, has many signals that are not needed. One drawback of using the Avalon interface is that it is only supported on Intel FPGAs; this is not a problem for the work presented here since we restricted ourselves to Intel SoC FPGAs at the onset.

For the audio applications presented here, the `data`, `channel`, `valid`, and `error` signals in Intel's Avalon Streaming interface are used. Multi-channel audio data is time-division-multiplexed on the Avalon bus, where the `channel` signal indicates which audio channel the data represents (e.g. left or right). `valid` indicates that the incoming data is valid, which in this context typically means a new sample is available.

[1] `https://github.com/fpga-open-speech-tools/simulink_models`

In simulation, the "Generate Avalon Signal" subsystem drives the simulation stimuli into the algorithm model; in hardware, this subsystem gets replaced with a streaming **source**. The "Capture Avalon Signal" subsystem simply captures the algorithm outputs in simulation for further test and verification; this subsystem gets replaced by a streaming **sink** in the hardware design.

Register Control Subsystem

The "Register Control Signals" subsystem in the Simulink model defines the hardware registers that control algorithm parameters. During simulation, these registers only take on one value. Multiple simulations are performed if multiple register values need to be simulated. When the algorithm is implemented on an SoC FPGA, registers are controlled via Linux device drivers running on the HPS. Software can change register values at any time, allowing algorithm parameters to be modified while the algorithm is running.

Dataplane Subsystem

The dataplane subsystem is where the algorithm is implemented. Streaming data comes into the dataplane from the Avalon Streaming sink subsystem. Register values also come into the dataplane.

The algorithm typically only processes incoming data when the valid signal is asserted. The downside of this is that all processing must happen within one clock cycle, as that is how long the valid signal is asserted.

Different processing can be performed on the left and right channels by having left and right channel processing subsystems that are only enabled when the channel signal is the appropriate number. Alternatively, if both channels will have the same processing done on them, both channels can be processed by the same subsystem.

MATLAB's HDL Coder is used to generate HDL from dataplane subsystem. The dataplane needs to be implemented with HDL-compatible Simulink blocks. Users can restrict the available blocks to HDL-compatible blocks by typing `hdllib` in MATLAB; in practice, this still makes some incompatible blocks available. Additionally, developers need to use HDL-compatible data types, which are generally fixed-point numbers. Some FPGAs support single-precision floating point numbers as well, so singles can be used as well if one is targeting an FPGA that supports singles.

Hardware implementation needs to be considered when creating an HDL-compatible algorithm. Unfortunately, this does decrease accessibility for people who are not familiar with FPGAs. The need for FPGA knowledge is somewhat remedied by abstracting common design patterns into custom library blocks. All in all, designing HDL-compatible models in Simulink is still significantly faster than designing the HDL by hand.

System Autogeneration

One of the primary goals of this research is to enable people who are unfamiliar with FPGAs to create audio signal processing algorithms on FPGAs. Another goal is to shorten the audio signal processing development process for people who are familiar with FPGAs. Both of these goals are accomplished through automatic code generation. MATLAB's HDL Coder, along with various python scripts presented here, create all of the code needed to implement and control the Simulink model in real-time on an SoC FPGA. Code is generated by clicking a button in the Simulink model, like that shown in Fig. 3.2; the button calls a MATLAB script that then runs all of the code generation processes.

To implement and control the Simulink model in real-time on an SoC FPGA, the following needs to be created:

1. HDL code to implement the algorithm

2. Device drivers to change algorithm parameters

3. A GUI to allow users to easily control algorithm parameters

MATLAB's HDL Coder handles most of the HDL generation. Device driver generation is not well-supported in MATLAB, and standalone GUI generation is not supported at all in MATLAB (to the best of the author's knowledge). To support generating all of the necessary code, information from the Simulink model—register and port names, data types, default values, etc.—are parsed and put into a dataplane json file, an example excerpt of which is shown in Listing 3.1.

All of the ports in the Simulink model are parsed, and relevant information is put into a MATLAB struct. Additional information from MATLAB scripts associated with the Simulink model is also put into the MATLAB struct. Once all of necessary fields are put in the struct, it is converted into json with MATLAB's `jsonencode` function. The json string is then written to a file by calling python's `json.dumps` function; python is used instead of MATLAB because it supports pretty-printing the json string in a human-readable format.

The dataplane json file, an example of which can be seen in full in Appendix F, has objects for the Avalon memory-mapped register interface, the Avalon Streaming source interface, and the Avalon Streaming sink interface, as well as any external inputs or outputs (conduits). If any of those objects aren't present in the mode, the associated flag (e.g. `conduit_output_flag`) will be 0. Each signal or register in the objects has a datatype object and a name; registers also have information about min, max, and default values. There is a clocks object that contains the sampling rate and system clock rate. The json file also has fields for the model name, VHDL entity name, and device driver name and version.

```
{
  "avalon_memorymapped": {
    "register": [
      {
        "data_type": {
          "fractional_bits": 7,
          "signed": false,
          "type": "ufix8_En7",
          "width": 8
        },
        "default_value": 0.5,
        "max_value": 1,
        "min_value": 0,
        "name": "wet_dry_mix",
        "reg_num": 2
      },
```

Listing 3.1: An excerpt from a json file generated from a Simulink model. The excerpt shows a register from the Simulink model.

Various python scripts use the information in the dataplane json file to generate VHDL, C, and TCL files; these processes will be described in more detail in the following sections.

Algorithm

To implement the algorithm in the FPGA fabric, an intellectual property (IP) core needs to be generated. The IP core consists of the HDL files that implement the algorithm as well as a TCL file that creates a component that can be used in Quartus' Platform Designer. Several python scripts were created to support automatic generation of the IP core; Aaron Koenigsberg contributed to the aforementioned python scripts.

MATLAB's HDL Coder is used to generate the HDL code that implements the algorithm. Since the HDL Coder doesn't support Intel's Avalon interface, a python script, `vgenAvalonWrapper.py`, was created to autogenerate VHDL code that wraps the code generated by MATLAB and implements the Avalon interfaces required by the algorithm. This script performs the following steps:

1. Parse a `<model_abbreviation>_dataplane.json` file
2. Create boilerplate code at the top of the VHDL file
3. Create the VHDL entity, including all of the Avalon interfaces specified in `<model_abbreviation>_dataplane.json`
4. Declare register signals
5. Declare and instantiates the VHDL component created by MATLAB
6. Create the Avalon memory mapped bus read and write processes
7. Write the automatically generated code to a VHDL file

Another set of python scripts, written by Aaron Koenigsberg and me, generates the TCL file that creates the Platform Designer component associated with the algorithm. These scripts perform the following steps:

1. Parse a `<model_abbreviation>_dataplane.json` file
2. Create boilerplate code at the top of the file
3. Set IP core name and other metadata
4. Add all dependent files to a list of file dependencies
5. Set the compatibility flag used by the device driver and device tree associated with the Simulink model
6. Create clock and reset interfaces
7. Create Avalon memory mapped interface (if necessary)
8. Create Avalon Streaming sink interface (if necessary)
9. Create Avalon Streaming source interface (if necessary)

All of the python and MATLAB scripts that generate IP core code can be found in Appendix B and on GitHub[2]. Examples of automatically generated IP core code can be found in Appendix F.

Device Drivers

Device drivers are programs that live in *kernel space* and are used to communicate with and control hardware devices. In Linux, there are many different classes of device drivers, examples of which include character devices, block devices, and platform devices. All drivers must have an `init()` function that registers the driver, a `probe()` function that binds the driver to one or more devices, and a `remove()` function that unbinds devices from the driver.

[2] `https://github.com/fpga-open-speech-tools/simulink_codegen`

```
struct fe_bitcrusher_dev {
  struct cdev cdev;
  char *name;
  void __iomem *regs;
  int bypass;
  int bits;
  int wet_dry_mix;
};
```

Listing 3.2: An example platform device structure from an autogenerated device driver. The structure has a pointer to the device registers in the FPGA fabric, as well as integers to store a copy of each register value.

The drivers that are created in this work are platform device drivers [22]. An example platform device structure from a bitcrusher Simulink model is shown in Listing 3.2.

The device struct has a name, a pointer to where the device registers are located on the Avalon Memory Mapped bus, and fields for each register in the device. Each register is exported as a device attribute through the sysfs [23] virtual filesystem via `DEVICE_ATTR` macros as shown in Listing 3.3. sysfs provides a way to access kernel object data structures as files in *user space*. Since files are a universal interface in Linux, this makes interacting with device drivers with Linux command-line tools very easy and convenient.

The `DEVICE_ATTR` macros define the attribute name, the file permissions associated with the attribute, and functions to read and write the attribute. The read and write functions are where all of the heavy lifting happens in the device

```
DEVICE_ATTR(bypass, 0664, bypass_read, bypass_write);

DEVICE_ATTR(bits, 0664, bits_read, bits_write);

DEVICE_ATTR(wet_dry_mix, 0664, wet_dry_mix_read, wet_dry_mix_write);

DEVICE_ATTR(name, 0444, name_read, NULL);
```

Listing 3.3: An example of device attributes in a platform device driver. The device attributes show up as files in sysfs.

driver. For drivers associated with devices in the FPGA fabric, pseudocode for the read and write functions is shown in Listing 3.4. Concrete examples are shown in Appendix F.

```
register_read(...) {
  // get register value from device struct
  // convert fixed-point register value to a string
  // return the string
}
register_write(...) {
  // read string from external source (e.g. command line)
  // convert string to a fixed-point integer
  // write fixed-point integer to register in the fpga fabric
}
```

Listing 3.4: Pseudocode for reading and writing registers in the FPGA fabric from a device driver.

```
KDIR ?= ../../../../../linux-socfpga

default:
	$(MAKE) -C $(KDIR) ARCH=arm M=$(CURDIR)

clean:
	$(MAKE) -C $(KDIR) ARCH=arm M=$(CURDIR) clean

help:
	$(MAKE) -C $(KDIR) ARCH=arm M=$(CURDIR) help
```

Listing 3.5: Autogenerated Makefile that is used to compile device drivers.

```
ccflags-y := -I$(src)/../../../../../component_library/include

obj-m := bitcrusher.o
```

Listing 3.6: Autogenerated Kbuild file that is used to compile device drivers.

Much like the IP core generation in the previous section, python scripts were written to create Linux device driver code. They follow the same general procedure as the python scripts in the previous section. Aaron Koenigsberg wrote the majority of the scripts that generate device driver code; I assisted and have fixed bugs.

To aid in compiling the device drivers, I wrote a python script that automatically generate the Makefile and Kbuild files needed to compile the device drivers. Kbuild [24] is the build system used by the Linux kernel. Examples of generated Makefile and Kbuild files are shown in Listings 3.5 and 3.6 respectively. The MATLAB and python scripts that generate device driver code can be found in Appendix C and on GitHub[3].

[3] https://github.com/fpga-open-speech-tools/simulink_codegen

On Linux systems, a MATLAB script compiles the device drivers during the code generation process. Windows systems can't compile Linux device drivers, so Windows users must move the driver and build files to a Linux machine for compilation.

Control Application

For SoC FPGA algorithms that need real-time control, there are many different approaches to control. On one extreme, users write register values directly via the Linux command line on the HPS. On the other end, users can use a remote GUI to write register values. Since this research targets developers who are not SoC FPGA developers, we choose to autogenerate a remote GUI application that can control algorithm registers. Dustin Sobrero at Flat Earth Inc. wrote the remote GUI application and node js server described below.

The remote application is a web-based application written in C# that can run on any web-enabled device, though 8th generation Amazon Fire HD 8 tablets are currently the main devices being used for development. The tablets must be rooted to allow installing custom applications. For rooting instructions, see[4,5].

The application has three main UI elements: pages, panels, and controls. Each page can have multiple panels, and each panel can have multiple controls. Controls are used to send register values to a server on the SoC FPGA, which writes the values to the actual registers. Panels are used to group controls together. To the configure the GUI, the application reads a `UI.json` file that lives on the HPS. This json file, an example excerpt of which is shown in Listing 3.7, contains data structures for all of the pages, panels, and controls the GUI needs to create.

[4] `https://forum.xda-developers.com/hd8-hd10/orig-development/unlock-fire-hd-8-2018-karnak-amonet-3-t3963496`

[5] `https://forum.xda-developers.com/hd8-hd10/orig-development/fire-hd-8-2018-downgrade-unlock-root-t3894256`

```
{
  "name": "bitcrusher",
  "pages": [
    {
      "name": "Main",
      "panels": [
        {
          "controls": [
            {
              "dataType": "u6,0",
              "defaultValue": 32,
              "linkerName": "slider1bitcrusher",
              "max": 32,
              "min": 0,
              "module": "bitcrusher",
              "style": "default",
              "title": "Bits",
              "type": "slider",
              "units": "bits"
            },
```

Listing 3.7: An excerpt of a UI configuration JSON file. The JSON data structure has lists of pages, panels, and controls. Each control is associated with a module, which is the name of the Simulink model.

Most of the information in this configuration file is contained in the control objects. The controls contain the UI element name (linker name), register data type, min value, max value, default value, units, UI element type, and UI element style.

The server running on the HPS is a node js server. It receives commands from the remote GUI application that contain a linker name and value. It matches the linker name with the location of the register in Linux via information contained in a file called `Linker.json`, an example of which is shown in Listing 3.8. It reads the `Linker.json` file on startup. If a match is found, the server writes the register value to the register file, where the device driver then takes over. The `"majorNo"` field in Listing 3.8 currently needs to be manually changed from the wildcard to the actual major number since the host PC running MATLAB cannot known what the device driver's major number will be.

```
{
  "bitcrusher": {
    "links": {
      "slider1bitcrusher": "/bits",
      "slider2bitcrusher": "/wet_dry_mix",
      "toggle1bitcrusher": "/bypass"
    },
    "majorNo": "*"
  }
}
```

Listing 3.8: An example Linker configuration JSON file. The Linker file links device driver attributes in sysfs to controls in the GUI.

```
mp.register(ri).name       = 'Wet_Dry_Mix';
mp.register(ri).value      = 1;
mp.register(ri).min        = 0;
mp.register(ri).max        = 1;
mp.register(ri).default    = 0.5;
mp.register(ri).dataType = fixdt(0, 8, 7);
mp.register(ri).widgetType          = 'slider';
mp.register(ri).widgetDisplayUnits = 'ratio';
mp.register(ri).widgetStyle = 'default';
mp.register(ri).uiPageName = 'main';
mp.register(ri).uiPanelName = 'bitcrusher';
```

Listing 3.9: Example register structure in MATLAB showing the field needed to generate GUI configuration files.

MATLAB scripts were written to automatically generate the `UI.json` and `Linker.json` files. The register structures in `sm_init_control_signals.m` contain all of the fields these scripts need to generate the JSON files. An example of a register structure is shown in Listing 3.9.

First, `createLinkerWidgetNames.m` creates linker names for all of the registers in the Simulink model. Each control must have a unique name, especially when combining multiple Simulink models into one GUI, so the model name is appended to each linker name as seen in Listing 3.8.

After the linker names are created, `genLinkerConfig.m` and `genUIConfig.m` generate `Linker.json` and `UI.json`, respectively. `genLinkerConfig.m` only looks at the name and linker name for each register. `genUIConfig.m` assembles a control

structure for each register containing all of the info needed to create a UI control. Each control structure is put into a panel and a page according to the `uiPanelName` and `uiPageName` fields in Listing 3.9.

In general, user may want to run multiple Simulink models on an SoC FPGA at the same time. To accommodate this, python scripts were created to combine multiple `UI.json` and multiple `Linker.json` files. `combine_ui_configs.py` takes a list of UI json files and combines them into one UI json file. `combine_linker_configs.py` takes a list of linker json files and combines them into one linker json file. These scripts are run manually on the command line.

Code listings of the functions mentioned in this section can be found in Appendix D and on GitHub[6].

Software Infrastructure

In order to use audio processing algorithms developed with the methods described in the previous section, there needs to be hardware and software for the algorithms to run. Audio processing hardware platforms were presented in the background section. This section presents the software infrastructure used to deploy algorithms on an SoC FPGA.

Embedded Linux

Embedded Linux is commonly used as the operating system on the HPS, and that is the case in this research. Using Linux allows developers to extend system functionality with the myriad features available on Linux (e.g. webservers, databases, networking stacks, display and peripheral drivers) without much effort. Developers can writer user software in almost any programming language they desire

[6] `https://github.com/fpga-open-speech-tools/simulink_codegen`

```
CROSS_COMPILE=arm-linux-gnueabihf
ARCH=arm
```

Listing 3.10: Environment variables needed to cross-compile the Linux kernel.

on embedded Linux. Linux is not a real-time operating system, but that is not an issue on SoC FPGAs; latency-sensitive real-time algorithms can be implemented in the FPGA fabric, while other functionality can be implemented in Linux. The basic components of any embedded Linux system are the Linux kernel, a filesystem, and device tree blobs. Each of these will be described in more detail below.

Linux Kernel The Linux kernel is the core of the Linux operating system. It handles the boot process, input/output requests, task scheduling, memory management, and interacting with peripheral devices via device drivers, among many other things. All things that are controlled by the kernel happen in *kernel space*, whereas user programs reside in *user space*. The Linux kernel must be compiled into a zImage for the target CPU architecture, which, for SoC FPGAs, is the ARM architecture. To compile the Linux kernel for SoC FPGAs, we download the linux-socfpga[7] source code on a Linux desktop computer and cross-compile it for the HPS. The Linaro[8] `arm-linux-gnueabihf` toolchain is used for cross-compilation. Cross-compiling the kernel requires two environment variables to be defined: `CROSS_COMPILE`, which is the path to or name of the cross-compiler being used; and `ARCH`, which is the target CPU architecture. For the work presented here, these variables are shown in Listing 3.10.

[7] `https://github.com/altera-opensource/linux-socfpga`
[8] `https://www.linaro.org/`

Filesystem In addition to the kernel, filesystems are an integral part of operating systems. For SoC FPGAs, there are several choices for filesystems, including the following:

- Yocto[9]: a set of tools that can build custom kernels and filesystems
- Angstrom[10]: another set of tools related to the Yocto Project
- Buildroot[11]: *another* set of tools that can build custom kernels and filesystems
- Ubuntu Base[12]: a minimal Ubuntu filesystem intended for constrained environments

The first three options produce highly-customized filesystems. Software packages are typically built-in, rather than installed on-the-fly (though some support for package installation is available). Ubuntu Base, on the other hand, has full support for installing packages on-the-fly via `apt`. Since the platforms presented in this thesis are intended for prototyping, Ubuntu Base is used as the filesystem since it allows developers to install any packages they want without having to rebuild the filesystem.

For information on setting up the Ubuntu Base root filesystem, refer to Appendix G.

Device Trees On desktop systems, Linux automatically detects and configures hardware components and peripherals. On embedded Linux systems, device trees are used to inform Linux of what hardware exists. Device trees are a tree data structure where each node contains information (e.g. register addresses, interrupt pins, device driver compatibility) about a specific hardware device. Nodes can have multiple

[9] `https://www.yoctoproject.org/`
[10] `http://www.angstrom-distribution.org/`
[11] `https://buildroot.org/`
[12] `https://wiki.ubuntu.com/Base`

```
FE_AD1939 {
  compatible = "dev,fe-ad1939";
};
```

Listing 3.11: Example of a device tree node with a compatibility flag.

```
static struct of_device_id fe_ad1939_dt_ids[] = {
  {
  .compatible = "dev,fe-ad1939"
  },
  {}
};
```

Listing 3.12: Example of a device tree compatibility flag in a device driver.

children nodes, such as having multiple i2c devices on an i2c bus. Device tree source files are compiled into binary device tree blobs, which are loaded by the Linux kernel.

One particularly important device tree field is the compatibility flag. This flag tells the kernel how to associate a hardware device with a specific device driver. An example of a device tree node with a compatibility flag is shown in Listing 3.11

This compatibility flag needs to match what is in the associated device driver. For example, the device driver for the FE AD1939 would specify the compatibility flag like what is shown in Listing 3.12.

Bootloader

Before Linux can run, a multi-stage boot process, shown in Fig. 3.3 must occur. First, a small program known as the boot ROM runs. The boot ROM lives in read-only-memory on the PCB. It loads the bootloader into memory so the bootloader can run. The bootloader then configures hardware, programs the FPGA, and loads the Linux kernel. Depending on which SoC FPGA family is being used, the bootloader can either have one or two stages. For Cyclone V devices, the bootloader has two stages: the preloader and the main bootloader. Arria 10 devices use a single stage bootloader.

Das U-Boot[13] is typically the bootloader used on SoC FPGAs. Via the use of environment variables, U-Boot can be configured for many different boot scenarios. I have set up two boot scenarios: booting over a network, which is intended for development; and booting from an SD card, which is intended for standalone prototypes.

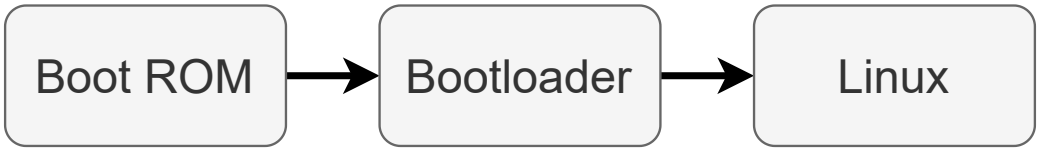


Figure 3.3: High-level SoC FPGA boot process.

SD Card Booting Booting from an SD card is the default method for many SoC FPGA development boards, including the Terasic DE10-Nano used in this project. In this scenario, the SD card contains everything needed to boot and run the system. An example SD card layout is shown in Fig. 3.4. The Master Boot Record contains a table of where the partitions are in memory, so the order and sizes of the partitions

[13] https://www.denx.de/wiki/U-Boot/

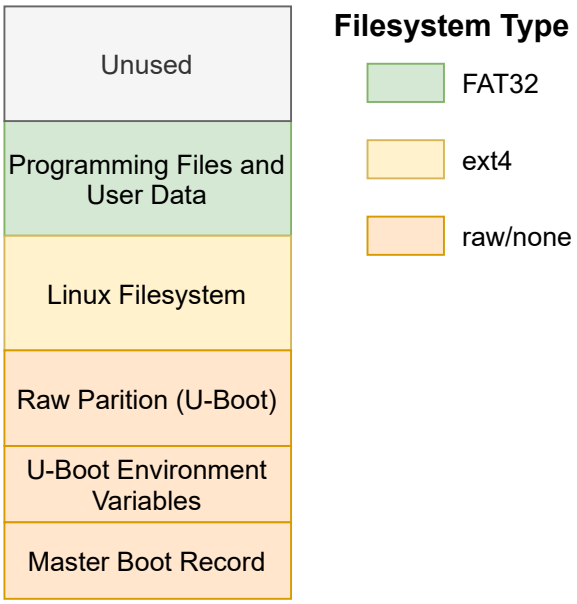


Figure 3.4: Example SD card layout.

(except for the Master Boot Record) can vary from what is shown in Fig 3.4.

The ext4 partition contains the Linux filesystem. The FAT32 partition contains an FPGA bitstream, a device tree blob, and the Linux kernel zImage. The raw partition contains the bootloader; if desired, a bootloader image file can be put on the FAT32 partition instead of having the bootloader live in the raw partition.

Booting from the SD card is convenient when one wants the system to be standalone, e.g. when testing a prototype in the field or shipping an end-product. However, constantly updating the SD card partitions during development is cumbersome.

Network Booting To avoid constantly updating SD card partitions during development, a network booting configuration was created. In this configuration, the SD card layout looks the same as Fig. 3.4, but the FAT32 and ext4 partitions don't get used. Instead, the Linux filesystem is mounted over a network via the

Network File System (NFS) protocol, and the files that were once on the FAT32 partition are downloaded via the Trivial File Transfer Protocol (TFTP).

To support this, a desktop Linux computer needs to run NFS and TFTP servers to host all of the necessary files. See Appendix H for instructions on setting up the NFS and TFTP servers. The directory structure of the NFS and TFTP servers are shown below:

```
srv
|-- tftp
    |-- arria10
    |-- de10nano
        |-- AudioMini_Passthrough
            |-- soc_system.dtb
            |-- soc_system.rbf
        |-- bootscripts
            |-- audiomini_passthrough.scr
            |-- audiomini_passthrough.script
            |-- make_uboot_script.sh
        |-- kernel
            |-- zImage
|-- nfs
    |-- arria10
    |-- de10nano
        |-- ubuntu-rootfs
```

Each target development board has its own folder in the tftp and nfs directories. Within the tftp directory, each board has a folder for the Linux Kernel, a folder for

U-Boot bootscripts, and project folders that contain files related to a specific FPGA project. Within the NFs directory, each board has a folder for the Ubuntu Base filesystem described in section 3

On boot, U-Boot is configured to download and run a bootscript (.scr file) for a particular project. The bootscript contains the other commands U-Boot needs to run to properly program the FPGA and run Linux. In particular, U-Boot needs to download an FPGA programming image (.rbf file), a device tree blob, and the Linux kernel zImage.

The following U-Boot environment variables were defined to support networking booting:

- `tftp-kernel-dir`: the directory in the TFTP server where the desired Linux kernel is located
- `tftp-project-dir`: the directory in the TFTP server where the desired FPGA project files are located
- `nfs-rootfs-dir`: the directory where the Linux filesystem is located
- `serverip`: the ip address of the TFTP server
- `nfsip`: the ip address of the nfs server
- `ipaddr`: the ip address of the SoC FPGA
- `nfsboot`: enable/disable network booting
- `bootnfs`: command to load the Linux kernel
- `bootcmdnfs`: command to download the U-Boot script
- `bootcmdmmc`: command to boot from SD card

- `bootcmd`: main boot command that chooses between network and SD card booting

See Appendix H for an example U-Boot script and an example of U-Boot environment variable values.

SOUND EFFECTS PROCESSOR

A sound effects processor, presented at the 147th Audio Engineering Society Convention [8], was developed to demonstrate the development process described in the Development Framework chapter and the low-cost hardware platform presented in the Low-cost Hardware section of the Background chapter. Below is a brief summary of the development process:

1. Develop the algorithm in Simulink using HDL-compatible blocks
2. Automatically generate the code needed to run the algorithm on an SoC FPGA
3. Deploy the algorithm to an SoC FPGA

The sound effects processor is a user-controllable effects chain comprising three effects: a bitcrusher, an echo, and a flanger. Examples of each effect are available here[1] and as supplementary files. The effects will be described below, along with details of how they were mapped to an FPGA-compatible hardware implementation. Simulink models for the sound effects can be found on here[2], and the Quartus project can be found here[3].

Sound Effect Algorithms

Bitcrusher

Bitcrusher is a sound effect that distorts audio by two main methods: downsampling and bit depth reduction. Downsampling reduces the sampling rate of the audio, which will make the audio waveform more course—especially if an anti-aliasing

[1] `https://montana.box.com/s/c06517yn0bvb8gp9g10nanfn9ohwzy9g`
[2] `https://github.com/fpga-open-speech-tools/simulink_models`
[3] `https://github.com/fpga-open-speech-tools/de10nano_projects`

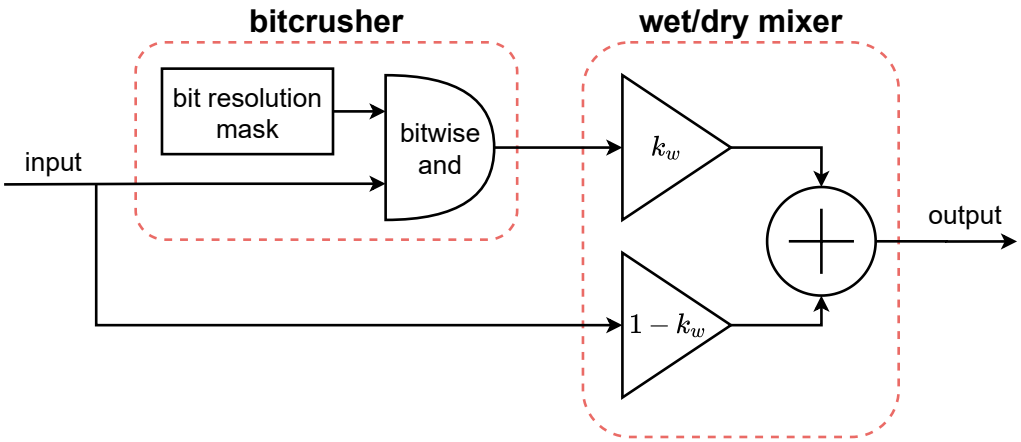


Figure 4.1: Block diagram of the bitcrusher sound effect. The effect has a bit depth reduction section and a wet/dry mixer.

low-pass filter is not used to properly bandlimit the audio to the new sampling rate. Bit depth reduction decreases the resolution of the digital values used to store each audio sample. In our case, the digital values are 32-bit fixed-point numbers with 28 fractional bits. Reducing the number of bits used to represent a sample reduces the number of possible values, which makes the waveform more "choppy" and distorted.

A block diagram of the bitcrusher sound effect Ross Snider and I created is shown in Fig. 4.1. We opted to only implement bit depth reduction because creating a tunable anti-aliasing filter for downsampling was too complicated to implement in the timeframe we had. In hindsight, simply decimating the signal by the desired ratio without an associated anti-aliasing filter would have been desirable for distortion effects; it is likely that some, maybe all, commercial bitcrusher sound effect plugins do not use anti-aliasing filters.

A wet/dry mixer was added to the output of the bitcrusher. This allows the effected "wet" signal to be mixed with the original "dry" signal in proportion given by the wet gain, k_w.

Hardware Architecture Implementation of bit depth reduction in hardware is accomplished with bit masking. For a desired bit depth, N_{bits}, a bit mask is created where the N_{bits} most significant bits are ones and the rest of the bits are zeros. For a 32-bit integer, this bit mask can be created with

$$\text{bit mask} = (2^{N_{bits}} - 1) << 32 \tag{4.1}$$

The incoming audio samples are ANDed with this bitmask; all bit positions that are zero in the mask become zero in the audio sample, while all other bits are left untouched. This operation truncates the audio sample to N_{bits} bits of precision.

Implementation of a tunable anti-aliasing filter would require a programmable low-pass filter. One possible architecture for this involves using a dual-port RAM to store filter coefficients that can be changed during runtime. Implementing a programmable filter before the Audio Engineering Society Convention this sound effect demo was presented at was not feasible, though a programmable filter Simulink block has now been made by our research group.

Since there are two channels in stereo audio, the bitcrusher hardware is duplicated for both channels. The channel and valid signals in the Avalon Streaming interface are used to enable the hardware for each channel. When valid is asserted and channel is 0, the left channel hardware is enabled as long as valid is asserted. The right channel follows the same paradigm when channel is 1.

No special considerations are needed when implementing the wet/dry mixer in an FPGA since it is just a handful of multiplications and additions.

Resource Usage The FPGA resources used by the bitcrusher IP core are shown in Table 4.1. This table shows Adaptive Logic Modules (ALMs), Combinational adaptive lookup tables (ALUTs), registers, memory usage in bits, and M10K memory

ALMs	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	M10K Memory Blocks	DSP Blocks
192.2	354	354	78	1	12

Table 4.1: Bitcrusher IP core resource usage.

blocks (10 Kb size). More information on resource types can be found in the Cyclone V device overview [11].

Echo

Echo is a delay-based effect that delays the input signal and adds the delayed signal on top of the original signal. This emulates the repetition of sound waves reflected off of surfaces. A block diagram of the echo implementation used in this work is shown in Fig. 4.2.

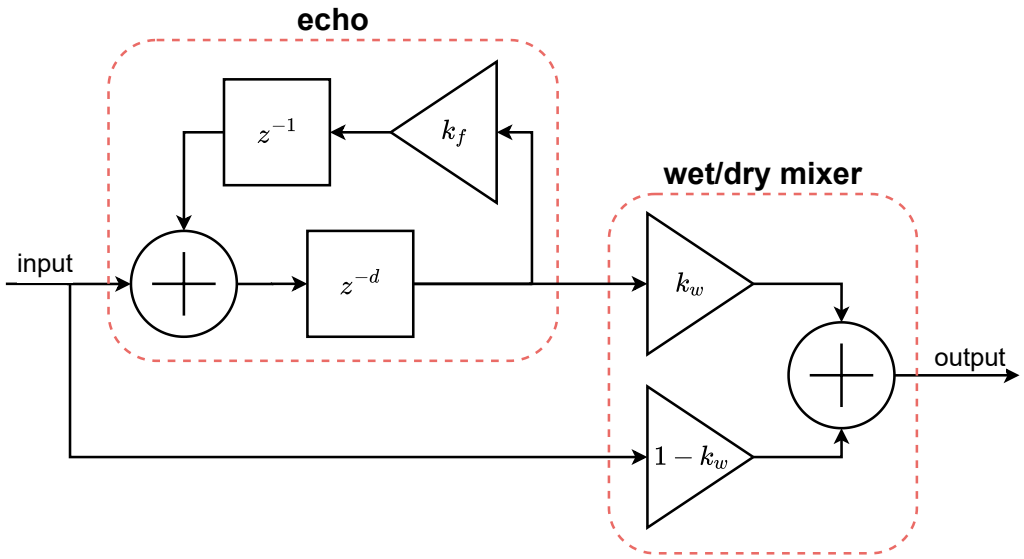


Figure 4.2: Block diagram of the echo sound effect. The effect uses a variable delay line which is fed back to the input. A wet/dry mixer is at the output.

The amount of delay can be varied with the z^{-d} block in Fig. 4.2, where d is the number of samples to delay the input signal. How quickly the echo decays is controlled by the feedback gain, k_f. A wet/dry mixer was added to the output of the echo to control how much of the echo is present compared to the original signal.

Hardware Architecture The variable delay, z^{-d}, requires special attention when being implemented in hardware. In hardware, one can't change the length of the delay line at will, so the delay line needs to have a fixed length.

One way to implement a variable delay line is to use a tapped delay line [25] where the signal can be extracted at multiple places in the delay line. A hardware implementation of a tapped delay line for the echo effect could be a series of registers, each with a tap, and a multiplexer to select which tap to output, as shown in Fig. 4.3. This implementation would use logic elements for storage.

A more efficient variable delay implementation uses a circular buffer, like that shown in Fig. 4.4. The circular buffer uses block memory in the FPGA, which is more efficient than using normal logic elements to implement memory. This is the

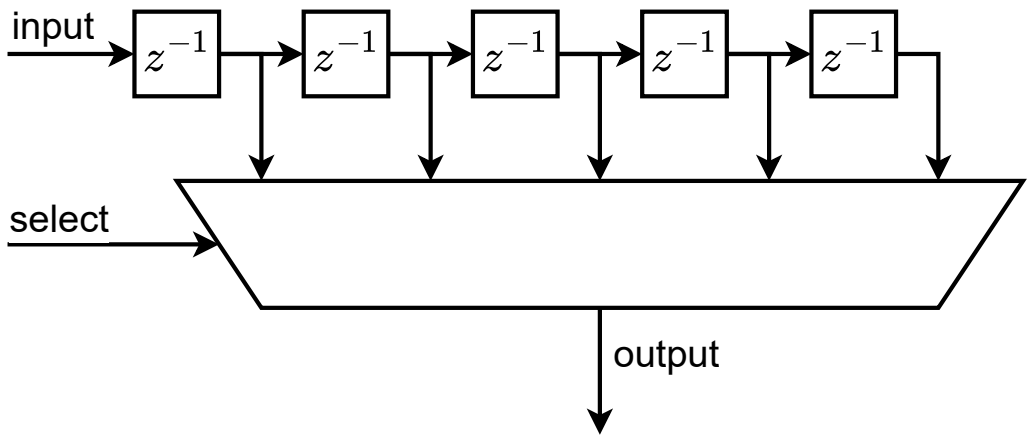


Figure 4.3: Example hardware implementation of a tapped delay line comprising registers and a multiplexer.

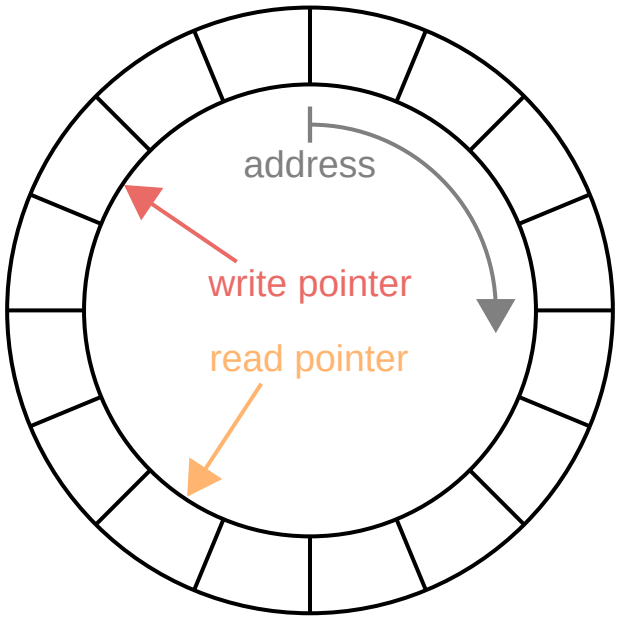


Figure 4.4: Circular buffer overview. The addresses in the circular buffer increase in the clockwise direction. Circular buffers have a write pointer, which points to the address data is written to; and a read pointer, which points to the address data is read from. Memory is shown in a ring for visualization purposes, but the physical memory layout is not circular.

implementation used in the echo sound effect. Rather than shifting data in memory each time a new sample comes in, a circular buffer writes the new sample to the next address in memory, which is more power-efficient than shifting each sample. This is well-suited to streaming data applications, such as audio. The difference between the read and write pointer addresses determines the delay through the buffer. For a variable delay line, changing the difference between the read and write pointers changes the delay length. To implement a circular buffer in an FPGA, a dual-port RAM was used. While there are many variations of dual-port RAMs, the one used here has one write port and one read port, as shown in Fig. 4.5; this is the simplest dual-port RAM implementation. In order to avoid undefined behavior when reading and writing to the same address, the read pointer is always at least one address

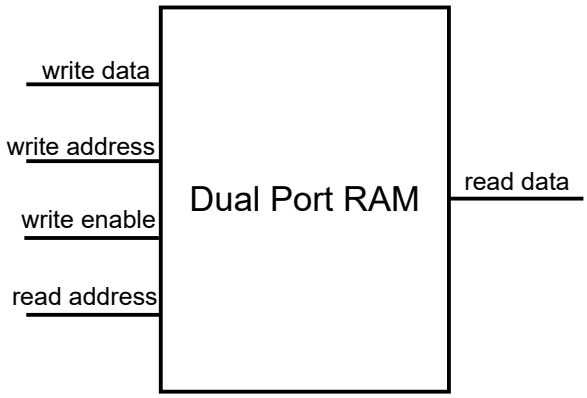


Figure 4.5: A simple dual-port RAM with one write port and one read port.

behind the write pointer. Additionally, the dual-port RAM is sized as a power of two. Making the RAM size a power of two allows the read and write pointer counters to reset back to zero at the end of buffer by using wrap-on-overflow behavior instead of comparators. This reduces resource usage. The hardened memory blocks in FPGAs are sized as powers of two.

The wet/dry mixer and left/right channel implementations are the same as in the bitcrusher model.

Resource Usage The resource usage for the echo IP core is shown in Table 4.2. Since this is a delay-based effect, memory block usage is high. The Cyclone V device used on the DE10 Nano only has 553 memory blocks. This translates to 5.66272×10^6 bits of RAM:

$$553 \text{ M10k RAM blocks} \implies 5530[\text{kb}] \times 1024 = 5.66272 \times 10^6[\text{bits}] \tag{4.2}$$

ALMs	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	M10K Memory Blocks	DSP Blocks
348.5	594	351	2162688	264	12

Table 4.2: Echo IP core resource usage. The echo uses a lot of memory blocks to implement long delays.

Using 32-bit words, this results in a maximum delay of 3.687 seconds:

$$\text{max memory usage} = \frac{5.66272 \times 10^6[\text{bits}]}{32[\text{bits/sample}]} = 176960[\text{samples}] \tag{4.3}$$

$$\text{max delay} = \frac{176960[\text{samples}]}{48000[\text{samples/sec}]} \approx 3.687[\text{s}] \tag{4.4}$$

The echo was set to have a maximum delay of 0.5 seconds to converse memory usage. Some optimizations could be made to reduce memory usage, such as decreasing word size. Memory usage is a bottleneck on low-cost FPGAs like the Cyclone V family.

Flanger

Flanging is a delay-based effect that originated in analog recording studios [26]. Two reel-to-reel tape machines are set to play the same tape, and their outputs are added together. Pressing the flange of a reel slows the playback speed of the reel, causing a slight delay between the two tape machines. The flange of each tape machine is pressed in alternating fashion, resulting in a modulation of the delay between the two tape machines. This effect can be modeled with a feedforward comb filter [27] whose delay is slowly modulated over time. Fig. 4.6 shows a block a diagram of the flanger effect I created.

In Fig. 4.6, the delay value of the delay line is modulated with a low-frequency sine wave, also known as a low-frequency oscillator (LFO). The rate of the LFO is

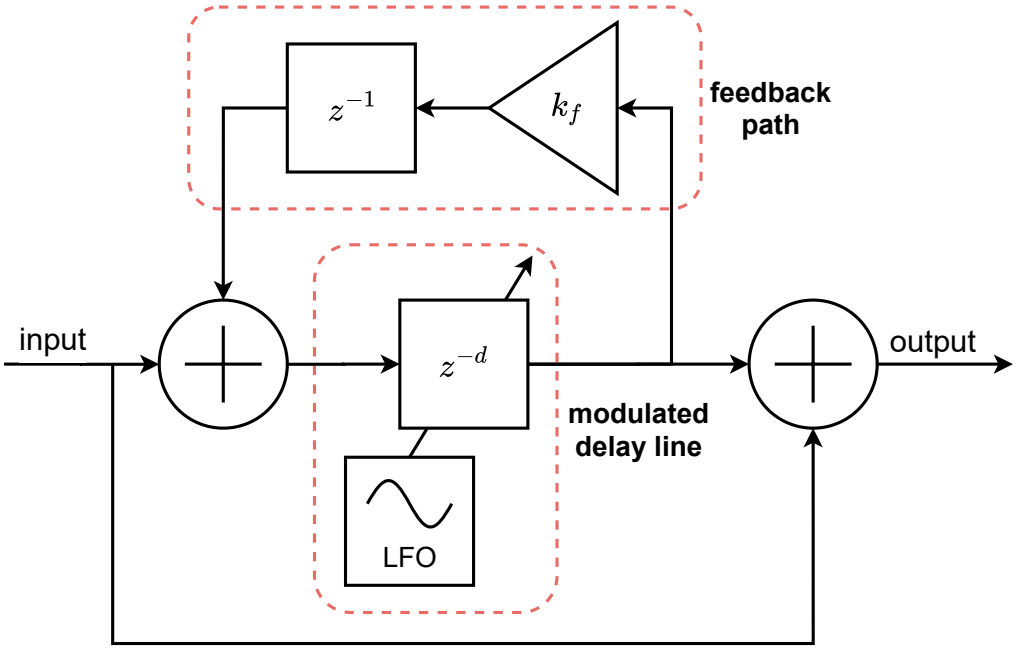


Figure 4.6: Block diagram of the flanger sound effect. A variable delay line is modulated by a low-frequency oscillator, and the delay line output is added to the original signal. The delayed signal can be fed back to the input.

variable between 0.1 Hz and 5 Hz. The delay is variable between 0.5 ms and 10 ms; longer delays will be perceived as an echo rather than a flanging effect. The delayed signal is fed back to the input to intensify the effect.

Hardware Architecture The variable delay z^{-d} block is implemented with a circular buffer as was done in the echo sound effect. The difference here is that the difference between the write and read pointers is sinusoidally varied.

Creating a sinusoidal oscillator in an FPGA is unfortunately not as easy as computing $sin(x)$ because FPGA synthesis tools don't allow this. To synthesize a periodic waveform in HDL, direct digital synthesis [28] is typically used. In this approach, an accumulator is used to index into a lookup table. The accumulator increment value determines the rate at which the lookup table is gone through. Larger

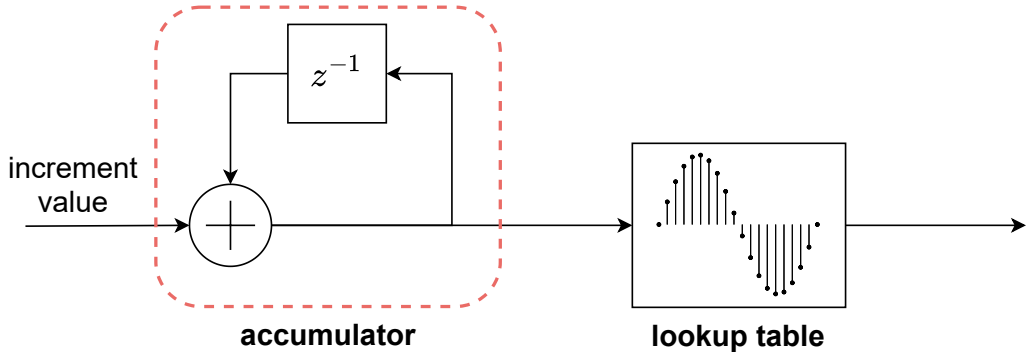


Figure 4.7: Overview of direct digital synthesis. The output of an accumulator is used to index into a lookup table containing the desired waveform. The accumulator increment value controls the frequency of the output waveform.

increment values skip more lookup table addresses between each clock cycle, which results in a faster frequency waveform; the opposite is true for smaller increment values. A high-level diagram of this approach is shown in Fig. 4.7.

The HDL-optimized numerically controlled oscillator (NCO) Simulink block [29] uses direct digital synthesis to create sinusoids. For a given output frequency, f_0, the accumulator phase increment is given by

$$\text{phase increment} = \frac{f_0 \times 2^N}{f_s} \tag{4.5}$$

where f_s is the sampling frequency, and N is the accumulator word length [29]. The accumulator word length was chosen so the minimum desired output frequency results in a phase increment of 1

$$N = \left\lceil \log_2 \frac{f_s}{f_0} \right\rceil \tag{4.6}$$

In general, $\log_2 \frac{f_s}{f_0}$ will not be an integer, so the ceiling is taken since N must be an integer. Doing this will make the phase increment slightly larger than 1, but the NCO ignores any fractional bits, so the phase increment is still effectively 1.

Dither [30,31] is added to the output of the NCO to decorrelate the quantization

error from the signal. Quantization error that is correlated to the signal can lead to harmonic structure that undesirably color the output signal.

Resource Usage The resource usage for the flanger IP core is shown in Table 4.3. About half of the ALMs are used by the LFOs.

ALMs	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	M10K Memory Blocks	DSP Blocks
411.7	655	597	40960	4	8

Table 4.3: Flanger IP core resource usage.

Demonstration

Using the development framework from the previous chapter, HDL code, device drivers, and a GUI were generated from Simulink models for each of the three sound effects. The sound effect IP cores were chained together in a Quartus project, which was then deployed on our low-cost hardware platform. During demonstrations, a synthesizer or microphone is plugged into the line input on the Flat Earth Audio Mini, and the effects are controlled with a GUI, which is shown in Fig. 4.8.

For the bitcrusher, bit depth and wet/dry mix are controllable. For the echo, the delay (in number samples), feedback gain, and wet/dry mix are controllable. For the flanger, the LFO rate and feedback gain (regen) are controllable.

The networking setup used during demonstrations is shown in Fig. 4.9. Since we don't need access to the internet, we use our own router to simplify networking and avoid dependence on other people's networks. The DE10 Nano plugs into the router via ethernet, and the Amazon Fire HD8 tablet connects to the router via WiFi.

Figure 4.8: Sound effects processor GUI, showing controls for a bitcrusher, echo, and flanger.

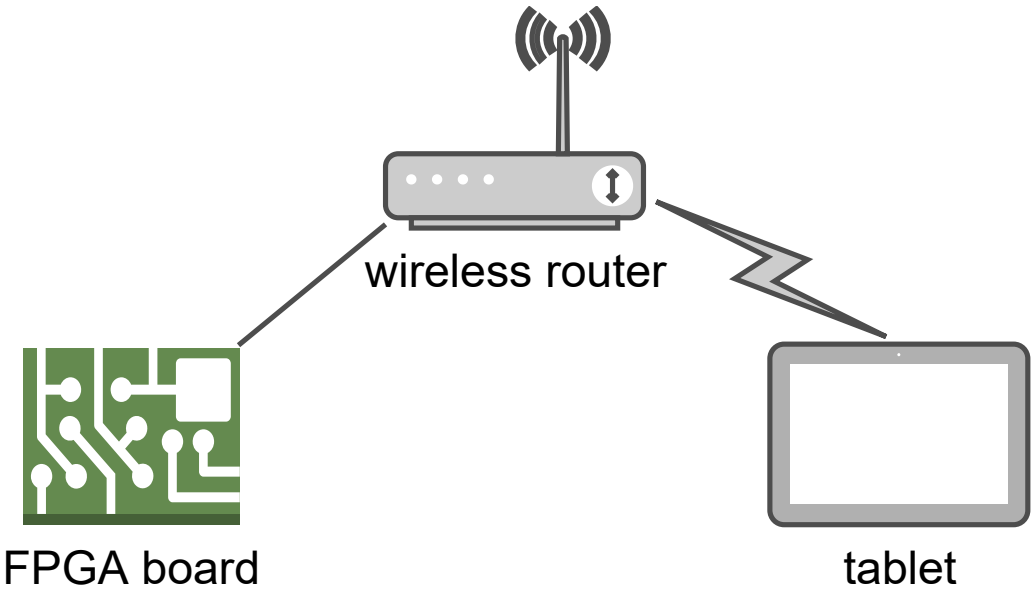


Figure 4.9: Networking setup used during demonstrations. An Amazon FIRE HD8 tablet and the FPGA connect to a standalone router.

REAL-TIME BEAMFORMER

Imagine a scenario where there are two sound sources at different locations in a room, but we only want to listen to one of them. In other words, we want to *filter* out the interfering source so we can focus on the source we're interested in. This type of spatial filtering is often called beamforming. Beamforming is a technique in which an array of sensors is used to focus on signals coming from a particular direction [32]. Beamforming finds uses in many areas, including RADAR, communications, astronomy, and acoustics. Beamforming can be used for both transmitting and receiving signals, though the focus here is on signal reception.

The basic idea behind beamforming is that the outputs of each sensor in an array can be combined in such a way that signals from a given direction add *constructively*, while signals from all other directions add *destructively*. The simplest beamforming algorithm is the delay and sum beamformer, where the outputs from each sensor are delayed relative to each and then added together. Other popular techniques include the minimum variance distortionless response (MVDR) beamformer and the linear constraint minimum variance (LCMV) beamformer. Beamforming and array signal processing are rich fields with a long history; see [32, 33] for a good introduction to the field, and see [34] for a reference on microphone arrays in particular.

This chapter focuses on implementing a delay and sum beamformer. The development process described in the Development Framework chapter is used to implement the algorithm. The algorithm is deployed on the Arria 10 FPGA mentioned in the High Performance Hardware section of the Background chapter, with a custom daughter card developed for beamforming.

The rest of this chapter will discuss

1. Sensor array theory

2. The theory behind the delay and sum algorithm

3. The hardware implementation of the algorithm

4. Additional hardware developed to create a real-time beamforming system

5. Verification of the real-time beamforming system

Sensor Array Theory

Array Directivity

Since sensor arrays are often used to enhance signals propagating from certain directions, a common array performance measure is the array's directivity. The directivity of an array can be interpreted as the array's gain as a function of angle and frequency [33].

In this work, a TDK InvenSense microphone array is used to sample acoustic wavefields. This array is a 4x4 rectangular array with 15 mm spacing between microphones. Fig. 5.1 shows the directivity pattern for the TDK microphone array as a function of frequency and azimuth (horizontal) angle. Fig. 5.2 shows the directivity pattern at 500 Hz, 1 kHz, 5 kHz, and 10 kHz, specifically.

Both Fig. 5.1 and Fig. 5.2 show that the array directivity for low frequencies is nearly zero. This is because the wavelengths at those frequencies are very large compared to the spacing between the microphones.

For sources that cover a broad band of frequencies, like audio, it is common for directivity to vary widely as a function of frequency for simple array geometries. Many array processing algorithms assume that the signal of interest is *narrowband*, occupying a small band of frequencies, rather than *broadband*.

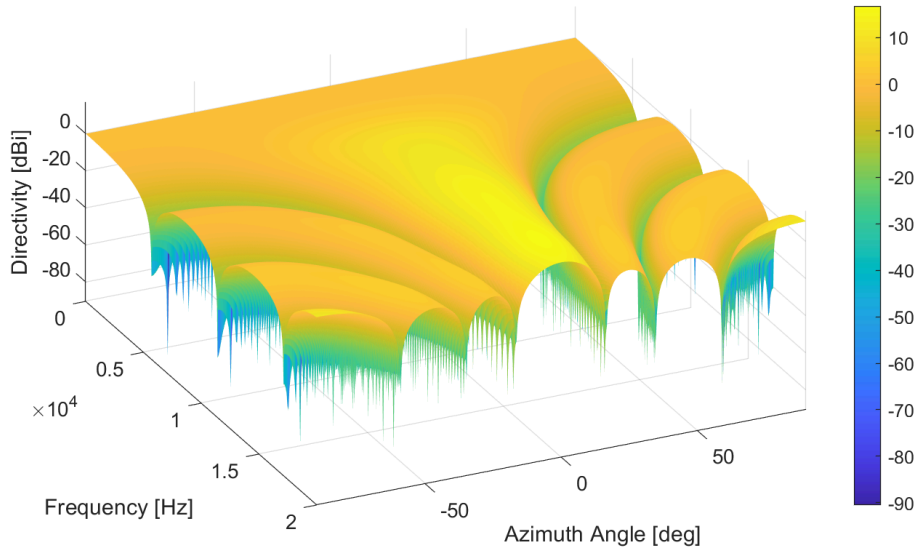


Figure 5.1: 3D directivity pattern for a 4x4 rectangular array with 15 mm spacing. Note how the directivity is nearly zero at lower frequencies.

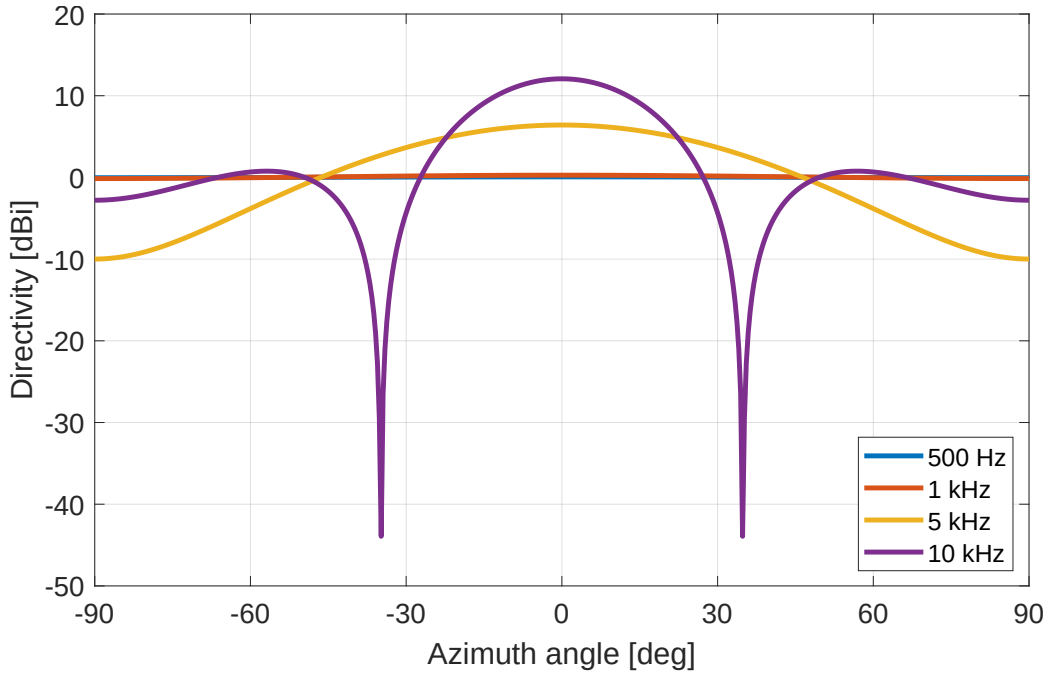


Figure 5.2: 2D directivity patterns for a 4x4 rectangular array with 15 mm spacing. Patterns for 500 Hz, 1 kHz, 5 kHz, and 10 kHz are shown. Note how the directivity is nearly zero at lower frequencies.

Spatial Sampling

Sensors arrays are used to spatially and temporally sample propagating wavefields. The Shannon-Nyquist sampling theorem, which is typically presented in the time-domain, also applies to spatial sampling.

Recall the the Shannon-Nyquist sampling theorem:

$$f_s > 2f_{max} \tag{5.1}$$

where f_s is the sampling frequency, and f_{max} is the highest frequency in the signal being sampled. Or, equivalently in terms of period instead of frequency:

$$T_s < \frac{1}{2f_{max}} \tag{5.2}$$

For spatial sampling, we are interested in spatial frequency, ν, which is expressed in cycles/meter. The spatial sampling version of the sampling theorem says

$$\nu_s > 2\nu_{max} \tag{5.3}$$

where ν_s is the spatial sampling frequency, and ν_{max} is largest spatial frequency in the signal. It is often more useful to think of this in terms of distances:

$$d < \lambda_{min}/2 \tag{5.4}$$

where d is the spatial sampling period and λ_{min} is the smallest wavelength of the signal being sampled. Wavelength is related to frequency by $\nu = 1/\lambda$.

The sampling theorem relies on signals being bandlimited to some maximum frequency. Signals are very rarely strictly bandlimited, so they are often filtered

before being sampled to guarantee the bandlimited condition. For temporal electrical signals, anti-aliasing filters are easy to implement. For spatial wavefields, one can't easily spatially filter the wavefield prior to sampling; as such, spatial aliasing is a common issue.

Spatial aliasing manifests as the appearance of spurious mainlobes in the array's directivity pattern [32]. Some spurious mainlobes can be seen around ± 90 for high frequencies in Fig. 5.1. This can cause confusion as to what direction a signal is propagating from. For example, take a wave with wavelength $\lambda = d$. When this signal forms a 90 degree angle with the array, each sensor samples the same value on successive wavefronts. When the wave forms a 0 degree angle with the array, each sensor also samples the same value because they are sampling a plane of constant phase. In this scenario, waves propagating from 90 degrees and 0 degrees are indistinguishable. Some spurious mainlobes can be seen around ± 90 for high frequencies in Fig. 5.1.

The effects of spurious mainlobes can be minimized by applying weights to each sensor. For simplicity, the delay and sum beamformer implementation described later does not include weights, as the focus here is on the hardware and design process rather than algorithm performance.

Nearfield and Farfield Sources

Most sources emit spherical wavefronts. When such a source is close to the array, the wavefronts will be curved when they impinge upon the array, causing the wave propagation direction at each sensor to depend upon how far away the source is [32]. On the other hand, when a source is very far away from the array, the curvature of the wavefronts will be small compared to the array dimensions; in this scenario, the wavefronts can be approximated as plane waves. When the source is far away from

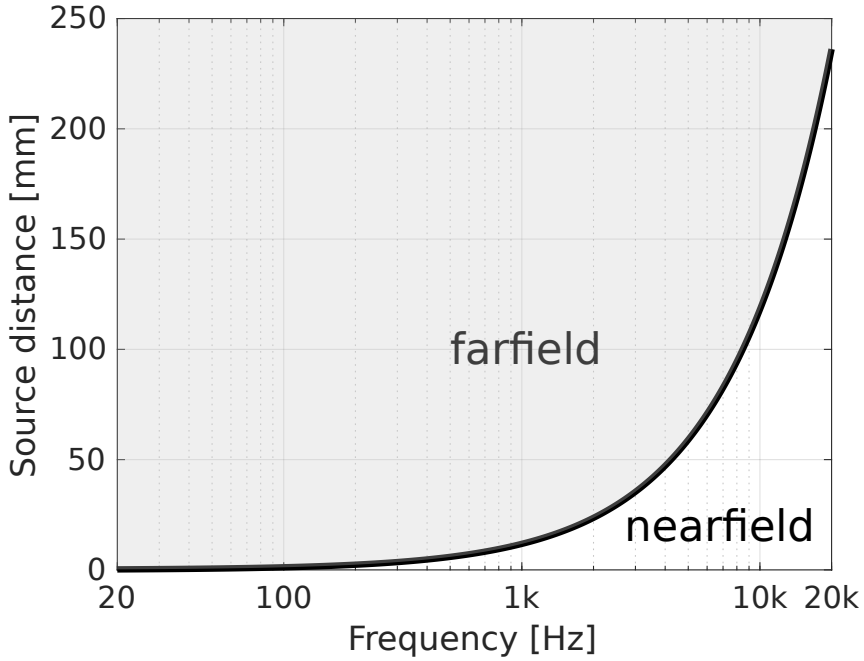


Figure 5.3: Plot of the nearfield/farfield boundary for an array with a largest dimension of 45 mm. The dark line indicates the boundary between the nearfield and farfield regions. The farfield region is above the line and is shaded.

the array, one can only determine the direction of the source, not the distance.

The boundary between the nearfield and farfield is often taken as the Fraunhofer distance [35]

$$R \geq \frac{2L^2}{\lambda} \tag{5.5}$$

where R is the distance from the array to the source, L is the largest dimension of the array, and λ is the wavelength of interest.

A plot of the farfield boundary for the TDK InvenSense microphone array used in this work is shown in Fig. 5.3. The farfield region above the boundary is shaded. The TDK microphone array is a 4x4 array with inner-element spacing of 15 mm, giving a maximum array dimension of 45 mm.

As shown in Fig. 5.3, the farfield approximation is valid for the entire audio band

as long as the source is about 250 mm, or about 10 inches, away from the array. For a 1 kHz source, the farfield approximation is valid for distances

$$R \geq \frac{2 \times (0.045 \text{ [m]})^2}{343 \text{ [m/s]}/1000 \text{ [1/s]}} \approx 0.0118 \text{ [m]} = 11.8 \text{ [mm]} \approx 0.465 \text{ [in]}$$

Delay and Sum Beamforming Theory

Algorithm

When a wavefront impinges on an array from a given direction, as shown in Fig. 5.4, the wavefront will arrive at each sensor at different moments in time. The basic idea behind delay and sum beamforming is that the signals received at each sensor can be delayed relative to each other, then added together to reinforce the desired signal, as depicted in Fig. 5.5.

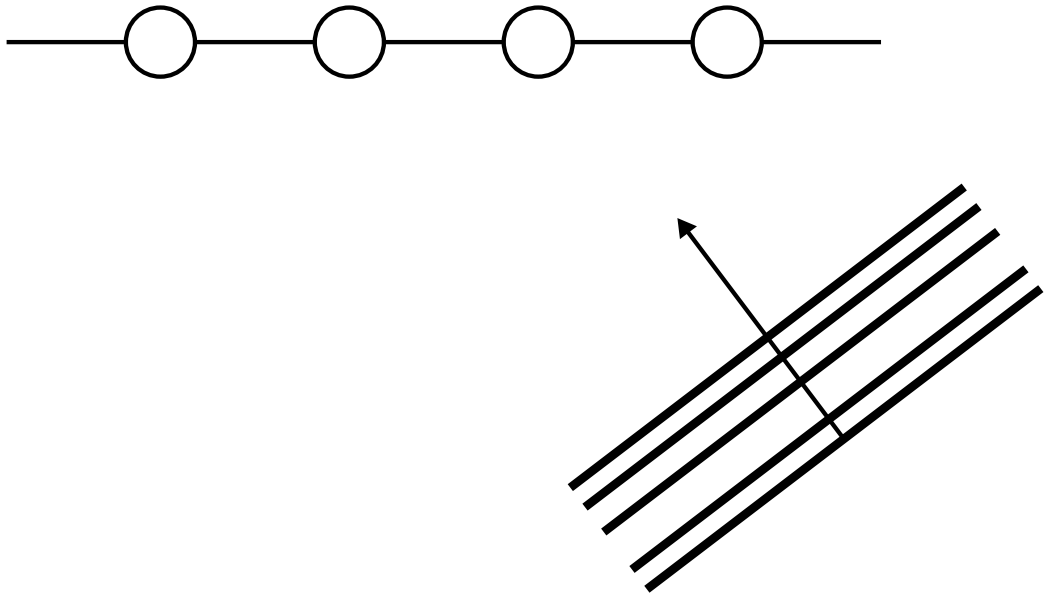


Figure 5.4: A wavefront impinging on a sensor array. The sensors towards the right will receive the wavefront before the sensors on the left.

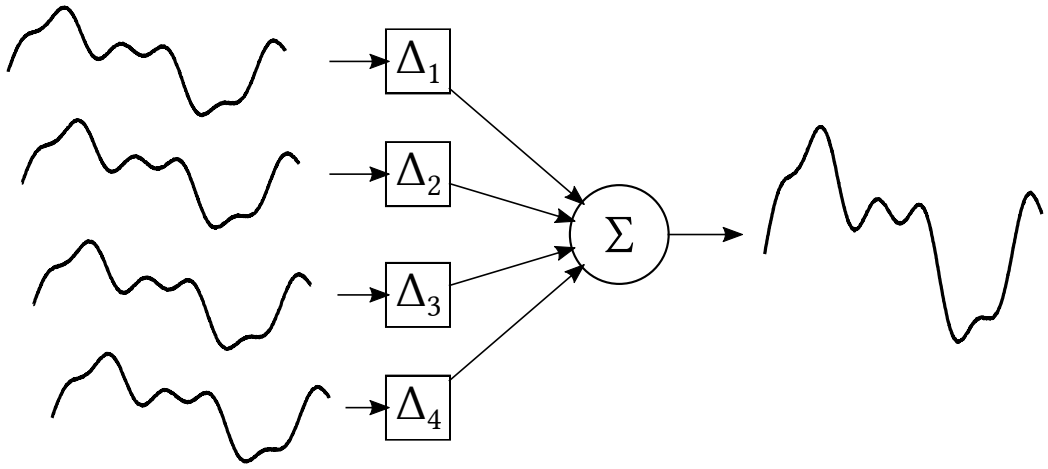


Figure 5.5: Sensor signals being delayed and added together to reinforce the desired signal.

Let

$$f(\boldsymbol{x}, t) = s(t - \boldsymbol{\alpha} \cdot \boldsymbol{x}) \tag{5.6}$$

where $\boldsymbol{x}$ is a position vector, and $\boldsymbol{\alpha} = \hat{\zeta}/c$ is the slowness vector [32]. The waveform measured at the mth sensor is

$$y_m(t) = f(\boldsymbol{x}_m, t) \tag{5.7}$$

where $\boldsymbol{x}_m$ is the position on the mth sensor. The output of the delay and sum beamformer is then defined as [32]

$$z(t) \equiv \sum_{m=0}^{M-1} w_m y_m(t - \Delta_m) \tag{5.8}$$

where w_m is the weight of the mth sensor, and Δ_m is the delay applied to the mth sensor. In this work, all weights are set to 1 for simplicity.

Delay Computation

Given a wave propagating from some direction $\hat{\zeta}$, the set of delays $\{\, \Delta_m \mid 0 \leq m \leq M-1 \,\}$ need to be computed to reinforce waves coming from that direction. Essentially, the choice of delays defines where the array is "looking". The delay equations shown in this section assume that the farfield approximation is valid.

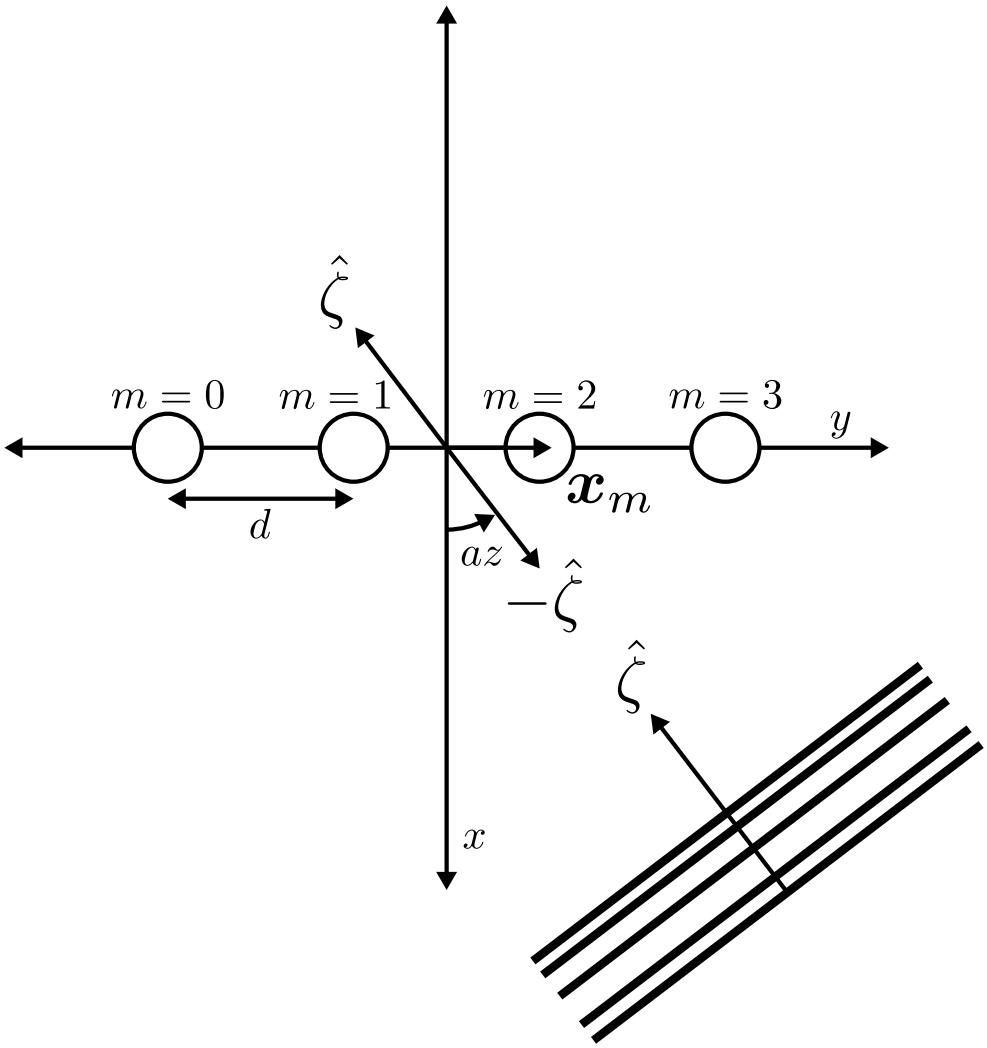


Figure 5.6: A uniform linear array with spacing d. az is the angle with respect to the positive x-axis, known as the azimuth angle; an azimuth of 0 is consider broadside. $\hat{\zeta}$ is direction vector of the wavefront; the wavefront is a plane wave, represented by the varyingly-spaced lines.

For the uniform linear array (ULA) shown in Fig. 5.6, the delay at the mth sensor, relative to the center of the array, also known as the phase center, is [32]

$$\begin{aligned}\tau_m &= \frac{\hat{\zeta}\cdot\boldsymbol{x}_m}{c} = \frac{\|\boldsymbol{x}_m\|}{c}\cos\left(\frac{\pi}{2}+az\right) \\ &= \frac{\|\boldsymbol{x}_m\|}{c}\left(-\sin\left(az\right)\right) \\ &= -\frac{\left(m-\frac{M-1}{2}\right)d}{c}\sin\left(az\right)\end{aligned} \tag{5.9}$$

where az is the azimuth angle shown in Fig. 5.6, M is the total number of sensors, d is the distance between sensors, and c is the wave propagation speed. The signal processing delay used to align the signal to the phase center is $\Delta_m = -\tau_m$. A negative value for τ_m means that the wavefront arrived at the mth sensor before it arrived at the array's phase center, and vice versa for a positive τ_m.

The ULA is the simplest array geometry, but this work uses a uniform rectangular array (URA), like that shown in Fig. 5.7. The delay geometry from the ULA extends to the URA, but now there are two angles—azimuth and elevation—to account for. Computing delays is easiest using direction cosines for the y and z axes, commonly denoted as u and v [36]

$$u = \sin\left(az\right)\cos\left(el\right) \tag{5.10}$$

$$v = \sin\left(el\right) \tag{5.11}$$

where az is the azimuth angle and el is the elevation angle.

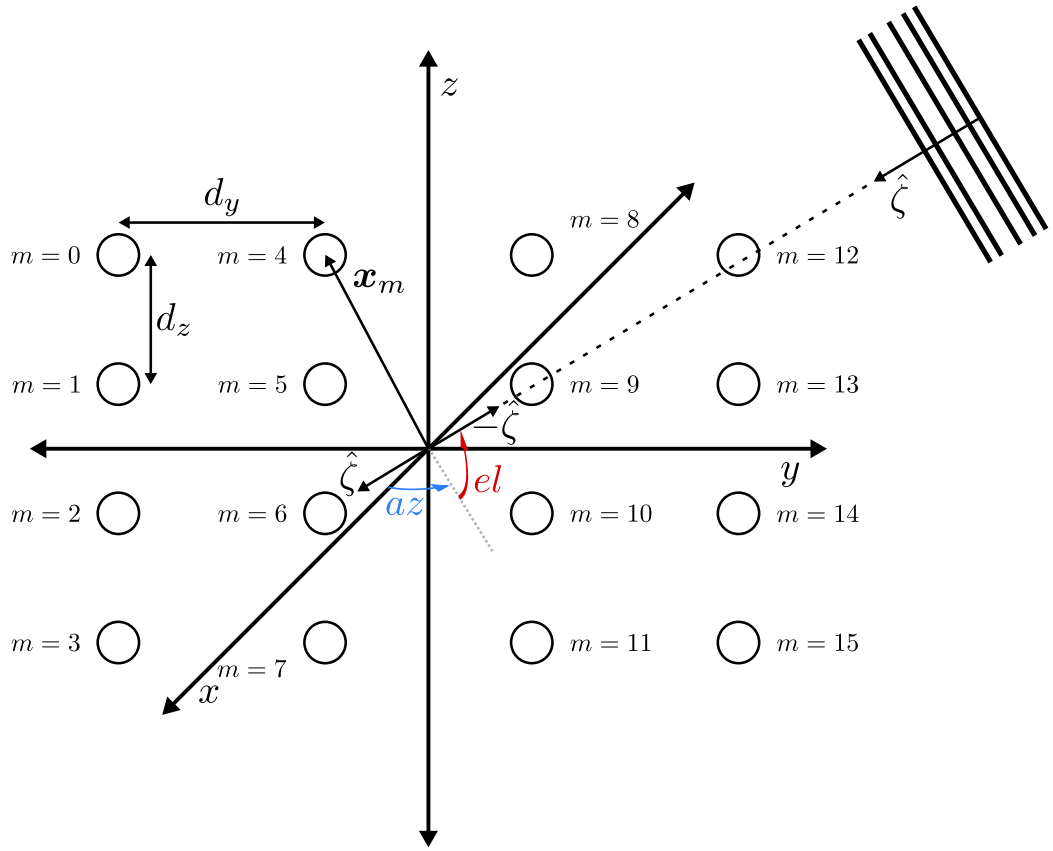


Figure 5.7: A uniform rectangular array. az is the angle with respect to the positive x-axis, known as the azimuth angle. el is the elevation angle, defined as the angle from the x-y plane. $\hat{\zeta}$ is direction vector of the wavefront; the wavefront is a plane wave, represented by the varyingly-spaced lines. d_y and d_z are the horizontal and vertical distances between sensors, respectively.

Using direction cosines, the delay at the mth sensor can be computed as

$$\begin{aligned}\tau_m &= \frac{\hat{\zeta} \cdot \boldsymbol{x}_m}{c} = \frac{1}{c}\left(-u\, x_{m,y} + -v\, x_{m,z}\right) \\ &= \frac{1}{c}\left(- \left(m_y - \frac{M_y - 1}{2}\right) d_y \sin\left(az\right)\cos\left(el\right) - \left(\frac{M_z - 1}{2} - m_z\right) d_z \sin\left(el\right)\right)\end{aligned} \tag{5.12}$$

where m_y and m_z are sensors indices in the y and z directions, and M_y and M_z are the total number of sensors in the y and z directions. For the array used in this work,

$d_y = d_z = d$ and $M_y = M_z$. The delay at the mth sensor can then be written as

$$\tau_m = -\frac{d}{c}\left(\left(m_y - \frac{M-1}{2} \right) \sin(az)\cos(el) + \left(\frac{M-1}{2} - m_z \right) \sin(el) \right) \tag{5.13}$$

Delay Interpolation

In general, the delays needed to reinforce signals coming from a particular direction will not be integer multiples of the temporal sampling period. Since the beamforming is being done digitally, we only have values at integers of the sampling period, thus we cannot exactly implement the required delays. This quantizes the angles we can steer the array to, and leads to errors between the "look" direction and the actual propagation direction.

The realizable steering angles are given by [32]

$$\theta = \sin^{-1}\left(\frac{-ncT_s}{d} \right) \tag{5.14}$$

where n is an integer representing the delay value, and T_s is the sampling period. For the microphone array used here, $d = 15$ mm and $T_s = \frac{1}{48\text{ kHz}}$. Since the domain of of $\sin^{-1}$ must be between ± 1, this leads to only 5 realizable steering angles given the previously-defined values of d and T_s.

To realize more steering angles, we need to interpolate between samples. Interpolation is often realized by upsampling the signal to a faster sampling rate, followed by lowpass filtering. Details of how interpolation was implemented will be discussed in the following section.

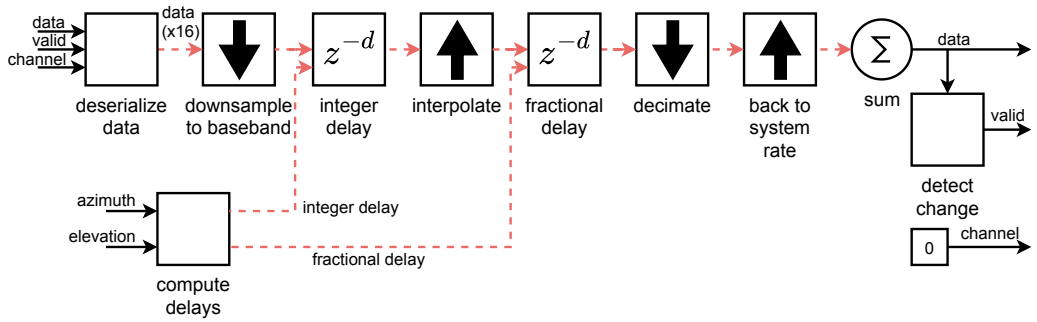


Figure 5.8: Delay and sum beamformer architecture. The dashed red lines indicate vectors of 16 values.

Delay and Sum Beamforming Implementation

The architecture of my delay and sum beamforming implementation is shown in Fig. 5.8. Data comes in as 16 time-division-multiplexed channels on an Avalon Streaming interface; azimuth and elevation angles are registers on the Avalon Memory Mapped bus, which are used to compute the delay for each sensor. Before processing, the data is deserialized into 16 parallel channels. Data is downsampled from the system clock rate to the baseband 48 kHz sampling rate before being delayed. The integer portion of the delay is performed at the 48 kHz rate; this delayed value is then interpolated to a higher sampling rate, after which the fractional part of the delay is implemented. After being completely delayed, the data is decimated back down to the original 48 kHz sampling rate. The data is moved back to the system clock rate, and the sum operation takes place at the system clock rate. After summing, the Avalon Streaming signals are regenerated.

Several of the operations in Fig. 5.8 required special consideration to create hardware-friendly implementations. These considerations are detailed in the following sections. The delay and sum beamforming Simulink model can be found on GitHub[1]

[1] `https://github.com/fpga-open-speech-tools/simulink_models`

Hardware Architecture

Sine and Cosine Lookup Tables The delay computation in Eq. (5.12) contains sine and cosine functions. FPGAs cannot compute transcendental functions like sine and cosine. Instead, these functions are often implemented as lookup tables that contain precomputed values for a set of inputs.

The domains of the azimuth and elevation angles are $[-90°, 90°]$. Taking advantage of sine's and cosine's symmetry about 0, the lookup table domain is reduced to $[0°, 90°]$, cutting the lookup table size in half. To make all angles going into the lookup table positive, the absolute value is taken. Since cosine has even symmetry about 0, no additional operations need to be performed to correct the lookup table output for negative angles. Sine, however, has odd symmetry about 0; consequently, the lookup table output has to be multiplied by -1 if the original angle was negative.

The lookup tables were implemented using MATLAB's Fixed-point Lookup Table Optimizer. This application can convert a Simulink block or a MATLAB function handle to a memory-efficient lookup table. I used MATLAB function handles for $\sin(\pi/180\,\theta)$ and $\cos(\pi/180\,\theta)$. The input data type to the lookup tables was chosen as 16-bit signed fixed-point with 8 fractional bits (s16,8). The output data type was chosen as s16,14. The absolute and relative error tolerances were set to $0.1\pi/180$, which is 0.1 degrees in radians. Linear interpolation between table values was used to further reduce table size, and power of two spacing was used to avoid division operations during linear interpolation. This resulted in tables that only used 48 *bytes* of memory! Comparison plots of the original functions and their lookup table approximations for sine and cosine are shown in Figs. 5.9 and 5.10, respectively.

The Fixed-point Lookup Table Optimizer chose a table data type that was different from the input data type. The HDL Coder didn't like that, so the table data type was changed to make the model HDL compatible.

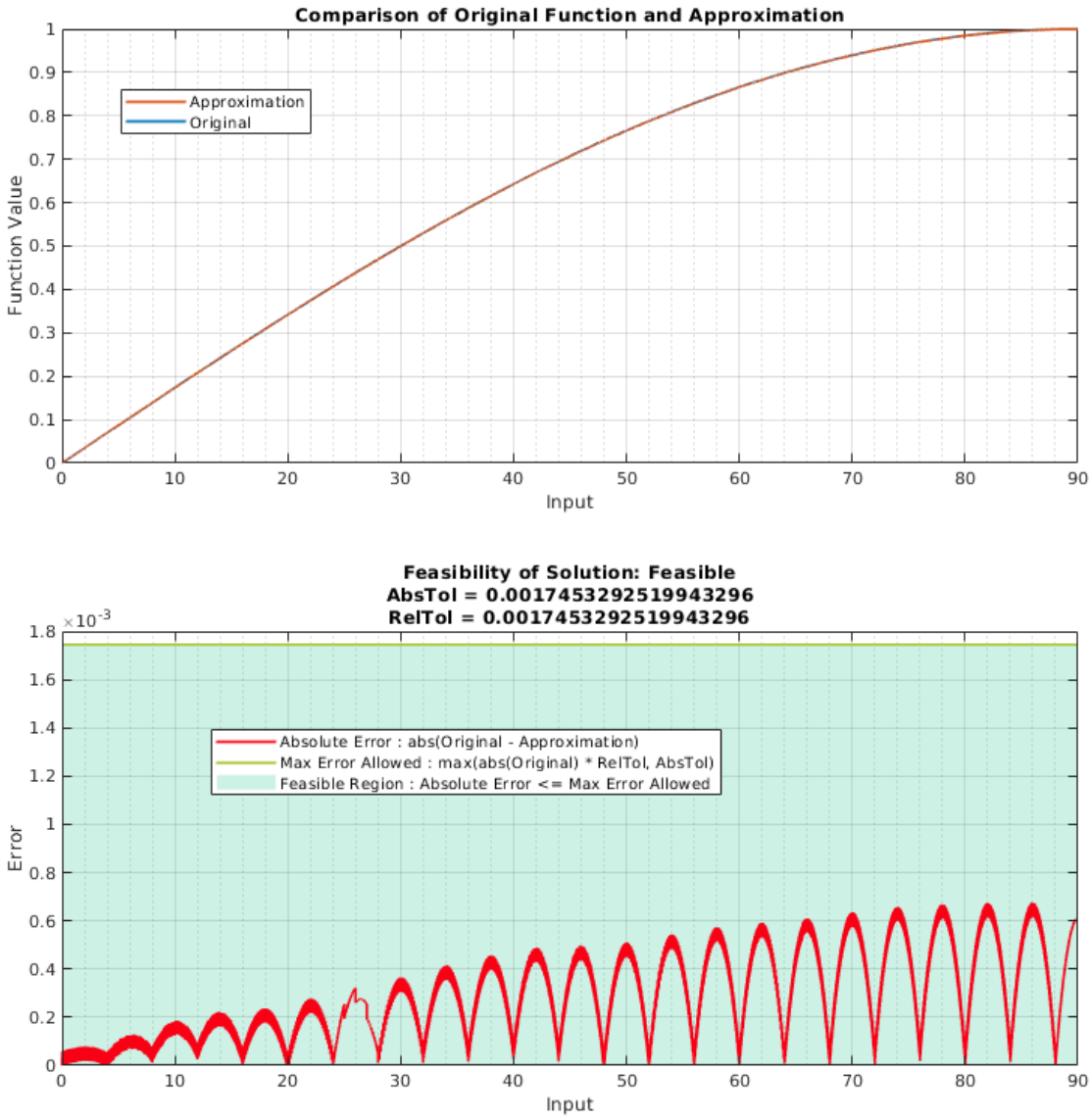


Figure 5.9: Comparison between the original sin function and the lookup table approximation. The absolute error between the approximation and the original function is less than 0.8×10^{-3}.

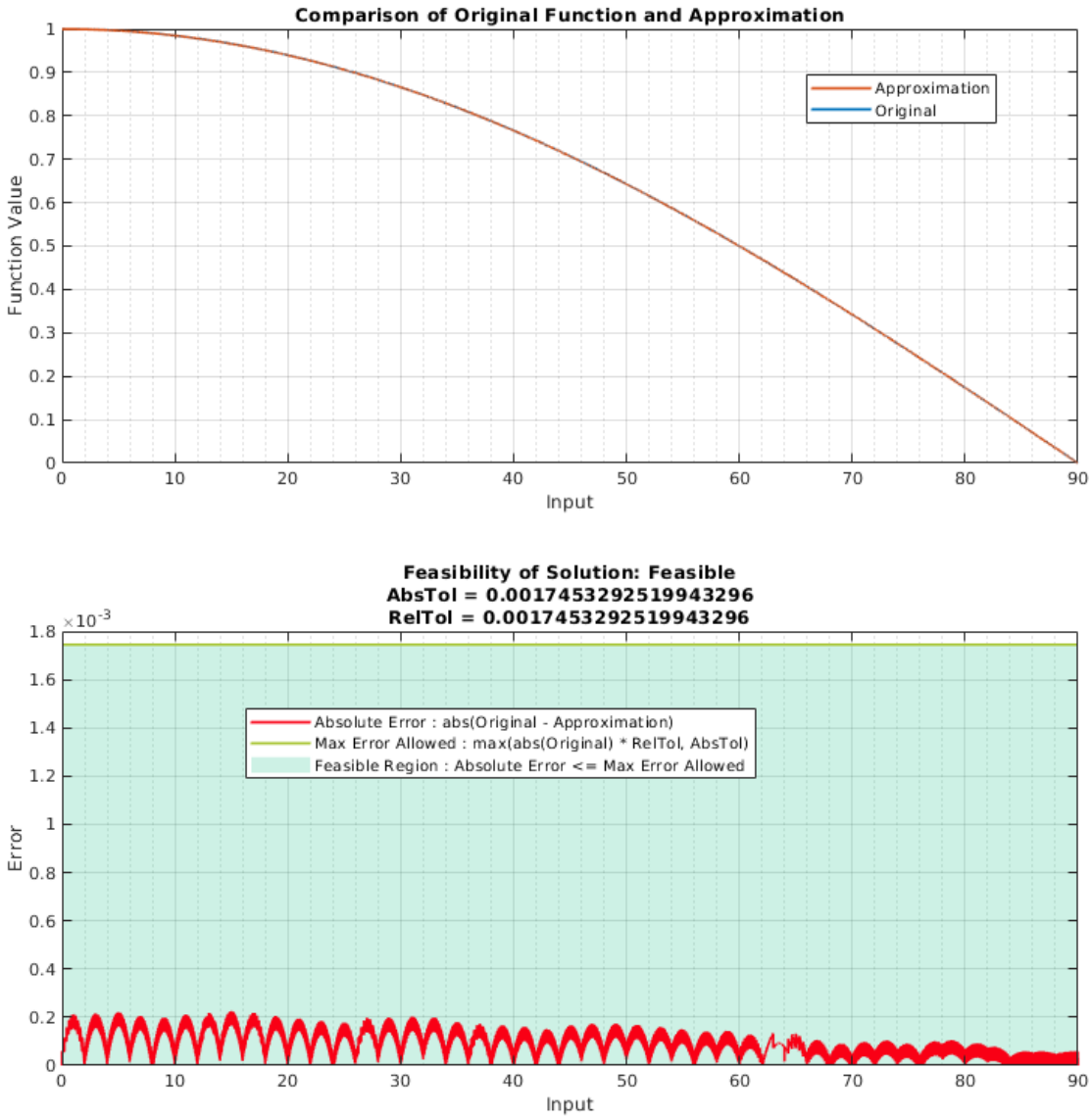


Figure 5.10: Comparison between the original cos function and the lookup table approximation. The absolute error between the approximation and the original function is less than 0.3×10^{-3}.

Fractional Delay As mentioned in the Delay Interpolation section, delays are generally not an integer number of samples, but we must somehow implement these fractional delays. The two primary options for implementing variable fractional delays are via Farrow structure fractional delay filters [37] or interpolation. Out of familiarity and simplicity, interpolation is used here.

Rather than implementing the entire delay with one buffer, separate buffers were used for the integer and fractional parts of the delay to reduce memory usage. To see this reduction in memory usage, consider a delay of 4.5 samples at the original sampling rate. If we interpolated by a factor of 16, the original delay of 4.5 samples is now a delay of 72 samples, requiring the buffer to be at least size 72 (though in practice a power of two would be used for the buffer size). If we instead handle the 4 sample delays at the original sampling rate, then interpolate by a factor of 16, we only need $4 + 0.5 \times 16 = 12$ buffer slots.

Circular buffers were used to implement the integer and fractional delays, just like in the Sound Effects Processor chapter. The interpolation factor was chosen to be 16 which resulted in 169 possible delays and steering angles. A power of two was used to make interpolation hardware-friendly, as discussed in the Interpolation and Decimation Filters section.

Circular Buffer Addressing While the delays in Eq. (5.12) can be both positive and negative, negative delays are non-causal and can't be implemented in a real-time system. To the make the system causal, a constant delay needed to be added to each microphone signal so none of the delays were negative. Each signal is delayed by the absolute value of the maximum delay from Eq. (5.12). This constant delay represents the phase center, or zero reference, of the array, as shown in Fig. 5.11. To avoid unwanted read-during-write behavior, the read pointer is delayed by an

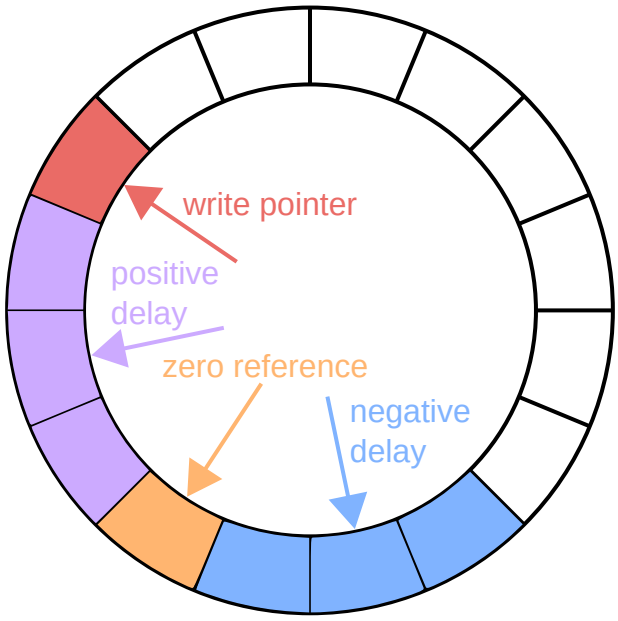


Figure 5.11: Circular buffer addressing scheme for the delay and sum beamformer. A constant delay, represented by the difference between the write pointer and zero reference, is added to each signal to make the system causal. Addresses larger than the zero reference represent positive delays, while addresses smaller than the zero reference represent negative delays.

additional sample so the read and write pointers are never equal.

Interpolation and Decimation Filters Interpolation is a process of upsampling followed by lowpass filtering, and decimation is a process using a lowpass anti-aliasing filter followed by downsampling. Since both processes use lowpass filters, we want to design those filters to be hardware-friendly. Originally, polyphase FIR [38] filters were used. Unfortunately, these filters were 383rd order filters. Having two of these filters—one for interpolation and one for decimation—for each of the 16 microphone signals resulted in a design with far too many multipliers; Quartus could not even synthesize the design.

For a more hardware-friendly implementation, cascaded integrator–comb (CIC) filters [38] were used. These filters don't require any multiplications, which makes

them particularly well-suited for hardware implementation. Unfortunately, the frequency response of these filters is non-ideal, and they have substantial gain. FIR filters are typically used to compensate for the frequency response of CIC filters by implementing the inverse of the CIC filter's frequency response; in practice, implementing the exact inverse response is not possible with an FIR filter. The FIR CIC compensation filters were both of order 18, which is much more reasonable for hardware implementation.

As mentioned earlier, the interpolation factor was chosen as a power of two. The advantage of this is that the CIC filter gains become powers of two, so bit-shift operations can be used to compensate for the CIC filter gain. Shifting to the right by N bits is far more efficient than dividing by 2^N—bit shifting requires very few resources.

In hindsight, the decimation filter was not necessary since the original signal was bandlimited to the decimated rate; just downsampling would have saved resources.

Resource Usage

The resource usage reported by Quartus is shown in Table 5.1. The design used 22% of the Arria 10's ALMs, 3% of the device's M20K memory blocks, and 43% of the DSP blocks. This Quartus project is on GitHub[2].

ALMs	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	M10K Memory Blocks	DSP Blocks
53715	100573	85300	69376	53	725

Table 5.1: Delay and sum beamformer IP core resource usage.

[2] https://github.com/fpga-open-speech-tools/arria10_projects/

Real-time Beamforming System

To enable real-world deployment of the beamforming Simulink model discussed in the previous section, a real-time beamforming hardware system was created. An overview of the hardware system is shown in Fig. 5.12. The system consists of up to 8 microphone arrays connected to an Arria 10 FPGA via a custom daughter card and Cat 6 cables. Data acquisition and beamforming are done inside the FPGA. The beamformed audio is sent to an audio codec on the daughter card for listening. The hardware and FPGA systems that comprise the real-time beamforming system will be described in the following sections.

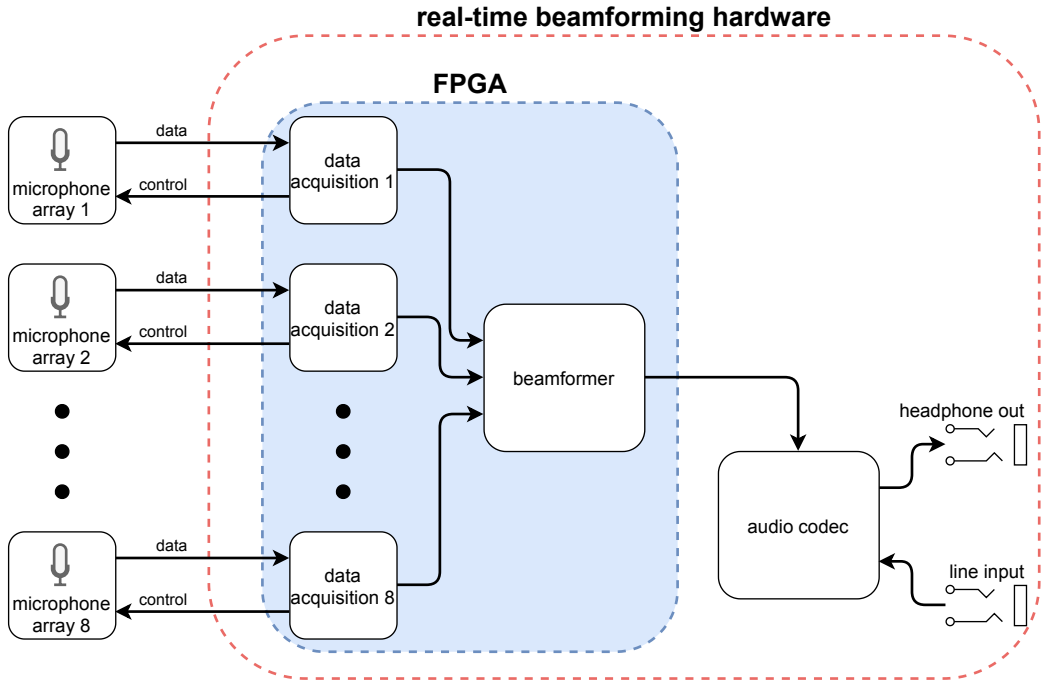


Figure 5.12: High-level block diagram of the real-time beamforming hardware system. The system supports up to 8 microphone arrays.

Microphone Array

The microphone array used in this work is a TDK InvenSense ICS-52000 TDM Array Demo Board, shown in Fig. 5.13. It is a 4x4 array of ICS-52000 microelectromechanical systems (MEMS) microphones, each spaced 15 mm apart. The microphone array outputs digital words for each microphone in a time-division multiplexed (TDM) format.

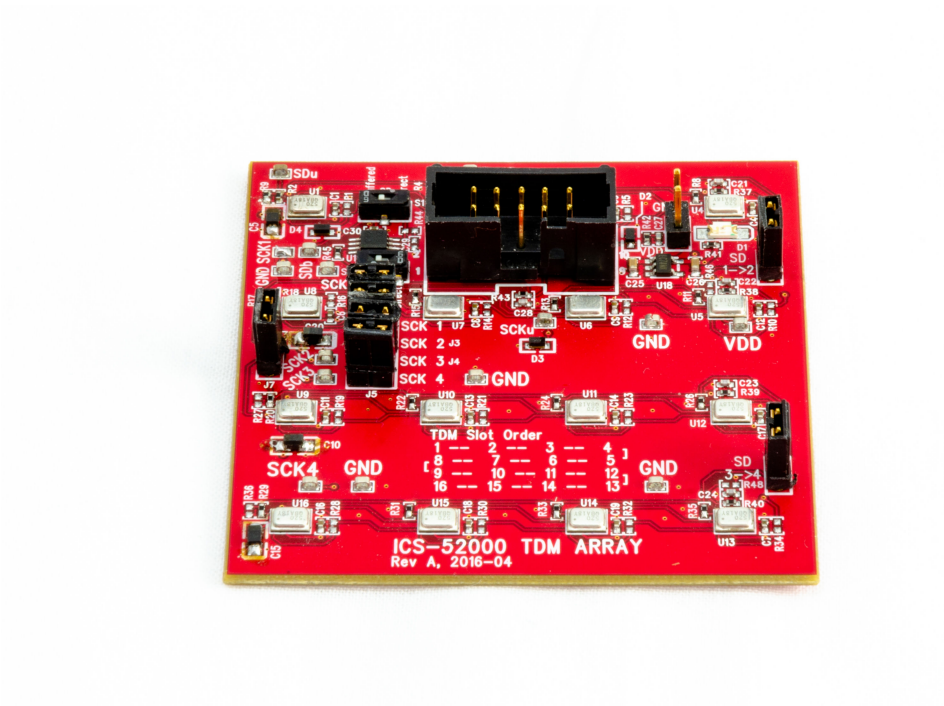


Figure 5.13: TDK InvenSense ICS-52000 TDM Array Demo Board. This board has 16 ICS-52000 MEMs microphones arranged in a 4x4 grid with 15 mm spacing.

The TDM format uses a single serial data line for all 16 microphones. The 24-bit words from each microphone are put into adjacent 32-bit slots on the serial data line. The TDM interface uses the following lines:

- `sck`: serial data clock
- `ws`: serial data word-select

- `sd`: serial data output

`sck` and `ws` are inputs driven from the FPGA, and `sd` is an output. The `ws` line is used to tell the microphones when to sample; this line is asserted every 48 kHz sampling period.

Microphone Array Adapter Board

As mentioned earlier, the microphone arrays are connected to the FPGA daughter card via Cat 6 cables. Since the TDK microphone array doesn't have an RJ45 connector for Cat 6 cables, I designed an adapter board that does. A block diagram of the adapter board is shown in Fig. 5.14.

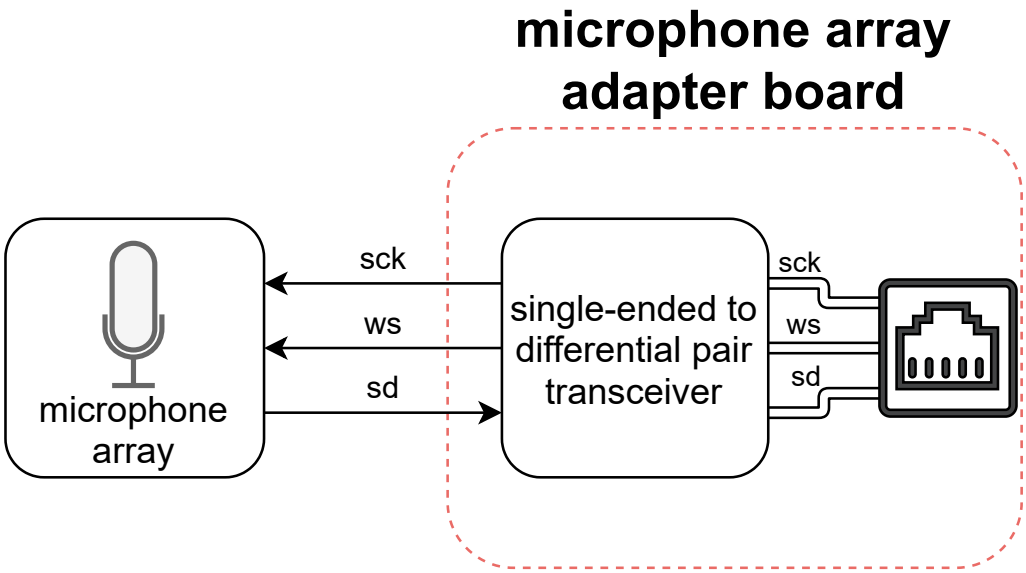


Figure 5.14: Block diagram of the microphone array adapter board. Differential pairs are indicated by double lines.

The adapter board simply takes the TDM interface signals from the TDK array, converts them to differential pairs, and routes them to an RJ45 connector. Cat 6 cable was chosen to transport the microphone array signals because Cat 6 cables are the most readily-available twisted pair cable; using twisted pairs provides better noise

immunity when transmitting over long distances, which was a primary consideration in this work because the microphone arrays may be quite far away from the FPGA.

Fig. 5.15 shows the finished microphone array adapter board. The printed circuit boards were designed with KiCad, fabricated by OSH Park, and assembled by myself. Before fabrication, mechanical fit with the TDK array was verified by 3D printing a mockup of the adapter board.

Schematics and PCB files for the microphone array adapter board are located in Appendix I; source code and original design files are located on GitHub[3].

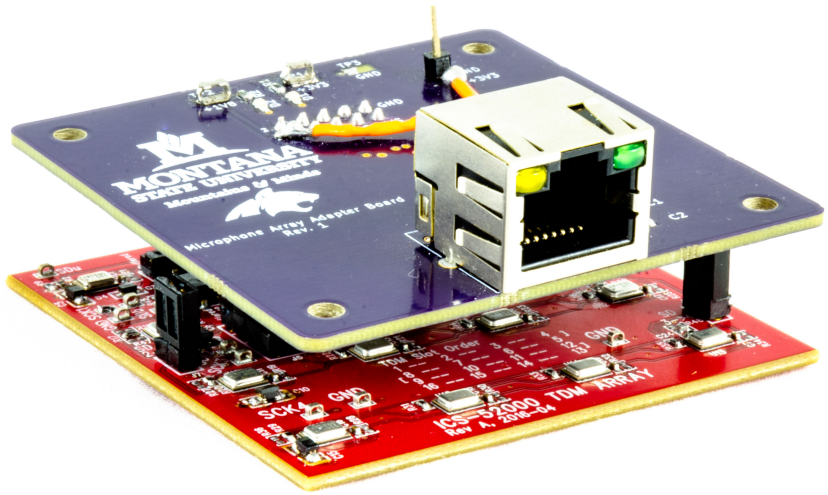


Figure 5.15: The microphone array adapter board mounted on the TDK microphone array.

[3]https://github.com/fpga-open-speech-tools/hardware/tree/old

Beamforming Daughter Card

A custom daughter card was developed for the Reflex Achilles Arria 10 SoC SoM so the microphone arrays could be connected to the FPGA. A block diagram of this beamforming daughter card is shown in Fig. 5.16. The board consists of 8 microphone array interfaces, a connector for general purpose input/output (GPIO), an audio codec, and two tip-ring-sleeve audio connectors. The microphone array interfaces are identical to the hardware on the microphone array adapter board. The daughter card connects to the Reflex Achilles via a high-pin count FPGA mezzanine connector.

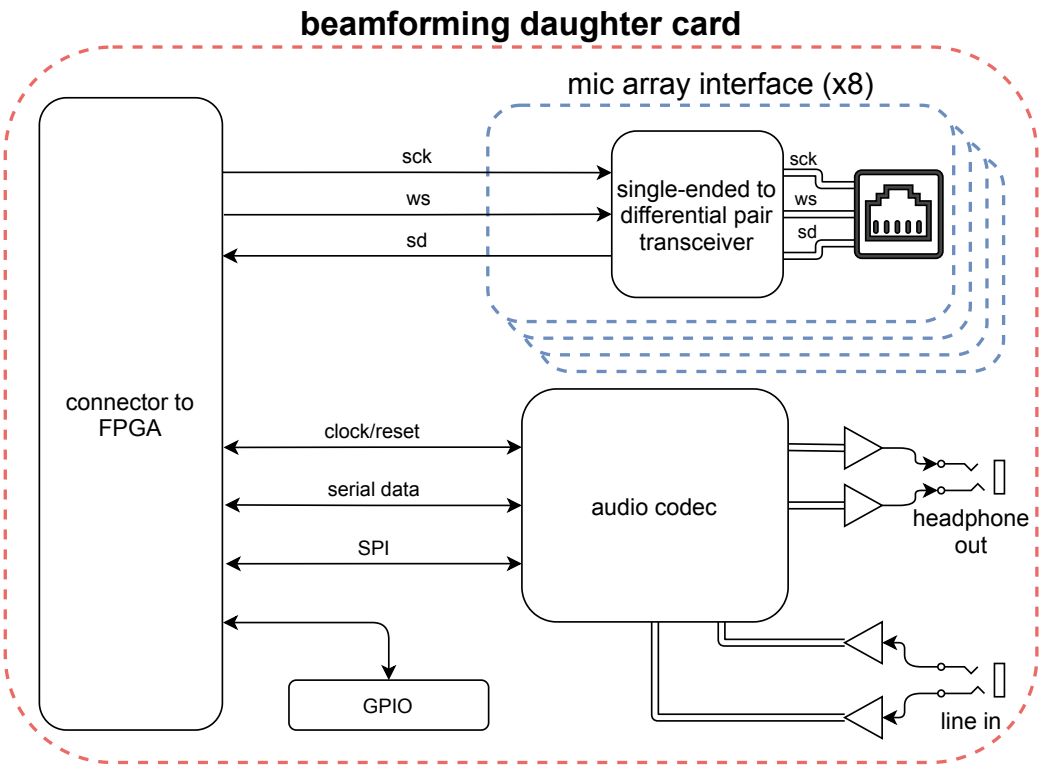


Figure 5.16: Beamforming daughter card block diagram. The board has 8 microphone array interfaces, an audio codec, and GPIO. Differential pairs are indicated by double lines.

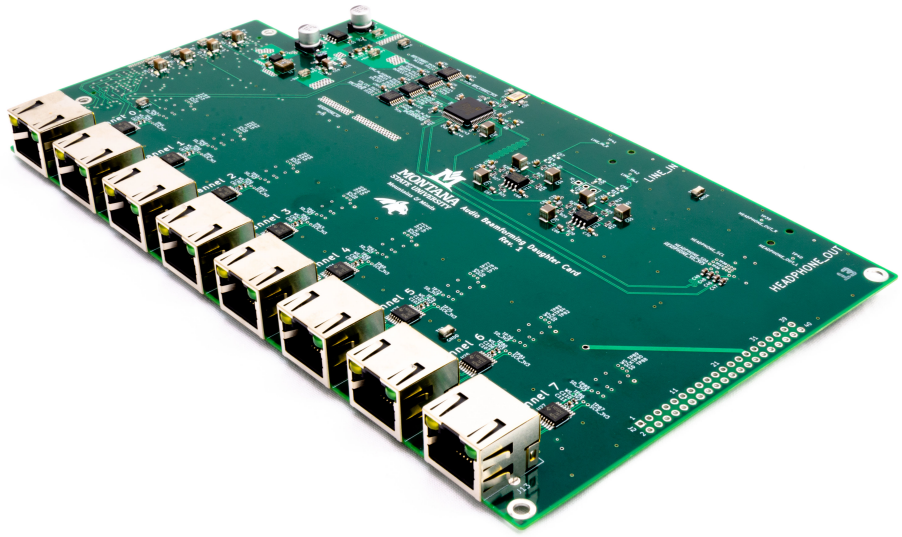


Figure 5.17: Picture of the assembled beamforming daughter card.

A picture of the finished circuit board is shown in Fig. 5.17. The board was designed with KiCad, fabricated by Sunstone Circuits, and assembled by Screaming Circuits. Schematics and PCB files are located in Appendix J; source code and design files are located on GitHub[4].

FPGA Dataplane

With the two circuit boards described in the previous sections, microphone array data is able to get into the FPGA. The block diagram in Fig. 5.18 shows an overview of the FPGA design created to capture and process the microphone array data. All of the clocks (`sck`, `sck rcv`, and `sys clk`) are generated with a phase-locked loop not shown in the figure. Fig. 5.18 shows the dataplane for one microphone array; the dataplane is replicated for each microphone array being used (only one array is used

[4] `https://github.com/fpga-open-speech-tools/hardware/tree/old`

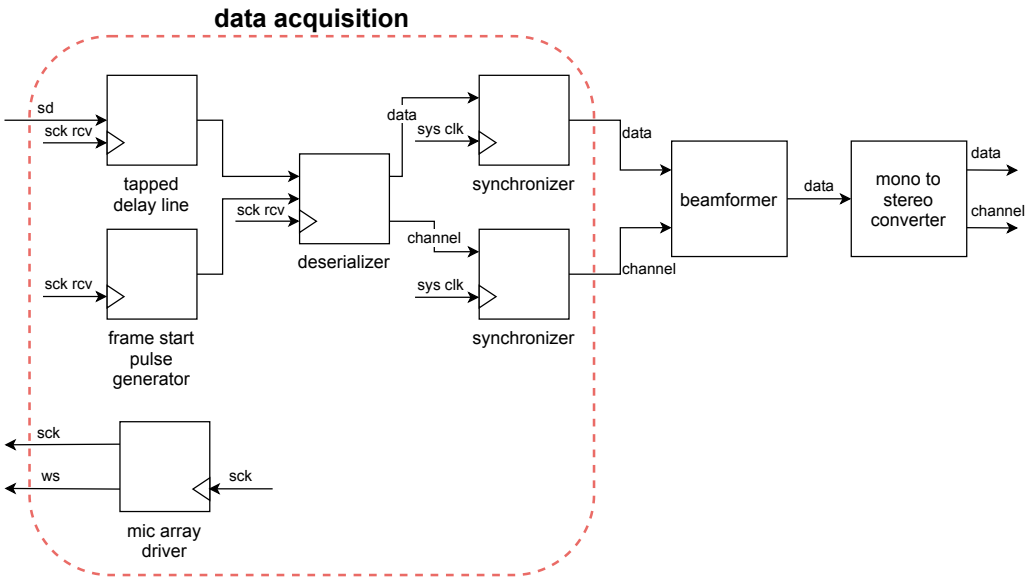


Figure 5.18: FPGA dataplane block diagram for the real-time beamforming system. The dataplane comprises a data acquisition subsystem, the beamformer, and a mono to stereo adapter.

in this work).

The main subsystem, besides the beamformer, in the dataplane deals with data acquisition, as shown in Fig. 5.18. A mic array driver creates the `sck` and ws signals needed to drive the microphone array. When the microphone array receives a rising edge of the ws signal, it samples and sends data back to the FPGA. The data will take some time to propagate through the Cat 6 cable before getting to the FPGA. In general, the propagation delay will not be an integer multiple of the `sck` clock rate, so the data will not line up with the rising-edge of the `sck` clock. This causes problems since we want to sample the data while it is stable, not while it is transitioning between bits. To remedy this, a phase-shifted version of `sck`, `sck rcv` (short for sck receive), is used to sample the data. The desired phase-shift is calculated by measuring the time difference between the middle of a received data bit and the rising edge of `sck`; `sck rcv` is shifted relative to `sck` by this time difference using a phase-locked loop.

Since the cables connecting the microphone arrays to the FPGA daughter card might be different lengths, the serial data coming from the arrays can arrive at different points in time. A tapped delay line is used to align data coming from multiple microphone arrays. The time difference between sending out a `ws` pulse and receiving the first serial data bit in a sampling frame (16 TDM slots) is measured empirically and rounded to the nearest integer number of `sck` clock cycles. This propagation delay is used to determine which tap to select in the tapped delay line: tap number = max tap number − propagation delay in clock cycles. Choosing the tap number in this way ensures that the effective delay between sending a `ws` pulse and receiving data is always the tapped delay line length; cable lengths are chosen such that the propagation delay is never larger than the delay line length.

Once data goes through the tapped delay line, it is then converted into parallel data by a deserializer, shown in Fig. 5.19. The deserializer shifts the serial data

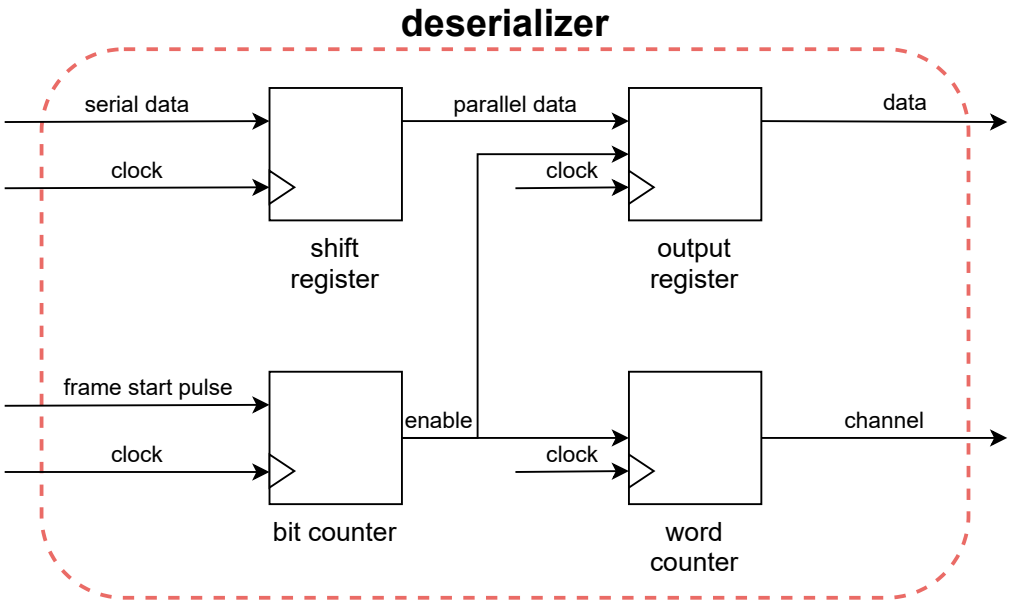


Figure 5.19: Deserializer block diagram. The deserializer comprises a serial-to-parallel shift register, two counters, and an output register.

through a serial-to-parallel shift register. The bit counter starts counting when it receives a frame start pulse, which indicates that the first TDM slot of the sampling frame has arrived. The frame start pulse is generated by the frame start pulse generator, which is a counter that sends out a pulse every multiple of the tapped delay line length. The bit counter counts 24 cycles and then asserts it's enable output. The bit counter continues counting until 32 cycles have passed, since this is the TDM slot length, and then starts back at 0. The enable output of the bit counter is used to enable the output register, which registers the current microphone sample. A word counter increments the channel number (which is equivalent to the TDM slot number) every time the bit counter enable is asserted.

The data and channel coming from the deserializer are synchronized to the system clock (`sys clk`) domain. When transferring data between clocks that have different phases and/or frequencies, synchronizers are used to prevent metastability. Metastability can occur when data is sampled during a transition. The synchronizers used here are a chain of two D flip-flops. For a good tutorial on synchronizers, see [39].

After data acquisition is done, beamforming is performed using the delay and sum beamformer developed in the Delay and Sum Beamforming Implementation section. Since the beamformer sums all 16 microphones into one signal, it's output is mono. A mono to stereo converted is used to create stere audio to send to the audio codec. This converter sets the channel number as zero (left channel) for the first half of the 48 kHz sampling period, then sets the channel number as one (right channel) for the second half of the sampling period.

VHDL code for the subsystems shown in Fig. 5.18 can be found in Appendix K and on GitHub[5].

[5] `https://github.com/fpga-open-speech-tools/component_library`

Real-time Beamforming Test

Some basic testing was performed to verify that the real-time beamforming system and algorithm were working. These tests were performed using Flat Earth's Audio Research daughter card, shown in the High Performance Hardware section of the Background chapter. The daughter card discussed in the Real-time Beamforming System section was not used due to a hardware bug that caused the on-board headphone amplifier to not work, resulting in no audio output; this bug is present on the Audio Research daughter card as well, but that board has additional audio outputs besides the headphone output.

The testing setup is shown in Fig. 5.20. A speaker was placed approximately 200 mm away from the array, pointing directly at the center of the array. 1 kHz and 10 kHz test tones were played out of the speaker; for each test tone, the array's look direction was set to the following angles: $az = 0°$ $el = 0°$, looking directly at the speaker, and $az = 45°$ $el = 0°$. The beamformed output was sent to the line output on the Audio Research board. The output was then recorded on a computer using Audacity.

The results for the 1 kHz and 10 kHz test tones are shown in in Figs. 5.21b and 5.22b, respectively. Figs. 5.21a and 5.22a show the theoretical directivity patterns

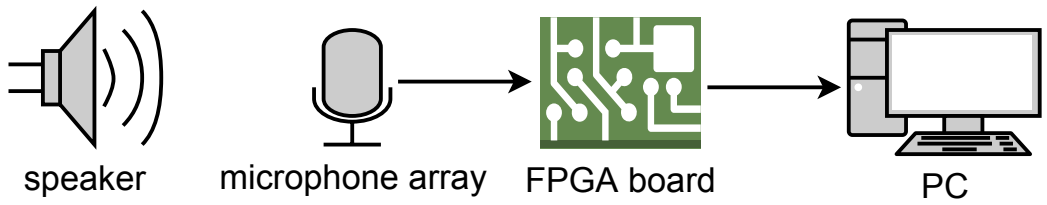
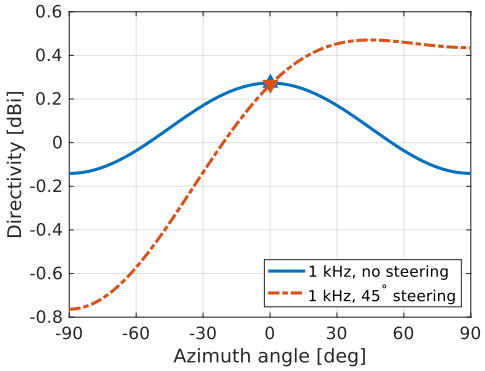
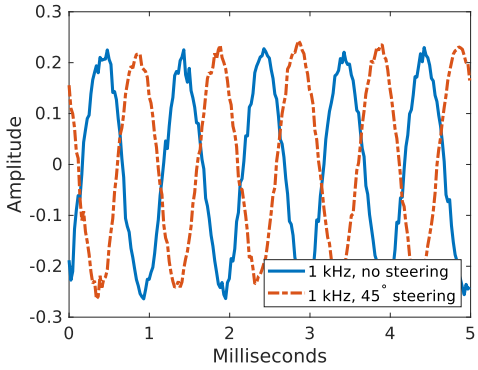


Figure 5.20: Real-time beamforming test setup. Test tones were played through a speaker, and the output of the beamformer was recorded on a computer with Audacity.

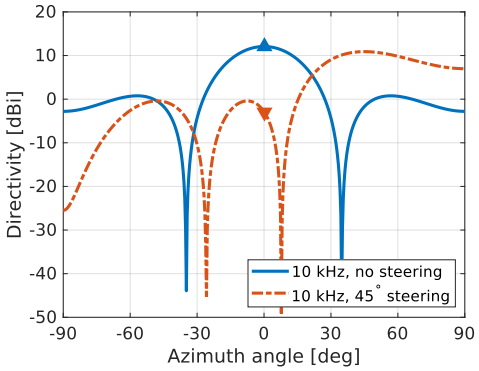


(a) Directivity patterns at 1 kHz for steering angles of 0° and 45°. Markers are placed to indicate the gain for a source at 0°.

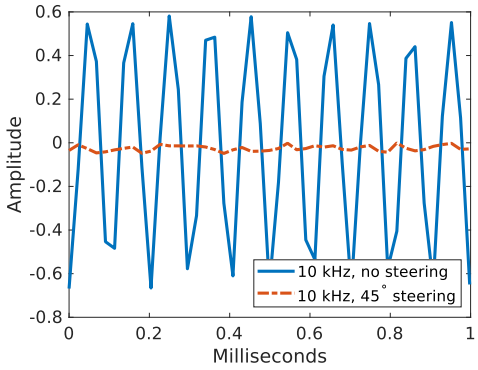


(b) Testing results for a 1 kHz test tone. The responses when the array was set to look broadside and at 45° were nearly identical.

Figure 5.21: Expected and measured results for the 1 kHz beamforming test.



(a) Directivity patterns at 10 kHz for steering angles of 0° and 45°. Markers are placed to indicate the gain for a source at 0°.



(b) Testing results for a 10 kHz test tone. The response at broadside, where the source was located, is much larger than the response when the array was set to look at 45°.

Figure 5.22: Expected and measured results for the 10 kHz beamforming test.

for the tested steering angles at 1 kHz and 10 kHz, respectively. Fig. 5.21b shows no discernable difference between the two steering angles for the 1 kHz test tone, which is to be expected since the markers in Fig. 5.21a are at nearly the same gain

value. Moreover, the wavelength of a 1 kHz sound wave is ∼343 mm, which is much larger than the 15 mm spacing between microphones. Fig. 5.22b shows a significant difference in gain between steering angles for the 10 kHz test, which is also reflected in Fig. 5.22a; the 45° response has a very small amplitude, indicating significant attenuation.

The relative gain between the broadside and 45° 1 kHz test cases was measured as ~ 0.01274 dB; the theoretical value based upon the curves shown in Fig. 5.21a is ~ 0.00693. The relative gain between the broadside and 45° 10 kHz test cases was measured as ~ 26.82 dB; the theoretical value based upon the curves shown in Fig. 5.22a is ~ 15.08 dB. The difference between the measured and theoretical gains is likely due to a mismatch between the actual microphone spacing and the spacing the algorithm was designed for: the algorithm was designed for an in-development microphone array with 25 mm spacing, not the 15 mm spacing of the TDK array that was use. Due to time constraints, this test was not run again with the proper microphone spacing.

Although the measured results do not exactly match the theoretical results, the general behavior of the array is consistent with the expected behavior. For the purposes of verifying basic hardware system functionality, this test was successful.

CONCLUSION

The development framework presented here makes developing audio processing applications on SoC FPGAs significantly easier. Designs are developed graphically in Simulink, which matches the block diagram abstraction level used when sketching designs on paper. Automatic code generation eliminates the need to be an expert in low-level programming, but some familiarity with hardware and FPGA design is still beneficial, if not necessary. All in all, the development framework makes FPGAs more accessible to non-experts; FPGA engineers using the development framework will find a significant reduction in development time.

A real-time sound effects processor was developed to showcase the development process and provide a reference design for developing audio processing systems. The sound effects processor demonstrated two important hardware architectures: variable-delay circular buffers and direct digital synthesis of waveforms; these architectures are common design patterns that can be used in many different applications. Real-time control of the sound effects is performed via a tablet application that communicates with the ARM processor in the SoC FPGA; automatic generation of the control application makes interfacing with the FPGA design quick and easy.

The real-time delay and sum beamformer provides a reference design for high-performance beamforming applications. Hardware was developed to support real-time beamforming applications using up to 8 microphone arrays. In addition to serving as a reference design for beamforming, the delay and sum beamformer Simulink model showcases important design considerations for multi-rate audio processing algorithms, as well as an example of creating highly-optimized lookup tables for trigonometric functions.

Overall, the work presented here is a step towards the dream of making the

performance of FPGAs accessible to all algorithm developers. Current proprietary solutions to high-level FPGA design often don't allow users to easily control algorithms during deployment; the development framework presented in the Development Framework chapter remedies this. All of the code I developed is open source, and will hopefully prompt further creation of open source development tools.

Future Work

Future work includes automating the remaining manual development processes, with hopes of creating a one-click solution to go from the algorithm model to a deployable implementation. The delay and sum beamformer's resource usage should be further optimized by removing the decimation filter and implementing resource folding/reuse. I would also like to create a multi-array beamforming reference design that utilizes 8 microphone arrays.

Lessons Learned

Throughout the process of designing the development framework and the example designs, many lessons were learned:

- To take advantage of the modulo wrap-around behavior of binary numbers, circular buffers need to be sized as a power of two.
- Always read software documentation and release notes to learn what functionality already exists; I found MATLAB features after I had already implemented things myself far too many times.
- Polyphase filters, while efficient, are not hardware-friendly for large interpolation/decimation factors because the filtering takes place at the higher sampling

rate; this causes the filters to need more coefficients to achieve the desired cutoff frequency. CIC filters are more hardware-friendly because they don't use any multiplications; the CIC compensation filters can be implemented at the lower sampling rate, resulting in lower-order filters compared to implementing them at the higher sampling rate.

- Although Simulink helps make FPGA development more accessible, developers still need to design the Simulink model with hardware architecture in mind; developers still need to know how to design real-time DSP hardware systems, but they don't need to be experts in all of the lowest-level details.

- Unfortunately, Simulink's high-level abstraction can sometimes hide important design considerations.

- Things that are simple to implement in code, like conditional statements and loops, can be difficult to implement in Simulink; MATLAB function blocks are useful here.

REFERENCES CITED

[1] G. E. Moore *et al.*, "Cramming more components onto integrated circuits."

[2] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted MOSFET's with very small physical dimensions," vol. 9, no. 5, pp. 256–268.

[3] L. B. Kish, "End of moore's law: thermal (noise) death of integration in micro and nano electronics," vol. 305, no. 3-4, pp. 144–149.

[4] M. M. Waldrop, "The chips are down for moore's law," vol. 530, no. 7589, pp. 144–147.

[5] K.Rupp. 42 years of microprocessor trend data. https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/.

[6] R. Weber and R. Snider, "Development of an open speech signal processing platform," 2016-2021, r44DC015443.

[7] R. K. Snider, C. N. Casebeer, and R. J. Weber, "An open computational platform for low-latency real-time audio signal processing using field programmable gate arrays," vol. 143, no. 3, pp. 1737–1737, invited talk.

[8] T. Vannoy, T. Davis, C. Dack, D. Sobrero, and R. Snider, "An open audio processing platform using soc fpgas and model-based development," in *Audio Engineering Society Convention 147*. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=20623

[9] R. K. Snider, T. Vannoy, J. Eaton, M. Blunt, E. B. Galacci, J. Williams, and T. B. Davis, "Real-time audio signal processing using system-on-chip field programmable gate arrays," vol. 146, no. 4, pp. 2879–2879.

[10] B. J. LaMeres, *Introduction to Logic Circuits & Logic Design with VHDL*. Springer International Publishing.

[11] Intel. Intel cyclone v device overview. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_51001.pdf

[12] ——. Intel cyclone v fpga floorplan. [Online]. Available: https://www.intel.com/content/www/us/en/products/programmable/fpga/cyclone-v/features.html

[13] ——. Intel agilex f-series fpga and soc fpga product table. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/pt/intel-agilex-f-series-product-table.pdf

[14] ——. Intel agilex variable precision dsp blocks user guide. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/agilex/ug-ag-dsp.pdf

[15] ——. Export compliance metrics. [Online]. Available: https://www.intel.com/content/dam/support/us/en/documents/processors/APP-for-Intel-Core-Processors.pdf

[16] ——. Intel cyclone v fpga and soc fpga product table. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/pt/cyclone-v-product-table.pdf

[17] EDN. General motors develops two-mode hybrid powertrain with model-based design. [Online]. Available: https://www.edn.com/electronics-products/electronic-product-releases/other/4367477/General-Motors-Develops-Two-Mode-Hybrid-Powertrain-With-Model-Based-Design

[18] Sparkfun. PCB Basics - learn.sparkfun.com. [Online; accessed 31. Jan. 2020]. [Online]. Available: https://learn.sparkfun.com/tutorials/pcb-basics/all

[19] H. W. Ott, "Partitioning and layout of a mixed-signal pcb," vol. 18, no. 6, pp. 8–11. [Online]. Available: http://hottconsultants.com/pdf_files/june2001pcd_mixedsignal.pdf

[20] S. Zaporowski, M. Blaszke, and D. Weber, "Measurement of latency in the android audio path," in *Audio Engineering Society Convention 144*. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=19526

[21] A. McPherson and V. Zappi, "An environment for submillisecond-latency audio and sensor processing on beaglebone black," in *Audio Engineering Society Convention 138*. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=17755

[22] Linux kernel community. Platform devices and drivers. The kernel development community. [Online]. Available: https://www.kernel.org/doc/html/latest/driver-api/driver-model/platform.html

[23] ——. sysfs manual page. [Online]. Available: http://man7.org/linux/man-pages/man5/sysfs.5.html

[24] ——. Kernel build system. The kernel development community. [Online]. Available: https://www.kernel.org/doc/html/latest/kbuild/index.html

[25] J. O. S. III. Tapped delay line. [Online]. Available: https://ccrma.stanford.edu/~jos/pasp/Tapped_Delay_Line_TDL.html

[26] ——. Flanging. [Online]. Available: https://ccrma.stanford.edu/~jos/pasp/Flanging.html

[27] ——. Feedforward comb filters. [Online]. Available: https://ccrma.stanford.edu/~jos/pasp/Feedforward_Comb_Filters.html

[28] L. Cordesses, "Direct digital synthesis: a tool for periodic wave generation (part 1)," vol. 21, no. 4, pp. 50–54.

[29] Mathworks. Nco hdl optimized. Mathworks. [Online]. Available: https://www.mathworks.com/help/dsp/ref/ncohdloptimized.html

[30] J. Vanderkooy and S. P. Lipshitz, "Dither in digital audio," vol. 35, no. 12, pp. 966–975. [Online]. Available: http://www.aes.org/e-lib/browse.cfm?elib=5173

[31] L. Schuchman, "Dither signals and their effect on quantization noise," vol. 12, no. 4, pp. 162–165.

[32] D. E. D. Don H. Johnson, *Array Signal Processing*. Pearson Education (US), 1993.

[33] H. Trees, *Optimum array processing*. New York: Wiley-Interscience, 2002.

[34] D. W. Michael Brandstein, *Microphone Arrays*. Springer Berlin Heidelberg, 2001.

[35] R. Mailloux, *Phased array antenna handbook*. Boston: Artech House, 2005.

[36] Mathworks. Spherical coordinates. Mathworks. [Online]. Available: https://www.mathworks.com/help/phased/ug/spherical-coordinates.html

[37] C. Farrow, "A continuously variable digital delay element," in *1988., IEEE International Symposium on Circuits and Systems*. IEEE.

[38] R. Lyons, *Understanding digital signal processing*. Upper Saddle River, NJ: Prentice Hall, 2011.

[39] R. Ginosar, "Metastability and synchronizers: A tutorial," *IEEE Design & Test of Computers*, vol. 28, no. 5, pp. 23–35, sep 2011.

APPENDICES

APPENDIX A

SIMULINK MODEL SCRIPTS

sm_run_me_first.m

```
% Run_me_first
%
% This scripts sets up the computer environment parameters
% (directories, paths, tools, etc) that the model needs.  It places these
% parameters in the data structure SG, which is the model name initials.
% This setup is needed for the Matlab HDL Coder to generate VHDL code.

% Copyright 2019 Flat Earth Inc
%
% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
% INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
% PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
% FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
% ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
%
% Ross K. Snider
% Flat Earth Inc
% 985 Technology Blvd
% Bozeman, MT 59718
% support@flatearthinc.com

%% Clear the workspace
clear all   % clear all workspace variables
close all   % close all open Matlab windows
clc         % clear command window

%% Fast simulation
% Fast simulation reduces the Fs_system clock rate to reduce the number of
% simulated clock cycles between each sample in the Avalon bus signals.
% It also reduces the number of stimulus samples.  This allow for faster
% development iterations when developing the simulink model.
mp.fastsim_flag = 1;     % perform fast simulation  Note: fast simulation will be turned off when generating VHDL code since we need to run at the system clock rate.
mp.fastsim_Fs_system_N = 8;     % (typical value 2 or 4) Simulate a much slower system clock than what is specified in sm_callback_init.m   - The reduce rate will be a multiple of the sample rate, i.e. mp.Fs_system = mp.Fs*mp.fastsim_Fs_system_N
mp.fastsim_Nsamples = 12000; % set to the string 'all' to use all the samples from the input signal specified in sm_init_test_signals.m


%% Model parameters
```

```
38 % Model parameters are placed in a data structure called mp that can be passed to
   ↪ functions
39 mp.model_name = 'bitcrusher';
40 mp.model_abbreviation = 'BC';
41
42 % Device driver version for the Linux device tree. Typically set as the Quartus
   ↪ version
43 mp.linux_device_name = mp.model_name;
44 mp.linux_device_version = '18.0';
45
46 %% Setup the directory paths & tool settings
47 % TODO: these paths should ideally be contained in a toolbox. the one exception
   ↪ is the model path, which is many cases is the pwd, though it doesn't have to
   ↪ be.
48 addpath('../../config');
49 if isunix  % setup for a Linux platform
50     path_setup_linux;
51 elseif ispc % setup for a Windows platform
52     path_setup_windows;
53 end
54
55 % TODO: remove python path and version information. All of the code should be
   ↪ python3 and python2 compatible. If not, we should make it python2/3
   ↪ compatible if possible.
56 % mp.python_path = 'F:\Python\Python37\python.exe';
57
58 % Note: addpath() only sets the paths for the current Matlab session
59 addpath(mp.model_path)
60 addpath(mp.driver_codegen_path)
61 addpath(mp.vhdl_codegen_path)
62 addpath(mp.config_path)
63 hdlsetuptoolpath('ToolName', 'Altera Quartus II', 'ToolPath', mp.quartus_path);
   ↪ % setup the HDL toochain path that needs to be set before calling HDL
   ↪ workflow process
64 eval(['cd ' mp.model_path])  % change the working directory to the model directory
65
66 %% python
67 % [python_version, python_exe, python_loaded] = pyversion;
68 % if  python_loaded
69 %     disp(['Using Python version ' python_version])
70 % else
71 %     pyversion(mp.python_path);    % Note: if the version changes from what is
   ↪ already loaded in Matlab, you will need to restart Matlab
72 %     [python_version, python_exe, python_loaded] = pyversion;
73 %     disp(['Setting Python to version ' python_version])
74 % end
75 % add the codegen_path to python's search path
76 if count(py.sys.path,mp.vhdl_codegen_path) == 0
77     insert(py.sys.path,int32(0),mp.vhdl_codegen_path);
78 end
79 if count(py.sys.path,mp.driver_codegen_path) == 0
```

```
80      insert(py.sys.path,int32(0),mp.driver_codegen_path);
81  end
82
83
84  %% Open the model
85  disp(['Please wait while the Simulink Model: '  mp.model_name  ' is being
    ↪  opened.'])
86  disp(['Note: Before generating VHDL, you will need to run a model simulation.'])
87  open_system([mp.model_abbreviation])
88  % display popup reminder message
89  helpdlg(sprintf(['NOTE: You will need to first run the Simulation for model '
    ↪  mp.model_name ' to initialize variables in the Matlab workspace before
    ↪  converting to VHDL.']),'Reminder Message')
90  mp.sim_prompts = 1;  % turn on the simulation prompts/comments (these will be
    ↪  turned off when the HDL conversion process starts).
```

sm_callback_init.m

```
1   % sm_callback_init
2   %
3   % This scripts initializes the model variables and parameters. The script
4   % runs before the simulation starts.  This is called in the InitFcn callback
5   % found in Model Explorer.
6   %
7   % Copyright 2019 Flat Earth Inc
8   %
9   % THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    ↪  IMPLIED,
10  % INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
    ↪  PARTICULAR
11  % PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
    ↪  BE LIABLE
12  % FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
    ↪  TORT OR OTHERWISE,
13  % ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
    ↪  DEALINGS IN THE SOFTWARE.
14  %
15  % Ross K. Snider
16  % Flat Earth Inc
17  % 985 Technology Blvd
18  % Bozeman, MT 59718
19  % support@flatearthinc.com
20
21  %% Make sure that sm_run_me_first has actually been run first.
22  % if not, run it first since it sets up paths and toolchains
23  if exist('mp') == 0 || isfield(mp,'sim_prompts') == 0
24      sm_run_me_first;
25  end
26
27  %% Set Audio Data Sample Rate
```

```
mp.Fs = 48000;    % sample rate frequency of AD1939 codec in Hz
mp.Ts = 1/mp.Fs;  % sample period

%% Set the FPGA system clock frequency (frequency of the FPGA fabric)
% The system clock frequency should be an integer multiple of the Audio codec AD1939 Mclk frequency (12.288 MHz)
if mp.fastsim_flag == 0
    mp.Fs_system = 98304000;        % System clock frequency in Hz of Avalon Interface  Mclk*8 = 12.288MHz*8=98304000
else
    mp.Fs_system = mp.Fs * mp.fastsim_Fs_system_N;          % Note: For faster development runs (faster sim times), reduce the number of system clocks between samples.  mp.fastsim_Fs_system_N is set in sm_run_me_first.m
end
mp.Ts_system = 1/mp.Fs_system;         % System clock period
mp.rate_change = mp.Fs_system/mp.Fs;   % how much faster the system clock is to the sample clock

%% Set the data type for audio signal (left and right channels) in data plane
mp.W_bits = 32;  % Word length
mp.F_bits = 28;  % Number of fractional bits in word

%% Create the control signals
mp = sm_init_control_signals(mp);  % create the control signals

%% Create test signals for the left and right channels
mp = sm_init_test_signals(mp);  % create the test signals that will go through the model
stop_time = mp.test_signal.duration;  % simulation time is based on the number of audio samples to go through the model
if mp.sim_prompts == 1  % Note: sim_prompts is set in Run_me_first.m and is set to zero when hdl code generation is run
    Nsamples_avalon = mp.test_signal.Nsamples * mp.rate_change;
    disp(['Simulation time has been set to ' num2str(stop_time) ' seconds'])
    disp(['    Processing ' num2str(Nsamples_avalon) ' Avalon streaming samples.'])
    disp(['          To reduce simulation time for development iterations,'])
    disp(['          reduce the system clock variable Fs_system (current set to ' num2str(mp.Fs_system)  ')'])
    disp(['          and/or reduce the test signal length (current set to ' num2str(mp.test_signal.duration)  ' sec = ' num2str(mp.test_signal.Nsamples)  ' samples)'])
end

%% Put the test signals into the Avalon Streaming Bus format
% i.e. put the test signals into the data-channel-valid protocol
mp = sm_init_avalon_signals(mp);  % create the avalon streaming signals

%% place variables into workspace directly (debug)
% Avalon_Source_Data    = SG.Avalon_Source_Data;    % place into workspace directly so that the "From Workspace" blocks can read from these variables
```

```
% Avalon_Source_Valid   = SG.Avalon_Source_Valid;
% Avalon_Source_Channel = SG.Avalon_Source_Channel;
% Avalon_Source_Error   = SG.Avalon_Source_Error;
% Ts_system = SG.Ts_system;
% W_bits = SG.W_bits;
% F_bits = SG.F_bits;
```

sm_callback_stop.m

```
% sm_callback_stop
%
% This scripts captures the output signals and then verifies that
% these signals are correct. The script runs after the simulation stops.
% This is called in the StopFcn callback found in Model Explorer.
%
% Copyright 2019 Flat Earth Inc
%
% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
% INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
% PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
% FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
% ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
%
% Ross K. Snider
% Flat Earth Inc
% 985 Technology Blvd
% Bozeman, MT 59718
% support@flatearthinc.com

%% Put the output into the SG data struct
% The "To Workspace" block won't accept structs
mp.Avalon_Sink_Data.Time    = Avalon_Sink_Data.Time;    % time
mp.Avalon_Sink_Data.Data    = Avalon_Sink_Data.Data;    % data
mp.Avalon_Sink_Channel.Data = Avalon_Sink_Channel.Data; % channel
mp.Avalon_Sink_Valid.Data   = Avalon_Sink_Valid.Data;   % valid


if mp.sim_prompts == 1  % sim_prompts is set in Run_me_first.m   This is turned off when the model is converted to VHDL since we don't want to run the verification multiple times at this point (HDL coder runs the simulation multiple times)

    mp = sm_stop_process_output(mp);  % get the output and convert from Avalon to vector
    mp = sm_stop_verify(mp);          % verify that the output is correct
```

```
33
34  end
```

sm_check_control_signals.m

```
1   function sm_check_control_signals(mp)
2
3
4
5   %% Check if all the register fields have been included by the user
6   if mp.sim_prompts == 1  % don't keep displaying when vhdl code is being generated
    ↪  by HDL coder
7       disp(' ')
8       disp('Checking if the control register fields have been setup correctly in
        ↪  sm_init_control_signals.m')
9   end
10
11  registers = mp.register;
12  Nsr = length(registers);
13
14  rf = readjson('register_control_fields.json');
15  Nrf = length(rf);
16
17  %% check for field names
18  for i=1:Nsr
19      supplied_fields = fieldnames(registers(i));
20      Nsf = length(supplied_fields);
21      found_flags = zeros(1,Nrf);
22      for j=1:Nrf
23          for k=1:Nsf
24              if strcmp(rf(j).name,supplied_fields(k))
25                  found_flags(j) = 1;
26                  break;
27              end
28          end
29      end
30      for j=1:Nrf
31          if found_flags(j) == 0
32              disp(['Register control is missing field ' rf(j).name ' for register
                ↪  ' registers(i).name])
33          end
34      end
35  end
36
37  %% check for empty fields
38  error_flag = zeros(1,Nsr);
39  for i=1:Nsr
40      supplied_fields = fieldnames(registers(i));
41      Nsf = length(supplied_fields);
42      for j=1:Nsf
```

```
        a = registers(i).(supplied_fields{j});
        if isempty(a)
            if error_flag(i) == 0
                disp(['The following fields are missing for control register '
                ↪  num2str(i) ': ' mp.register(i).name])
            end
            error_flag(i) = 1;
            disp(['    ' supplied_fields{j}])
        end
    end
end
if error_flag == 1
    error('Register fields are missing.  Please add them in
    ↪  sm_init_control_signals.m')
end


%% Check if the widget name is correct
wl = readjson('widget_list.json');
Nw = length(wl);

error_flag = 0;
for i=1:Nsr
    found_flag = 0;
    for j=1:Nw
        if strcmp(registers(i).widget_type, wl(j).name) == 1
            found_flag = 1;
            break
        end
    end
    if found_flag == 0
        error_flag = 1;
        disp(['There is no widget called  ***' registers(i).widget_type '***
        ↪  associated with register ' num2str(i) ': '  mp.register(i).name])
        disp(['Please specify a known widget.  Avaliable widget names are:'])
        for j=1:Nw
            disp(['     '  wl(j).name])
        end
    end
end
if error_flag == 1
    error('Widget names are wrong.  Please correct in sm_init_control_signals.m')
end
```

sm_init_avalon_signals.m

```
% mp = sm_init_avalon_signals(mp)
%
% Matlab function that puts the test signals into the Avalon Streaming
% interface format that uses the data-channel-valid protocol.
```

```
%
% Inputs:
%   mp, which is the model data structure that holds the model parameters
%
% Outputs:
%   mp, the model data structure that now contains the Avalon signals:
%          mp.Avalon_Source_Data    - The Avalon streaming data bus
%          mp.Avalon_Source_Valid   - The Avalon streaming valid signal
%          mp.Avalon_Source_Channel - The Avalon streaming channel bus
%          mp.Avalon_Source_Error   - The Avalon streaming error bus
%
% Copyright 2019 Flat Earth Inc
%
% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
% INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
% PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
% FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
% ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
%
% Ross K. Snider
% Flat Earth Inc
% 985 Technology Blvd
% Bozeman, MT 59718
% support@flatearthinc.com


function mp = sm_init_avalon_signals(mp)

%% create the data-channel-valid signals from the test signals
mp.Nsamples_avalon = mp.test_signal.Nsamples * mp.rate_change;
datavals_data    = zeros(1,mp.Nsamples_avalon);   % preallocate arrays
datavals_valid   = zeros(1,mp.Nsamples_avalon);
datavals_channel = zeros(1,mp.Nsamples_avalon);
datavals_error   = zeros(1,mp.Nsamples_avalon);
dataval_index = 1;
for sample_index = 1:mp.test_signal.Nsamples
    %-----------------------------
    % left channel
    %-----------------------------
    datavals_data(dataval_index)     = mp.test_signal.left(sample_index);
    datavals_valid(dataval_index)    = 1;   % data is valid in this time bin
    datavals_channel(dataval_index)  = 0;   % channel 0 = left
    datavals_error(dataval_index)    = 0;   % no error
    dataval_index                    = dataval_index + 1;
    %-----------------------------
    % right channel
```

```
51      %-----------------------------
52      datavals_data(dataval_index)     = mp.test_signal.right(sample_index);
53      datavals_valid(dataval_index)    = 1;  % data is valid in this time bin
54      datavals_channel(dataval_index)  = 1;  % channel 1 = right
55      datavals_error(dataval_index)    = 0;  % no error
56      dataval_index                    = dataval_index + 1;
57      %-------------------------------------------
58      % fill in the invalid data slots with zeros
59      %-------------------------------------------
60      for k=1:(mp.rate_change-2)
61          datavals_data(dataval_index)    = 0;  % no data (put in zeros)
62          datavals_valid(dataval_index)   = 0;  % data is not valid in these time
            ↪  bins
63          datavals_channel(dataval_index) = 3;  % channel 3 = no data
64          datavals_error(dataval_index)   = 0;  % no error
65          dataval_index                   = dataval_index + 1;
66      end
67  end
68
69  %% Convert to time series objects that can be read from "From Workspace" blocks
70  Ndatavals = length(datavals_data);  % get number of data points
71  timevals  =  [0 1:(Ndatavals-1)]*mp.Ts_system;  % get associated times assuming
    ↪  system clock
72  mp.Avalon_Source_Data     = timeseries(datavals_data,timevals);
73  mp.Avalon_Source_Valid    = timeseries(datavals_valid,timevals);
74  mp.Avalon_Source_Channel  = timeseries(datavals_channel,timevals);
75  mp.Avalon_Source_Error    = timeseries(datavals_error,timevals);
```

sm_init_control_signals.m

```
1   function mp = sm_init_control_signals(mp)
2   % mp = sm_init_control_signals(mp)
3   %
4   % Matlab function that creates and initializes the control signals.
5   % The min/max values the control signals are expected to take need to be
6   % defined.
7   %
8   % Inputs:
9   %   mp, which is the model data structure that holds the model parameters
10  %
11  % Outputs:
12  %   mp, the model data structure that now contains the registers in the
13  %   field mp.register(i), indexed by i, which has the following fields:
14  %         mp.register(i).name       -  The register name
15  %         mp.register(i).value      -  The value the control register is
    ↪  currently set to
16  %         mp.register(i).min        -  The minimum value the control signal will
    ↪  ever take
17  %         mp.register(i).max        -  The maximum value the control signal will
    ↪  ever take
```

```
18 %         mp.register(i).timeseries - The timeseries data format that is needed
   ↪  for the from workspace block
19 %
20 % Copyright 2019 Flat Earth Inc
21 %
22 % THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   ↪  IMPLIED,
23 % INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
   ↪  PARTICULAR
24 % PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
   ↪  BE LIABLE
25 % FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
   ↪  TORT OR OTHERWISE,
26 % ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
   ↪  DEALINGS IN THE SOFTWARE.
27 %
28 % Ross K. Snider
29 % Flat Earth Inc
30 % 985 Technology Blvd
31 % Bozeman, MT 59718
32 % support@flatearthinc.com
33 %
34 % Note: to display the widget types available, run the function
35 % \simulink_models\config\widget_list_display.m
36 %
37
38
39 %% Create bypass control signal
40 ri = 1;   % register index
41 mp.register(ri).name       = 'Bypass';  % control signal name
42 mp.register(ri).value      = 0;         % value control signal will take during
   ↪  simulation
43 mp.register(ri).min        = 0;         % The minimum value the control signal
   ↪  will ever take
44 mp.register(ri).max        = 1;         % The maximum value the control signal
   ↪  will ever take
45 mp.register(ri).default    = 0;         % default (initial) value
46 mp.register(ri).widget_type          = 'toggle';
47 mp.register(ri).widget_display_units = 'bypass';
48
49 %% Create bit control signal
50 ri = ri + 1;  % register index
51 mp.register(ri).name       = 'Bits';   % control signal name
52 mp.register(ri).value      = 4;        % value control signal will take during
   ↪  simulation
53 mp.register(ri).min        =  0;       % The minimum value the control signal
   ↪  will ever take
54 mp.register(ri).max        = 32;       % The maximum value the control signal
   ↪  will ever take
55 mp.register(ri).default    = 32;       % default (initial) value
56 mp.register(ri).widget_type          = 'slider';
```

```
mp.register(ri).widget_display_units = 'bits';

%% Create wet_dry_mix signal
ri = ri + 1;  % register index
mp.register(ri).name       = 'Wet_Dry_Mix';   % control signal name   Note:
↪  wet_gain=wet_dry_mix; dry_gain=1-wet_dry_mix
mp.register(ri).value      = 1;             % value control signal will take
↪  during simulation
mp.register(ri).min        = 0;               % The minimum value the control
↪  signal will ever take
mp.register(ri).max        = 1;               % The maximum value the control
↪  signal will ever take
mp.register(ri).default    = 0.5;             % default (initial) value
mp.register(ri).widget_type          = 'slider';
mp.register(ri).widget_display_units = 'ratio';

% Any other register control signals should be created in a similar manner


%% convert to time series data
for i=1:length(mp.register)
    mp.register(i).timeseries = timeseries(mp.register(i).value,0);  %
    ↪  timeseries(datavals,timevals);  % A timeseries data format is needed for
    ↪  the from workspace block
end

%% Check if the control signals have valid entries
sm_check_control_signals(mp)
```

sm_init_test_signals.m

```
% mp = sm_init_test_signals(mp)
%
% This function creates and initializes the test signals that are used
% in the model simulation.
%
% Inputs:
%   mp, which is the model data structure that holds the model parameters
%
% Outputs:
%   mp, the model data structure that now contains the test signals in the
%   field mp.test_signal, which has the following fields:
%         mp.test_signal.duration  -  length of test signals in seconds
%         mp.test_signal.Nsamples  -  number of samples in test signals
%         mp.test_signal.left      -  signal for left channel
%         mp.test_signal.right     -  signal for right channel
%
% Copyright 2019 Flat Earth Inc
%
```

```
% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
% INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
% PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
% FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
% ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
%
% Ross K. Snider
% Flat Earth Inc
% 985 Technology Blvd
% Bozeman, MT 59718
% support@flatearthinc.com

function mp = sm_init_test_signals(mp)

signal_option = 3;  % set which test signal to use

switch signal_option
    case 1 % Simple tones
        mp.test_signal.duration = 1;  % duration of tone in seconds
        Nsamples = round(mp.test_signal.duration*mp.Fs);
        if mp.fastsim_flag == 1 % perform fast simulation by reducing the number of samples
           mp.test_signal.Nsamples = min(Nsamples, mp.fastsim_Nsamples);
        else
           mp.test_signal.Nsamples = mp.fastsim_Nsamples;
        end
        sample_times = [0 1:(mp.test_signal.Nsamples-1)]*mp.Ts;
        mp.test_signal.left  = cos(2*pi*2000*sample_times);
        mp.test_signal.right = cos(2*pi*3000*sample_times);
    case 2  % speech
        [y,Fs] = audioread('SpeechDFT-16-8-mono-5secs.wav');  % speech sample found in the Matlab Audio Toolbox
        y_resampled = resample(y,mp.Fs,Fs);  % resample to change the sample rate to SG.Fs
        Nsamples = length(y_resampled);
        if mp.fastsim_flag == 1 % perform fast simulation by reducing the number of samples
           mp.test_signal.Nsamples = min(Nsamples, mp.fastsim_Nsamples);
        else
           mp.test_signal.Nsamples = mp.fastsim_Nsamples;
        end
        mp.test_signal.left  = y_resampled(1:mp.test_signal.Nsamples);
        mp.test_signal.right = y_resampled(1:mp.test_signal.Nsamples);
        mp.test_signal.Nsamples = length(mp.test_signal.left);
        mp.test_signal.duration = mp.test_signal.Nsamples * mp.Ts;
    case 3 % user supplied music
```

```
61          [y,Fs] = audioread([mp.test_signals_path filesep
            ↪  'Urban_Light_HedaMusic_Creative_Commons.mp3']);
62          y_resampled = resample(y,mp.Fs,Fs);  % resample to change the sample rate
            ↪  to SG.Fs
63          Nsamples = length(y_resampled);
64          if mp.fastsim_flag == 1 % perform fast simulation by reducing the number
            ↪  of samples
65             mp.test_signal.Nsamples = min(Nsamples, mp.fastsim_Nsamples);
66          else
67             mp.test_signal.Nsamples = mp.fastsim_Nsamples;
68          end
69          mp.test_signal.left  = y_resampled(1:mp.test_signal.Nsamples);
70          mp.test_signal.right = y_resampled(1:mp.test_signal.Nsamples);
71          mp.test_signal.Nsamples = length(mp.test_signal.left);
72          mp.test_signal.duration = mp.test_signal.Nsamples * mp.Ts;
73      otherwise
74          error('Please choose a viable option for the test signal (see
            ↪  sm_init_test_signals)')
75  end
```

sm_stop_process_output.m

```
1   % mp = sm_stop_process_output(mp)
2   %
3   % Matlab function that gets the Avalon Streaming output and converts it
4   % to a vector format that is useful for verification.
5   %
6   % Inputs:
7   %   mp, which is the model data structure that holds the model parameters
8   %
9   % Outputs:
10  %   mp, the model data structure that now contains the left/right channel
11  %   data, which is in the following format:
12  %          mp.left_data_out         - The processed left channel data
13  %          mp.left_time_out         - time of left channel data
14  %          mp.right_data_out        - The processed right channel data
15  %          mp.right_time_out        - time of right channel data
16  %
17  % Copyright 2019 Flat Earth Inc
18  %
19  % THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    ↪  IMPLIED,
20  % INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
    ↪  PARTICULAR
21  % PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
    ↪  BE LIABLE
22  % FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
    ↪  TORT OR OTHERWISE,
23  % ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
    ↪  DEALINGS IN THE SOFTWARE.
```

```
%
% Ross K. Snider
% Flat Earth Inc
% 985 Technology Blvd
% Bozeman, MT 59718
% support@flatearthinc.com

function mp = sm_stop_process_output(mp)

%% Get the Avalon streaming signals from the model
t = mp.Avalon_Sink_Data.Time;             % time
d = squeeze(mp.Avalon_Sink_Data.Data);    % data      Note: the Matlab squeeze()
↪  function removes singleton dimensions (i.e. dimensions of length 1)
c = squeeze(mp.Avalon_Sink_Channel.Data); % channel
v = squeeze(mp.Avalon_Sink_Valid.Data);   % valid
left_index = 1;
right_index = 1;
for i=1:length(v)
    if v(i) == 1  % check if valid, valid is asserted when there is data
        if c(i) == 0  % if the channel number is zero, it is left channel data
            mp.left_data_out(left_index) = double(d(i));
            mp.left_time_out(left_index) = t(i);
            left_index            = left_index + 1;
        end
        if c(i) == 1  % if the channel number is one, it is right channel data
            mp.right_data_out(right_index) = double(d(i));
            mp.right_time_out(right_index) = t(i);
            right_index             = right_index + 1;
        end
    end
end
```

sm_stop_verify.m

```
% mp = sm_stop_verify(mp)
%
% Matlab function that verifies the model output

% Inputs:
%   mp, which is the model data structure that holds the model parameters
%
% Outputs:
%   mp, the model data structure that now contains the left/right channel
%   data, which is in the following format:
%         mp.left_data_out         - The processed left channel data
%         mp.left_time_out         - time of left channel data
%         mp.right_data_out        - The processed right channel data
%         mp.right_time_out        - time of right channel data
%
% Copyright 2019 Flat Earth Inc
```

```
17 %
18 % THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   ↪  IMPLIED,
19 % INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
   ↪  PARTICULAR
20 % PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
   ↪  BE LIABLE
21 % FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
   ↪  TORT OR OTHERWISE,
22 % ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
   ↪  DEALINGS IN THE SOFTWARE.
23 %
24 % Ross K. Snider
25 % Flat Earth Inc
26 % 985 Technology Blvd
27 % Bozeman, MT 59718
28 % support@flatearthinc.com
29
30 function mp = sm_stop_verify(mp)
31
32 %% Verify that the test data got encoded, passed through the model, and
33 % decoded correctly.  The input (modified by gain) and output values should be
   ↪  identical.
34
35 % figure(1)
36 % subplot(2,1,1)
37 % plot(mp.test_signal.left); hold on
38 % plot(mp.left_data_out)
39 % title(['Bit Control = ' num2str(mp.register(2).value) ' Bypass = '
   ↪  num2str(mp.register(1).value) ' Wet/Dry Mix = '
   ↪  num2str(mp.register(3).value)])
40
41 % subplot(2,1,2)
42 % plot(mp.test_signal.right); hold on
43 % plot(mp.right_data_out)
44 % title(['Bit Control = ' num2str(mp.register(2).value) ' Bypass = '
   ↪  num2str(mp.register(1).value) ' Wet/Dry Mix = '
   ↪  num2str(mp.register(3).value)])
45
46 % original_audio = [mp.test_signal.left(:) mp.test_signal.right(:)];
47 processed_audio = [mp.left_data_out(:) mp.right_data_out(:)];
48 % soundsc(original_audio, mp.Fs);
49 % pause(mp.test_signal.duration*1.1);
50 soundsc(processed_audio, mp.Fs);
```

APPENDIX B

IP CORE AUTOGENERATION SCRIPTS

VHDL Autogeneration

hdlworkflow.m

```
%% Load the Model
model = mp.model_abbreviation;
dataplane_name = 'dataplane';
load_system(model);

%% Model HDL Parameters
hdlset_param(model, ...
    'DateComment', 'off', ...
    'ModulePrefix', [model '_'], ...
    'CriticalPathEstimation', 'off', ...
    'GenerateHDLTestBench', 'off', ...
    'HDLCodingStandardCustomizations',hdlcodingstd.IndustryCustomizations(), ...
    'HDLGenerateWebview', 'on', ...
    'HDLSubsystem', [model '/' dataplane_name], ...
    'OptimizationReport', 'on', ...
    'ResourceReport', 'on', ...
    'SynthesisTool', 'Altera QUARTUS II', ...
    'SynthesisToolPackageName', '', ...
    'SynthesisToolSpeedValue', '', ...
    'MapPipelineDelaysToRAM', 'on', ...
    'UseRisingEdge', 'on', ...
    'TargetDirectory', 'hdlsrc', ...
    'TargetFrequency', mp.Fs_system);

% TODO: should we always map pipeline delays to ram?
% TODO: add ability to customize code generation parameters on a per-model basis

%% Workflow Configuration Settings
% Construct the Workflow Configuration Object with default settings
hWC = hdlcoder.WorkflowConfig('SynthesisTool','Altera QUARTUS
↪  II','TargetWorkflow','Generic ASIC/FPGA');

% Specify the top level project directory
hWC.ProjectFolder = mp.model_path;

% Set Workflow tasks to run
hWC.RunTaskGenerateRTLCodeAndTestbench = true;
hWC.RunTaskVerifyWithHDLCosimulation = false;
hWC.RunTaskCreateProject = false;
hWC.RunTaskPerformLogicSynthesis = false;
hWC.RunTaskPerformMapping = false;
hWC.RunTaskPerformPlaceAndRoute = false;
hWC.RunTaskAnnotateModelWithSynthesisResult = false;

% Set properties related to 'RunTaskGenerateRTLCodeAndTestbench' Task
hWC.GenerateRTLCode = true;
hWC.GenerateTestbench = false;
hWC.GenerateValidationModel = false;
```

```
48
49 % Set properties related to 'RunTaskCreateProject' Task
50 hWC.Objective = hdlcoder.Objective.None;
51 hWC.AdditionalProjectCreationTclFiles = '';
52
53 % Set properties related to 'RunTaskPerformMapping' Task
54 hWC.SkipPreRouteTimingAnalysis = false;
55
56 % Set properties related to 'RunTaskPerformPlaceAndRoute' Task
57 hWC.IgnorePlaceAndRouteErrors = false;
58
59 % Set properties related to 'RunTaskAnnotateModelWithSynthesisResult' Task
60 hWC.CriticalPathSource = 'pre-route';
61 hWC.CriticalPathNumber =  1;
62 hWC.ShowAllPaths = false;
63 hWC.ShowDelayData = false;
64 hWC.ShowUniquePaths = false;
65 hWC.ShowEndsOnly = false;
66
67 % Validate the Workflow Configuration Object
68 hWC.validate;
69
70 %% Run the workflow
71 hdlcoder.runWorkflow([model '/' dataplane_name], hWC);
```

vgen_get_simulink_block_interfaces.m

```
1  % avalon = vgen_get_simulink_block_interfaces(model_params)
2  %
3  % This function parses the simulink model and extracts the interface signals
4  % Note: The model simulation needs to be run first before this function
5  % is called since there are workspace variables that need to be set/created
6  % in the initiallization callback function first
7
8  % Inputs:
9  %   model_params, which is the model data structure that holds the model
   ↪  parameters
10 %
11 % Outputs:
12 %   The data structure avalon that contains the interface signals for the
13 %   dataplane block
14 %
15 % Copyright 2019 Flat Earth Inc
16 %
17 % THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   ↪  IMPLIED,
18 % INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
   ↪  PARTICULAR
19 % PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
   ↪  BE LIABLE
```

```
% FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
% ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
%
% Ross K. Snider
% Flat Earth Inc
% 985 Technology Blvd
% Bozeman, MT 59718
% support@flatearthinc.com

% XXX: it appears that you can't have multiple models open at the same time, otherwise this functions errors

function avalon1 = vgen_get_simulink_block_interfaces(model_params)
%% Put the model in compile mode  needed to get the CompiledPortDataTypes
% https://www.mathworks.com/matlabcentral/answers/8679-how-to-get-the-port-types-and-dimensions-for-a-block
modelName = bdroot;  % get model name
disp(['Placing the model in compile mode.'])
eval([modelName,'([],[],[],''compile'');']);  % put in compile mode

%% Get the Avalon streaming sink signals
avalon_sink_signals = find_system('SearchDepth','2','regexp','on','BlockType','In port','BlockDialogParams','avalon_sink*');
if (isempty(avalon_sink_signals)==0) % The avalon streaming sink exists
    avalon1.avalon_sink_flag = 1;
    Na = length(avalon_sink_signals);
    index = 1;
    for i=1:Na
        signal_name = avalon_sink_signals{i};
        ind = strfind(signal_name,'gm_');  % if the signal name contains gm_ we will ignore it (generated model stuff)
        if sum(size(ind)) == 0 % signal name doesn't contain 'gm_' so process it
            h = getSimulinkBlockHandle(signal_name);  % get block handle
            if index==1 % get parent name, which will be the entity name
                parent = get_param(h,'Parent');  % get parent name with path
                split = strsplit(parent,'/');    % split string into parts separated by '/'
                avalon1.entity = [model_params.model_abbreviation '_' split{end}];     % get the last string
            end
            p=get_param(h,'CompiledPortDataTypes');
            avalon1.avalon_sink.signal{index}.name        = get(h,'PortName');
            avalon1.avalon_sink.signal{index}.data_type   = parse_data_type(p.Outport{1});
            index = index + 1;
        end
    end
else
    avalon1.avalon_sink_flag = 0;  % No avalon streaming sink interface
```

```
62      avalon1.avalon_sink = [];
63  end
64
65  %% Get the Avalon streaming source signals
66  avalon_source_signals = find_system('SearchDepth','2','regexp','on','BlockType','
    ↪  Outport','BlockDialogParams','avalon_source*');
67  if (isempty(avalon_source_signals)==0) % The avalon streaming source exists
68      avalon1.avalon_source_flag = 1;
69      Na = length(avalon_source_signals);
70      index = 1;
71      for i=1:Na
72          signal_name = avalon_source_signals{i};
73          ind = strfind(signal_name,'gm_');  % if the signal name contains gm_ we
            ↪  will ignore it (generated model stuff)
74          if sum(size(ind)) == 0 % signal name doesn't contain 'gm_' so process it
75              h = getSimulinkBlockHandle(signal_name);  % get block handle
76              p=get_param(h,'CompiledPortDataTypes');
77              avalon1.avalon_source.signal{index}.name      = get(h,'PortName');
78              avalon1.avalon_source.signal{index}.data_type =
                ↪  parse_data_type(p.Inport{1});
79              index = index + 1;
80          end
81      end
82  else
83      avalon1.avalon_source_flag = 0;  % No avalon streaming source interface
84      avalon1.avalon_source = [];
85  end
86
87  %% Get the Avalon memory mapped signals
88  % These are the registers that Linux will interact with
89  register_names = find_system('SearchDepth','2','regexp','on','BlockType','Inport'
    ↪  ,'BlockDialogParams','register_control*');
90  if (isempty(register_names)==0) % The avalon memory mapped interface exists
91      avalon1.avalon_memorymapped_flag = 1;
92      Na = length(register_names);
93      index = 1;
94      for i=1:Na
95          register_name = register_names{i};
96          ind = strfind(register_name,'gm_');  % if the register name contains gm_
            ↪  we will ignore it (generated model stuff)
97          if sum(size(ind)) == 0 % register name doesn't contain 'gm_' so process it
98              h = getSimulinkBlockHandle(register_name);  % get block handle
99              p=get_param(h,'CompiledPortDataTypes');
100             register_name = get(h,'PortName');
101             register_name = register_name(length('register_control_')+1:end);  %
                ↪  remove "register_control_" from name
102             avalon1.avalon_memorymapped.register{index}.name      = register_name;
103             avalon1.avalon_memorymapped.register{index}.data_type =
                ↪  parse_data_type(p.Outport{1});
104             % register numbers start at 0, so we have to subtract 1
105             avalon1.avalon_memorymapped.register{index}.reg_num   = index - 1;
```

```
106             Nregisters = length(model_params.register);
107             for j=1:Nregisters
108                  if strcmpi(register_name,model_params.register(j).name)  % get
                     ↪  the register with the same name
109                     avalon1.avalon_memorymapped.register{index}.default_value =
                        ↪  model_params.register(j).default;
110                     avalon1.avalon_memorymapped.register{index}.min_value     =
                        ↪  model_params.register(j).min;
111                     avalon1.avalon_memorymapped.register{index}.max_value     =
                        ↪  model_params.register(j).max;
112                 end
113             end
114             index = index + 1;
115         end
116     end
117 else
118     avalon1.avalon_memorymapped_flag = 0;  % No avalon streaming sink interface
119     avalon1.avalon_memorymapped = [];
120 end
121
122 %% Get the Exported Input signals
123 % These are the input signals coming in from outside the FPGA
124 conduit_names = find_system('SearchDepth','2','regexp','on','BlockType','Inport',
    ↪  'BlockDialogParams','export*');
125 if (isempty(conduit_names)==0)
126     avalon1.conduit_input_flag = 1;
127     Na = length(conduit_names);
128     index = 1;
129     for i=1:Na
130         conduit_name = conduit_names{i};
131         ind = strfind(conduit_name,'gm_');  % if the conduit name contains gm_ we
            ↪  will ignore it (generated model stuff)
132         if sum(size(ind)) == 0 % conduit name doesn't contain 'gm_' so process it
133             h = getSimulinkBlockHandle(conduit_name);  % get block handle
134             p=get_param(h,'CompiledPortDataTypes');
135             avalon1.conduit_input.signal{index}.name      = get(h,'PortName');
136             avalon1.conduit_input.signal{index}.data_type =
                ↪  parse_data_type(p.Outport{1});
137             index = index + 1;
138         end
139     end
140 else
141     avalon1.conduit_input_flag = 0;
142     avalon1.conduit_input = [];
143 end
144
145 %% Get the Exported Output signals
146 % These are the output signals going outside the FPGA
147 conduit_names = find_system('SearchDepth','2','regexp','on','BlockType','Outport'
    ↪  ,'BlockDialogParams','export*');
148 if (isempty(conduit_names)==0)
```

```
149     avalon1.conduit_output_flag = 1;
150     Na = length(conduit_names);
151     index = 1;
152     for i=1:Na
153         conduit_name = conduit_names{i};
154         ind = strfind(conduit_name,'gm_');  % if the conduit name contains gm_ we
            ↪  will ignore it (generated model stuff)
155         if sum(size(ind)) == 0 % conduit name doesn't contain 'gm_' so process it
156             h = getSimulinkBlockHandle(conduit_name);  % get block handle
157             p=get_param(h,'CompiledPortDataTypes');
158             avalon1.conduit_output.signal{index}.name      = get(h,'PortName');
159             avalon1.conduit_output.signal{index}.data_type =
                ↪  parse_data_type(p.Inport{1});
160             index = index + 1;
161         end
162     end
163 else
164     avalon1.conduit_output_flag = 0;
165     avalon1.conduit_output = [];
166 end
167
168 %% Misc Info
169 % Other compiled port information:
170 % CompiledPortWidths: [1x1 struct]
171 %  CompiledPortDimensions: [1x1 struct]
172 %  CompiledPortDataTypes: [1x1 struct]
173 %  CompiledPortComplexSignals: [1x1 struct]
174 %  CompiledPortFrameData: [1x1 struct]
175 %https://www.mathworks.com/matlabcentral/answers/8679-how-to-get-the-port-types-a
    ↪  nd-dimensions-for-a-block
176 %get(gcbh)  % get block parameters when block is selected
177
178
179 %% Turn off the compile mode
180 % Otherwise you won't be able to modify the model
181 % If you can't modify, you may need to call
182 % the function below multiple times (a terminate call is needed for each compile
    ↪  call)
183 % because the termination has been deferred.  This can happen if an error
184 % occurs in the function.
185 eval([modelName,'([],[],[],''term'');']);  % terminate the compile mode
186
187 end
188
189 function typeinfo = parse_data_type(datatype)
190 % parse_data_type Create a structure containing relevant type info from a
    ↪  datatype string
191 %
192 % examples of type strings: int16, uint8, ufix13_en5, sfix32_en28, boolean
193 typeinfo = struct();
194
```

```
195 typeinfo.type = datatype;
196
197 % boolean
198 if strcmp(typeinfo.type, 'boolean')
199     typeinfo.width = 1;
200     typeinfo.fractional_bits = 0;
201     typeinfo.signed = false;
202
203 % integer
204 elseif ~isempty(strfind(typeinfo.type, 'int'))
205     % extract sign
206     if typeinfo.type(1) == 'u'
207         typeinfo.signed = false;
208     else
209         typeinfo.signed = true;
210     end
211
212     % extract width
213     width = str2double(regexp(typeinfo.type, '\d+', 'match'))
214     if length(width) == 1
215         typeinfo.width = width;
216         typeinfo.fractional_bits = 0;
217     else
218         error(['Malformed or unexpected fixed-point datatype string ', datatype])
219     end
220
221 % fixed-point, but could be an integer (0 fractional bits)
222 elseif ~isempty(strfind(typeinfo.type, 'fix'))
223
224     % extract sign; regexp returns a cell
225     signstr = typeinfo.type(1);
226     if signstr == 's'
227         typeinfo.signed = true;
228     elseif signstr == 'u'
229         typeinfo.signed = false;
230     else
231         error(['Could not extract sign from ', datatype])
232     end
233
234     % extact integer and fractional widths
235     widths = str2double(regexp(typeinfo.type, '\d+', 'match'));
236     if length(widths) == 1
237         % no fractional bits, so it's an integer
238         typeinfo.width = widths(1);
239         typeinfo.fractional_bits = 0;
240     elseif length(widths) == 2
241         typeinfo.width = widths(1);
242         typeinfo.fractional_bits = widths(2);
243     else
244         error(['Malformed or unexpected fixed-point datatype string ', datatype])
245     end
```

```
end
end
```

vgen_process_simulink_model.m

```
% vgen_process_simulink_model
%
% This script parses the simulink model and extracts the interface signals
% and puts this information in a JSON file.

% Copyright 2019 Flat Earth Inc
%
% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
% INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
% PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
% FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
% ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
%
% Ross K. Snider, Trevor Vannoy
% Flat Earth Inc
% 985 Technology Blvd
% Bozeman, MT 59718
% support@flatearthinc.com


%% Parse the Simulink Model (currently opened model)
% We parse the model to get the Avalon signals and control registers we need for the Avalon vhdl wrapper
disp(['vgen: Parsing Simulink model: ' mp.model_name '. Please wait until you see the message "vgen: Finished."'])
try
    % turn off fast sim so that the model runs at the system clock rate
    mp.fastsim_flag = 0;
    % turn off the simulation prompts and the stop callbacks when running HDL workflow (otherwise this runs at each HDL workflow step)
    mp.sim_prompts = 0;
    avalon = vgen_get_simulink_block_interfaces(mp);
catch ME
    % Terminate the compile mode if an error occurs while the model
    % has been placed in compile mode. Otherwise the model will be frozen
    % and you can't quit Matlab
    cmd = [bdroot,'([],[],[],''term'');'];
    eval(cmd)

```

```
37      disp('**************************************************************************
        ↪  ****');
38      disp('Error occurred in function vgen_get_simulink_block_interfaces(mp)');
39      disp(['line number: ' num2str(ME.stack(1).line)])
40      disp(ME.message)
41      disp('**************************************************************************
        ↪  ****');
42
43      % reset fast simulation flag so running the model simulation isn't so slow
        ↪  after generating code.
44      mp.fastsim_flag = 1;
45
46      % turn simulation prompts and callbacks back on for normal simulation.
47      mp.sim_prompts = 1;
48  end
49
50  %% save the specified clock frequencies
51  avalon.clocks.sample_frequency_Hz   = mp.Fs;
52  avalon.clocks.sample_period_seconds = mp.Ts;
53  avalon.clocks.system_frequency_Hz   = mp.Fs_system;
54  avalon.clocks.system_period_seconds = mp.Ts_system;
55
56  %% save the device info
57  avalon.model_name           = mp.model_name;
58  avalon.model_abbreviation   = mp.model_abbreviation;
59  avalon.linux_device_name    = mp.linux_device_name;
60  avalon.linux_device_version = mp.linux_device_version;
61
62  %% Save the avalon structure to a json file and a .mat file
63  writejson(avalon, [avalon.entity,'.json'])
64  save([avalon.entity '_avalon'], 'avalon')
65
66  %% Generate the Simulink model VHDL code
67
68  % run the hdl coder
69  hdlworkflow
70
71  % this is where hdlworkflow puts the vhdl files
72  hdlpath = [mp.model_path filesep 'hdlsrc' filesep mp.model_abbreviation];
73
74  %% Generate the Avalon VHDL wrapper for the VHDL code generated by the HDL Coder
75  disp('vgen: Creating Avalon VDHL wrapper.')
76  infile = [avalon.entity '.json'];
77  outfile = [hdlpath filesep avalon.entity '_avalon.vhd'];
78  vgenAvalonWrapper(infile, outfile, false, false);
79  disp(['      created vhdl file: ' outfile])
80
81  %% Generate the .tcl script to be used by Platform Designer in Quartus
82  disp('vgen: Creating .tcl script for Platform Designer.')
83  infile = [avalon.entity '.json'];
84  % NOTE: platform designer only adds components if they have the _hw.tcl suffix
```

```
85  outfile = [hdlpath filesep avalon.entity '_avalon_hw.tcl'];
86  vgenTcl(infile, outfile, hdlpath);
87  disp(['      created tcl file: ' outfile])
88
89  %% Generate the device driver code
90  disp('Creating device driver.')
91  outfile = [hdlpath filesep mp.model_name '.c'];
92  genDeviceDriver(infile, outfile)
93  disp(['      created device driver: ' outfile])
94
95  %% Generate kernel module build files
96  disp('Creating Makefile and Kbuild.')
97  genMakefile([hdlpath filesep], mp.model_name)
98  disp(['      created Makefile: ' [hdlpath filesep 'Makefile']])
99  disp(['      created Kbuild: ' [hdlpath filesep 'Kbuild']])
100
101 %% Build kernel module
102 % TODO: this needs to be platform independent, but how? Our Windows users
103 %       use a virtual machine to compile the device driver, but that
104 %       won't automate very well. Maybe we can build the kernel module
105 %       with Quartus' embedded command shell instead?
106 disp('Building kernel module.')
107 cd(hdlpath)
108 !make clean
109 !make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
110
111 % TODO: this file now generates C code, but "vgen" make it seem like it is just
    ↪  VHDL still. This should be changed, and the repository should be reorganized
    ↪  a bit.
112 %       This file shouldn't live in the vhdl folder anymore.
113
114
115 disp('vgen: Finished.')
116
117 % reset fast simulation flag so running the model simulation isn't so slow after
    ↪  generating code.
118 mp.fastsim_flag = 1;
119
120 % turn simulation prompts and callbacks back on for normal simulation.
121 mp.sim_prompts = 1;
```

vgenAvalonWrapper.m

```
1  % vgenAvalonWrapper Generate an Avalon vhdl wrapper from a json input file
2  %
3  % vgenAvalonWrapper(infile, outfile, verbose, print_output)
4  %
5  % The json file this function expects has a specific format that contains
   ↪  information about the vhdl entity and
```

```
% the Avalon streaming interface parameters. This will often be generated by MATLAB/Simulink, but it can also be written
% by hand. An example of the format is shown below. This function calls a python script of the same name that generates
% the vhdl.
% TODO: add example json file
%
% Inputs:
%   infile = the json input filename
%   outfile = the vhdl output filename
%   verbose = verbose output
%   print_output = print the output vhdl in the console

% Copyright 2019 Flat Earth Inc, Montana State University
%
% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
% INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
% PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
% FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
% ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
%
% Trevor Vannoy
% Flat Earth Inc
% 985 Technology Blvd
% Bozeman, MT 59718
% support@flatearthinc.com

function vgenAvalonWrapper(infile, outfile, verbose, print_output)

% TODO: add default values to input arguments

% call the python file that autogens the vhdl code
py.vgenAvalonWrapper.main(infile, outfile, verbose, print_output)
```

vgenAvalonWrapper.py

```
#!/usr/bin/python

# @file vgenAvalonWrapper.py
#
#     Python function to auto generate vhdl code given json generated from Simulink/Matlab
#
#     @author Trevor Vannoy, Aaron Koenigsberg
#     @date 2019
```

```
9   #     @copyright 2019 Flat Earth Inc
10  #
11  #     THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    ↪  IMPLIED,
12  #     INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR
    ↪  A PARTICULAR
13  #     PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
    ↪  HOLDERS BE LIABLE
14  #     FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
    ↪  CONTRACT, TORT OR OTHERWISE,
15  #     ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
    ↪  DEALINGS IN THE SOFTWARE.
16  #
17  #     Trevor Vannoy
18  #     Flat Earth Inc
19  #     985 Technology Blvd
20  #     Bozeman, MT 59718
21  #     support@flatearthinc.com
22
23  import json
24  import argparse
25  import re
26  from textwrap import dedent
27  from math import ceil, log, fabs
28
29  # TODO: error checking
30
31  indent = '  '
32
33  def create_library():
34      LIBRARIES=dedent("""\
35          library ieee;
36          use ieee.std_logic_1164.all;
37          use ieee.numeric_std.all;\n
38          """)
39      return LIBRARIES
40
41  def create_entity(name, sink_enabled, sink, source_enabled, source,
42      registers_enabled, registers, conduit_out_enabled, conduit_out,
43      conduit_in_enabled, conduit_in):
44
45      global indent
46
47      ENTITY_BEGIN=dedent("""\
48          entity {}_avalon is
49            port (
50            """.format(name))
51      ENTITY_END=dedent("""\
52            );
53          end entity {}_avalon;\n
54          """.format(name))
```

```
55
56      entity = ENTITY_BEGIN
57
58      entity += indent*2 + "clk".ljust(26, ' ') + ": in  std_logic;\n"
59      entity += indent*2 + "reset".ljust(26, ' ') + ": in  std_logic;\n"
60
61      # avalon streaming sink
62      if sink_enabled:
63          for signal in sink:
64              datatype = convert_data_type(signal['data_type'])
65              entity += indent*2 + (signal['name']).ljust(26, ' ') + ": in  " +
                ↪  datatype + "; --" +\
66                  signal['data_type']['type'] + '\n'
67
68
69      # avalon streaming source
70      if source_enabled:
71          for signal in source:
72              datatype = convert_data_type(signal['data_type'])
73              entity += indent*2 + signal['name'].ljust(26, ' ') + ": out " +
                ↪  datatype + "; --" +\
74                  signal['data_type']['type'] + '\n'
75
76      # avalon memorymapped bus
77      if registers_enabled:
78          entity += indent*2 + "avalon_slave_address".ljust(26, ' ') + ": in
            ↪  std_logic_vector({} downto 0);\
79              \n".format(str(int(ceil(log(len(registers),2)) - 1)).ljust(3, ' '))
80
81          entity += indent*2 + "avalon_slave_read".ljust(26, ' ') + ": in
            ↪  std_logic;\n"
82          entity += indent*2 + "avalon_slave_readdata".ljust(26, ' ') + ": out
            ↪  std_logic_vector(31  downto 0);\n"
83
84          entity += indent*2 + "avalon_slave_write".ljust(26, ' ') + ": in
            ↪  std_logic;\n"
85          entity += indent*2 + "avalon_slave_writedata".ljust(26, ' ') +": in
            ↪  std_logic_vector(31  downto 0);\n"
86
87      # input conduit signals
88      if conduit_in_enabled:
89          for signal in conduit_in:
90              name = re.search('Export_(.*)', signal['name']).group(1)
91              datatype = convert_data_type(signal['data_type'])
92
93              entity += indent*2 + name.ljust(30, ' ') + ": in " + datatype + ";
                ↪  --" + \
94                  signal['data_type']['type'] + '\n'
95
96      # output conduit signals
97      if conduit_out_enabled:
```

```
98          for signal in conduit_out:
99              name = re.search('Export_(.*)', signal['name']).group(1)
100             datatype = convert_data_type(signal['data_type'])
101
102             entity += indent*2 + name.ljust(26, ' ') + ": out " + datatype + ";
                ↪  --" + \
103                 signal['data_type']['type'] + '\n'
104
105     # remove the semicolon from the last entity port definition
106     semicolon_idx = entity.rfind(';')
107     entity = entity[:semicolon_idx] + entity[semicolon_idx+1:]
108
109     entity += ENTITY_END
110
111     return entity
112
113 def create_architecture(name, registers_enabled, registers, register_defaults,
        component_declaration, component_instantiation, clock,
115     sink_flag, sink_signal, mm_flag, mm_signal, ci_flag, ci_signal, source_flag,
        ↪  source_signal, co_flag, co_signal):
116     global indent
117
118     ARCH_BEGIN = "architecture {}_avalon_arch of {}_avalon is\n\n".format(name,
        ↪  name)
119     ARCH_END = "end architecture;"
120
121     architecture = ARCH_BEGIN
122
123     if registers_enabled:
124         # sort registers according to register number
125         registers = sorted(registers, key=lambda k: k['reg_num'])
126
127         # declare register signals
128         (register_defaults, data_widths) = create_component_reg_defaults(mm_flag,
            ↪  mm_signal)
129         for register, data_width in zip(register_defaults, data_widths):
130             if data_width == 1:
131                 architecture += indent + "signal " + register.replace('<=',
132                 ': std_logic :=') + "\n"
133             else:
134                 architecture += indent + "signal " + register.replace('<=',
135                 ': std_logic_vector({}  downto 0) :='.format(data_width-1)) + "\n"
136
137
138
139     architecture += "\n"
140     architecture += create_component_declaration2(clock=clock, entity=name,
        ↪  sink_flag=sink_flag, sink_signal=sink_signal,
141                             mm_flag=mm_flag, mm_signal=mm_signal,
                                ↪  ci_flag=ci_flag, ci_signal=ci_signal,
```

```
142                             source_flag=source_flag, source_signal=source_signal,
                                ↪  co_flag=co_flag, co_signal=co_signal)
143
144     # begin architecture
145     architecture += "\nbegin\n\n"
146
147     architecture += create_component_instantiation2(ts_system=clock, entity=name,
        ↪  sink_flag=sink_flag, sink_signal=sink_signal,
148                             mm_flag=mm_flag, mm_signal=mm_signal,
                                ↪  ci_flag=ci_flag, ci_signal=ci_signal,
149                             source_flag=source_flag, source_signal=source_signal,
                                ↪  co_flag=co_flag, co_signal=co_signal)
150
151     architecture += "\n"
152
153     if registers_enabled:
154         addr_width = int(ceil(log(len(registers),2)))
155
156         # create read process
157         architecture += indent + "bus_read : process(clk)\n" + indent + "begin\n"
158         architecture += indent*2 + "if rising_edge(clk) and avalon_slave_read =
            ↪  '1' then\n"
159         architecture += indent*3 + "case avalon_slave_address is\n"
160
161         for register, data_width in zip(registers, data_widths):
162             if data_width == 1:
163                 value = "(31 downto 1 => '0') & {0}".format(register['name'])
164             elif data_width != 32:
165                 if 'sfix' in register['data_type']:
166                     value = '(31 downto {0} => {2}({1})) &
                        ↪  {2}'.format(data_width, data_width-1, register['name'])
167                 else:
168                     value = '(31 downto {0} => \'0\') & {1}'.format(data_width,
                        ↪  register['name'])
169             else:
170                 value = register['name']
171
172             architecture += indent*4 + \
173                 "when \"{0:0{1}b}\" => avalon_slave_readdata <=
                    ↪  {2};\n".format(register['reg_num'],\
174                 addr_width, value)
175
176         architecture += indent*4 + "when others => avalon_slave_readdata <=
            ↪  (others => '0');\n"
177         architecture += indent*3 + "end case;\n" + indent*2 + "end if;\n" + \
178             indent + "end process;\n\n"
179
180         # create write process
181         architecture += indent + "bus_write : process(clk, reset)\n" + indent +
            ↪  "begin\n"
182         architecture += indent*2 + "if reset = '1' then\n"
```

```
183
184         for reg in register_defaults:
185             architecture += indent * 3 + reg + "\n"
186
187         architecture += indent*2 + "elsif rising_edge(clk) and " + \
188             "avalon_slave_write = '1' then\n"
189         architecture += indent*3 + "case avalon_slave_address is\n"
190
191         for register, data_width in zip(registers, data_widths):
192             if data_width == 1:
193                 architecture += indent*4 + \
194                     "when \"{0:0{1}b}\" => {2} <=
                        ↪ avalon_slave_writedata(0);\n".format(
195                         register['reg_num'], addr_width, register['name'])
196             else:
197                 architecture += indent*4 + \
198                     "when \"{0:0{1}b}\" => {2} <= avalon_slave_writedata({3}
                        ↪ downto 0);\n".format(
199                         register['reg_num'], addr_width, register['name'],
                            ↪ data_width-1)
200
201         architecture += indent*4 + "when others => null;\n"
202         architecture += indent*3 + "end case;\n" + indent*2 + "end if;\n" + \
203             indent + "end process;\n\n"
204
205     architecture += ARCH_END
206
207     return architecture
208
209 # TODO: rename this to create_datatype_string?
210 def convert_data_type(typedict):
211     # TODO: add support for more data types (e.g. integer, signed, unsigned)?
212     if typedict['type'] in {'boolean', 'ufix1'}:
213         typestr = 'std_logic'
214     else:
215         typestr = 'std_logic_vector({} downto
            ↪ 0)'.format(str(typedict['width']-1).ljust(3, ' '))
216     return typestr
217
218 def num_to_bitstring(value, tot_bits, frac_bits):
219     # make value positive, then take the two's complement later if value is
        ↪ supposed to be negative
220     is_negative = value < 0
221     value = fabs(value)
222
223     # Get rid of the binary point by shifting the value left by frac_bits.
224     # The value must be an int to be converted to a binary string.
225     # The bits in this new value are the closest possible representation
226     # to the original floating point value
227     value = int(round(2**frac_bits * value))
228
```

```
229     if is_negative:
230         # take the two's complement; this also handles the sign extension
231         toggle_mask = 2**tot_bits - 1
232         value ^= toggle_mask
233         value += 1
234 
235     # [2:] removes '0b' from the binary string
236     bitstring = bin(value)[2:]
237 
238     if not is_negative:
239         # sign extend with 0's
240         bitstring = bitstring.rjust(tot_bits, "0")
241 
242     # wrap the string in quotes
243     bitstring = '"{0}"'.format(bitstring)
244 
245     return bitstring
246 
247 
248 def create_component_declaration2(clock, entity, sink_flag, sink_signal, mm_flag,
    ↪  mm_signal, ci_flag, ci_signal, source_flag, source_signal, co_flag,
    ↪  co_signal):
249     global indent
250     decl = "component " + entity + "\n"
251     decl += indent * 1 + "port(\n"
252     decl += indent * 2 + "clk".ljust(28, ' ') + ": in  std_logic; -- clk_freq = "
        ↪  + str(clock['frequency']) + " Hz, period = " + str(clock['period']) + "\n"
253     decl += indent * 2 + "reset".ljust(28, ' ') + ": in  std_logic;\n"
254     decl += indent * 2 + "clk_enable".ljust(28, ' ') + ": in  std_logic;\n"
255     if sink_flag == 1:
256         for i in range(len(sink_signal)):
257             name = sink_signal[i]["name"]
258             data_type = sink_signal[i]["data_type"]
259             decl += (indent * 2 + name).ljust(32, ' ') + (": in  " +
                ↪  convert_data_type(data_type) + ";").ljust(45, ' ') + " -- " +
                ↪  data_type['type'] + "\n"
260     if mm_flag == 1:
261         for i in range(len(mm_signal)):
262             # the vhdl that matlab generates has register_control prefixed to
                ↪  each register name, so we need to do the same
263             name = "register_control_" + mm_signal[i]["name"]
264             data_type = mm_signal[i]["data_type"]
265             decl += (indent * 2 + name).ljust(32, ' ') + (": in  " +
                ↪  convert_data_type(data_type) + ";").ljust(45, ' ') + " -- " +
                ↪  data_type['type'] + "\n"
266     if ci_flag == 1:
267         for i in range(len(ci_signal)):
268             name = ci_signal[i]["name"]
269             data_type = ci_signal[i]["data_type"]
```

```
270             decl += (indent * 2 + name).ljust(32, ' ') + (": in  " +
                ↪  convert_data_type(data_type) + ";").ljust(45, ' ') + " -- " +
                ↪  data_type['type'] + "\n"
271     decl += (indent * 2 + "ce_out").ljust(32, ' ') + ": out std_logic;\n"
272     if source_flag == 1:
273         for i in range(len(source_signal)):
274             name = source_signal[i]["name"]
275             data_type = source_signal[i]["data_type"]
276             decl += (indent * 2 + name).ljust(32, ' ') + (": out " +
                ↪  convert_data_type(data_type) + ";").ljust(45, ' ') + " -- " +
                ↪  data_type['type'] + "\n"
277     if co_flag == 1:
278         for i in range(len(co_signal)):
279             name = co_signal[i]["name"]
280             data_type = co_signal[i]["data_type"]
281             decl += (indent * 2 + name).ljust(32, ' ') + (": out " +
                ↪  convert_data_type(data_type) + ";").ljust(45, ' ') + " -- " +
                ↪  data_type['type'] + "\n"
282     last_semi_ind = decl.rfind(";")
283     decl = decl[:last_semi_ind] + ' ' + decl[last_semi_ind + 1:]
284     decl += indent * 1 + ");\n"
285     decl += "end component;\n"
286     return decl
287
288 def create_component_instantiation2(ts_system, entity, sink_flag, sink_signal,
    ↪  mm_flag, mm_signal, ci_flag, ci_signal, source_flag, source_signal, co_flag,
    ↪  co_signal):
289     global indent
290     inst = "u_" + entity + " : " + entity + "\n"
291     inst += indent * 1 + "port map(\n"
292     inst += (indent * 2 + "clk").ljust(32, ' ') + "=>  clk,\n"
293     inst += (indent * 2 + "reset").ljust(32, ' ') + "=>  reset,\n"
294     inst += (indent * 2 + "clk_enable").ljust(32, ' ') + "=>  '1',\n"
295     if sink_flag == 1:
296         for i in range(len(sink_signal)):
297             name = sink_signal[i]["name"]
298             data_type = sink_signal[i]["data_type"]
299             inst += (indent * 2 + name).ljust(32, ' ') + "=>  " + (name +
                ↪  ",").ljust(32, ' ') + " -- " + data_type['type'] + "\n"
300     if mm_flag == 1:
301         for i in range(len(mm_signal)):
302             name = "register_control_" + mm_signal[i]["name"]
303             name2 = mm_signal[i]["name"]
304             data_type = mm_signal[i]["data_type"]
305             inst += (indent * 2 + name).ljust(32, ' ') + "=>  " + (name2 +
                ↪  ",").ljust(32, ' ') + " -- " + data_type['type'] + "\n"
306     if ci_flag == 1:
307         for i in range(len(ci_signal)):
308             name = ci_signal[i]["name"]
309             name2 = name.replace("Export_", "")
310             data_type = ci_signal[i]["data_type"]
```

```
311             inst += (indent * 2 + name).ljust(32, ' ') + "=>  " + (name2 +
                ↪  ",").ljust(32, ' ') + " -- " + data_type['type'] + "\n"
312     if source_flag == 1:
313         for i in range(len(source_signal)):
314             name = source_signal[i]["name"]
315             data_type = source_signal[i]["data_type"]
316             inst += (indent * 2 + name).ljust(32, ' ') + "=>  " + (name +
                ↪  ",").ljust(32, ' ') + " -- " + data_type['type'] + "\n"
317     if co_flag == 1:
318         for i in range(len(co_signal)):
319             name = co_signal[i]["name"]
320             name2 = name.replace("Export_", "")
321             data_type = co_signal[i]["data_type"]
322             inst += (indent * 2 + name).ljust(32, ' ') + "=>  " + (name2 +
                ↪  ",").ljust(32, ' ') + " -- " + data_type['type'] + "\n"
323     last_comma_ind = inst.rfind(',')
324     inst = inst[:last_comma_ind] + ' ' + inst[last_comma_ind + 1:]
325     inst += indent * 1 + ");\n"
326     return inst
327
328
329 def create_component_reg_defaults(mm_flag, mm_signal):
330     reg_defs = []
331     data_widths = []
332
333     if mm_flag == 1:
334         for i in range(len(mm_signal)):
335             name = mm_signal[i]["name"]
336             name2 = name.replace("Register_Control_", "")
337             def_val = mm_signal[i]["default_value"]
338             datatype = mm_signal[i]["data_type"]
339             typestr = datatype['type']
340
341             (value_str, data_width) = convert_default_value(def_val, datatype)
342             reg_defs.append(name2.ljust(24, ' ') + "  <=  " + value_str +
343                 "; -- " + str(def_val) + " (" + typestr + ")")
344             data_widths.append(data_width)
345
346     return (reg_defs, data_widths)
347
348
349 def convert_default_value(value, datatype):
350     is_int = False
351     is_bool = False
352     data_width = 32
353
354     if datatype['type'] == 'boolean':
355         is_bool = True
356         data_width = 1
357     else:
358         data_width = datatype['width']
```

```
359         frac_width = datatype['fractional_bits']
360
361     # create the default value string
362     if is_int:
363         value_str = "std_logic_vector(to_unsigned({}, {}))".format(value,
            ↪  data_width)
364     elif is_bool:
365         value_str = "'{}'".format(value)
366     else:
367         value_str = num_to_bitstring(value, data_width, frac_width)
368
369     return (value_str, data_width)
370
371
372 def parseargs():
373     parser = argparse.ArgumentParser(description=\
374         "Generate VHDL code for Avalon streaming and memory-mapped interfaces.")
375     parser.add_argument('infile',
376         help="json file containing the interface and register specifications")
377     parser.add_argument('-v', '--verbose', action='store_true',
378         help="verbose output")
379     parser.add_argument('-p', '--print', action='store_true', dest='print_output',
380         help="print out the generated vhdl code")
381     parser.add_argument('outfile', help="the name of the output vhdl file")
382     args = parser.parse_args()
383     return (args.infile, args.outfile, args.verbose, args.print_output)
384
385
386 # TODO: make a default filename?
387 def main(infile, outfile, verbose, print_output):
388     with open(infile) as f:
389         avalon = json.load(f)
390
391     name = avalon['entity']
392     sink_enabled = avalon['avalon_sink_flag']
393     sink = avalon['avalon_sink']['signal'] if sink_enabled else None
394     source_enabled = avalon['avalon_source_flag']
395     source = avalon['avalon_source']['signal'] if source_enabled else None
396     registers_enabled = avalon['avalon_memorymapped_flag']
397     registers = avalon['avalon_memorymapped']['register'] if registers_enabled
        ↪  else None
398     conduit_in_enabled = avalon['conduit_input_flag']
399     conduit_in = avalon['conduit_input']['signal'] if conduit_in_enabled else None
400     conduit_out_enabled = avalon['conduit_output_flag']
401     conduit_out = avalon['conduit_output']['signal'] if conduit_out_enabled else
        ↪  None
402     register_defaults = None#avalon['vhdl']['register_defaults']
403     component_declaration = None#avalon['vhdl']['component_declaration']
404     component_instantiation = None#avalon['vhdl']['component_instantiation']
405     clock = {'frequency': 1, 'period': .1}#avalon['clock']
406
```

```
407
408     vhdl_out = create_library()
409     vhdl_out += create_entity(name=name, sink_enabled=sink_enabled, sink=sink,
        ↪  source_enabled=source_enabled, source=source,
410         registers_enabled=registers_enabled, registers=registers,
            ↪  conduit_out_enabled=conduit_out_enabled,
411         conduit_out=conduit_out, conduit_in_enabled=conduit_in_enabled,
            ↪  conduit_in=conduit_in)
412     vhdl_out += create_architecture(name, registers_enabled, registers,
413         register_defaults, component_declaration, component_instantiation,
            ↪  clock=clock, sink_flag=sink_enabled, sink_signal=sink,
414         source_flag=source_enabled, source_signal=source,
            ↪  mm_flag=registers_enabled, mm_signal=registers,
415         co_flag=conduit_out_enabled, co_signal=conduit_out,
            ↪  ci_flag=conduit_in_enabled,
416         ci_signal=conduit_in)
417
418     if print_output:
419         print(vhdl_out)
420
421     with open(outfile, 'w') as f:
422         f.write(vhdl_out)
423
424 if __name__ == '__main__':
425     (infile, outfile, verbose, print_output) = parseargs()
426     main(infile, outfile, verbose, print_output)
```

vgenTcl.m

```
1  % vgenTcl Generate an TCL script from a json input file
2  %
3  % vgenTcl(infile, outfile, additionalFilesetAbsDir)
4  %
5  % The json file this function expects has a specific format that contains
   ↪  information about the vhdl entity and
6  % the Avalon streaming interface parameters. This will often be generated by
   ↪  MATLAB/Simulink, but it can also be written
7  % by hand. An example of the format is shown below. This function calls a
8  % python script autogen_tcl.py that generates the tcl script that allows
9  % the simulink model to be seen as a custom component in Platform Designer
10 % in Quartus
11 %
12 % Inputs:
13 %   infile = the json input filename
14 %   additionalFilesetAbsDir = the VHDL source directory the the HDL coder creates
   ↪  for the IP core generation
15 %   outfile = the tcl output filename
16 % Copyright 2019 Flat Earth Inc, Montana State University
17 %
```

```
% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
% INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
% PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
% FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
% ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
%
% Ross Snider
% Flat Earth Inc
% 985 Technology Blvd
% Bozeman, MT 59718
% support@flatearthinc.com

function vgenTcl(infile, outfile, additionalFilesetAbsDir)

% call the python file that autogens the tcl script
py.autogen_tcl.main(infile, outfile, additionalFilesetAbsDir)
```

TCL Autogeneration

Aaron Koenigsberg was the primary author of the following code listings, with assistance and bug fixes from me.

autogen_tcl.py

```
from create_connection_point_clock import create_connection_point_clock
from create_connection_point_reset import create_connection_point_reset
from create_file_sets import create_file_sets
from create_module import create_module
from create_module_assignments import create_module_assignments
from create_header_file_stuff import create_header_file_stuff
from create_mm_connection_point import create_mm_connection_point
from create_sink_connection_point import create_sink_connection_point
from create_source_connection_point import  create_source_connection_point
from input_structure import InputStructure
from parse_json import parse_json
from populate_additional_filesets import populate_additional_filesets
def main(inputFilename, outputFilename, additionalFilesetAbsDir):
    input_struct = parse_json(inputFilename)
    populate_additional_filesets(input_struct, additionalFilesetAbsDir)
    write_tcl(input_struct, outputFilename)

def write_tcl(input_struct, outputFilename):
    with open(outputFilename, "w") as out_file:
        out_file.write(create_header_file_stuff(input_struct))
        out_file.write(create_module(input_struct))
```

```
        out_file.write(create_file_sets(input_struct))
        out_file.write(create_module_assignments(input_struct))
        out_file.write(create_connection_point_clock(input_struct))
        out_file.write(create_connection_point_reset(input_struct))
        out_file.write(create_mm_connection_point(input_struct))
        out_file.write(create_sink_connection_point(input_struct))
        out_file.write(create_source_connection_point(input_struct))
```

create_connection_point_clock.py

```
def create_connection_point_clock(input_struct):
    built_string = "# # # # # # # # # # # # # # # # # # # # # #\n"
    built_string += "# Created by create_connection_point_clock\n"
    built_string += "# # # # # # # # # # # # # # # # # # # # # #\n\n"
    built_string += "add_interface clock clock end\n"
    built_string += "set_interface_property clock clockRate " +
    ↪  str(input_struct.clock_rate) + "\n"
    built_string += "set_interface_property clock ENABLED true\n"
    built_string += "set_interface_property clock EXPORT_OF \"\"\n"
    built_string += "set_interface_property clock PORT_NAME_MAP \"\"\n"
    built_string += "set_interface_property clock CMSIS_SVD_VARIABLES \"\"\n"
    built_string += "set_interface_property clock SVD_ADDRESS_GROUP \"\"\n"
    built_string += "add_interface_port clock " + input_struct.clock_abbrev + " "
    ↪  + input_struct.clock_abbrev + " Input 1\n"
    built_string += "# End create_connection_point_clock\n\n\n"
    return built_string
```

create_connection_point_reset.py

```
def create_connection_point_reset(input_struct):
    built_string = "# # # # # # # # # # # # # # # # # # # # # #\n"
    built_string += "# Created by create_connection_point_reset\n"
    built_string += "# # # # # # # # # # # # # # # # # # # # # #\n\n"
    built_string += "add_interface reset reset end\n"
    built_string += "set_interface_property reset associatedClock clock\n"
    built_string += "set_interface_property reset synchronousEdges DEASSERT\n"
    built_string += "set_interface_property reset ENABLED true\n"
    built_string += "set_interface_property reset EXPORT_OF true\n"
    built_string += "set_interface_property reset PORT_NAME_MAP \"\"\n"
    built_string += "set_interface_property reset CMSIS_SVD_VARIABLES \"\"\n"
    built_string += "set_interface_property reset SVD_ADDRESS_GROUP \"\"\n"
    built_string += "add_interface_port reset reset reset Input 1\n"
    built_string += "# End create_connection_point_reset\n\n\n"
    return built_string
```

create_file_sets.py

```
def create_file_sets(input_struct):
    built_string = "# # # # # # # # # # # # # # # # # #\n"
    built_string += "# created in create_file_sets\n"
    built_string += "# # # # # # # # # # # # # # # # # #\n\n"
    built_string += "add_fileset QUARTUS_SYNTH QUARTUS_SYNTH \"\" \"\"\n"

    # add additional dependent files before the top level file; quartus needs to synthesize dependencies first
    for i in range(len(input_struct.additional_filesets)):
        built_string += ("add_fileset_file " + input_struct.additional_filesets[i] + " VHDL PATH " +
                         input_struct.additional_filesets[i] + "\n")

    # add the top level file
    built_string += "set_fileset_property QUARTUS_SYNTH TOP_LEVEL " + input_struct.quartus_synth_top_level + "\n"
    built_string += ("set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS " +
                     str(input_struct.enable_rel_inc_paths).lower() + "\n")
    built_string += ("set_fileset_property QUARTUS_SYNTH ENABLE_FILE_OVERWRITE_MODE " +
                     str(input_struct.enable_file_overwrite).lower() + "\n")
    built_string += ("add_fileset_file " + input_struct.vhdl_top_level_file + " VHDL PATH " +
                     input_struct.vhdl_top_level_file + " TOP_LEVEL_FILE\n")
    built_string += "# end create_file_sets\n\n\n"

    return built_string
```

create_header_file_stuff.py

```
def create_header_file_stuff(input_struct):
    built_string = "# # # # # # # # # # # # # # # # # #\n"
    built_string += "# Built in create_header_file_stuff\n"
    built_string += "# # # # # # # # # # # # # # # # # #\n\n"
    built_string += "package require -exact qsys 16.1\n"
    built_string += "# End create_header_file_stuff\n\n\n"
    return built_string
```

create_mm_connection_point.py

```
def create_mm_connection_point(input_struct):
    if not input_struct.has_avalon_mm_slave_signal:
        return ""
    built_string = "# # # # # # # # # # # # # # # # # # # #\n"
    built_string += "# Created by create_mm_connection_point\n"
    built_string += "# # # # # # # # # # # # # # # # # # # #\n\n"
```

```
7       mmname = input_struct.avalon_mm_slave_signal_name
8       built_string += "add_interface " + mmname + " avalon end\n"
9       built_string += "set_interface_property " + mmname + " addressUnits WORDS\n"
10      built_string += "set_interface_property " + mmname + " associatedClock
        ↪  clock\n"
11      built_string += "set_interface_property " + mmname + " associatedReset
        ↪  reset\n"
12      built_string += "set_interface_property " + mmname + " bitsPerSymbol 8\n"
13      built_string += "set_interface_property " + mmname + "
        ↪  burstOnBurstBoundariesOnly false\n"
14      built_string += "set_interface_property " + mmname + " burstcountUnits
        ↪  WORDS\n"
15      built_string += "set_interface_property " + mmname + " explicitAddressSpan
        ↪  0\n"
16      built_string += "set_interface_property " + mmname + " holdTime 0\n"
17      built_string += "set_interface_property " + mmname + " linewrapBursts false\n"
18      built_string += "set_interface_property " + mmname + "
        ↪  maximumPendingReadTransactions 0\n"
19      built_string += "set_interface_property " + mmname + "
        ↪  maximumPendingWriteTransactions 0\n"
20      built_string += "set_interface_property " + mmname + " readLatency 0\n"
21      built_string += "set_interface_property " + mmname + " readWaitTime 1\n"
22      built_string += "set_interface_property " + mmname + " setupTime 1\n"
23      built_string += "set_interface_property " + mmname + " timingUnits Cycles\n"
24      built_string += "set_interface_property " + mmname + " writeWaitTime 0\n"
25      built_string += "set_interface_property " + mmname + " ENABLED true\n"
26      built_string += "set_interface_property " + mmname + " EXPORT_OF \"\"\n"
27      built_string += "set_interface_property " + mmname + " PORT_NAME_MAP \"\"\n"
28      built_string += "set_interface_property " + mmname + " CMSIS_SVD_VARIABLES
        ↪  \"\"\n"
29      built_string += "set_interface_property " + mmname + " SVD_ADDRESS_GROUP
        ↪  \"\"\n\n"
30
31      built_string += "add_interface_port " + mmname + " " + mmname + "_address
        ↪  address Input " + str(input_struct.address_bus_size) + "\n"
32      built_string += "add_interface_port " + mmname + " " + mmname + "_read read
        ↪  Input 1\n"
33      built_string += "add_interface_port " + mmname + " " + mmname + "_readdata
        ↪  readdata Output " + str(input_struct.data_bus_size) + "\n"
34      built_string += "add_interface_port " + mmname + " " + mmname + "_write write
        ↪  Input 1\n"
35      built_string += "add_interface_port " + mmname + " " + mmname + "_writedata
        ↪  writedata Input " + str(input_struct.data_bus_size) + "\n"
36      built_string += "set_interface_assignment " + mmname + "
        ↪  embeddedsw.configuration.isFlash 0\n"
37      built_string += "set_interface_assignment " + mmname + "
        ↪  embeddedsw.configuration.isMemoryDevice 0\n"
38      built_string += "set_interface_assignment " + mmname + "
        ↪  embeddedsw.configuration.isNonVolatileStorage 0\n"
39      built_string += "set_interface_assignment " + mmname + "
        ↪  embeddedsw.configuration.isPrintableDevice 0\n"
```

```
    built_string += "# End create_mm_connection_point\n\n\n"
    return built_string
```

create_module_assignments.py

```
def create_module_assignments(input_structs):
    built_string = "# # # # # # # # # # # # # # # # # #\n"
    built_string += "# Created in create_module_assignments\n"
    built_string += "# # # # # # # # # # # # # # # # # #\n\n"
    built_string += "set_module_assignment embeddedsw.dts.compatible " +
    ↪  input_structs.compatible_flag + "\n"
    built_string += "set_module_assignment embeddedsw.dts.group " +
    ↪  input_structs.group + "\n"
    built_string += "set_module_assignment embeddedsw.dts.vendor " +
    ↪  input_structs.vendor + "\n"
    built_string += "# End create_module_assignments\n\n\n"
    return built_string
```

create_module.py

```
def create_module(input_struct):
    built_string = "# # # # # # # # # # # # # # # # # #\n"
    built_string += "# Created in create_module\n"
    built_string += "# # # # # # # # # # # # # # # # # #\n\n"
    built_string += "set_module_property DESCRIPTION \"" +
    ↪  input_struct.description + "\"\n"
    built_string += "set_module_property NAME \"" + input_struct.name + "\"\n"
    built_string += "set_module_property VERSION " + input_struct.version + "\n"
    built_string += "set_module_property OPAQUE_ADDRESS_MAP " +
    ↪  str(input_struct.opaque_address_map).lower() + "\n"
    built_string += "set_module_property AUTHOR \"" + input_struct.author + "\"\n"
    built_string += "set_module_property DISPLAY_NAME \"" +
    ↪  input_struct.display_name + "\"\n"
    built_string += "set_module_property INSTANTIATE_IN_SYSTEM_MODULE " +
    ↪  str(input_struct.inst_in_sys_mod).lower() + "\n"
    built_string += "set_module_property EDITABLE " +
    ↪  str(input_struct.editable).lower() + "\n"
    built_string += "set_module_property REPORT_TO_TALKBACK " +
    ↪  str(input_struct.report_to_talkback).lower() + "\n"
    built_string += "set_module_property ALLOW_GREYBOX_GENERATION " +
    ↪  str(input_struct.allow_greybox_generation).lower() + "\n"
    built_string += "set_module_property VERSION " + input_struct.version + "\n"
    built_string += "set_module_property REPORT_HIERARCHY " +
    ↪  str(input_struct.report_hierarchy).lower() + "\n"
    for i in range(len(input_struct.additional_module_properties)):
        built_string += ("set_module_property " +
        ↪  input_struct.additional_module_properties[i][0] +
                         " " + input_struct.additional_module_properties[i][1] +
                         ↪  "\n")
```

```
    built_string += "# end of create_module\n\n\n"
    return built_string
```

create_sink_connection_point.py

```
def create_sink_connection_point(input_struct):
    if not input_struct.has_sink_signal:
        return ""
    built_string = "# # # # # # # # # # # # # # # # # # # # # #\n"
    built_string += "# Created by create_sink_connection_point\n"
    built_string += "# # # # # # # # # # # # # # # # # # # # # #\n\n"
    ssname = input_struct.sink_signal_name
    built_string += "add_interface " + ssname + " avalon_streaming end\n"
    built_string += "set_interface_property " + ssname + " associatedClock
    ↪  clock\n"
    built_string += "set_interface_property " + ssname + " associatedReset
    ↪  reset\n"
    built_string += "set_interface_property " + ssname + " dataBitsPerSymbol " +
    ↪  str(input_struct.data_bus_size) + "\n"
    built_string += "set_interface_property " + ssname + " errorDescriptor \"\"\n"
    built_string += "set_interface_property " + ssname + "
    ↪  firstSymbolInHighOrderBits true\n"
    built_string += "set_interface_property " + ssname + " maxChannel " +
    ↪  str(input_struct.sink_max_channel) + "\n"
    built_string += "set_interface_property " + ssname + " readyLatency 0\n"
    built_string += "set_interface_property " + ssname + " ENABLED true\n"
    built_string += "set_interface_property " + ssname + " EXPORT_OF \"\"\n"
    built_string += "set_interface_property " + ssname + " PORT_NAME_MAP \"\"\n"
    built_string += "set_interface_property " + ssname + " CMSIS_SVD_VARIABLES
    ↪  \"\"\n"
    built_string += "set_interface_property " + ssname + " SVD_ADDRESS_GROUP
    ↪  \"\"\n"

    for port_name, num_bits in
    ↪  input_struct.sink_signal_port_names_and_widths.items():
        port_type = port_name.rpartition('_')[2]
        built_string += ("add_interface_port " + ssname + " " + port_name + " " +
        ↪  port_type + " Input "
                         + str(num_bits) + "\n")
    built_string += "# End create_sink_connection_point\n\n\n"
    return built_string
```

create_source_connection_point.py

```
def create_source_connection_point(input_struct):
    if not input_struct.has_source_signal:
        return ""
    built_string = "# # # # # # # # # # # # # # # # # # # # # #\n"
```

```
    built_string += "# Created in create_source_connection_point\n"
    built_string += "# # # # # # # # # # # # # # # # # # # # # # #\n\n"
    ssname = input_struct.source_signal_name
    built_string += "add_interface " + ssname + " avalon_streaming start\n"
    built_string += "set_interface_property " + ssname + " associatedClock
    ↪  clock\n"
    built_string += "set_interface_property " + ssname + " associatedReset
    ↪  reset\n"
    built_string += "set_interface_property " + ssname + " dataBitsPerSymbol " +
    ↪  str(input_struct.data_bus_size) + "\n"
    built_string += "set_interface_property " + ssname + " errorDescriptor \"\"\n"
    built_string += "set_interface_property " + ssname + "
    ↪  firstSymbolInHighOrderBits true\n"
    built_string += "set_interface_property " + ssname + " maxChannel " +
    ↪  str(input_struct.source_max_channel) + "\n"
    built_string += "set_interface_property " + ssname + " readyLatency 0\n"
    built_string += "set_interface_property " + ssname + " ENABLED true\n"
    built_string += "set_interface_property " + ssname + " EXPORT_OF \"\"\n"
    built_string += "set_interface_property " + ssname + " PORT_NAME_MAP \"\"\n"
    built_string += "set_interface_property " + ssname + " CMSIS_SVD_VARIABLES
    ↪  \"\"\n"
    built_string += "set_interface_property " + ssname + " SVD_ADDRESS_GROUP
    ↪  \"\"\n"

    for port_name, num_bits in
    ↪  input_struct.source_signal_port_names_and_widths.items():
        port_type = port_name.rpartition('_')[2]
        built_string += ("add_interface_port " + ssname + " " + port_name + " " +
        ↪  port_type + " Output "
                         + str(num_bits) + "\n")
    built_string += "# End create_sink_connection_point\n\n\n"
    return built_string
```

input_structure.py

```
class InputStructure:
    def __init__(self):
        self.description = ""
        self.name = "Fill me in"
        self.version = "1.0"
        self.internal = False
        self.opaque_address_map = True
        self.author = ""
        self.display_name = "Fill me in"
        self.model_abbreviation = "Fill me in"
        self.inst_in_sys_mod = True
        self.editable = True
        self.report_to_talkback = False
        self.allow_greybox_generation = False
        self.report_hierarchy = False
```

```
16          self.additional_module_properties = []
17
18          self.quartus_synth_top_level = "Fill me in"
19          self.enable_rel_inc_paths = False
20          self.enable_file_overwrite = False
21          self.vhdl_top_level_file = "Fill me in"
22          self.additional_filesets = []
23
24          self.parameters = []
25
26          self.compatible_flag = "Fill me in"
27          self.group = "Fill me in"
28          self.vendor = "fe"
29
30          self.clock_rate = 0
31          self.clock_abbrev = "clk"
32
33          self.has_avalon_mm_slave_signal = True
34          self.avalon_mm_slave_signal_name = "Fill me in (or not)"
35          self.data_bus_size = 32
36          self.address_bus_size = 0
37
38          self.has_sink_signal = True
39          self.sink_signal_name = "Fill me in (or not)"
40          self.sink_signal_port_names_and_widths = {} #{"channel": 2, "data": 32,
            ↪  "error": 3, "valid": 1}
41          self.sink_max_channel = 0
42
43          self.has_source_signal = True
44          self.source_signal_name = "Fill me in (or not)"
45          self.source_signal_port_names_and_widths = {} #{"channel": 2, "data": 32,
            ↪  "error": 3, "valid": 1}
46          self.source_max_channel = 0
```

parse_json.py

```
1   import json
2   import math
3   import re
4   from input_structure import InputStructure
5   def parse_json(inputFilename):
6       with open(inputFilename, "r") as file:
7           in_str = file.read()
8           json_dict = json.loads(in_str)
9           input_struct = InputStructure()
10          input_struct.name = json_dict['model_name']
11          input_struct.quartus_synth_top_level = json_dict['entity'] + "_avalon"
12          input_struct.vhdl_top_level_file = json_dict['model_abbreviation'] +
            ↪  "_dataplane_avalon.vhd"
13          input_struct.compatible_flag = "dev,fe-" + json_dict['linux_device_name']
```

```
        input_struct.group = json_dict['model_name']
        input_struct.vendor = "fe"
        input_struct.clock_rate = json_dict['clocks']['system_frequency_Hz']
        input_struct.clock_abbrev = "clk"
        input_struct.display_name = json_dict['model_name']
        input_struct.model_abbreviation = json_dict['model_abbreviation']

        if json_dict['avalon_memorymapped_flag'] == 1:
            input_struct.avalon_mm_slave_signal_name = 'avalon_slave'
            input_struct.has_avalon_mm_slave_signal = True
            input_struct.data_bus_size = 32
            input_struct.address_bus_size = int(math.ceil(math.log(len(json_dict[
            ↪  'avalon_memorymapped']['register'])))
        if json_dict['avalon_sink_flag'] == 1:
            input_struct.sink_signal_name = 'avalon_streaming_sink'
            input_struct.has_sink_signal = True
            for signal in json_dict['avalon_sink']['signal']:
                input_struct.sink_signal_port_names_and_widths[signal['name']] =
                ↪  signal['data_type']['width']
                if 'channel' in signal['name']:
                    input_struct.sink_max_channel = int(math.pow(2, input_struct.
                    ↪  sink_signal_port_names_and_widths[signal['name']]) -
                    ↪  1)
        if json_dict['avalon_source_flag'] == 1:
            input_struct.source_signal_name = 'avalon_streaming_source'
            input_struct.has_source_signal = True
            for signal in json_dict['avalon_source']['signal']:
                input_struct.source_signal_port_names_and_widths[signal['name']]
                ↪  = signal['data_type']['width']
                if 'channel' in signal['name']:
                    input_struct.source_max_channel = int(math.pow(2, input_struc
                    ↪  t.source_signal_port_names_and_widths[signal['name']]) -
                    ↪  1)
    return input_struct
```

populate_additional_filesets.py

```
import os
def populate_additional_filesets(input_struct, additionalFilesetAbsDir):
    for filename in os.listdir(additionalFilesetAbsDir):
        if filename.endswith(".vhd") and '_avalon' not in filename:
            input_struct.additional_filesets.append(filename)
```

APPENDIX C

DEVICE DRIVER AUTOGENERATION SCRIPTS

Aaron Koenigsberg was the primary author of the following code listings, with assistance from me, except where stated otherwise in the file comment headers. I wrote the MATLAB scripts that call the python scripts, and I also the makefile autogeneration script.

AvalonJsonParser.py

```
from InputStructureFile import InputStructure
import json
def AvalonJsonParse(jsonFileName):
    file = open(jsonFileName, "r")
    fileStr = file.read()
    jsonDict = json.loads(fileStr)
    inputStruct = InputStructure()

    inputStruct.deviceName = jsonDict['linux_device_name']
    inputStruct.deviceType = 2
    inputStruct.deviceNameAbbrev = jsonDict['model_abbreviation']
    inputStruct.compatibleFlag = 'dev,fe-' + inputStruct.deviceName
    attributes = jsonDict['avalon_memorymapped']['register']
    inputStruct.deviceAttributes = []
    inputStruct.attributePerms = []
    inputStruct.attributeWriteIsNormal = []
    inputStruct.attributeReadIsNormal = []
    inputStruct.attributeWriteOffsets = []
    inputStruct.attributeDataTypes = []
    inputStruct.attributeDataTypeSigned = []
    inputStruct.attributeDataTypeWidth = []
    inputStruct.attributeDataTypeFraction = []
    for attr in attributes:
        inputStruct.deviceAttributes.append(attr['name'])
        inputStruct.attributeDataTypes.append(attr['data_type'])
        inputStruct.attributeDataTypeSigned.append(attr['data_type']['signed'])
        inputStruct.attributeDataTypeWidth.append(attr['data_type']['width'])
        inputStruct.attributeDataTypeFraction.append(attr['data_type']['fractiona
        ↪  l_bits'])
        inputStruct.attributePerms.append('0664')
        inputStruct.attributeWriteIsNormal.append(True)
        inputStruct.attributeReadIsNormal.append(True)
        inputStruct.attributeWriteOffsets.append([str(attr['reg_num'])])
    return inputStruct
```

CreateCFunctionStubs.py

```
def CreateCFunctionStubs(inputParams):
    devName = inputParams.deviceName
    functionString = "/*****************************************\n"
    functionString += "  Generated by CreateCFunctionStubs\n"
```

```
    functionString += "****************************************/\n"
    functionString += "static int " + devName + "_open(struct inode *inode,
    ↪  struct file *file) {\n"
    functionString += "  // TODO: fill this in (if its needed, it might not be)\n"
    functionString += "  return 0;\n"
    functionString += "}\n\n"
    functionString += "static int " + devName + "_release(struct inode *inode,
    ↪  struct file *file) {\n"
    functionString += "  // TODO: fill this in (if its needed, it might not be)\n"
    functionString += "  return 0;\n"
    functionString += "}\n\n"
    functionString += "static ssize_t " + devName + "_read(struct file *file,
    ↪  char *buffer, size_t len, loff_t *offset) {\n"
    functionString += "  // TODO: fill this in (if its needed, it might not be)\n"
    functionString += "  return 0;\n"
    functionString += "}\n\n"
    functionString += "static ssize_t " + devName + "_write(struct file *file,
    ↪  const char *buffer, size_t len, loff_t *offset) {\n"
    functionString += "  // TODO: fill this in (if its needed, it might not be)\n"
    functionString += "  return 0;\n"
    functionString += "}\n"
    functionString += " /* End CreateCFunctionStubs */\n\n\n"
    return functionString
```

CreateCustomFunctions.py

```
def CreateCustomFunctions(inputParams):
    functionString =  "/*********************************************\n"
    functionString += "Generated by CreateCustomFunctions\n"
    functionString += "*********************************************/\n"
    if inputParams.needsVolumeTable:
        file = open("venv/files/FindVolumeLevelTable", "r")
        functionString += file.read()
    elif inputParams.deviceType != 2:  #FPGAS don't need find_volume?
        file = open("venv/files/FindVolumeLevelNoTable", "r")
        functionString += file.read()
    functionString += "/* End CreateCustomFunctions */\n\n\n"
    return functionString
```

CreateDeviceAttrMacros.py

```
def WriteDeviceAttributes(inputParams):
    functionString = "/************************************************\n"
    functionString += "Generated in WriteDeviceAttributes\n"
    functionString += "************************************************/\n"
    for i in range(len(inputParams.deviceAttributes)):
        functionString += ("DEVICE_ATTR(" + inputParams.deviceAttributes[i] + ",
        ↪  " + inputParams.attributePerms[i]
```

```
                            + ", " + inputParams.deviceAttributes[i] + "_read, " +
                            ↪  inputParams.deviceAttributes[i]
                            + "_write);\n")
    functionString += "DEVICE_ATTR(name, 0444, name_read, NULL);\n"
    functionString += "/* End WriteDeviceAttributes */\n\n\n"
    return functionString
```

CreateDriverStuff.py

```
def CreateDriverStuff(inputParams):
    functionString = "/*****************************************************\n"
    functionString += "Generated in CreateDriverStuff\n"
    functionString += "*****************************************************/\n"
    devName = inputParams.deviceName
    functionString += CreateDevStruct(inputParams)
    functionString += "typedef struct fe_" + devName + "_dev fe_" + devName +
    ↪  "_dev_t;\n"
    functionString += CreateIDMatchingStruct(inputParams)
    if inputParams.deviceType == 1:  # if I2C device it needs the next thing
        functionString += CreateI2CDriverStructs(inputParams)
    functionString += "MODULE_DEVICE_TABLE(of, fe_" + devName + "_dt_ids);\n"
    functionString += CreatePlatformDriverStruct(inputParams)
    functionString += CreateFileOpsStruct(inputParams)
    functionString += "/* End of CreateDriverStuff */\n\n\n"
    return functionString


def CreateDevStruct(inputParams):
    functionString = "\n/* Device struct */\n"
    functionString += "struct fe_" + inputParams.deviceName + "_dev {\n"
    functionString += "  struct cdev cdev;\n"
    functionString += "  char *name;\n"
    functionString += "  void __iomem *regs;\n"
    for i in range(len(inputParams.deviceAttributes)):
        try:  # try to get the type of the attribute from input params, otherwise
        ↪  just assume its int;
            functionString += "  " + inputParams.attributeType[i]
        except:
            functionString += "  int"
        functionString += " " + inputParams.deviceAttributes[i] + ";\n"
    functionString += "};\n\n"
    return functionString


def CreateIDMatchingStruct(inputParams):
    functionString = "/* ID Matching struct */\n"
    functionString += "static struct of_device_id fe_" + inputParams.deviceName +
    ↪  "_dt_ids[] = {\n"
    functionString += "  {\n"
    functionString += "    .compatible = \"" + inputParams.compatibleFlag + "\"\n"
```

```
39      functionString += "  },\n"
40      functionString += "  { }\n"
41      functionString += "};\n\n"
42      return functionString
43
44
45  def CreatePlatformDriverStruct(inputParams):
46      functionString = "/* Platform driver struct */\n"
47      functionString += "static struct platform_driver " + inputParams.deviceName +
        ↪  "_platform = {\n"
48      functionString += "  .probe = " + inputParams.deviceName + "_probe,\n"
49      functionString += "  .remove = " + inputParams.deviceName + "_remove,\n"
50      functionString += "  .driver = {\n"
51      functionString += "    .name = \"Flat Earth " + inputParams.deviceName + "
        ↪  Driver\",\n"
52      functionString += "    .owner = THIS_MODULE,\n"
53      functionString += "    .of_match_table = fe_" + inputParams.deviceName +
        ↪  "_dt_ids\n"
54      functionString += "  }\n"
55      functionString += "};\n\n"
56      return functionString
57
58
59  def CreateFileOpsStruct(inputParams):
60      functionString = "/* File ops struct */\n"
61      functionString += "static const struct file_operations fe_" +
        ↪  inputParams.deviceName + "_fops = {\n"
62      functionString += "  .owner = THIS_MODULE,\n"
63      functionString += "  .read = "    + inputParams.deviceName + "_read,\n"
64      functionString += "  .write = "   + inputParams.deviceName + "_write,\n"
65      functionString += "  .open = "    + inputParams.deviceName + "_open,\n"
66      functionString += "  .release = " + inputParams.deviceName + "_release,\n"
67      functionString += "};\n\n"
68      return functionString
69
70
71  def CreateI2CDriverStructs(inputParams):
72      devNameAbbrev = inputParams.deviceNameAbbrev
73      functionString = "/* I2C Driver stuff */\n"
74      functionString += "static struct i2c_device_id " + devNameAbbrev + "_id[] =
        ↪  {\n"
75      functionString += "  {\n"
76      functionString += "    \"" + devNameAbbrev + "_i2c\"," + inputParams.deviceID
        ↪  + "\n"
77      functionString += "  },\n"
78      functionString += "  { }\n"
79      functionString += "};\n\n"
80      functionString += "static struct i2c_board_info " + devNameAbbrev +
        ↪  "_i2c_info = {\n"
81      functionString += "  I2C_BOARD_INFO(\"" + devNameAbbrev + "_i2c\"," +
        ↪  inputParams.deviceID + "),\n"
```

```
82      functionString += "};\n\n"
83      functionString += "static int " + devNameAbbrev + "_i2c_probe(struct
        ↪  i2c_client *client, const struct i2c_device_id *id) {\n"
84      functionString += "  return 0;\n"
85      functionString += "}\n"
86      functionString += "static int " + devNameAbbrev + "_i2c_remove(struct
        ↪  i2c_client *client) {\n"
87      functionString += "  return 0;\n"
88      functionString += "}\n"
89      functionString += "struct i2c_driver " + devNameAbbrev + "_i2c_driver = {\n"
90      functionString += "  .driver = {\n"
91      functionString += "    .name=\"" + devNameAbbrev + "_i2c\",\n"
92      functionString += "  },\n"
93      functionString += "  .probe = " + devNameAbbrev + "_i2c_probe,\n"
94      functionString += "  .remove = " + devNameAbbrev + "_i2c_remove,\n"
95      functionString += "  .id_table = " + devNameAbbrev + "_id,\n"
96      functionString += "};\n"
97      return functionString
```

CreateEndOfFileStuff.py

```
1   def CreateEndOfFileStuff(inputParams):
2       functionString = "/*********************************************\n"
3       functionString += "Generated by CreateEndOfFileStuff\n"
4       functionString += "*********************************************/\n"
5       functionString += "module_init(" + inputParams.deviceName + "_init);\n"
6       functionString += "module_exit(" + inputParams.deviceName + "_exit);\n"
7       functionString += "/* End CreateEndOfFileStuff */\n"
8       return functionString
```

CreateExitFunction.py

```
1   def CreateExitFunction(inputParams):
2       functionString =  "/*********************************************\n"
3       functionString += "Generated by CreateExitFunction\n"
4       functionString += "*********************************************/\n"
5       functionString += "static void " + inputParams.deviceName + "_exit(void) {\n"
6       functionString += "  pr_info(\"Flat Earth " + inputParams.deviceName + "
        ↪  module exit\\n\");\n"
7       functionString += "  platform_driver_unregister(&" + inputParams.deviceName +
        ↪  "_platform);\n"
8       if inputParams.deviceType == 0:  # SPI device
9           functionString += "  if (spi_device) {\n"
10          functionString += "    spi_unregister_device(spi_device);\n"
11          functionString += "  }\n"
12      functionString += "  pr_info(\"Flat Earth " + inputParams.deviceName + "
        ↪  module successfully unregistered\\n\");\n"
13      functionString += "}\n"
```

```
    functionString += "/* End CreateExitFunction */\n\n\n"
    return functionString
```

CreateFileHeaderInfo.py

```
def CreateFileHeaderInfo(inputParams):
    functionString =
    ↪  "/**************************************************************************\n"
    functionString += "Generated in CreateFileHeaderInfo \n"
    functionString +=
    ↪  "**************************************************************************/\n"
    functionString += "#include <linux/module.h>\n"
    functionString += "#include <linux/platform_device.h>\n"
    functionString += "#include <linux/io.h>\n"
    functionString += "#include <linux/fs.h>\n"
    functionString += "#include <linux/types.h>\n"
    functionString += "#include <linux/uaccess.h>\n"
    functionString += "#include <linux/init.h>\n"
    functionString += "#include <linux/cdev.h>\n"
    functionString += "#include <linux/regmap.h>\n"
    if inputParams.deviceType == 0: # SPI Device, fpga needs struct of_device_id
    ↪  from here
        functionString += "#include <linux/spi/spi.h>\n"
    elif inputParams.deviceType == 1: # I2C Device
        functionString += "#include <linux/i2c.h>\n"
    elif inputParams.deviceType == 2:
        functionString += "#include <linux/of.h>\n"
    functionString += "#include \"custom_functions.h\"\n"
    functionString += "\nMODULE_LICENSE(\"GPL\");\n"
    functionString += "MODULE_AUTHOR(\"Tyler Davis
    ↪  <support@flatearthinc.com\");\n"
    functionString += "MODULE_DESCRIPTION(\"Loadable kernel module for the " +
    ↪  inputParams.deviceName + "\");\n"
    functionString += "MODULE_VERSION(\"1.0\");\n"
    functionString += "/* End CreateFileHeaderInfo */\n\n\n"
    return functionString
```

CreateFunctionPrototypes.py

```
def CreateFunctionPrototypes(inputParams):
    functionString = "/*****************************************************\n"
    functionString += "Generate in CreateFunctionPrototypes\n"
    functionString += "*****************************************************/\n"
    devName = inputParams.deviceName
    functionString += "static int " + devName + "_probe(struct platform_device
    ↪  *pdev);\n"
    functionString += "static int " + devName + "_remove(struct platform_device
    ↪  *pdev);\n"
```

```
8       functionString += "static ssize_t " + devName + "_read(struct file *file,
        ↪  char *buffer, size_t len, loff_t *offset);\n"
9       functionString += "static ssize_t " + devName + "_write(struct file *file,
        ↪  const char *buffer, size_t len, loff_t *offset);\n"
10      functionString += "static int " + devName + "_open(struct inode *inode,
        ↪  struct file *file);\n"
11      functionString += "static int " + devName + "_release(struct inode *inode,
        ↪  struct file *file);\n"
12      functionString += "static ssize_t name_read(struct device *dev, struct
        ↪  device_attribute *attr, char *buf);\n\n"
13      functionString += "/*************** Generate device specific prototypes
        ↪  ********************/\n"
14      if inputParams.deviceType == 0:    # SPI device
15          functionString += CreateSPIAttributePrototypes(inputParams)
16      elif inputParams.deviceType == 1:  # I2c device
17          functionString += CreateI2CAttributePrototypes(inputParams)
18      elif inputParams.deviceType == 2:  # fpga device
19          functionString += CreateFPGAAttributePrototypes(inputParams)
20      functionString += "\n/* Custom function declarations */\n"
21      if not inputParams.deviceType == 2:
22          functionString += "uint8_t find_volume_level(" +
            ↪  inputParams.volumeLevelParams + ");\n"
23          functionString += "uint32_t decode_volume(uint8_t code);\n"
24      functionString += "/* End CreateFunctionPrototypes */\n\n\n"
25      return functionString
26
27
28  def CreateSPIAttributePrototypes(inputParams):
29      functionString = "// SPI Device funcs\n"
30      for i in range(len(inputParams.deviceAttributes)):
31          functionString += "static ssize_t " + inputParams.deviceAttributes[i] + \
32                            "_write (struct device *dev, struct device_attribute
                              ↪  *attr, const char *buf, size_t count);\n"
33          functionString += "static ssize_t " + inputParams.deviceAttributes[i] + \
34                            "_read  (struct device *dev, struct device_attribute
                              ↪  *attr, char *buf);\n"
35      return functionString
36
37
38  def CreateI2CAttributePrototypes(inputParams):
39      functionString = "// I2C Device funcs\n"
40      for i in range(len(inputParams.deviceAttributes)):
41          functionString += "static ssize_t " + inputParams.deviceAttributes[i] + \
42                            "_write (struct device *dev, struct device_attribute
                              ↪  *attr, const char *buf, size_t count);\n"
43          functionString += "static ssize_t " + inputParams.deviceAttributes[i] + \
44                            "_read  (struct device *dev, struct device_attribute
                              ↪  *attr, char *buf);\n"
45      return functionString
46
47
```

```
# in the HAv8 driver these are named attr_show_left/right. im assuming every attr has both a left and a right
def CreateFPGAAttributePrototypes(inputParams):
    functionString = "// FPGA device funcs\n"
    for i in range(len(inputParams.deviceAttributes)):
        functionString += "static ssize_t " + inputParams.deviceAttributes[i] + \
                          "_write (struct device *dev, struct device_attribute *attr, const char *buf, size_t count);\n"
        functionString += "static ssize_t " + inputParams.deviceAttributes[i] + \
                          "_read  (struct device *dev, struct device_attribute *attr, char *buf);\n"
    return functionString
```

CreateInitFunctionFile.py

```
from InputStructureFile import InputStructure


def CreateInitFunction(inputParams):
    if not isinstance(inputParams, InputStructure):
        print("inputParams must be an InputStructure object")
    return CreateInitFunctionHelper(inputParams)


def CreateInitFunctionHelper(inputParams):
    functionString = "/*************************************************************\n"
    functionString += "Generated in CreateInitFunction\n"
    functionString += "*************************************************************/\n"
    functionString += "static int " + inputParams.deviceName + "_init(void) {\n"
    functionString += "  int ret_val = 0;\n"
    functionString += "  printk(KERN_ALERT \"FUNCTION AUTO GENERATED AT: " + inputParams.currTime + "\\n\");\n"
    if inputParams.deviceType == 0:    # SPI device
        functionString += CreateInitFunctionSPI(inputParams)
    elif inputParams.deviceType == 1:  # I2C device
        functionString += CreateInitFunctionI2C(inputParams)
    elif inputParams.deviceType == 2:  # FPGA
        functionString += CreateInitFunctionFPGA(inputParams)
    else:
        print("FAILURE, device type not recognized")
        return "\nINIT NOT CREATED\n\n"
    functionString += "/* End CreateInitFunction */\n\n"
    return functionString


def CreateInitFunctionSPI(inputParams):
    functionString = "  // Add the spi master\n"
    functionString += "  struct spi_master *master;\n"
```

```
33      functionString += "  pr_info(\"Initializing the Flat Earth " +
        ↪  inputParams.deviceName + " module\\n\");\n"
34      functionString += "  // Register our driver with the \"Platform Driver\"
        ↪  bus\n"
35      functionString += "  ret_val = platform_driver_register(&" +
        ↪  inputParams.deviceName + "_platform);\n"
36      functionString += "  if (ret_val != 0) {\n"
37      functionString += "    pr_err(\"platform_driver_register returned %d\\n\",
        ↪  ret_val);\n"
38      functionString += "    return ret_val;\n"
39      functionString += "  }\n"
40      functionString += "  // Register the device\n"
41      functionString += "  struct spi_board_info spi_device_info = {\n"
42      functionString += "    .modalias = \"fe_" + inputParams.deviceName + "_\",\n"
43      functionString += "    .max_speed_hz = " + inputParams.speed + ",\n"
44      functionString += "    .bus_num = 0,\n"
45      functionString += "    .chip_select = " + inputParams.chipSelect + ",\n"
46      functionString += "    .mode = " + inputParams.mode + ",\n"
47      functionString += "  };\n"
48      functionString += "  /* To send data we have to know what spi port/pins
        ↪  should be used. This information\n"
49      functionString += "  can be found in the device-tree. */\n"
50      functionString += "  master = spi_busnum_to_master( spi_device_info.bus_num
        ↪  );\n"
51      functionString += "  if( !master ) {\n"
52      functionString += "    printk(\"MASTER not found.\\n\");\n"
53      functionString += "    return -ENODEV;\n"
54      functionString += "  }\n"
55      functionString += "  // ceate a new slave device, given the master and device
        ↪  info\n"
56      functionString += "  spi_device = spi_new_device(master, &spi_device_info);\n"
57      functionString += "  printk(\"Setting up new slave device\\n\");\n"
58      functionString += "  if (!spi_device) {\n"
59      functionString += "    printk(\"FAILED to create slave.\\n\");\n"
60      functionString += "    return -ENODEV;\n"
61      functionString += "  }\n"
62      functionString += "  printk(\"Set the bits per word\\n\");\n"
63      functionString += "  spi_device->bits_per_word = bits;\n"
64      functionString += "  printk(\"Setting up the device\\n\");\n"
65      functionString += "  ret_val = spi_setup(spi_device);\n"
66      functionString += "  if (ret_val) {\n"
67      functionString += "    printk(\"FAILED to setup slave.\\n\");\n"
68      functionString += "    spi_unregister_device(spi_device);\n"
69      functionString += "    return -ENODEV;\n"
70      functionString += "  }\n"
71      functionString += "  printk(\"Sending SPI initialization commands...\\n\");\n"
72      functionString += "  // Sending init commands\n"
73      functionString += WriteInitCommands(inputParams)
74      functionString += "
        ↪  /*******************************************************/\n"
```

```
75      functionString += "  pr_info(\"Flat Earth " + inputParams.deviceName + "
        ↪  module successfully initialized!\\n\");\n"
76      functionString += "  return 0;\n"
77      functionString += "}\n"
78      return functionString
79
80
81  def CreateInitFunctionI2C(inputParams):
82      functionString = "  struct i2c_adapter *i2c_adapt;\n"
83      functionString += "  struct i2c_board_info i2c_info;\n"
84      functionString += "  pr_info(\"Initializing the Flat Earth " +
        ↪  inputParams.deviceName + " module\\n\");\n"
85      functionString += "  // Register our driver with the \"Platform Driver\"
        ↪  bus\n"
86      functionString += "  ret_val = platform_driver_register(&" +
        ↪  inputParams.deviceName + "_platform);\n"
87      functionString += "  if (ret_val != 0) {\n"
88      functionString += "    pr_err(\"platform_driver_register returned %d\\n\",
        ↪  ret_val);\n"
89      functionString += "    return ret_val;\n"
90      functionString += "  }\n"
91      functionString += "
        ↪  /*-----------------------------------------------------\n"
92      functionString += "     I2C communication\n"
93      functionString += "
        ↪  -----------------------------------------------------*/\n"
94      functionString += "  // Register the device\n"
95      functionString += "  ret_val = i2c_add_driver(&" +
        ↪  inputParams.deviceNameAbbrev + "_i2c_driver);\n"
96      functionString += "  if (ret_val < 0) {\n"
97      functionString += "    pr_err(\"Failed to register I2C driver\");\n"
98      functionString += "    return ret_val;\n"
99      functionString += "  }\n"
100     functionString += "  i2c_adapt = i2c_get_adapter(0);\n"
101     functionString += "  memset(&i2c_info,0,sizeof(struct i2c_board_info));\n"
102     functionString += "  strlcpy(i2c_info.type, \"" +
        ↪  inputParams.deviceNameAbbrev + "_i2c\",I2C_NAME_SIZE);\n"
103     functionString += "  " + inputParams.deviceNameAbbrev + "_i2c_client =
        ↪  i2c_new_device(i2c_adapt,&" + inputParams.deviceNameAbbrev +
        ↪  "_i2c_info);\n"
104     functionString += "  i2c_put_adapter(i2c_adapt);\n"
105     functionString += "  if (!" + inputParams.deviceNameAbbrev + "_i2c_client)
        ↪  {\n"
106     functionString += "    pr_err(\"Failed to connect to I2C client\\n\");\n"
107     functionString += "    ret_val = -ENODEV;\n"
108     functionString += "    return ret_val;\n"
109     functionString += "  }\n"
110     functionString += "  // Send some initialization commands\n"
111     functionString += WriteInitCommands(inputParams)
112     functionString += "  /*****************************************************
        ↪  ******************/\n"
```

```
113     functionString += "  pr_info(\"Flat Earth " + inputParams.deviceName + "
        ↪  module successfully initialized!\\n\");\n"
114     functionString += "  return 0;\n"
115     functionString += "}\n"
116     return functionString
117
118
119 def CreateInitFunctionFPGA(inputParams):
120     functionString = "  pr_info(\"Initializing the Flat Earth " +
        ↪  inputParams.deviceName + " module\\n\");\n"
121     functionString += "  // Register our driver with the \"Platform Driver\"
        ↪  bus\n"
122     functionString += "  ret_val = platform_driver_register(&" +
        ↪  inputParams.deviceName + "_platform);"
123     functionString += "  if (ret_val != 0) {\n"
124     functionString += "    pr_err(\"platform_driver_register returned %d\\n\",
        ↪  ret_val);\n"
125     functionString += "    return ret_val;\n"
126     functionString += "  }\n"
127     functionString += "  pr_info(\"Flat Earth " + inputParams.deviceName + "
        ↪  module successfully initialized!\\n\");\n"
128     functionString += "  return 0;\n"
129     functionString += "}\n"
130     return functionString
131
132
133 def WriteInitCommands(inputParams):
134     functionString = ""
135     if inputParams.initCommLen > 0:
136         functionString += "  char cmd[" + str(inputParams.initCommBytes) + "];\n"
137     for i in range(inputParams.initCommLen):
138         try:
139             functionString += "  // desc: " + inputParams.initCommDesc[i] + "\n"
140         except:
141             functionString += "  // no command description\n"
142         for j in range(inputParams.initCommBytes):
143             functionString += "  cmd[" + str(j) + "] = " +
                ↪  inputParams.initComm[i][j] + ";"
144         functionString += "\n"
145         functionString += "  ret_val = " + inputParams.initCommSendFunc + "(" +
            ↪  inputParams.initCommSendParams + ");\n\n"
146     return functionString
```

CreateMiscTopOfFileStuff.py

```
1  def CreateMiscTopOfFile(inputParams):
2      functionString = "/*********************************************\n"
3      functionString += "  Generated in CreateMiscTopOfFile\n"
4      functionString += "*********************************************/\n"
5      functionString += "struct fixed_num {\n"
```

```
6       functionString += "  int integer;\n"
7       functionString += "  int fraction;\n"
8       functionString += "  int fraction_len;\n"
9       functionString += "};\n"
10      functionString += "static struct class *cl;  // Global variable for the
        ↪  device class\n"
11      functionString += "static dev_t dev_num;\n\n"
12      functionString += "/*********** Device type specific things **************/\n"
13      if inputParams.deviceType == 0:   # SPI Device
14          functionString = TopOfFileSPIStuff(inputParams, functionString)
15      elif inputParams.deviceType == 1: # I2C Device
16          functionString = TopOfFileI2CStuff(inputParams, functionString)
17      elif inputParams.deviceType == 2: # FPGA Device
18          functionString = TopOfFileFPGAStuff(inputParams, functionString)
19      else:
20          print("NOT A SPI I2C OR FPGA DEVICE?")
21          return ""
22      if inputParams.needsVolumeTable:
23          functionString += "\n/* Volume table */\n"
24          functionString = WriteVolumeTable(inputParams, functionString)
25      functionString += "/* End of CreateMiscTopOfFile */\n\n\n"
26      return functionString
27
28
29  def TopOfFileSPIStuff(inputParams, functionString):
30      functionString += "// Define some SPI stuff\n"
31      functionString += "static uint8_t bits = 8;\n"
32      functionString += "static uint32_t speed = " + inputParams.speed + ";\n"
33      functionString += "static struct spi_device *spi_device;\n"
34      return functionString
35
36
37  def TopOfFileI2CStuff(inputParams, functionString):
38      functionString += "// Define some I2C stuff\n"
39      functionString += "struct i2c_driver " + inputParams.deviceNameAbbrev +
        ↪  "_i2c_driver;\n"
40      functionString += "struct i2c_client * " + inputParams.deviceNameAbbrev +
        ↪  "_i2c_client;\n"
41      functionString += "static const unsigned short normal_i2c[] = {0x35,
        ↪  I2C_CLIENT_END}; // remove? -Tyler\n"
42      return functionString
43
44
45  def TopOfFileFPGAStuff(inputParams, functionString):
46      try: # to lookup the pound defined constants from the input params, otherwise
        ↪  use what the HAv8 used and hope that works
47          for i in range(len(inputParams.definedDataConstants)):
48              functionString += "#define " + inputParams.definedDataConstants[i][0]
                ↪  + " " + str(inputParams.definedDataConstants[i][1]) + "\n"
49      except:
```

```
        # These are almost certainly not universal. Should pull these from the params probably
        functionString += "//TODO: check this. Register memory map. Was not pulled from input params, might want to verify this\n"
        functionString += "#define BAND_ALL_OFFSET 0x00\n"
        functionString += "#define BAND1_OFFSET 0x01\n"
        functionString += "#define BAND2_OFFSET 0x02\n"
        functionString += "#define BAND3_OFFSET 0x03\n"
        functionString += "#define BAND4_OFFSET 0x04\n"
        functionString += "#define LEFT_OFFSET 0x00\n"
        functionString += "#define RIGHT_OFFSET 0x08\n"
        functionString += "#define GAIN_OFFSET 0\n"
    return functionString


def WriteVolumeTable(inputParams, functionString):
    functionString += "typedef struct {\n"
    functionString += "  uint16_t value;\n"
    functionString += "  uint8_t  code;\n"
    functionString += "} volumeLevel;\n\n"
    try:
        functionString += "#define PN_INDEX " + str(inputParams.PN_INDEX) + "\n"
    except:
        functionString += "// No PN_INDEX specified in input params\n"
    functionString += "static volumeLevel VolumeLevels[] = {\n"
    for i in range(len(inputParams.volumeTable)):
        functionString += "  {.value = " + str(inputParams.volumeTable[i][0]) + ", .code = " + str(inputParams.volumeTable[i][1]) + "},\n"
    functionString += "};\n"
    return functionString
```

CreateProbeFunction.py

```
def CreateProbeFunction(inputParams):
    functionString = "/************************************************\n"
    functionString += "Generated by CreateProbeFunction\n"
    functionString += "************************************************/\n"
    functionString += "static int " + inputParams.deviceName + "_probe(struct platform_device *pdev) {\n"
    functionString += "  int ret_val = -EBUSY;\n"
    functionString += ("  char deviceName[" + str(len(inputParams.deviceName) + 12) +"] = \"fe_" + inputParams.deviceName
                       + "_\";\n")  # adding 12 to len is arbitrary
    functionString += "  char deviceMinor[20];\n"
    functionString += "  int status;\n"
    functionString += "  struct device *device_obj;\n"
    devpString = "fe_" + inputParams.deviceName + "_devp"
    functionString += "  fe_" + inputParams.deviceName + "_dev_t * " + devpString + ";\n"
    functionString += "  struct resource *r = NULL;\n"
```

```
15      functionString += "  pr_info(\"" + inputParams.deviceName + "_probe
        ↪  enter\\n\");\n"
16      if inputParams.deviceType == 2:  # FPGA device
17          functionString += "  r = platform_get_resource(pdev, IORESOURCE_MEM,
            ↪  0);\n"
18          functionString += "  if (r == NULL) {\n"
19          functionString += "    pr_err(\"IORESOURCE_MEM (register space) does not
            ↪  exist\\n\");\n"
20          functionString += "    goto bad_exit_return;"
21          functionString += "  }\n"
22      functionString += ("  " + devpString + " = devm_kzalloc(&pdev->dev,
        ↪  sizeof(fe_" + inputParams.deviceName +
23                         "_dev_t), GFP_KERNEL);\n")
24      if inputParams.deviceType == 2:  # FPGA device type
25          functionString += "  " + devpString + "->regs =
            ↪  devm_ioremap_resource(&pdev->dev, r);\n"
26          functionString += "  if (IS_ERR(" + devpString + "->regs))\n"
27          functionString += "    goto bad_ioremap;\n"
28      functionString += "  platform_set_drvdata(pdev, (void *)" + devpString +
        ↪  ");\n"
29      functionString += "  " + devpString + "->name = devm_kzalloc(&pdev->dev, 50,
        ↪  GFP_KERNEL);\n"
30      functionString += "  if (" + devpString + "->name == NULL)\n"
31      functionString += "    goto bad_mem_alloc;\n"
32      functionString += "  strcpy(" + devpString + "->name, (char *)pdev->name);\n"
33      functionString += "  pr_info(\"%s\\n\", (char *)pdev->name);\n"
34      functionString += "  status = alloc_chrdev_region(&dev_num, 0, 1, \"fe_" +
        ↪  inputParams.deviceName + "_\");\n"
35      functionString += "  if (status != 0)\n"
36      functionString += "    goto bad_alloc_chrdev_region;\n"
37      functionString += "  sprintf(deviceMinor, \"%d\", MAJOR(dev_num));\n"
38      functionString += "  strcat(deviceName, deviceMinor);\n"
39      functionString += "  pr_info(\"%s\\n\", deviceName);\n"
40      functionString += "  cl = class_create(THIS_MODULE, deviceName);\n"
41      functionString += "  if (cl == NULL)\n"
42      functionString += "    goto bad_class_create;\n"
43      functionString += "  cdev_init(&" + devpString + "->cdev, &fe_" +
        ↪  inputParams.deviceName + "_fops);\n"
44      functionString += "  status = cdev_add(&" + devpString + "->cdev, dev_num,
        ↪  1);\n"
45      functionString += "  if (status != 0)\n"
46      functionString += "    goto bad_cdev_add;\n"
47      functionString += "  device_obj = device_create(cl, NULL, dev_num, NULL,
        ↪  deviceName);\n"
48      functionString += "  if (device_obj == NULL)\n"
49      functionString += "    goto bad_device_create;\n"
50      functionString += "  dev_set_drvdata(device_obj, " + devpString + ");\n"
51      functionString += CreateAttributeFilesAndErrorHandling(inputParams)
52      functionString += "/* End of CreateProbeFunction */\n\n\n"
53      return functionString
54
```

```
55
56  def CreateAttributeFilesAndErrorHandling(inputParams):
57      functionString = "/* Beginning attribute file stuff */\n"
58      for i in range(len(inputParams.deviceAttributes)):
59          functionString += "  status = device_create_file(device_obj, &dev_attr_"
            ↪  + inputParams.deviceAttributes[i] + ");\n"
60          functionString += "  if (status)\n"
61          functionString += "    goto bad_device_create_file_" + str(i) + ";\n"
62          functionString += "\n"
63      functionString += "  status = device_create_file(device_obj,
        ↪  &dev_attr_name);\n"
64      functionString += "  if (status)\n"
65      functionString += "    goto bad_device_create_file_" +
        ↪  str(len(inputParams.deviceAttributes) + 1) + ";\n"
66      functionString += "  pr_info(\"HA_probe exit\\n\");\n"
67      functionString += "  return 0;\n"
68      functionString += "bad_device_create_file_" +
        ↪  str(len(inputParams.deviceAttributes) + 1) + ":\n"
69      functionString += "  device_remove_file(device_obj, &dev_attr_name);\n"
70      for i in range(len(inputParams.deviceAttributes) - 1, -1, -1):
71          functionString += "bad_device_create_file_" + str(i) + ":\n"
72          functionString += "  device_remove_file(device_obj, &dev_attr_" +
            ↪  inputParams.deviceAttributes[i] + ");\n\n"
73      functionString += "bad_device_create:\n"
74      if inputParams.deviceType == 2:
75          functionString += "  device_destroy(cl, dev_num);\n\n"
76      functionString += "  cdev_del(&fe_" + inputParams.deviceName +
        ↪  "_devp->cdev);\n"
77      functionString += "bad_cdev_add:\n"
78      functionString += "  class_destroy(cl);\n"
79      functionString += "bad_class_create:\n"
80      functionString += "  unregister_chrdev_region(dev_num, 1);\n"
81      functionString += "bad_alloc_chrdev_region:\n"
82      functionString += "bad_mem_alloc:\n"
83      if inputParams.deviceType == 2: # FPGA
84          functionString += "bad_ioremap:\n"
85          functionString += "  ret_val = PTR_ERR(fe_" + inputParams.deviceName +
            ↪  "_devp->regs);\n"
86          functionString += "bad_exit_return:\n"
87          functionString += "  pr_info(\"" + inputParams.deviceName + "_probe bad
            ↪  exit\\n\");\n"
88      functionString += "  return ret_val;\n"
89      functionString += "}\n"
90      return functionString
```

CreateReadAndWriteFuncs.py

```
1   def CreateAttrReadWriteFuncs(inputParams):
2       functionString = "/*****************************************************\n"
3       functionString += "Generated by CreateAttrReadWriteFuncs\n"
```

```
    functionString += "*****************************************************/\n"
    if inputParams.deviceType == 0:    # SPI device
        functionString += CreateSPIAttrReadWrite(inputParams)
    elif inputParams.deviceType == 1:  # I2C device
        functionString += CreateI2CAttrReadWrite(inputParams)
    elif inputParams.deviceType == 2:  # FPGA device
        functionString += CreateFPGAAttrReadWrite(inputParams)
    functionString += "/* End of CreateAttrReadWriteFuncs */\n\n\n"
    return functionString


def CreateSPIAttrReadWrite(inputParams):
    functionString = "// SPI Attribute functions\n"
    for i in range(len(inputParams.deviceAttributes)):
        functionString += "static ssize_t " + inputParams.deviceAttributes[i] +
        ↪  "_read(struct device *dev, struct device_attribute *attr, char *buf)
        ↪  {\n"
        if inputParams.attributeReadIsNormal[i]:
            functionString += "  fe_" + inputParams.deviceName + "_dev_t * devp =
            ↪  (fe_" + inputParams.deviceName + "_dev_t
            ↪  *)dev_get_drvdata(dev);\n"
            functionString += "  fp_to_string(buf, devp->" +
            ↪  inputParams.deviceAttributes[i] + ", 16, true, 9);\n"
            functionString += "  strcat2(buf,\"\\n\");\n"
            functionString += "  return strlen(buf);\n"
        else:
            functionString += "  // TODO: fill me in\n"
            functionString += "  return -1;\n"
        functionString += "}\n\n"
        functionString += "static ssize_t " + inputParams.deviceAttributes[i] +
        ↪  "_write(struct device *dev, struct device_attribute *attr, const char
        ↪  *buf, size_t count) {\n"
        if inputParams.attributeWriteIsNormal[i]:
            functionString += CreateSPIWriteFunc(inputParams, i)
        else:
            functionString += "  // TODO: Fill this in\n"
            functionString += "  return 0;\n"
        functionString += "}\n\n"
    functionString += "static ssize_t name_read(struct device *dev, struct
    ↪  device_attribute *attr, char *buf) {\n"
    functionString += "  // TODO: Fill this in\n"
    functionString += "  return 0;\n"
    functionString += "}\n"
    return functionString


def CreateI2CAttrReadWrite(inputParams):
    functionString = "// I2C Attribute functions\n"
    for i in range(len(inputParams.deviceAttributes)):
```

```
45          functionString += "static ssize_t " + inputParams.deviceAttributes[i] +
            ↪  "_read(struct device *dev, struct device_attribute *attr, char *buf)
            ↪  {\n"
46          if inputParams.attributeReadIsNormal[i]:
47              functionString += "  fe_" + inputParams.deviceName + "_dev_t * devp =
                ↪  (fe_" + inputParams.deviceName + "_dev_t
                ↪  *)dev_get_drvdata(dev);\n"
48              functionString += "  fp_to_string(buf, devp->" +
                ↪  inputParams.deviceAttributes[i] + ", 16, true, 9);\n"
49              functionString += "  strcat2(buf,\"\\n\");\n"
50              functionString += "  return strlen(buf);\n"
51          else:
52              functionString += "  // TODO: fill me in\n"
53              functionString += "  return -1;\n"
54          functionString += "}\n\n"
55          functionString += "static ssize_t " + inputParams.deviceAttributes[i] +
            ↪  "_write(struct device *dev, struct device_attribute *attr, const char
            ↪  *buf, size_t count) {\n"
56          if inputParams.attributeWriteIsNormal[i]:
57              functionString += CreateI2CWriteFunc(inputParams, i)
58          else:
59              functionString += "  // TODO: Fill this in\n"
60              functionString += "  return 0;\n"
61          functionString += "}\n\n"
62      functionString += "static ssize_t name_read(struct device *dev, struct
        ↪  device_attribute *attr, char *buf) {\n"
63      functionString += "  // TODO: Fill this in\n"
64      functionString += "  return 0;\n"
65      functionString += "}\n"
66      return functionString
67
68
69  def CreateFPGAAttrReadWrite(inputParams):
70      functionString = "// FPGA Attribute functions\n"
71      for i in range(len(inputParams.deviceAttributes)):
72          functionString += "static ssize_t " + inputParams.deviceAttributes[i] +
            ↪  "_read(struct device *dev, struct device_attribute *attr, char *buf)
            ↪  {\n"
73          if inputParams.attributeReadIsNormal[i]:
74              functionString += "  fe_" + inputParams.deviceName + "_dev_t * devp =
                ↪  (fe_" + inputParams.deviceName + "_dev_t
                ↪  *)dev_get_drvdata(dev);\n"
75              # TODO: add support for int, bool, etc.
76              # TODO: fix parsing of fractional bits; this code currently doesn't
                ↪  return the correct part of the string. we should maybe do the
                ↪  parsing elsewhere and have a field in inputParams for fractional
                ↪  bits
77              functionString += "  fp_to_string(buf, devp->" +
                ↪  inputParams.deviceAttributes[i] + \
78                  ", " + str(inputParams.attributeDataTypeWidth[i]) + ", " + \
79                  str(inputParams.attributeDataTypeSigned[i]).lower()+ ", " + \
```

```
80                  str(inputParams.attributeDataTypeFraction[i]) + ");\n"
81              functionString += "  strcat2(buf,\"\\n\");\n"
82              functionString += "  return strlen(buf);\n"
83          else:
84              functionString += "  // TODO: fill me in\n"
85              functionString += "  return -1;\n"
86          functionString += "}\n\n"
87          functionString += "static ssize_t " + inputParams.deviceAttributes[i] +
            ↪  "_write(struct device *dev, struct device_attribute *attr, const char
            ↪  *buf, size_t count) {\n"
88          if inputParams.attributeWriteIsNormal[i]:
89              functionString += CreateFPGAWriteFunc(inputParams, i)
90          else:
91              functionString += "  // TODO: Fill this in\n"
92              functionString += "  return 0;\n"
93          functionString += "}\n\n"
94      functionString += "static ssize_t name_read(struct device *dev, struct
        ↪  device_attribute *attr, char *buf) {\n"
95      functionString += "  fe_" + inputParams.deviceName + "_dev_t *devp = (fe_" +
        ↪  inputParams.deviceName + "_dev_t *)dev_get_drvdata(dev);\n"
96      functionString += "  sprintf(buf, \"%s\\n\", devp->name);\n"
97      functionString += "  return strlen(buf);\n"
98      functionString += "}\n"
99      return functionString
100
101
102 def CreateSPIWriteFunc(inputParams, i):
103     functionString = "  uint32_t tempValue = 0;\n"
104     functionString += "  uint8_t volume_level;\n"
105     functionString += "  char cmd[" + str(inputParams.attrWriteCommBytes) + "] =
        ↪  {"
106     for j in range(inputParams.attrWriteCommBytes):
107         functionString += " " + inputParams.attrWriteComm[i][j]
108         if j != inputParams.attrWriteCommBytes - 1:
109             functionString += ","
110     functionString += "};\n"
111     functionString += "  char substring[80];\n"
112     functionString += "  int substring_count = 0;\n"
113     functionString += "  int i;\n"
114     functionString += "  fe_" + inputParams.deviceName + "_dev_t * devp = (fe_" +
        ↪  inputParams.deviceName + "_dev_t *) dev_get_drvdata(dev);\n"
115     functionString += "  for (i = 0; i < count; i++) {\n"
116     functionString += "    if ((buf[i] != ',') && (buf[i] != ' ') && (buf[i] !=
        ↪  '\\0') && (buf[i] != '\\r') && (buf[i] != '\\n')) {\n"
117     functionString += "      substring[substring_count] = buf[i];\n"
118     functionString += "      substring_count ++;\n"
119     functionString += "    }\n"
120     functionString += "  }\n"
121     functionString += "  substring[substring_count] = '\\0';\n"
122     functionString += "  if (buf[0] == '-') {\n"
123     functionString += "    for (i = 0; i < 79; i++) {\n"
```

```
    functionString += "      substring[i] = substring[i + 1];\n"
    functionString += "    }\n"
    functionString += "  }\n"
    functionString += "  tempValue = set_fixed_num(substring, 28, true);\n"
    functionString += "  volume_level = find_volume_level(tempValue);\n"
    functionString += "  tempValue = decode_volume(volume_level);\n"
    functionString += "  devp->" + inputParams.deviceAttributes[i] + " = -1 *
    ↪  tempValue;\n"
    functionString += "  cmd[" + str(inputParams.attrWriteCommBytes - 1) + "] =
    ↪  volume_level;\n"
    functionString += "  spi_write(spi_device, &cmd, sizeof(cmd));\n"
    functionString += "  return count;\n"
    return functionString


def CreateI2CWriteFunc(inputParams, i):
    functionString =  "  uint32_t tempValue = 0;\n"
    functionString += "  char substring[80];\n"
    functionString += "  int substring_count = 0;\n"
    functionString += "  int i;\n"
    functionString += "  char cmd[" + str(inputParams.attrWriteCommBytes) + "] =
    ↪  {"
    for j in range(inputParams.attrWriteCommBytes):
        functionString += " " + inputParams.attrWriteComm[i][j]
        if j != inputParams.attrWriteCommBytes - 1:
            functionString += ","
    functionString += "};\n"
    functionString += "  uint8_t code = 0x00;\n"
    functionString += "  fe_" + inputParams.deviceName + "_dev_t * devp = (fe_" +
    ↪  inputParams.deviceName + \
                      "_dev_t *) dev_get_drvdata(dev);\n"
    functionString += "  for (i = 0; i < count; i++) {\n"
    functionString += "    if ((buf[i] != ',') && (buf[i] != ' ') && (buf[i] !=
    ↪  '\\0') && (buf[i] != '\\r') && (buf[i] != '\\n')) {\n"
    functionString += "      substring[substring_count] = buf[i];\n"
    functionString += "      substring_count ++;\n"
    functionString += "    }\n"
    functionString += "  }\n"
    functionString += "  substring[substring_count] = '\\0';\n"
    functionString += "  if (buf[0] == '-') {\n"
    functionString += "    for (i = 0; i < 79; i++) {\n"
    functionString += "      substring[i] = substring[i + 1];\n"
    functionString += "    }\n"
    functionString += "    tempValue = set_fixed_num(substring, 16, true);\n"
    functionString += "    code = find_volume_level(tempValue, 0);\n"
    functionString += "  } else {\n"
    functionString += "    tempValue = set_fixed_num(substring, 16, true);\n"
    functionString += "    code = find_volume_level(tempValue, 1);\n"
    functionString += "  }\n"
    functionString += "  tempValue = decode_volume(code);\n"
```

```
    functionString += "  devp->" + inputParams.deviceAttributes[i] + " = tempValue;\n"
    functionString += "  cmd[" + str(inputParams.attrWriteCommBytes - 1) + "] = code;\n"
    functionString += "  i2c_master_send(" + inputParams.deviceNameAbbrev + "_i2c_client, &cmd[0], " + str(inputParams.attrWriteCommBytes) + ");\n"
    functionString += "  return count;\n"
    return functionString


def CreateFPGAWriteFunc(inputParams, i):
    functionString =  "  uint32_t tempValue = 0;\n"
    functionString += "  char substring[80];\n"
    functionString += "  int substring_count = 0;\n"
    functionString += "  int i;\n"
    functionString += "  fe_" + inputParams.deviceName + "_dev_t *devp = (fe_" + inputParams.deviceName + \
                      "_dev_t *)dev_get_drvdata(dev);\n"
    functionString += "  for (i = 0; i < count; i++) {\n"
    functionString += "    if ((buf[i] != ',') && (buf[i] != ' ') && (buf[i] != '\\0') && (buf[i] != '\\r') && (buf[i] != '\\n')) {\n"
    functionString += "      substring[substring_count] = buf[i];\n"
    functionString += "      substring_count ++;\n"
    functionString += "    }\n"
    functionString += "  }\n"
    functionString += "  substring[substring_count] = 0;\n"
    is_signed = inputParams.attributeDataTypeSigned[i]
    functionString += "  tempValue = set_fixed_num(substring, " + \
        str(inputParams.attributeDataTypeWidth[i]) + ", " + (str(is_signed)).lower() + ");\n"
    functionString += "  devp->" + inputParams.deviceAttributes[i] + " = tempValue;\n"
    functionString += "  iowrite32(devp->" + inputParams.deviceAttributes[i] + ", (u32 *)devp->regs"
    for j in range(len(inputParams.attributeWriteOffsets[i])):
        functionString += " + " + inputParams.attributeWriteOffsets[i][j]
    functionString += ");\n"
    functionString += "  return count;\n"
    return functionString
```

CreateRemoveFunction.py

```
def CreateRemoveFunction(inputParams):
    functionString =  "/*******************************************************\n"
    functionString += "Generated in CreateRemoveFunction\n"
    functionString += "*******************************************************/\n"
    functionString += "static int " + inputParams.deviceName + "_remove(struct platform_device *pdev) {\n"
    functionString += "  fe_" + inputParams.deviceName + "_dev_t *dev = (fe_" + inputParams.deviceName + "_dev_t *)platform_get_drvdata(pdev);\n"
```

```
    functionString += "  pr_info(\"" + inputParams.deviceName + "_remove
    ↪  enter\\n\");\n"
    functionString += "  cdev_del(&dev->cdev);\n"
    functionString += "  unregister_chrdev_region(dev_num, 2);\n"
    if inputParams.deviceType == 2:  # FPGA device
        functionString += "  iounmap(dev->regs);\n"
    functionString += "  pr_info(\"" + inputParams.deviceName + "_remove
    ↪  exit\\n\");\n"
    functionString += "  return 0;\n"
    functionString += "}\n"
    functionString += "/* End CreateRemoveFunction */\n\n\n"
    return functionString
```

FileDriver.py

```
from FileOutput import WriteDriver
from InputStructureFile import InputStructure
from ExampleInputStructs import AD1939inputStruct
from ExampleInputStructs import TPAinputStruct
from ExampleInputStructs import HAinputStruct
from CreateCustomFunctions import CreateCustomFunctions


WriteDriver("ADfile.c", AD1939inputStruct, overwrite=True)
WriteDriver("TPAfile.c", TPAinputStruct, overwrite=True)
WriteDriver("HAfile.c", HAinputStruct, overwrite=True)
```

FileOutput.py

```
from CreateInitFunctionFile import CreateInitFunction
from CreateFileHeaderInfo import CreateFileHeaderInfo
from CreateFunctionPrototypes import CreateFunctionPrototypes
from CreateCFunctionStubs import CreateCFunctionStubs
from CreateMiscTopOfFileStuff import CreateMiscTopOfFile
from CreateDeviceAttrMacros import WriteDeviceAttributes
from CreateDriverStuff import CreateDriverStuff
from CreateProbeFunction import CreateProbeFunction
from CreateReadAndWriteFuncs import CreateAttrReadWriteFuncs
from CreateCustomFunctions import CreateCustomFunctions
from CreateEndOfFileStuff import CreateEndOfFileStuff
from CreateRemoveFunction import CreateRemoveFunction
from CreateExitFunction import CreateExitFunction
import os.path
from time import strftime


def WriteDriver(fileName, inputParams):
    timeString = strftime("%Y-%m-%d %H:%M")
```

```
    inputParams.currTime = timeString

    output = open(fileName, "w")
    output.write(  CreateFileHeaderInfo(inputParams)      )
    output.write(  CreateMiscTopOfFile(inputParams)       )
    output.write(  CreateFunctionPrototypes(inputParams)  )
    output.write(  WriteDeviceAttributes(inputParams)     )
    output.write(  CreateDriverStuff(inputParams)         )
    output.write(  CreateInitFunction(inputParams)        )
    output.write(  CreateProbeFunction(inputParams)       )
    output.write(  CreateAttrReadWriteFuncs(inputParams)  )
    output.write(  CreateCFunctionStubs(inputParams)      )
    output.write(  CreateRemoveFunction(inputParams)      )
    output.write(  CreateExitFunction(inputParams)        )
    output.write(  CreateCustomFunctions(inputParams)     )
    output.write(  CreateEndOfFileStuff(inputParams)      )

    print("Keep in mind custom_functions.h should be included when you compile
    ↪  the driver.")
```

InputStructureFile.py

```
class InputStructure:
    def __init__(self):

        # device specifications
        self.deviceName = None
        self.deviceType = None  # 0 for spi, 1 for i2c, 2 for memory mapped device
        self.deviceNameAbbrev = None  # some vars only have an abbrev of the
        ↪  device name, i.e. tpa instead of TPA6130A2
        self.compatibleFlag = None  # something like "dev,fe-AD1939" gets
        ↪  inserted into .compatible
        self.deviceID = None  # An address(?), only used i think for i2c

        # device attribute specifications
        self.deviceAttributes = None  # list of string attributes
        self.attributeDataTypes = None  # list of string data types
        self.attributeDataTypeSigned = None # list of signed/unsigned
        self.attributeDataTypeFraction = None # list of fraction widths
        self.attributeDataTypeWidth = None # list of fraction widths
        self.attributePerms = None  # attribute permissions, list of "0446" or
        ↪  whatever the desired permission is
        #                             for the attribute at the same index
        self.attributeReadIsNormal = None  # is a list of bool vals that informs
        ↪  whether to insert a stub or definition
        self.attributeWriteIsNormal = None  # same as above but for writes
        self.attrWriteCommBytes = None  # number of bytes in a command. Should be
        ↪  same as len(self.attrWritComm[0])
        self.attrWriteComm = None  # of the form [ ["0xNN", "0xNN", ..] ..] for
        ↪  however many bytes a command contains,
```

```
23          #                           and however many attributes there are. They
            ↪  are matched according to index
24          self.attributeWriteOffsets = None  # used for fpga's only i think. should
            ↪  be of the form [ [ "OFFSET_N", "OFFSET_N" ..] ..]
25          #                           for each attribute it should be the defined
            ↪  constants which are added to devp->regs
26
27          # initial command information
28          self.initComm = None  # initial commands that get run in init. 2d list [
            ↪  ["0xNN", "0xNN", ..] ..]
29          self.initCommBytes = None  # number of bytes in a initialization command.
            ↪  Should be same as len(initComm[0])
30          self.initCommLen = None  # number of initialization commands. Should be
            ↪  same as len(initComm)
31          self.initCommSendFunc = None  # function that sends each command,
            ↪  spi_master_write, for instance
32          self.initCommSendParams = None  # parameters that should be inserted for
            ↪  the commSendFunc. ie "dev,cmd,sizeof(cmd)"
33
34          # top of file stuff
35          self.needsVolumeTable = None  # true if the device needs a volume table
36          self.volumeTable = None  # should be filled in with [ [ <val>, "0xNN"] ..
            ↪  ] in decreasing order of val
37          self.PN_INDEX = None  # index where the val goes from positive to negative
38
39          # stuff
40          self.speed = 500000  # not even sure what this is
41          self.mode = None  # rising edge or whatever
42          self.chipSelect = None  # not even sure what this is either
43
44
45          # min and max for each attribute
46          # look for data type var
```

JsonToDriver.py

```
1  from FileOutput import WriteDriver
2  from AvalonJsonParser import AvalonJsonParse
3
4
5  def JsonToDriver(inFileName, outFileName):
6    inputStructure = AvalonJsonParse(inFileName)
7    WriteDriver(outFileName, inputStructure)
```

genDeviceDriver.m

```
1  % genDeviceDriver Generate a device driver from a json input file
2  %
```

```
% genDeviceDriver(infile, outfile)
%
% The json file this function expects has a specific format that contains information about the vhdl entity and
% the Avalon streaming interface parameters. This will often be generated by MATLAB/Simulink, but it can also be written
% by hand.
%
% Inputs:
%   infile = the json input filename
%   outfile = the vhdl output filename

% Copyright 2019 Flat Earth Inc, Montana State University
%
% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
% INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
% PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
% FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
% ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
%
% Trevor Vannoy
% Flat Earth Inc
% 985 Technology Blvd
% Bozeman, MT 59718
% support@flatearthinc.com

function genDeviceDriver(infile, outfile)

% call the python file that autogens the code
py.JsonToDriver.JsonToDriver(infile, outfile)
```

genMakefile.m

```
% genMakefile Generate Makefile and Kbuild for compiling kernel modules
%
% genMakefile(modulePath, modelName)
%
% In order for this function to be platform-independent, the path has to include
% the proper trailing slash for your operating system. The easiest way to do this
% is with the _filesep_ command.
%
% Inputs:
%   modulePath = path to where the kernel module is located
%   modelName = name of the kernel module, without any file extensions

```

```
% Copyright 2020 Flat Earth Inc, Montana State University
%
% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
% INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
% PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
% FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
% ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
%
% Trevor Vannoy
% Flat Earth Inc
% 985 Technology Blvd
% Bozeman, MT 59718
% support@flatearthinc.com

function genMakefile(modulePath, modelName)
py.gen_makefile.main(modulePath, modelName)
```

gen_makefile.py

```
#!/usr/bin/python

# @file gen_makefile.py
#
#     Python function to auto generate files used to build kernel modules
#
#     @author Trevor Vannoy
#     @date 2020
#     @copyright 2020 Flat Earth Inc
#
#     THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
#     INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
#     PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
#     FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
#     ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
#     Trevor Vannoy
#     Flat Earth Inc
#     985 Technology Blvd
#     Bozeman, MT 59718
#     support@flatearthinc.com
```

```
22
23  import argparse
24
25  def create_kbuild(path, model_name):
26      """
27      Create Kbuild file for kernel module compilation.
28
29      This creates a minimal Kbuild file so we can build our
30      autogenerated kernel modules. Our kernel modules rely on
31      a separate header file that defines fixed-point/string
32      conversion functions, so we include that directory in the
33      include search path. This function writes the Kbuild file to
34      the kernel module directory.
35
36      Inputs:
37          path = path, including trailing slash, to the kernel module location
38          model_name = name of the kernel module source file (without file
    ↪   extension)
39      """
40      # by convention, all of our repositories get cloned into the same
41      # root folder; this relative paths reflect that setup, so as long
42      # the convention is followed, this path will work
43      INCLUDE_PATH = '../../../../../component_library/include'
44
45      output = 'ccflags-y := -I$(src)/' + INCLUDE_PATH + '\n'
46      output += 'obj-m := ' + model_name + '.o\n'
47
48      with open(path + 'Kbuild', 'w') as outfile:
49          outfile.write(output)
50
51
52  def create_makefile(path):
53      """
54      Create Makefile for kernel module compilation.
55
56      This function writes the Makefile to the kernel module directory.
57
58      Inputs:
59          path = path, including trailing slash, to the kernel module location
60      """
61      # by convention, all of our repositories get cloned into the same
62      # root folder; this relative paths reflect that setup, so as long
63      # the convention is followed, this path will work
64      KERNEL_SOURCE_PATH = '../../../../../linux-socfpga'
65
66      output = 'KDIR ?= ' + KERNEL_SOURCE_PATH + '\n'
67      output += 'default:\n'
68      output += '\t$(MAKE) -C $(KDIR) ARCH=arm M=$(CURDIR)\n'
69      output += 'clean:\n'
70      output += '\t$(MAKE) -C $(KDIR) ARCH=arm M=$(CURDIR) clean\n'
71      output += 'help:\n'
```

```
    output += '\t$(MAKE) -C $(KDIR) ARCH=arm M=$(CURDIR) help\n'

    with open(path + 'Makefile', 'w') as outfile:
        outfile.write(output)


def parseargs():
    """
    Parse commandline input arguments.
    """
    parser = argparse.ArgumentParser(description=\
        "Generate files used for building kernel modules.")
    parser.add_argument('path',
        help="path where kernel module lives; must contain trailing slash for
        ↪  your operating system!")
    parser.add_argument('model_name',
        help="name of kernel module; e.g. <model_name>.c")
    args = parser.parse_args()
    return (args.path, args.model_name)


# NOTE: path MUST contain a trailing slash so this code can be platform
↪  independent
def main(path, model_name):
    """
    Run the program and build files needed for building a kernel module

    Inputs:
        path = path, including trailing slash, to the kernel module location
        model_name = name of the kernel module source file (without file
↪  extension)
    """
    create_kbuild(path, model_name)
    create_makefile(path)


if __name__ == '__main__':
    (path, model_name) = parseargs()
    main(path, model_name)
```

APPENDIX D

USER INTERFACE AUTOGENERATION SCRIPTS

combine_linker_configs.py

```
#!/usr/bin/python

# @file combine_linker_configs.py
#
#     Script to combine multiple linker json files into one file
#
#     @author Trevor Vannoy
#     @date 2020
#     @copyright 2020 Flat Earth Inc
#
#     THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
#     INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
#     PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
#     FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
#     ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
#
#     Trevor Vannoy
#     Flat Earth Inc
#     985 Technology Blvd
#     Bozeman, MT 59718
#     support@flatearthinc.com

import json
import argparse

def main(configs, outfile='Linker.json', verbose=False):
    """
    Combine multiple linker json files into one.

    Each linker json file is an object of one or more objects
    representing a device driver interface for a given IP core.
    This function combines each of these objects such that the output
    is an object containing all of the device driver interface objects.

    Inputs:
        configs = list of linker json filenames
        outfile = name of the output linker json file
        verbose = verbose printing
    """
    combined = {}

    for config in configs:
        with open(config) as fd:
            config_dict = json.load(fd)
```

```
45              combined.update(config_dict)
46
47      if verbose:
48          print('\n')
49          print(json.dumps(combined, indent=4, sort_keys=True))
50
51      if verbose:
52          print('\n')
53          print('wrote new linker file: ' + outfile)
54      json.dump(combined, open(outfile, 'w'), indent=4, sort_keys=True)
55
56
57  def parseargs():
58      """
59      Parse commandline input arguments.
60      """
61      parser = argparse.ArgumentParser(description=\
62          "Combine multiple linker json files into one")
63      parser.add_argument('configs',
64          help="list of linker files to combine", nargs='+')
65      parser.add_argument('-o', '--outfile',
66          help="name of combined linker json file", default='Linker.json')
67      parser.add_argument('-v', '--verbose', action='store_true',
68          help="verbose output")
69      args = parser.parse_args()
70      return (args.configs, args.outfile, args.verbose)
71
72
73  if __name__ == "__main__":
74      (configs, outfile, verbose) = parseargs()
75      main(configs, outfile, verbose)
```

combine_ui_configs.py

```
1   #!/usr/bin/python
2
3   # @file combine_ui_configs.py
4   #
5   #   Script to combine multiple UI json files into one file
6   #
7   #   @author Trevor Vannoy
8   #   @date 2020
9   #   @copyright 2020 Flat Earth Inc
10  #
11  #   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    ↪  IMPLIED,
12  #   INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
    ↪  PARTICULAR
13  #   PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
    ↪  HOLDERS BE LIABLE
```

```
14  #   FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
    ↪  TORT OR OTHERWISE,
15  #   ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
    ↪  DEALINGS IN THE SOFTWARE.
16  #
17  #   Trevor Vannoy
18  #   Flat Earth Inc
19  #   985 Technology Blvd
20  #   Bozeman, MT 59718
21  #   support@flatearthinc.com
22
23  import json
24  import argparse
25
26  def main(configs, module_name, outfile='UI.json', verbose=False):
27      """
28      Combine multiple UI json files into one.
29
30      Each UI json file contains a list of pages, each of which has a name and a
    ↪  list panels.
31      Each panel has a name and a list of controls. This function merges pages,
    ↪  panels, and
32      controls from multiple UI json files into one json file that can be used to
    ↪  configure
33      a GUI with the UI elements from each UI json file.
34
35      Inputs:
36          configs     = list of linker json filenames
37          module_name = UI module name (what the UI represents), e.g. hearing aid
38          outfile     = name of the output linker json file
39          verbose     = verbose printing
40      """
41      combined = {'module': module_name, 'pages': []}
42
43      # lambda functions to get lists of page and panel names
44      page_names = lambda: [combined['pages'][i]['name'] for i in
        ↪  range(len(combined['pages']))]
45      panel_names = lambda m :[combined['pages'][m]['panels'] for i in
        ↪  range(len(combined['pages'][m]['panels']))]
46
47      for config in configs:
48          with open(config) as fd:
49              config_dict = json.load(fd)
50              for page in config_dict['pages']:
51                  try:
52                      # check if the page exists in the combined config
53                      page_idx = page_names().index(page['name'])
54
55                      # page exists, so loop through all panels in the page
56                      for panel in page['panels']:
57                          try:
```

```
58                              # check if panel exists in the combined config
59                              panel_idx = panel_names(page_idx).index(panel['name'])
60
61                              # panel already exists, so add register controls to
                                ↪  the controls list
62                              combined['pages'][page_idx]['panels'][panel_idx]['con
                                ↪  trols'].extend(panel['controls'])
63                              if verbose:
64                                  print('adding controls to panel
                                    ↪  {}'.format(panel['name']))
65                          except ValueError:
66                              # panel isn't in the page, so add it
67                              combined['pages'][page_idx]['panels'].append(panel)
68                              if verbose:
69                                  print('adding panel {}'.format(panel['name']))
70                  except ValueError:
71                      # page isn't in the combined dictionary, so add it
72                      combined['pages'].append(page)
73                      if verbose:
74                          print('adding page {}'.format(page['name']))
75
76      if verbose:
77          print('\n')
78          print(json.dumps(combined, indent=4, sort_keys=True))
79
80      if verbose:
81          print('\n')
82          print('wrote new UI file: ' + outfile)
83      json.dump(combined, open(outfile, 'w'), indent=4, sort_keys=True)
84
85
86  def parseargs():
87      """
88      Parse commandline input arguments.
89      """
90      parser = argparse.ArgumentParser(description=\
91          "Combine multiple UI json files into one")
92      parser.add_argument('module_name',
93          help="what the UI represents (e.g. hearing aid, multi-effects, etc.)")
94      parser.add_argument('configs',
95          help="list of UI files to combine", nargs='+')
96      parser.add_argument('-o', '--outfile',
97          help="name of combined UI json file", default='UI.json')
98      parser.add_argument('-v', '--verbose', action='store_true',
99          help="verbose output")
100     args = parser.parse_args()
101     return (args.configs, args.module_name, args.outfile, args.verbose)
102
103
104 if __name__ == "__main__":
105     (configs, module_name, outfile, verbose) = parseargs()
```

```
106     main(configs, module_name, outfile, verbose)
```

createLinkerWidgetNames.m

```
1   % createLinkerWidgetNames Create the widget names used in the UI and linker
    ↪  configs for each register
2   %
3   % mp = createLinkerWidgetNames(mp)
4   %
5   % To avoid name conflicts between different devices/models, the widget names
6   % are of the form <ui element type><#><model name>, e.g. toggle1flanger,
    ↪  slider3bitcrusher, etc.
7   %
8   % Inputs:
9   %   mp = simulink model parameters struct
10  % Outputs:
11  %   mp = simulink model parameters struct, with widget names for each register
    ↪  added
12
13  % Copyright 2020 Flat Earth Inc, Montana State University
14  %
15  % THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    ↪  IMPLIED,
16  % INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
    ↪  PARTICULAR
17  % PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
    ↪  BE LIABLE
18  % FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
    ↪  TORT OR OTHERWISE,
19  % ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
    ↪  DEALINGS IN THE SOFTWARE.
20  %
21  % Trevor Vannoy
22  % Flat Earth Inc
23  % 985 Technology Blvd
24  % Bozeman, MT 59718
25  % support@flatearthinc.com
26  function mp = createLinkerWidgetNames(mp)
27
28  numWidgets = containers.Map();
29  for i = 1:length(mp.register)
30      % keep track of how many registers have a given widget type so
31      % we can increment the widget name accordingly
32      widgetType = mp.register(i).widgetType;
33      if numWidgets.isKey(widgetType)
34          numWidgets(widgetType) = numWidgets(widgetType) + 1;
35      else
36          numWidgets(widgetType) = 1;
37      end
38
```

```
39      % widget name is of the form: slider<#><model name>
40      widgetName = [widgetType, num2str(numWidgets(widgetType)), mp.model_name];
41      mp.register(i).widgetName = widgetName;
42
43  end
```

genLinkerConfig.m

```
1   % genLinkerConfig Generate linker json file
2   %
3   % genLinkerConfig(mp, outfile)
4   %
5   % The linker json file tells our node js server on the HPS where
6   % registers for a device driver are located in sysfs. This function
7   % generates a linker file for the model represented by mp. If
8   % multiple models are to be run on the FPGA, use combine_linker_configs.py
9   % to combine linker files for all models of interest.
10  %
11  % Inputs:
12  %   mp = simulink model parameters struct
13  %   outfile = name of the output linker json file
14
15  % Copyright 2020 Flat Earth Inc, Montana State University
16  %
17  % THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    ↪  IMPLIED,
18  % INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
    ↪  PARTICULAR
19  % PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
    ↪  BE LIABLE
20  % FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
    ↪  TORT OR OTHERWISE,
21  % ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
    ↪  DEALINGS IN THE SOFTWARE.
22  %
23  % Trevor Vannoy
24  % Flat Earth Inc
25  % 985 Technology Blvd
26  % Bozeman, MT 59718
27  % support@flatearthinc.com
28
29  function genLinkerConfig(mp, outfile)
30  linkerConfig = struct(mp.model_name, struct());
31  linkerConfig.(mp.model_name).majorNo = "*";
32
33  links = struct();
34
35  % map widget name to sysfs file
36  for register = mp.register
37      links.(register.widgetName) = ['/', lower(register.name)];
```

```
38 end
39
40 linkerConfig.(mp.model_name).links = links;
41
42 writejson(linkerConfig, outfile);
```

genUiConfig.m

```
1  % genUiConfig Generate json file used to autogenerate gui
2  %
3  % genUiConfig(mp, outfile)
4  %
5  % This function generates a json file that can be used to autogenerate
6  % GUI controls at runtime. To generate GUI controls at runtime, the
7  % generated json file needs to be called UI.json and be in the same
8  % directory as the nodejs server on the HPS.
9  %
10 % To create a GUI that controls multiple Simulink models running on
11 % an FPGA, the individual UI json files for each model need to be
12 % combined into one json file with combine_ui_configs.py
13 %
14 % Inputs:
15 %   mp = simulink model parameters struct
16 %   outfile = name of the output linker json file
17
18 % Copyright 2020 Flat Earth Inc, Montana State University
19 %
20 % THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   ↪  IMPLIED,
21 % INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
   ↪  PARTICULAR
22 % PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
   ↪  BE LIABLE
23 % FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
   ↪  TORT OR OTHERWISE,
24 % ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
   ↪  DEALINGS IN THE SOFTWARE.
25 %
26 % Trevor Vannoy
27 % Flat Earth Inc
28 % 985 Technology Blvd
29 % Bozeman, MT 59718
30 % support@flatearthinc.com
31
32 function genUiConfig(mp, outfile)
33 % NOTE: to get jsonencode to create lists when there is only one struct,
34 %       I had to use cell arrays instead of normal arrays. A scalar is the same
   ↪  as a
35 %       1x1 array, so that all makes sense. Creating the cell arrays seemed to
   ↪  work
```

```
%       best when I created the cell array outside of the struct constructor.
%       Essentially, I needed a cell array of [1x1 structs] so jsonencode
%       would create the intended data structure. The data structure creation
%       in this function could probably be cleaner...

% The GUI name defaults to the model name, but can be changed
% when running combine_ui_configs.py
uiConfig.name = mp.model_name;

module = mp.model_name;

% create the first page and panel
registerControls = createControlObject(mp.register(1), module);
pageName = formatTitle(mp.register(1).uiPageName);
panelName = formatTitle(mp.register(1).uiPanelName);
controls = {registerControls};
panel = {struct('name', panelName)};
panel{1}.controls = controls;
uiConfig.pages = {struct('name', pageName, 'panels', ...
    {panel})};


% add the rest of the registers to the UI config
for i=2:length(mp.register)
    registerControls = createControlObject(mp.register(i), module);
    pageName = formatTitle(mp.register(i).uiPageName);
    panelName = formatTitle(mp.register(i).uiPanelName);

    % check if the desired page already exists
    pageIdx = findFieldValue(uiConfig.pages, 'name', pageName);
    if pageIdx
        % page exists, so check if the desired panel exists
        panelIdx = findFieldValue(uiConfig.pages{pageIdx}.panels, 'name',
        ↪  panelName);
        if panelIdx
            % add register to existing panel
            % NOTE: apparently appending to a cell array doesn't work quite like
            %       appending to a normal array. a = {..., ...}; a = {a, ...}
            ↪  DOES NOT
            %       behave the same as a = [..., ...]; a = [a, ...]. Maybe that's
            ↪  not
            %       even a good way to append to normal arrays either...
            %       To append to a cell array, you need to index into a new cell
            registerIdx =
            ↪  length(uiConfig.pages{pageIdx}.panels{panelIdx}.controls) + 1;
            uiConfig.pages{pageIdx}.panels{panelIdx}.controls{registerIdx} =
            ↪  registerControls;
        else
            % panel doesn't exist, so create one and add the register controls
            panelIdx = length(uiConfig.pages{pageIdx}.panels) + 1;
            panel = {struct('name', panelName)};
```

```
            panel{1}.controls = {registerControls};
            uiConfig.pages{pageIdx}.panels{panelIdx} = panel{1};

        end
    else
        % page doesn't exist, so create a new page and panel for the register
        ↪ controls
        panel = {struct('name', panelName)};
        panel{1}.controls = {registerControls};
        pageIdx = length(uiConfig.pages) + 1;
        uiConfig.pages{pageIdx} = struct('name', pageName, 'panels', {panel});
    end
end

writejson(uiConfig, outfile);
end


function control = createControlObject(register, module)
% createControlObject Assemble register info into a struct
%
% control = createControlObject(register, module)
%
% This function essentially just renames some fields in the register struct
%
% Inputs:
%   register = register struct from Simulink model init scripts
%   module = the model / device driver name
% Outputs:
%   control = register info packed into a struct for json UI config file
control.linkerName = register.widgetName;
control.type = register.widgetType;
control.min = register.min;
control.max = register.max;
control.dataType = register.dataType.qpointstr;
control.defaultValue = register.default;
control.units = register.widgetDisplayUnits;
control.style = register.widgetStyle;
control.title = formatTitle(register.name);
control.module = module;
end

function idx = findFieldValue(arrayOfStruct, field, value)
% findFieldValue Find a field value in an array of structs
%
% idx = findFieldValue(arrayOfStruct, field, value)
%
% This function loops through an array of structures, looks at
% the given field in each struct, and sees if the given value
% is in any of the structs. If so, it returns the index where that is
% is true, otherwise the index is 0.
```

```
%
% Inputs:
%   arrayOfStruct = array of structures to loop through
%   field = the field to look at in each struct
%   value = the value to look for
%
% Outputs:
%   idx = the index where the field was found
idx = 0;
for i = 1:length(arrayOfStruct)
    if strcmpi(arrayOfStruct{i}.(field), value)
        idx = i;
    end
end
end

function s = formatTitle(str)
% formatTitle Format strings into "Title Case"
%
% s = formatTitle(str)
%
% Formats strings into by separating words and capitalizing
% each word. This is used for display purposes because we would
% rather display "Some Register Name" than "some_register_name" in GUIs.
% This function only works on strings where words are separated by
% spaces, hyphens, or underscores.
%
% Inputs:
%   str = the string to format
% Outputs:
%   s = the formatted string

% TODO: add support for camelCase and TitleCase strings?
% split string into words
words = split(str, [" " "_" "-"]);

% capitalize each word
words = cellfun(@(s) [upper(s(1)) s(2:end)], words, 'uniformoutput', false);

% put the words back together with spaces in between
s = join(words, ' ');

% turn cell array into character array (string)
s = s{:};
end
```

APPENDIX E

JSON UTILITY SCRIPTS

readjson.m

```
% READJSON Read a json file and return a json-formatted string
%
% json = READJSON(filename)
%
% Inputs:
%   filename = the input filename
%   json = json-formatted output string

% Copyright 2019 Flat Earth Inc, Montana State University
%
% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
% INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
% PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
% FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
% ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
%
% Trevor Vannoy
% Flat Earth Inc
% 985 Technology Blvd
% Bozeman, MT 59718
% support@flatearthinc.com

function json = readjson(filename)

%https://www.mathworks.com/matlabcentral/answers/326764-how-can-i-read-a-json-file
json = jsondecode(fileread(filename));
```

writejson.m

```
% WRITEJSON Take a json-compatible Matlab object and save it to a file
%
% WRITEJSON(data, filename)
%
% This function uses python to pretty-print the string before we save it to a file
%
% Inputs:
%   data = json-compatible Matlab object
%   filename = the output json file name

% Copyright 2019 Flat Earth Inc, Montana State University
%
% THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED,
```

```
% INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
% PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE
% FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
% ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
%
% Trevor Vannoy
% Flat Earth Inc
% 985 Technology Blvd
% Bozeman, MT 59718
% support@flatearthinc.com

function writejson(data, filename)

% pretty-printing taken from https://blogs.mathworks.com/pick/2016/11/11/r2016b-features-markerindices-jsonencode-jsondecode/

% import python libraries
py.importlib.import_module('json');
collections = py.importlib.import_module('collections');

% encode the data
jsonstr = jsonencode(data);

% create the pretty-printed json string
prettyjson = char(py.json.dumps(py.json.loads(jsonstr), pyargs('indent', int32(4))));

% save the string to a file
fid = fopen(filename, 'w');
fprintf(fid, '%s', prettyjson);
```

APPENDIX F

AUTOGENERATED CODE EXAMPLES

BC_dataplane.json

```
{
  "avalon_memorymapped": {
    "register": [
      {
        "data_type": {
          "fractional_bits": 7,
          "signed": false,
          "type": "ufix8_En7",
          "width": 8
        },
        "default_value": 0.5,
        "max_value": 1,
        "min_value": 0,
        "name": "wet_dry_mix",
        "reg_num": 2
      },
      {
        "data_type": {
          "fractional_bits": 0,
          "signed": false,
          "type": "boolean",
          "width": 1
        },
        "default_value": 0,
        "max_value": 1,
        "min_value": 0,
        "name": "bypass",
        "reg_num": 0
      },
      {
        "data_type": {
          "fractional_bits": 0,
          "signed": false,
          "type": "ufix6",
          "width": 6
        },
        "default_value": 32,
        "max_value": 32,
        "min_value": 0,
        "name": "bits",
        "reg_num": 1
      }
    ]
  },
  "avalon_memorymapped_flag": 1,
  "avalon_sink": {
    "signal": [
      {
        "data_type": {
```

```
            "fractional_bits": 0,
            "signed": false,
            "type": "boolean",
            "width": 1
          },
          "name": "avalon_sink_valid"
        },
        {
          "data_type": {
            "fractional_bits": 28,
            "signed": true,
            "type": "sfix32_En28",
            "width": 32
          },
          "name": "avalon_sink_data"
        },
        {
          "data_type": {
            "fractional_bits": 0,
            "signed": false,
            "type": "ufix2",
            "width": 2
          },
          "name": "avalon_sink_channel"
        },
        {
          "data_type": {
            "fractional_bits": 0,
            "signed": false,
            "type": "ufix2",
            "width": 2
          },
          "name": "avalon_sink_error"
        }
      ]
    },
    "avalon_sink_flag": 1,
    "avalon_source": {
      "signal": [
        {
          "data_type": {
            "fractional_bits": 0,
            "signed": false,
            "type": "boolean",
            "width": 1
          },
          "name": "avalon_source_valid"
        },
        {
          "data_type": {
            "fractional_bits": 28,
```

```
            "signed": true,
            "type": "sfix32_En28",
            "width": 32
          },
          "name": "avalon_source_data"
        },
        {
          "data_type": {
            "fractional_bits": 0,
            "signed": false,
            "type": "ufix2",
            "width": 2
          },
          "name": "avalon_source_channel"
        },
        {
          "data_type": {
            "fractional_bits": 0,
            "signed": false,
            "type": "ufix2",
            "width": 2
          },
          "name": "avalon_source_error"
        }
      ]
    },
    "avalon_source_flag": 1,
    "clocks": {
      "sample_frequency_Hz": 48000,
      "sample_period_seconds": 2.0833333333333333e-05,
      "system_frequency_Hz": 98304000.0,
      "system_period_seconds": 1.0172526041666666e-08
    },
    "conduit_input": [],
    "conduit_input_flag": 0,
    "conduit_output": [],
    "conduit_output_flag": 0,
    "entity": "BC_dataplane",
    "linux_device_name": "bitcrusher",
    "linux_device_version": "18.0",
    "model_abbreviation": "BC",
    "model_name": "bitcrusher"
}
```

BC_dataplane_avalon.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

```

```
5  entity BC_dataplane_avalon is
6    port (
7      clk                       : in  std_logic;
8      reset                     : in  std_logic;
9      avalon_sink_valid         : in  std_logic; --boolean
10     avalon_sink_data          : in  std_logic_vector(31  downto 0); --sfix32_En28
11     avalon_sink_channel       : in  std_logic_vector(1   downto 0); --ufix2
12     avalon_sink_error         : in  std_logic_vector(1   downto 0); --ufix2
13     avalon_source_valid       : out std_logic; --boolean
14     avalon_source_data        : out std_logic_vector(31  downto 0); --sfix32_En28
15     avalon_source_channel     : out std_logic_vector(1   downto 0); --ufix2
16     avalon_source_error       : out std_logic_vector(1   downto 0); --ufix2
17     avalon_slave_address      : in  std_logic_vector(1   downto 0);
18     avalon_slave_read         : in  std_logic;
19     avalon_slave_readdata     : out std_logic_vector(31  downto 0);
20     avalon_slave_write        : in  std_logic;
21     avalon_slave_writedata    : in  std_logic_vector(31  downto 0)
22   );
23 end entity BC_dataplane_avalon;
24
25 architecture BC_dataplane_avalon_arch of BC_dataplane_avalon is
26
27   signal bypass                    : std_logic :=  '0'; -- 0
28   signal bits                      : std_logic_vector(5  downto 0) :=
     ↪  std_logic_vector(to_unsigned(32, 6)); -- 32
29   signal wet_dry_mix               : std_logic_vector(7  downto 0) :=
     ↪  "01000000"; -- 0.5
30
31 component BC_dataplane
32   port(
33     clk                         : in  std_logic; -- clk_freq = 1 Hz, period = 0.1
34     reset                       : in  std_logic;
35     clk_enable                  : in  std_logic;
36     avalon_sink_valid           : in  std_logic;                                --
       ↪  boolean
37     avalon_sink_data            : in  std_logic_vector(31  downto 0);         --
       ↪  sfix32_En28
38     avalon_sink_channel         : in  std_logic_vector(1   downto 0);         --
       ↪  ufix2
39     avalon_sink_error           : in  std_logic_vector(1   downto 0);         --
       ↪  ufix2
40     register_control_bypass     : in  std_logic;                              --
       ↪  boolean
41     register_control_bits       : in  std_logic_vector(5   downto 0);         --
       ↪  ufix6
42     register_control_wet_dry_mix: in  std_logic_vector(7   downto 0);         --
       ↪  ufix8_En7
43     ce_out                      : out std_logic;
44     avalon_source_valid         : out std_logic;                              --
       ↪  boolean
```

```
45      avalon_source_data          : out std_logic_vector(31  downto 0);         --
        ↪  sfix32_En28
46      avalon_source_channel       : out std_logic_vector(1   downto 0);         --
        ↪  ufix2
47      avalon_source_error         : out std_logic_vector(1   downto 0)          --
        ↪  ufix2
48    );
49  end component;
50
51  begin
52
53  u_BC_dataplane : BC_dataplane
54    port map(
55      clk                         =>  clk,
56      reset                       =>  reset,
57      clk_enable                  =>  '1',
58      avalon_sink_valid           =>  avalon_sink_valid,              -- boolean
59      avalon_sink_data            =>  avalon_sink_data,               --
        ↪  sfix32_En28
60      avalon_sink_channel         =>  avalon_sink_channel,            -- ufix2
61      avalon_sink_error           =>  avalon_sink_error,              -- ufix2
62      register_control_bypass     =>  bypass,                         -- boolean
63      register_control_bits       =>  bits,                           --
        ↪  sfix32_En28
64      register_control_wet_dry_mix=>  wet_dry_mix,                    -- ufix2
65      avalon_source_valid         =>  avalon_source_valid,            -- boolean
66      avalon_source_data          =>  avalon_source_data,             --
        ↪  sfix32_En28
67      avalon_source_channel       =>  avalon_source_channel,          -- ufix2
68      avalon_source_error         =>  avalon_source_error             -- ufix2
69    );
70
71    bus_read : process(clk)
72    begin
73      if rising_edge(clk) and avalon_slave_read = '1' then
74        case avalon_slave_address is
75          when "00" => avalon_slave_readdata <= (31 downto 1 => '0') & bypass;
76          when "01" => avalon_slave_readdata <= (31 downto 6 => '0') & bits;
77          when "10" => avalon_slave_readdata <= (31 downto 8 => '0') & wet_dry_mix;
78          when others => avalon_slave_readdata <= (others => '0');
79        end case;
80      end if;
81    end process;
82
83    bus_write : process(clk, reset)
84    begin
85      if reset = '1' then
86        bypass                    <=  '0'; -- 0
87        bits                      <=  std_logic_vector(to_unsigned(32, 6)); -- 32
88        wet_dry_mix               <=  "01000000"; -- 0.5
89      elsif rising_edge(clk) and avalon_slave_write = '1' then
```

```
90        case avalon_slave_address is
91          when "00" => bypass <= avalon_slave_writedata(0);
92          when "01" => bits <= avalon_slave_writedata(5 downto 0);
93          when "10" => wet_dry_mix <= avalon_slave_writedata(7 downto 0);
94          when others => null;
95        end case;
96      end if;
97    end process;
98
99  end architecture;
```

BC_dataplane_avalon_hw.tcl

```
1   # # # # # # # # # # # # # # # # # #
2   # Built in create_header_file_stuff
3   # # # # # # # # # # # # # # # # # #
4
5   package require -exact qsys 16.1
6   # End create_header_file_stuff
7
8
9   # # # # # # # # # # # # # # # # # #
10  # Created in create_module
11  # # # # # # # # # # # # # # # # # #
12
13  set_module_property DESCRIPTION ""
14  set_module_property NAME "bitcrusher"
15  set_module_property VERSION 1.0
16  set_module_property OPAQUE_ADDRESS_MAP true
17  set_module_property AUTHOR ""
18  set_module_property DISPLAY_NAME "bitcrusher"
19  set_module_property INSTANTIATE_IN_SYSTEM_MODULE true
20  set_module_property EDITABLE true
21  set_module_property REPORT_TO_TALKBACK false
22  set_module_property ALLOW_GREYBOX_GENERATION false
23  set_module_property VERSION 1.0
24  set_module_property REPORT_HIERARCHY false
25  # end of create_module
26
27
28  # # # # # # # # # # # # # # # # # #
29  # created in create_file_sets
30  # # # # # # # # # # # # # # # # # #
31
32  add_fileset QUARTUS_SYNTH QUARTUS_SYNTH "" ""
33  add_fileset_file BC_Left_Channel_Processing.vhd VHDL PATH
    ↪  BC_Left_Channel_Processing.vhd
34  add_fileset_file BC_dataplane_pkg.vhd VHDL PATH BC_dataplane_pkg.vhd
35  add_fileset_file BC_Right_Channel_Processing.vhd VHDL PATH
    ↪  BC_Right_Channel_Processing.vhd
```

```
36 add_fileset_file BC_dataplane.vhd VHDL PATH BC_dataplane.vhd
37 add_fileset_file BC_getbitmask.vhd VHDL PATH BC_getbitmask.vhd
38 add_fileset_file BC_Avalon_Data_Processing.vhd VHDL PATH
   ↪ BC_Avalon_Data_Processing.vhd
39 add_fileset_file BC_mixer.vhd VHDL PATH BC_mixer.vhd
40 add_fileset_file BC_mixer_block.vhd VHDL PATH BC_mixer_block.vhd
41 set_fileset_property QUARTUS_SYNTH TOP_LEVEL BC_dataplane_avalon
42 set_fileset_property QUARTUS_SYNTH ENABLE_RELATIVE_INCLUDE_PATHS false
43 set_fileset_property QUARTUS_SYNTH ENABLE_FILE_OVERWRITE_MODE false
44 add_fileset_file BC_dataplane_avalon.vhd VHDL PATH BC_dataplane_avalon.vhd
   ↪ TOP_LEVEL_FILE
45 # end create_file_sets
46
47
48 # # # # # # # # # # # # # # # # # #
49 # Created in create_module_assignments
50 # # # # # # # # # # # # # # # # # # #
51
52 set_module_assignment embeddedsw.dts.compatible dev,fe-bitcrusher
53 set_module_assignment embeddedsw.dts.group bitcrusher
54 set_module_assignment embeddedsw.dts.vendor fe
55 # End create_module_assignments
56
57
58 # # # # # # # # # # # # # # # # # # # # # #
59 # Created by create_connection_point_clock
60 # # # # # # # # # # # # # # # # # # # # # #
61
62 add_interface clock clock end
63 set_interface_property clock clockRate 98304000.0
64 set_interface_property clock ENABLED true
65 set_interface_property clock EXPORT_OF ""
66 set_interface_property clock PORT_NAME_MAP ""
67 set_interface_property clock CMSIS_SVD_VARIABLES ""
68 set_interface_property clock SVD_ADDRESS_GROUP ""
69 add_interface_port clock clk clk Input 1
70 # End create_connection_point_clock
71
72
73 # # # # # # # # # # # # # # # # # # # # # #
74 # Created by create_connection_point_reset
75 # # # # # # # # # # # # # # # # # # # # # #
76
77 add_interface reset reset end
78 set_interface_property reset associatedClock clock
79 set_interface_property reset synchronousEdges DEASSERT
80 set_interface_property reset ENABLED true
81 set_interface_property reset EXPORT_OF true
82 set_interface_property reset PORT_NAME_MAP ""
83 set_interface_property reset CMSIS_SVD_VARIABLES ""
84 set_interface_property reset SVD_ADDRESS_GROUP ""
```

```
85  add_interface_port reset reset reset Input 1
86  # End create_connection_point_reset
87
88
89  # # # # # # # # # # # # # # # # # # # #
90  # Created by create_mm_connection_point
91  # # # # # # # # # # # # # # # # # # # #
92
93  add_interface avalon_slave avalon end
94  set_interface_property avalon_slave addressUnits WORDS
95  set_interface_property avalon_slave associatedClock clock
96  set_interface_property avalon_slave associatedReset reset
97  set_interface_property avalon_slave bitsPerSymbol 8
98  set_interface_property avalon_slave burstOnBurstBoundariesOnly false
99  set_interface_property avalon_slave burstcountUnits WORDS
100 set_interface_property avalon_slave explicitAddressSpan 0
101 set_interface_property avalon_slave holdTime 0
102 set_interface_property avalon_slave linewrapBursts false
103 set_interface_property avalon_slave maximumPendingReadTransactions 0
104 set_interface_property avalon_slave maximumPendingWriteTransactions 0
105 set_interface_property avalon_slave readLatency 0
106 set_interface_property avalon_slave readWaitTime 1
107 set_interface_property avalon_slave setupTime 1
108 set_interface_property avalon_slave timingUnits Cycles
109 set_interface_property avalon_slave writeWaitTime 0
110 set_interface_property avalon_slave ENABLED true
111 set_interface_property avalon_slave EXPORT_OF ""
112 set_interface_property avalon_slave PORT_NAME_MAP ""
113 set_interface_property avalon_slave CMSIS_SVD_VARIABLES ""
114 set_interface_property avalon_slave SVD_ADDRESS_GROUP ""
115
116 add_interface_port avalon_slave avalon_slave_address address Input 2
117 add_interface_port avalon_slave avalon_slave_read read Input 1
118 add_interface_port avalon_slave avalon_slave_readdata readdata Output 32
119 add_interface_port avalon_slave avalon_slave_write write Input 1
120 add_interface_port avalon_slave avalon_slave_writedata writedata Input 32
121 set_interface_assignment avalon_slave embeddedsw.configuration.isFlash 0
122 set_interface_assignment avalon_slave embeddedsw.configuration.isMemoryDevice 0
123 set_interface_assignment avalon_slave
    ↪ embeddedsw.configuration.isNonVolatileStorage 0
124 set_interface_assignment avalon_slave embeddedsw.configuration.isPrintableDevice 0
125 # End create_mm_connection_point
126
127
128 # # # # # # # # # # # # # # # # # # # #
129 # Created by create_sink_connection_point
130 # # # # # # # # # # # # # # # # # # # #
131
132 add_interface avalon_streaming_sink avalon_streaming end
133 set_interface_property avalon_streaming_sink associatedClock clock
134 set_interface_property avalon_streaming_sink associatedReset reset
```

```
set_interface_property avalon_streaming_sink dataBitsPerSymbol 8
set_interface_property avalon_streaming_sink errorDescriptor ""
set_interface_property avalon_streaming_sink firstSymbolInHighOrderBits true
set_interface_property avalon_streaming_sink maxChannel 3
set_interface_property avalon_streaming_sink readyLatency 0
set_interface_property avalon_streaming_sink ENABLED true
set_interface_property avalon_streaming_sink EXPORT_OF ""
set_interface_property avalon_streaming_sink PORT_NAME_MAP ""
set_interface_property avalon_streaming_sink CMSIS_SVD_VARIABLES ""
set_interface_property avalon_streaming_sink SVD_ADDRESS_GROUP ""
add_interface_port avalon_streaming_sink avalon_sink_data data Input 32
add_interface_port avalon_streaming_sink avalon_sink_channel channel Input 2
add_interface_port avalon_streaming_sink avalon_sink_error error Input 2
add_interface_port avalon_streaming_sink avalon_sink_valid valid Input 1
# End create_sink_connection_point


# # # # # # # # # # # # # # # # # # # # # # # #
# Created in create_source_connection_point
# # # # # # # # # # # # # # # # # # # # # # # #

add_interface avalon_streaming_source avalon_streaming start
set_interface_property avalon_streaming_source associatedClock clock
set_interface_property avalon_streaming_source associatedReset reset
set_interface_property avalon_streaming_source dataBitsPerSymbol 8
set_interface_property avalon_streaming_source errorDescriptor ""
set_interface_property avalon_streaming_source firstSymbolInHighOrderBits true
set_interface_property avalon_streaming_source maxChannel 3
set_interface_property avalon_streaming_source readyLatency 0
set_interface_property avalon_streaming_source ENABLED true
set_interface_property avalon_streaming_source EXPORT_OF ""
set_interface_property avalon_streaming_source PORT_NAME_MAP ""
set_interface_property avalon_streaming_source CMSIS_SVD_VARIABLES ""
set_interface_property avalon_streaming_source SVD_ADDRESS_GROUP ""
add_interface_port avalon_streaming_source avalon_source_channel channel Output 2
add_interface_port avalon_streaming_source avalon_source_error error Output 2
add_interface_port avalon_streaming_source avalon_source_data data Output 32
add_interface_port avalon_streaming_source avalon_source_valid valid Output 1
# End create_sink_connection_point
```

bitcrusher.c

```
/*************************************************************************
Generated in CreateFileHeaderInfo
*************************************************************************/
#include <linux/module.h>
#include <linux/platform_device.h>
#include <linux/io.h>
#include <linux/fs.h>
#include <linux/types.h>
```

```
#include <linux/uaccess.h>
#include <linux/init.h>
#include <linux/cdev.h>
#include <linux/regmap.h>
#include <linux/of.h>
#include "custom_functions.h"

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Tyler Davis <support@flatearthinc.com");
MODULE_DESCRIPTION("Loadable kernel module for the bitcrusher");
MODULE_VERSION("1.0");
/* End CreateFileHeaderInfo */


/***********************************************
  Generated in CreateMiscTopOfFile
***********************************************/
struct fixed_num {
  int integer;
  int fraction;
  int fraction_len;
};
static struct class *cl;  // Global variable for the device class
static dev_t dev_num;

/*********** Device type specific things **************/
//TODO: check this. Register memory map. Was not pulled from input params, might want to verify this
#define BAND_ALL_OFFSET 0x00
#define BAND1_OFFSET 0x01
#define BAND2_OFFSET 0x02
#define BAND3_OFFSET 0x03
#define BAND4_OFFSET 0x04
#define LEFT_OFFSET 0x00
#define RIGHT_OFFSET 0x08
#define GAIN_OFFSET 0
/* End of CreateMiscTopOfFile */


/*****************************************************
Generate in CreateFunctionPrototypes
*****************************************************/
static int bitcrusher_probe(struct platform_device *pdev);
static int bitcrusher_remove(struct platform_device *pdev);
static ssize_t bitcrusher_read(struct file *file, char *buffer, size_t len, loff_t *offset);
static ssize_t bitcrusher_write(struct file *file, const char *buffer, size_t len, loff_t *offset);
static int bitcrusher_open(struct inode *inode, struct file *file);
static int bitcrusher_release(struct inode *inode, struct file *file);
```

```
56 static ssize_t name_read(struct device *dev, struct device_attribute *attr, char
   ↪  *buf);
57
58 /************** Generate device specific prototypes *******************/
59 // FPGA device funcs
60 static ssize_t bypass_write (struct device *dev, struct device_attribute *attr,
   ↪  const char *buf, size_t count);
61 static ssize_t bypass_read  (struct device *dev, struct device_attribute *attr,
   ↪  char *buf);
62 static ssize_t bits_write (struct device *dev, struct device_attribute *attr,
   ↪  const char *buf, size_t count);
63 static ssize_t bits_read  (struct device *dev, struct device_attribute *attr,
   ↪  char *buf);
64 static ssize_t wet_dry_mix_write (struct device *dev, struct device_attribute
   ↪  *attr, const char *buf, size_t count);
65 static ssize_t wet_dry_mix_read  (struct device *dev, struct device_attribute
   ↪  *attr, char *buf);
66
67 /* Custom function declarations */
68 /* End CreateFunctionPrototypes */
69
70
71 /***************************************************
72 Generated in WriteDeviceAttributes
73 ***************************************************/
74 DEVICE_ATTR(bypass, 0664, bypass_read, bypass_write);
75 DEVICE_ATTR(bits, 0664, bits_read, bits_write);
76 DEVICE_ATTR(wet_dry_mix, 0664, wet_dry_mix_read, wet_dry_mix_write);
77 DEVICE_ATTR(name, 0444, name_read, NULL);
78 /* End WriteDeviceAttributes */
79
80
81 /***************************************************
82 Generated in CreateDriverStuff
83 ***************************************************/
84
85 /* Device struct */
86 struct fe_bitcrusher_dev {
87   struct cdev cdev;
88   char *name;
89   void __iomem *regs;
90   int bypass;
91   int bits;
92   int wet_dry_mix;
93 };
94
95 typedef struct fe_bitcrusher_dev fe_bitcrusher_dev_t;
96 /* ID Matching struct */
97 static struct of_device_id fe_bitcrusher_dt_ids[] = {
98   {
99     .compatible = "dev,fe-bitcrusher"
```

```
100   },
101   { }
102 };
103 
104 MODULE_DEVICE_TABLE(of, fe_bitcrusher_dt_ids);
105 /* Platform driver struct */
106 static struct platform_driver bitcrusher_platform = {
107   .probe = bitcrusher_probe,
108   .remove = bitcrusher_remove,
109   .driver = {
110     .name = "Flat Earth bitcrusher Driver",
111     .owner = THIS_MODULE,
112     .of_match_table = fe_bitcrusher_dt_ids
113   }
114 };
115 
116 /* File ops struct */
117 static const struct file_operations fe_bitcrusher_fops = {
118   .owner = THIS_MODULE,
119   .read = bitcrusher_read,
120   .write = bitcrusher_write,
121   .open = bitcrusher_open,
122   .release = bitcrusher_release,
123 };
124 
125 /* End of CreateDriverStuff */
126 
127 
128 /*********************************************************
129 Generated in CreateInitFunction
130 *********************************************************/
131 static int bitcrusher_init(void) {
132   printk(KERN_ALERT "FUNCTION AUTO GENERATED AT: 2019-10-06 20:30\n");
133   int ret_val = 0;
134   pr_info("Initializing the Flat Earth bitcrusher module\n");
135   // Register our driver with the "Platform Driver" bus
136   ret_val = platform_driver_register(&bitcrusher_platform);  if (ret_val != 0) {
137     pr_err("platform_driver_register returned %d\n", ret_val);
138     return ret_val;
139   }
140   pr_info("Flat Earth bitcrusher module successfully initialized!\n");
141   return 0;
142 }
143 /* End CreateInitFunction */
144 
145 /*********************************************
146 Generated by CreateProbeFunction
147 *********************************************/
148 static int bitcrusher_probe(struct platform_device *pdev) {
149   int ret_val = -EBUSY;
150   char deviceName[22] = "fe_bitcrusher_";
```

```
151   char deviceMinor[20];
152   int status;
153   struct device *device_obj;
154   fe_bitcrusher_dev_t * fe_bitcrusher_devp;
155   pr_info("bitcrusher_probe enter\n");
156   struct resource *r = 0;
157   r = platform_get_resource(pdev, IORESOURCE_MEM, 0);
158   if (r == NULL) {
159     pr_err("IORESOURCE_MEM (register space) does not exist\n");
160     goto bad_exit_return;  }
161   fe_bitcrusher_devp = devm_kzalloc(&pdev->dev, sizeof(fe_bitcrusher_dev_t),
      ↪  GFP_KERNEL);
162   fe_bitcrusher_devp->regs = devm_ioremap_resource(&pdev->dev, r);
163   if (IS_ERR(fe_bitcrusher_devp->regs))
164     goto bad_ioremap;
165   platform_set_drvdata(pdev, (void *)fe_bitcrusher_devp);
166   fe_bitcrusher_devp->name = devm_kzalloc(&pdev->dev, 50, GFP_KERNEL);
167   if (fe_bitcrusher_devp->name == NULL)
168     goto bad_mem_alloc;
169   strcpy(fe_bitcrusher_devp->name, (char *)pdev->name);
170   pr_info("%s\n", (char *)pdev->name);
171   status = alloc_chrdev_region(&dev_num, 0, 1, "fe_bitcrusher_");
172   if (status != 0)
173     goto bad_alloc_chrdev_region;
174   sprintf(deviceMinor, "%d", MAJOR(dev_num));
175   strcat(deviceName, deviceMinor);
176   pr_info("%s\n", deviceName);
177   cl = class_create(THIS_MODULE, deviceName);
178   if (cl == NULL)
179     goto bad_class_create;
180   cdev_init(&fe_bitcrusher_devp->cdev, &fe_bitcrusher_fops);
181   status = cdev_add(&fe_bitcrusher_devp->cdev, dev_num, 1);
182   if (status != 0)
183     goto bad_cdev_add;
184   device_obj = device_create(cl, NULL, dev_num, NULL, deviceName);
185   if (device_obj == NULL)
186     goto bad_device_create;
187   dev_set_drvdata(device_obj, fe_bitcrusher_devp);
188 /* Beginning attribute file stuff */
189   status = device_create_file(device_obj, &dev_attr_bypass);
190   if (status)
191     goto bad_device_create_file_1;
192
193   status = device_create_file(device_obj, &dev_attr_bits);
194   if (status)
195     goto bad_device_create_file_2;
196
197   status = device_create_file(device_obj, &dev_attr_wet_dry_mix);
198   if (status)
199     goto bad_device_create_file_3;
200
```

```
201   status = device_create_file(device_obj, &dev_attr_name);
202   if (status)
203     goto bad_device_create_file_4;
204   pr_info("HA_probe exit\n");
205   return 0;
206 bad_device_create_file_4:
207   device_remove_file(device_obj, &dev_attr_name);
208 bad_device_create_file_3:
209   device_remove_file(device_obj, &dev_attr_wet_dry_mix);
210
211 bad_device_create_file_2:
212   device_remove_file(device_obj, &dev_attr_bits);
213
214 bad_device_create_file_1:
215   device_remove_file(device_obj, &dev_attr_bypass);
216
217 bad_device_create_file_0:
218   device_destroy(cl, dev_num);
219
220 bad_device_create:
221   cdev_del(&fe_bitcrusher_devp->cdev);
222 bad_cdev_add:
223   class_destroy(cl);
224 bad_class_create:
225   unregister_chrdev_region(dev_num, 1);
226 bad_alloc_chrdev_region:
227 bad_mem_alloc:
228 bad_ioremap:
229   ret_val = PTR_ERR(fe_bitcrusher_devp->regs);
230 bad_exit_return:
231   pr_info("bitcrusher_probe bad exit\n");
232   return ret_val;
233 }
234 /* End of CreateProbeFunction */
235
236
237 /*****************************************************
238 Generated by CreateAttrReadWriteFuncs
239 *****************************************************/
240 // FPGA Attribute functions
241 static ssize_t bypass_read(struct device *dev, struct device_attribute *attr,
    ↪  char *buf) {
242   fe_bitcrusher_dev_t * devp = (fe_bitcrusher_dev_t *)dev_get_drvdata(dev);
243   fp_to_string(buf, devp->bypass, 0, false, 0);
244   strcat2(buf,"\n");
245   return strlen(buf);
246 }
247
248 static ssize_t bypass_write(struct device *dev, struct device_attribute *attr,
    ↪  const char *buf, size_t count) {
249   uint32_t tempValue = 0;
```

```
250   char substring[80];
251   int substring_count = 0;
252   int i;
253   fe_bitcrusher_dev_t *devp = (fe_bitcrusher_dev_t *)dev_get_drvdata(dev);
254   for (i = 0; i < count; i++) {
255     if ((buf[i] != ',') && (buf[i] != ' ') && (buf[i] != '\0') && (buf[i] !=
      ↪  '\r') && (buf[i] != '\n')) {
256       substring[substring_count] = buf[i];
257       substring_count ++;
258     }
259   }
260   substring[substring_count] = 0;
261   tempValue = set_fixed_num(substring, 0, false);
262   devp->bypass = tempValue;
263   iowrite32(devp->bypass, (u32 *)devp->regs + 0);
264   return count;
265 }
266
267 static ssize_t bits_read(struct device *dev, struct device_attribute *attr, char
  ↪  *buf) {
268   fe_bitcrusher_dev_t * devp = (fe_bitcrusher_dev_t *)dev_get_drvdata(dev);
269   fp_to_string(buf, devp->bits, 0, false, 0);
270   strcat2(buf,"\n");
271   return strlen(buf);
272 }
273
274 static ssize_t bits_write(struct device *dev, struct device_attribute *attr,
  ↪  const char *buf, size_t count) {
275   uint32_t tempValue = 0;
276   char substring[80];
277   int substring_count = 0;
278   int i;
279   fe_bitcrusher_dev_t *devp = (fe_bitcrusher_dev_t *)dev_get_drvdata(dev);
280   for (i = 0; i < count; i++) {
281     if ((buf[i] != ',') && (buf[i] != ' ') && (buf[i] != '\0') && (buf[i] !=
      ↪  '\r') && (buf[i] != '\n')) {
282       substring[substring_count] = buf[i];
283       substring_count ++;
284     }
285   }
286   substring[substring_count] = 0;
287   tempValue = set_fixed_num(substring, 0, false);
288   devp->bits = tempValue;
289   iowrite32(devp->bits, (u32 *)devp->regs + 1);
290   return count;
291 }
292
293 static ssize_t wet_dry_mix_read(struct device *dev, struct device_attribute
  ↪  *attr, char *buf) {
294   fe_bitcrusher_dev_t * devp = (fe_bitcrusher_dev_t *)dev_get_drvdata(dev);
295   fp_to_string(buf, devp->wet_dry_mix, 7, false, 9);
```

```
296   strcat2(buf,"\n");
297   return strlen(buf);
298 }
299
300 static ssize_t wet_dry_mix_write(struct device *dev, struct device_attribute
    ↪  *attr, const char *buf, size_t count) {
301   uint32_t tempValue = 0;
302   char substring[80];
303   int substring_count = 0;
304   int i;
305   fe_bitcrusher_dev_t *devp = (fe_bitcrusher_dev_t *)dev_get_drvdata(dev);
306   for (i = 0; i < count; i++) {
307     if ((buf[i] != ',') && (buf[i] != ' ') && (buf[i] != '\0') && (buf[i] !=
      ↪  '\r') && (buf[i] != '\n')) {
308       substring[substring_count] = buf[i];
309       substring_count ++;
310     }
311   }
312   substring[substring_count] = 0;
313   tempValue = set_fixed_num(substring, 7, false);
314   devp->wet_dry_mix = tempValue;
315   iowrite32(devp->wet_dry_mix, (u32 *)devp->regs + 2);
316   return count;
317 }
318
319 static ssize_t name_read(struct device *dev, struct device_attribute *attr, char
    ↪  *buf) {
320   fe_bitcrusher_dev_t *devp = (fe_bitcrusher_dev_t *)dev_get_drvdata(dev);
321   sprintf(buf, "%s\n", devp->name);
322   return strlen(buf);
323 }
324 /* End of CreateAttrReadWriteFuncs */
325
326
327 /*****************************************
328   Generated by CreateCFunctionStubs
329 *****************************************/
330 static int bitcrusher_open(struct inode *inode, struct file *file) {
331   // TODO: fill this in (if its needed, it might not be)
332   return 0;
333 }
334
335 static int bitcrusher_release(struct inode *inode, struct file *file) {
336   // TODO: fill this in (if its needed, it might not be)
337   return 0;
338 }
339
340 static ssize_t bitcrusher_read(struct file *file, char *buffer, size_t len,
    ↪  loff_t *offset) {
341   // TODO: fill this in (if its needed, it might not be)
342   return 0;
```

```
}

static ssize_t bitcrusher_write(struct file *file, const char *buffer, size_t
↪  len, loff_t *offset) {
  // TODO: fill this in (if its needed, it might not be)
  return 0;
}
 /* End CreateCFunctionStubs */


/*****************************************************
Generated in CreateRemoveFunction
*****************************************************/
static int bitcrusher_remove(struct platform_device *pdev) {
  fe_bitcrusher_dev_t *dev = (fe_bitcrusher_dev_t *)platform_get_drvdata(pdev);
  pr_info("bitcrusher_remove enter\n");
  cdev_del(&dev->cdev);
  unregister_chrdev_region(dev_num, 2);
  iounmap(dev->regs);
  pr_info("bitcrusher_remove exit\n");
  return 0;
}
/* End CreateRemoveFunction */


/***********************************************
Generated by CreateExitFunction
***********************************************/
static void bitcrusher_exit(void) {
  pr_info("Flat Earth bitcrusher module exit\n");
  platform_driver_unregister(&bitcrusher_platform);
  pr_info("Flat Earth bitcrusher module successfully unregistered\n");
}
/* End CreateExitFunction */


/*********************************************
Generated by CreateCustomFunctions
*********************************************/
/* End CreateCustomFunctions */


/*******************************************
Generated by CreateEndOfFileStuff
*******************************************/
module_init(bitcrusher_init);
module_exit(bitcrusher_exit);
/* End CreateEndOfFileStuff */
```

Makefile

```
KDIR ?= ../../../../../linux-socfpga
default:
	$(MAKE) -C $(KDIR) ARCH=arm M=$(CURDIR)
clean:
	$(MAKE) -C $(KDIR) ARCH=arm M=$(CURDIR) clean
help:
	$(MAKE) -C $(KDIR) ARCH=arm M=$(CURDIR) help
```

Kbuild

```
ccflags-y := -I$(src)/../../../../../component_library/include
obj-m := bitcrusher.o
```

linker_bitcrusher.json

```
{
  "bitcrusher": {
    "links": {
      "slider1bitcrusher": "/bits",
      "slider2bitcrusher": "/wet_dry_mix",
      "toggle1bitcrusher": "/bypass"
    },
    "majorNo": "*"
  }
}
```

UI_bitcrusher.json

```
{
  "name": "bitcrusher",
  "pages": [
    {
      "name": "Main",
      "panels": [
        {
          "controls": [
            {
              "dataType": "u6,0",
              "defaultValue": 32,
              "linkerName": "slider1bitcrusher",
              "max": 32,
              "min": 0,
              "module": "bitcrusher",
              "style": "default",
              "title": "Bits",
              "type": "slider",
```

```
            "units": "bits"
          },
          {
            "dataType": "boolean",
            "defaultValue": 0,
            "linkerName": "toggle1bitcrusher",
            "max": 1,
            "min": 0,
            "module": "bitcrusher",
            "style": "default",
            "title": "Bypass",
            "type": "toggle",
            "units": "bypass"
          },
          {
            "dataType": "u8,7",
            "defaultValue": 0.5,
            "linkerName": "slider2bitcrusher",
            "max": 1,
            "min": 0,
            "module": "bitcrusher",
            "style": "default",
            "title": "Wet Dry Mix",
            "type": "slider",
            "units": "ratio"
          }
        ],
        "name": "Bitcrusher"
      }
    ]
  }
]
}
```

APPENDIX G

UBUNTU ROOT FILESYSTEM

The process of setting up Ubuntu Base for SoC FPGA development comprises the following steps, performed on an Ubuntu Linux desktop computer:

1. Download the Ubuntu Base archive for the armhf architecture and uncompress it
2. Make it so the desktop computer can run ARM executables in the Ubuntu Base filesystem during setup; this is done via user-mode emulation in QEMU[1]
   (a) Install `qemu_user_static`
   (b) Copy `qemu-arm-static` into `/usr/bin/` in the Ubuntu Base filesystem
3. Copy the desktop computer's DNS configuration file, `/etc/resolv.conf` into the Ubuntu Base filesystem to enable networking during setup
4. Mount `/proc`, `/sysfs`, `/dev/`, and `/dev/pts` in the Ubuntu Base filesystem
5. Change into the Ubuntu Base filesystem by using `chroot` to change the terminal's root directory
6. Install all desired packages
7. Create a user
8. Enable the serial login terminal
9. Exit the `chroot`
10. Unmount the previously mounted directories

The basic set of packages I install are

- text editors
  - vim
  - nano
  - emacs-nox
- build/dev tools
  - build-essential
  - kmod
  - busybox
  - python

[1] `https://wiki.debian.org/QemuUserEmulation`

  - python3
  - nodejs
- networking
  - iproute2
  - iputils-ping
  - net-tools
  - tftp
  - ssh
- misc
  - bash-completion

The following code listings show scripts that set up an Ubuntu Base rootfs according to the steps above.

```
#!/bin/bash

UBUNTU_VERSION=18.04.3
ROOTFS_ARCHIVE=ubuntu-base-$UBUNTU_VERSION-base-armhf.tar.gz
ROOT_DIR=ubuntu-base

# cleanup any previous rootfs
if [ -d $ROOT_DIR ]; then
    sudo rm -rf $ROOT_DIR
fi

# download the rootfs
if [ -f $ROOTFS_ARCHIVE ]; then
    echo "archive already exists; using that one..."
else
    wget http://cdimage.ubuntu.com/ubuntu-base/releases/$UBUNTU_VERSION/release/u
    ↪ buntu-base-$UBUNTU_VERSION-base-armhf.tar.gz
fi

# extract the rootfs
mkdir $ROOT_DIR
sudo tar -xpf $ROOTFS_ARCHIVE --directory=ubuntu-base

# install qemu-user-static so we can chroot into the armhf rootfs
sudo apt install qemu-user-static
sudo cp /usr/bin/qemu-arm-static $ROOT_DIR/usr/bin/

# copy our resolv.conf into the armhf rootfs so we have internet when we chroot
↪ into it
```

```
sudo cp /etc/resolv.conf $ROOT_DIR/etc/resolv.conf

# copy our package list into the armhf rootfs so we can install them once we
↪  chroot into the rootfs
cp packages $ROOT_DIR/

# mount stuff in the armhf rootfs
sudo mount -t proc /proc $ROOT_DIR/proc
sudo mount -t sysfs /sys $ROOT_DIR/sys
sudo mount -B /dev $ROOT_DIR/dev
sudo mount -B /dev/pts $ROOT_DIR/dev/pts

# copy the setup script into the armhf rootfs
cp setup_rootfs.sh $ROOT_DIR/

# chroot into the armhf rootfs and do the setup
sudo chroot $ROOT_DIR ./setup_rootfs.sh

# unmount directories in the armhf rootfs
sudo umount -l $ROOT_DIR/dev/pts
sudo umount -l $ROOT_DIR/dev
sudo umount -l $ROOT_DIR/proc
sudo umount -l $ROOT_DIR/sys

# clean up after ourselves
rm $ROOTFS_ARCHIVE

# package up the rootfs
sudo tar -czf rootfs.tar.gz $ROOT_DIR
```

```
#!/bin/bash

# install all of the packages
# https://askubuntu.com/questions/252734/apt-get-mass-install-packages-from-a-file
apt update && apt upgrade -y
xargs -a <(awk '! /^ *(#|$)/' packages) -r -- apt -y install

# setup our root user
# TODO: we really should not be using root for everything
#       we should create a normal user and disable the root account,
#       or at least give it a strong password...
echo "root:root" | chpasswd

# enable serial login getty
systemctl enable getty@ttyS0.service

# we're done; exit the chroot
exit
```

APPENDIX H

NETWORK BOOTING SETUP

This appendix assumes Ubuntu is the Linux distribution being used to setup the TFTP and NFS servers. Ubuntu 18.04 LTS was used here, but other versions should work. If using a different Linux distribution, package names and package managers may be different.

The basic ideas and framework for network booting were adapted from documentation and tutorials on rocketboards[1].

TFTP Server

The following steps install the TFTP server program:

1. Install TFTP

   ```
   sudo apt install tftpd-hpa
   ```

2. Verify that TFTP is running

   ```
   sudo systemctl status tftpd-hpa
   ```

3. If it's not running, start it

   ```
   sudo systemctl start tftpd-hpa
   ```

4. Create a tftp directory in `/srv/`; this is a common directory for storing files served by web servers, etc.

   ```
   sudo mkdir -p /srv/tftp
   ```

By default, directories in `/srv/*` are only writable by the root user. It is convenient to give our normal user permissions to the tftp directory tree with the following steps:

1. Add yourself to the `tftp` group

   ```
   sudo gpasswd -a user tftp
   ```

   where `user` is your username

2. Give the `tftp` group ownership of `/srv/tftp`

   ```
   sudo chown :tftp tftp
   ```

3. Ensure that subfolders/files inherit group ownership by setting the `setgid` (set group id) bit; this makes `tftp` the owner of all newly-created subfolders and files

   ```
   sudo chmod g+s tftp
   ```

[1] `rocketboards.org/`

4. Give the `tftp` group read-write-execute permissions

```
sudo chmod g+rwx tftp
```

5. Set the default access control list (acl) to give the `tftp` group read-write-execute permissions on directories and read-write permissions on files;

```
sudo setfacl -d --set u::rwx,g::rwx tftp
```

Now that proper permissions are set up, we can create directories to organize the files that will be served to the SoC FPGA:

1. Create a directory for the Linux Kernel image (zImage), e.g.

```
sudo mkdir -p /srv/tftp/kernel/de10-nano
```

2. Create a directory for the device tree blob and FPGA programming files, e.g.

```
sudo mkdir -p /srv/tftp/de10-nano
```

3. Create a directory for u-boot scripts, e.g.

```
sudo mkdir -p /srv/tftp/bootscripts
```

Finally, the TFTP server needs to be configured to serve the tftp directory:

1. Edit `/etc/default/tftpd-hpa` as root and change `TFTP_DIRECTORY="/var/lib/tftpboot"` to `TFTP_DIRECTORY="/srv/tftp/"`

The TFTP server needs to be restarted for the new configuration to take effect:

```
sudo systemctl restart tftpd-hpa
```

NFS Server

The NFS server is used to serve the root filesystem that the HPS uses. The following steps install and configure the NFS server:

1. Install the NFS server

```
sudo apt install nfs-kernel-server
```

2. Create a directory for the rootfs, e.g.

```
sudo mkdir -p /srv/nfs/de10-nano/ubuntu-rootfs
```

3. Edit `/etc/exports` as root; this file is the access control list for filesystems that may be exported to NFS clients Add the following line to `/etc/exports`:

```
/srv/nfs/de10-nano/ubuntu-rootfs/
↪  192.168.1.0/24(rw,no_subtree_check,sync,no_root_squash)
```

NOTE: 192.168.1.0/24 allows any computer on the 192.168.1.* subnet to mount the `ubuntu-rootfs` NFS folder. The SoC FPGA must be assigned an IP address on this subnet. If your network has a different subnet, `/etc/exports` will have to be changed accordingly.

4. Restart the NFS server

   ```
   sudo service nfs-kernel-server restart
   ```

5. Make sure your directory is exported

   ```
   sudo exportfs -v
   ```

   This should print the line you added to `/etc/exports`

Permissions could be set up to give our normal user access to the NFS directories, but certain permissions in the rootfs should not be changed, so permissions are left as is here. When working with the NFS directories as non-root user, `sudo` must be used.

U-Boot

To support network booting, some environment variables were changed and created in U-Boot. The following listing shows all of the U-Boot variables for the standard SD card images used in this work; the variables that need to be defined for network booting are highlighted.

```
1  axibridge=ffd0501c
2  axibridge_handoff=0x00000000
3  baudrate=115200
4  bootargs=console=ttyS0,115200 ip=dhcp root=/dev/nfs rw
   ↪  nfsroot=192.168.1.191:/srv/nfs/de10nano/ubuntu-rootfs,vers=4,tcp nfsrootdebug
   ↪  earlyprintk=serial
5  bootcmd=if $nfsboot; then run bootcmdnfs; else run bootcmdmmc; fi
6  bootcmdmmc=run callscript; run mmcload; run mmcboot
7  bootcmdnfs=tftp 0x100000 de10nano/bootscripts/${bootscript}; source 0x100000
8  bootdelay=5
9  bootimage=zImage
10 bootimagesize=0x600000
11 bootnfs=bootz ${kerneladdr} - ${dtbaddr}
12 bootscript=lab3.scr
13 bridge_disable=mw $fpgaintf 0; mw $fpga2sdram 0; go $fpga2sdram_apply; mw
   ↪  $axibridge 0; mw $l3remap 0x1
```

```
14 bridge_enable_handoff=mw $fpgaintf ${fpgaintf_handoff}; go $fpga2sdram_apply; mw
   ↪ $fpga2sdram ${fpga2sdram_handoff}; mw $axibridge ${axibridge_handoff}; mw
   ↪ $l3remap ${l3remap_handoff}
15 callscript=if fatload mmc 0:1 $fpgadata $scriptfile;then source $fpgadata; else
   ↪ echo Optional boot script not found. Continuing to boot normally; fi;
16 dtb-image=soc_system.dtb
17 dtbaddr=0x00000100
18 ethact=mii0
19 ethaddr=de:ad:be:ef:01:23
20 fdtaddr=0x00000100
21 fdtimage=socfpga.dtb
22 fdtimagesize=0x7000
23 fileaddr=100000
24 filesize=4E3
25 fpga=0
26 fpga-image=soc_system.rbf
27 fpga2sdram=ffc25080
28 fpga2sdram_apply=3ff79580
29 fpga2sdram_handoff=0x00000000
30 fpgadata=0x2000000
31 fpgadatasize=0x700000
32 fpgaintf=ffd08028
33 fpgaintf_handoff=0x00000001
34 get-dtb=tftp ${dtbaddr} ${tftp-de10nano-dir}/${dtb-image}
35 get-fpgadata=tftp ${fpgadata} ${tftp-de10nano-dir}/${fpga-image}
36 get-kernel=tftp ${kerneladdr} ${tftp-kernel-dir}/${kernel-image}
37 ipaddr=192.168.0.20
38 kernel-image=zImage
39 kerneladdr=0x8000
40 l3remap=ff800000
41 l3remap_handoff=0x00000019
42 load-fpga=fpga load 0 ${fpgadata} ${fpgadatasize}
43 loadaddr=0x8000
44 micrel-ksz9021-clk-skew=0xf0f0
45 micrel-ksz9021-data-skew=0x0
46 mmcboot=setenv bootargs console=ttys0,115200 earlyprintk-serial root-$(mmcroot}
   ↪ rw rootwait;bootz ${loadaddr} - ${fdtaddr}
47 mmcload=mmc rescan;${mmcloadcmd} mmc 0:${mmcloadpart} ${loadaddr}
   ↪ ${bootimage};${mmcloadcmd} mmc 0:${mmcloadpart} ${fdtaddr} ${fdtimage}
48 mmcloadcmd=fatload
49 mmcloadpart=1
50 mmcroot=/dev/mmcblk0p2
51 nandboot=setenv bootargs console=ttyS0,115200 root=${nandroot} rw
   ↪ rootfstype=${nandrootfstype};bootz ${loadaddr} - ${fdtaddr}
52 nandbootimageaddr=0x120000
53 nandfdtaddr=0xA0000
54 nandload=nand read ${loadaddr} ${nandbootimageaddr} ${bootimagesize};nand read
   ↪ ${fdtaddr} ${nandfdtaddr} ${fdtimagesize}
55 nandroot=/dev/mtdblock1
56 nandrootfstype=jffs2
57 netboot=dhcp ${bootimage} ; tftp ${fdtaddr} ${fdtimage} ; run ramboot
```

```
58 netmask=255.255.255.0
59 nfs-rootfs-dir=/srv/nfs/de10nano/ubuntu-rootfs
60 nfsboot=true
61 nfsip=192.168.0.127
62 qspiboot=setenv bootargs console=ttyS0,115200 root=${qspiroot} rw
   ↪ rootfstype=${qspirootfstype};bootz ${loadaddr} - ${fdtaddr}
63 qspibootimageaddr=0xa0000
64 qspifdtaddr=0x50000
65 qspiload=sf probe ${qspiloadcs};sf read ${loadaddr} ${qspibootimageaddr}
   ↪ ${bootimagesize};sf read ${fdtaddr} ${qspifdtaddr} ${fdtimagesize};
66 qspiloadcs=0
67 qspiroot=/dev/mtdblock1
68 qspirootfstype=jffs2
69 ramboot=setenv bootargs console=ttyS0,115200;bootz ${loadaddr} - ${fdtaddr}
70 scriptfile=u-boot.scr
71 serverip=192.168.0.127
72 setenv_ethaddr_eeprom=3ff9ac0d
73 stderr=serial
74 stdin=serial
75 stdout=serial
76 tftp-kernel-dir=de10nano/kernel
77 tftp-project-dir=de10nano/AudioMini_Passthrough
78 verify=n
```

The overall flow of network booting is as follows. Looking at the `bootcmd` variable in line 5 of the previous listing, we see that if the `nfsboot` variable is `true`, then the bootscript defined in the `bootscript` variable is downloaded via TFTP and executed. This bootscript defines all of the commands needed to continue booting from the network. An example bootscript is shown in the following listing.

```
1  # file directories
2  setenv tftp-kernel-dir de10nano/kernel
3  setenv tftp-project-dir de10nano/AudioMini_Passthrough
4  setenv nfs-rootfs-dir /srv/nfs/de10nano/ubuntu-rootfs
5
6  # kernel bootargs
7  setenv bootargs console=ttyS0,115200 ip=${ipaddr} root=/dev/nfs rw
   ↪ nfsroot=${nfsip}:${nfs-rootfs-dir},vers=4,tcp nfsrootdebug earlyprintk=serial
8
9  # file names
10 setenv fpga-image soc_system.rbf
11 setenv dtb-image soc_system.dtb
12 setenv kernel-image zImage
13
14 # memory addresses where files get loaded into
15 setenv fpgadata 0x2000000
16 setenv fpgadatasize 0x700000
17 setenv dtbaddr 0x00000100
18 setenv kerneladdr 0x8000
```

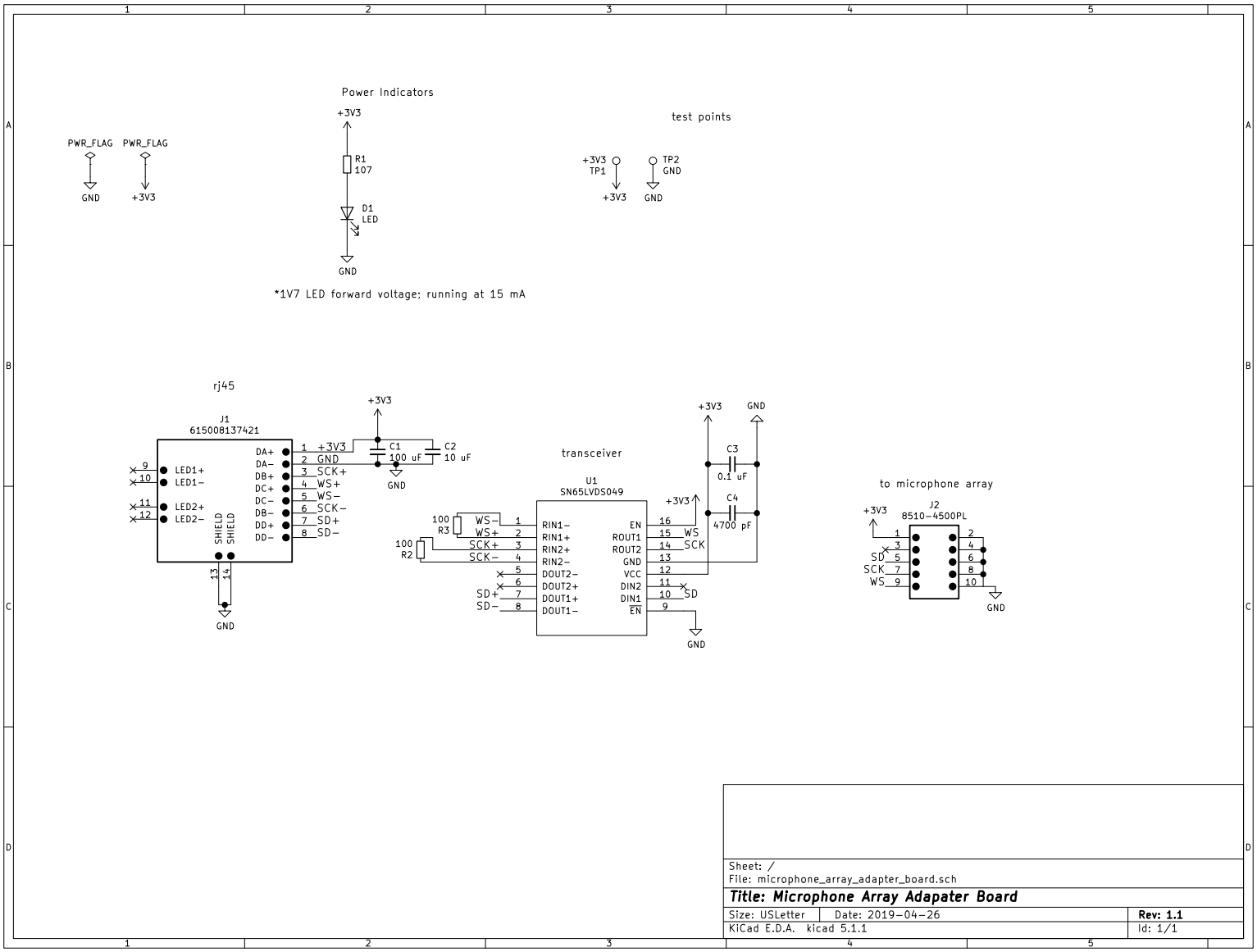
```
19
20 # commands to get files, configure the fpga, and load the kernel
21 setenv get-fpgadata 'tftp ${fpgadata} ${tftp-project-dir}/${fpga-image}'
22 setenv get-dtb 'tftp ${dtbaddr} ${tftp-project-dir}/${dtb-image}'
23 setenv get-kernel 'tftp ${kerneladdr} ${tftp-kernel-dir}/${kernel-image}'
24 setenv load-fpga 'fpga load 0 ${fpgadata} ${fpgadatasize}'
25 setenv bootnfs 'bootz ${kerneladdr} - ${dtbaddr}'
26
27 # get all of the files and boot the device
28 run get-fpgadata load-fpga get-dtb get-kernel bridge_enable_handoff bootnfs
```

U-Boot scripts, like those shown in the previous listing, need to be compiled in order to be used by U-Boot. This is done with `mkimage`, which is available in the `u-boot-tools` package in Ubuntu. The following line will compile a boot script named `boot.script`:

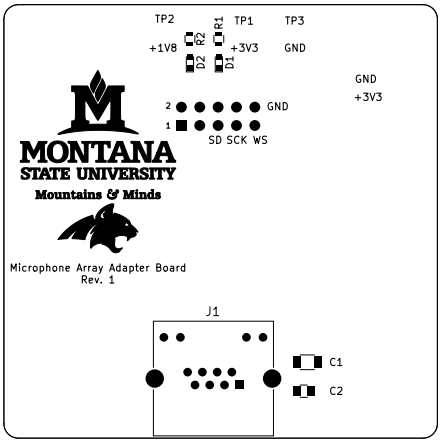
```
sudo mkimage -A arm -O u-boot -T script -C none -a 0 -e 0 -n "TFTP
↪  Boot Script" -d boot.script boot.scr
```

APPENDIX I

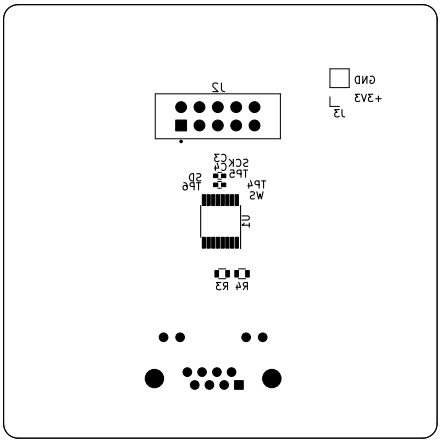
MICROPHONE ARRAY ADATPER BOARD DESIGN FILES


Power Indicators
PWR_FLAG
PWR_FLAG
GND
+3V3
+3V3
R1
107
D1
LED
GND
*1V7 LED forward voltage; running at 15 mA
test points
+3V3
TP1
TP2
GND
+3V3
GND
rj45
J1
615008137421
LED1+
LED1−
LED2+
LED2−
SHIELD
SHIELD
DA+
DA−
DB+
DC+
DC−
DB−
DD+
DD−
+3V3
GND
SCK+
WS+
WS−
SCK−
SD+
SD−
C1
100 uF
C2
10 uF
GND
transceiver
U1
SN65LVDS049
100
R3
100
R2
WS−
WS+
SCK+
SCK−
SD+
SD−
RIN1−
RIN1+
RIN2+
RIN2−
DOUT2−
DOUT2+
DOUT1+
DOUT1−
EN
ROUT1
ROUT2
GND
VCC
DIN2
DIN1
EN
WS
SCK
SD
+3V3
GND
C3
0.1 uF
C4
4700 pF
to microphone array
+3V3
J2
8510−4500PL
SD
SCK
WS
GND
Sheet: /
File: microphone_array_adapter_board.sch
Title: Microphone Array Adapater Board
Size: USLetter
Date: 2019−04−26
Rev: 1.1
KiCad E.D.A. kicad 5.1.1
Id: 1/1

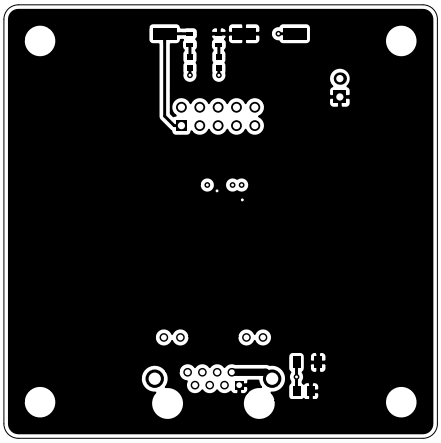
top view



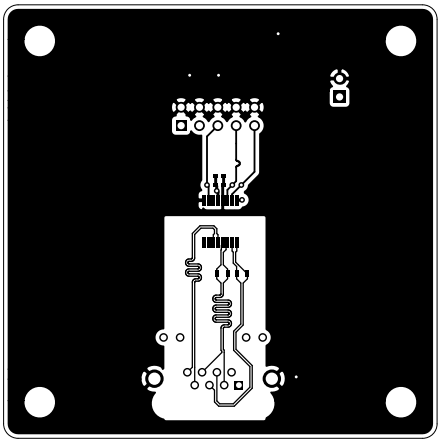
bottom view



top layer

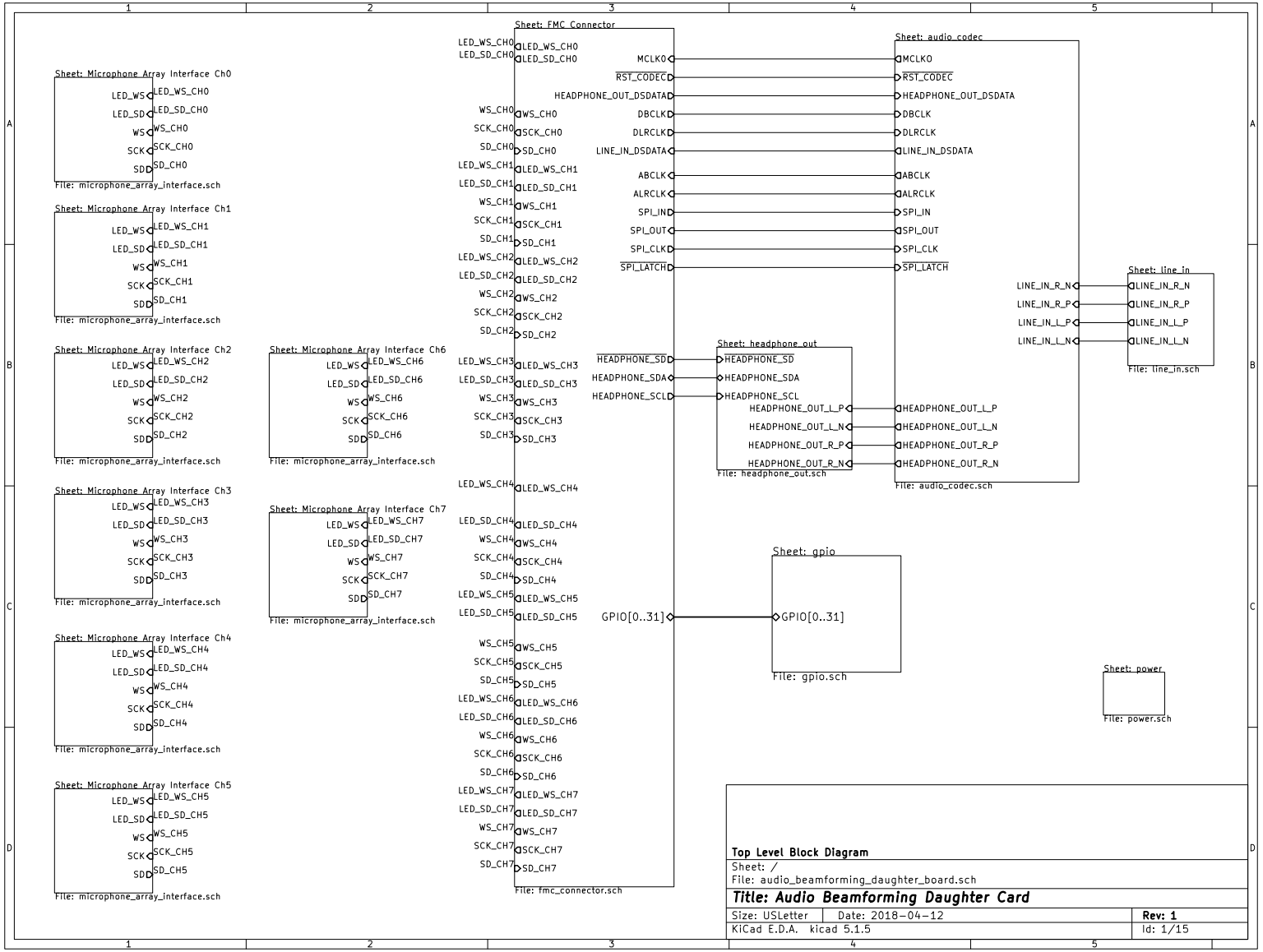


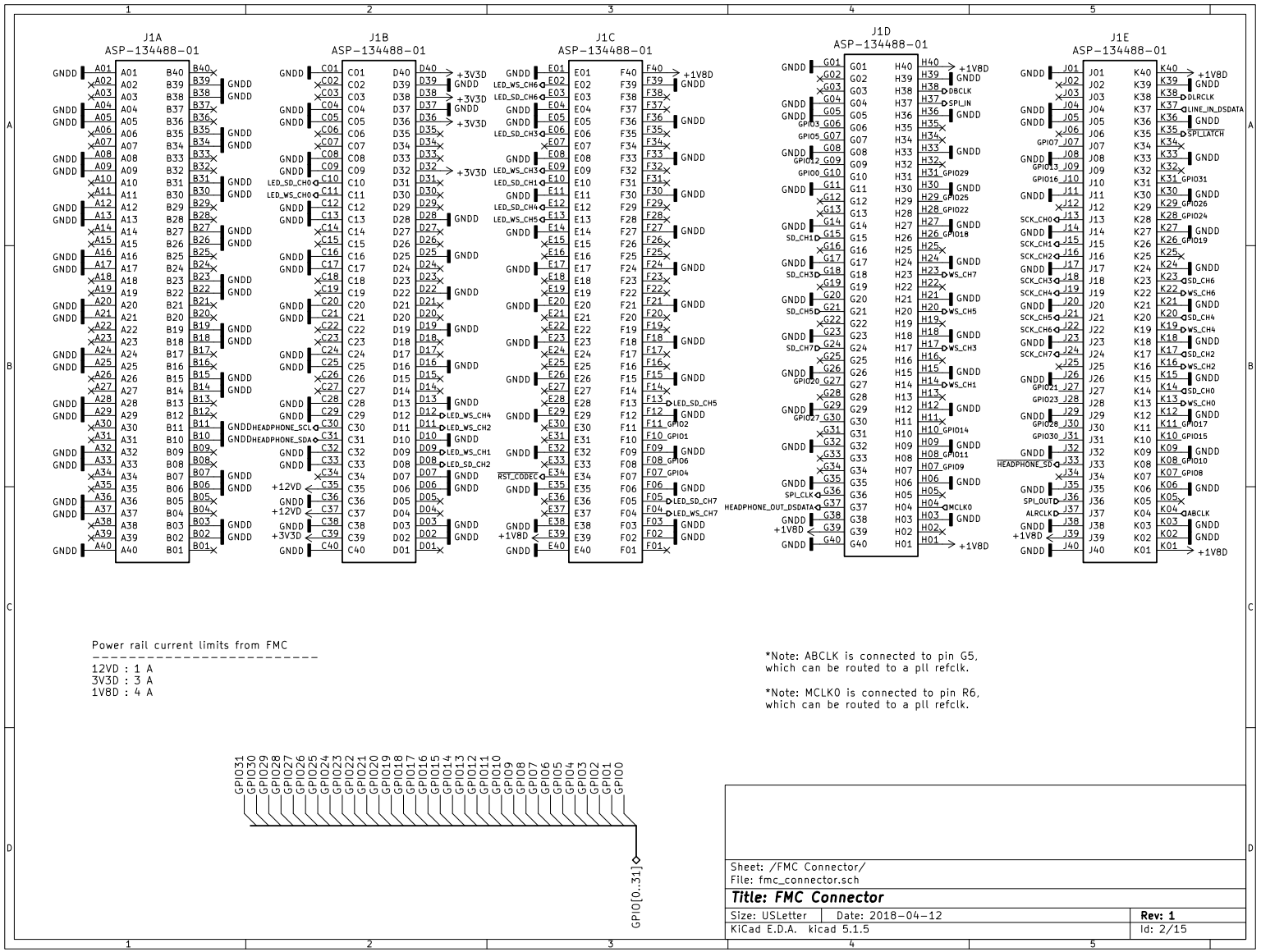
bottom layer

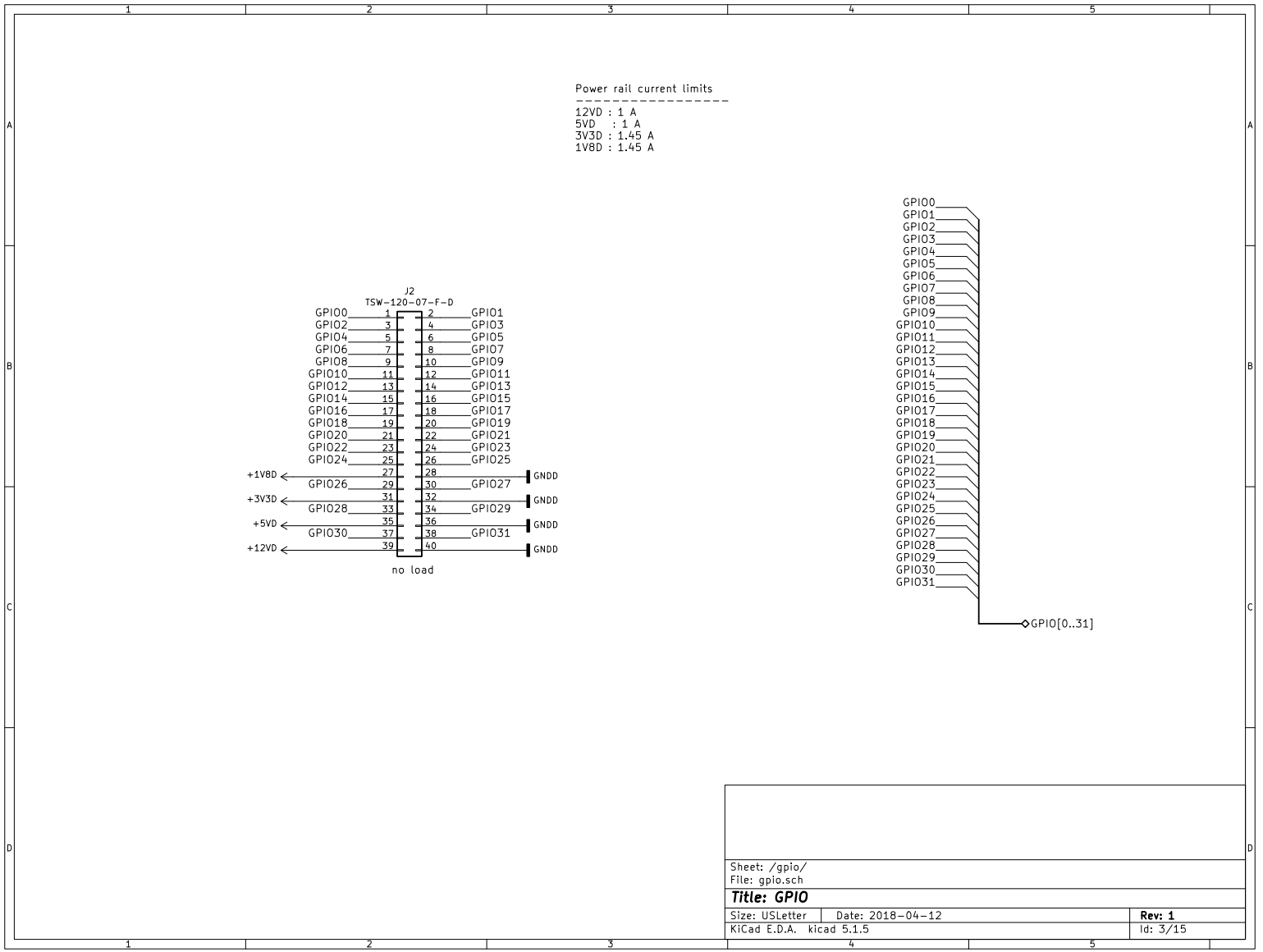


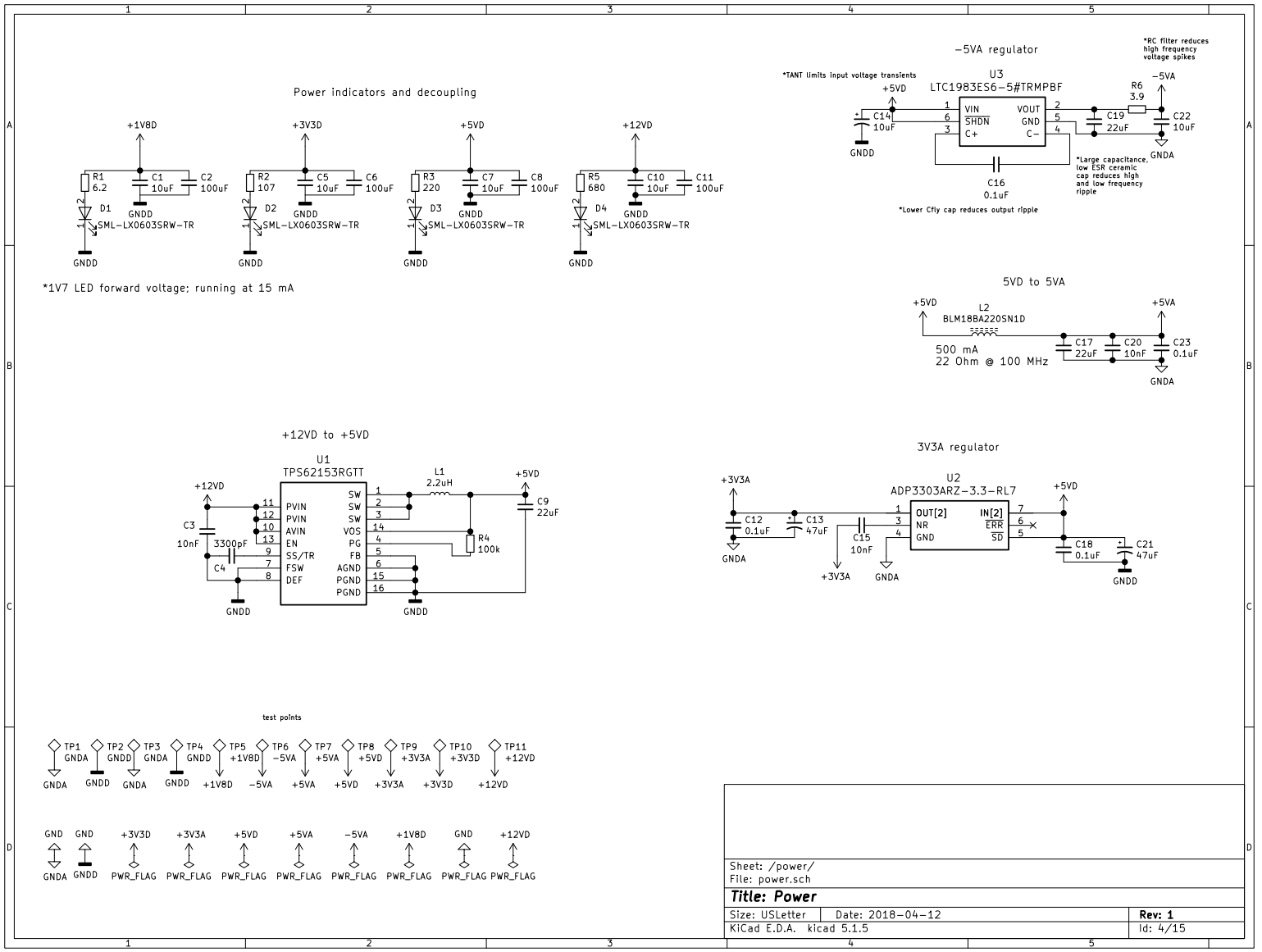
APPENDIX J

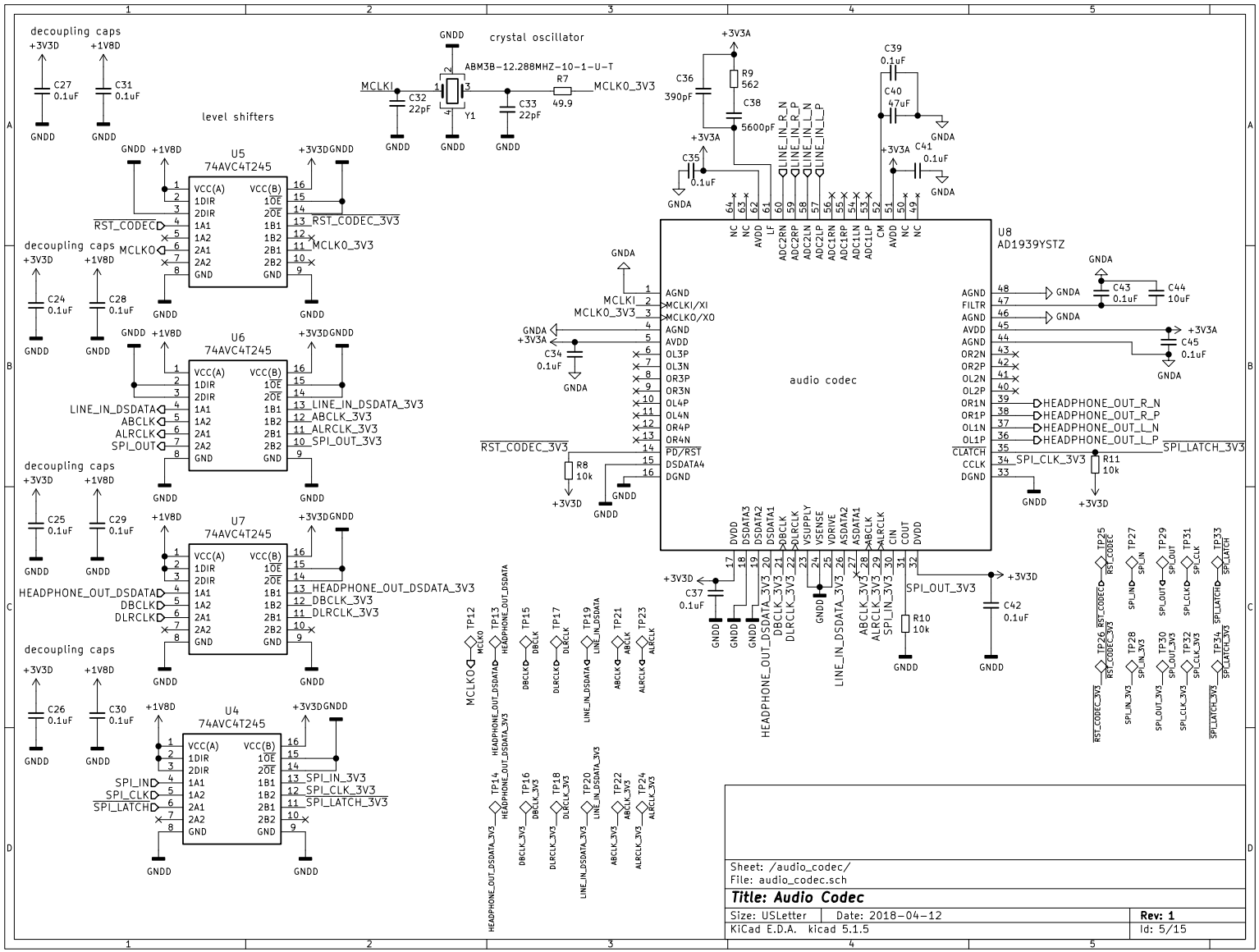
AUDIO BEAMFOMRING DAUGHTER CARD DESIGN FILES

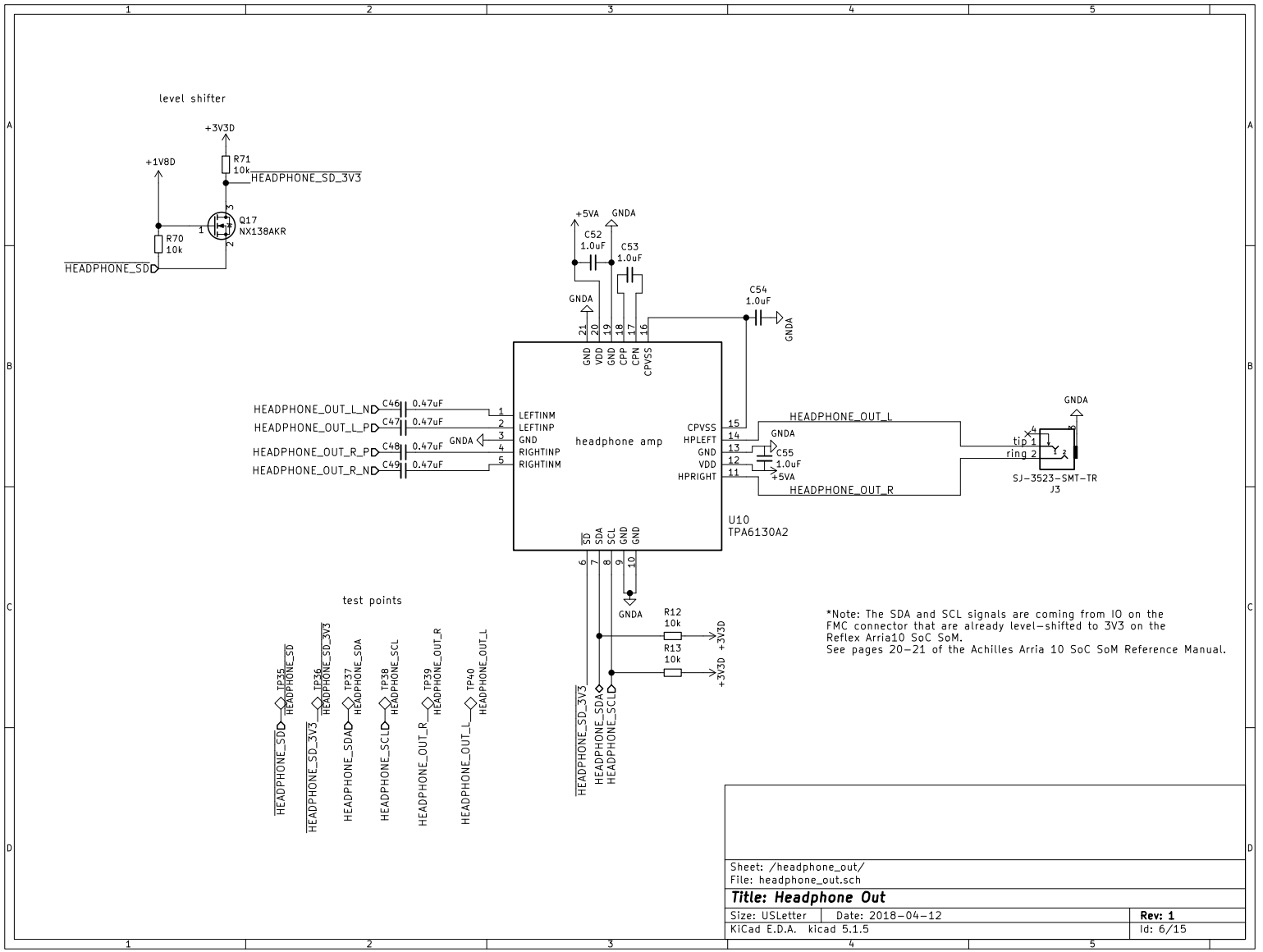

Sheet: Microphone Array Interface Ch0
LED_WS
LED_SD
WS
SCK
SDD
LED_WS_CH0
LED_SD_CH0
WS_CH0
SCK_CH0
SD_CH0
File: microphone_array_interface.sch
Sheet: Microphone Array Interface Ch1
LED_WS_CH1
LED_SD_CH1
WS_CH1
SCK_CH1
SD_CH1
File: microphone_array_interface.sch
Sheet: Microphone Array Interface Ch2
LED_WS_CH2
LED_SD_CH2
WS_CH2
SCK_CH2
SD_CH2
File: microphone_array_interface.sch
Sheet: Microphone Array Interface Ch3
LED_WS_CH3
LED_SD_CH3
WS_CH3
SCK_CH3
SD_CH3
File: microphone_array_interface.sch
Sheet: Microphone Array Interface Ch4
LED_WS_CH4
LED_SD_CH4
WS_CH4
SCK_CH4
SD_CH4
File: microphone_array_interface.sch
Sheet: Microphone Array Interface Ch5
LED_WS_CH5
LED_SD_CH5
WS_CH5
SCK_CH5
SD_CH5
File: microphone_array_interface.sch
Sheet: Microphone Array Interface Ch6
LED_WS_CH6
LED_SD_CH6
WS_CH6
SCK_CH6
SD_CH6
File: microphone_array_interface.sch
Sheet: Microphone Array Interface Ch7
LED_WS_CH7
LED_SD_CH7
WS_CH7
SCK_CH7
SD_CH7
File: microphone_array_interface.sch
Sheet: FMC Connector
LED_WS_CH0
LED_SD_CH0
WS_CH0
SCK_CH0
SD_CH0
LED_WS_CH1
LED_SD_CH1
WS_CH1
SCK_CH1
SD_CH1
LED_WS_CH2
LED_SD_CH2
WS_CH2
SCK_CH2
SD_CH2
LED_WS_CH3
LED_SD_CH3
WS_CH3
SCK_CH3
SD_CH3
LED_WS_CH4
LED_SD_CH4
WS_CH4
SCK_CH4
SD_CH4
LED_WS_CH5
LED_SD_CH5
WS_CH5
SCK_CH5
SD_CH5
LED_WS_CH6
LED_SD_CH6
WS_CH6
SCK_CH6
SD_CH6
LED_WS_CH7
LED_SD_CH7
WS_CH7
SCK_CH7
SD_CH7
MCLKO
RST_CODEC
HEADPHONE_OUT_DSDATA
DBCLK
DLRCLK
LINE_IN_DSDATA
ABCLK
ALRCLK
SPI_IN
SPI_OUT
SPI_CLK
SPI_LATCH
HEADPHONE_SD
HEADPHONE_SDA
HEADPHONE_SCL
GPIO[0..31]
File: fmc_connector.sch
Sheet: audio_codec
MCLKO
RST_CODEC
HEADPHONE_OUT_DSDATA
DBCLK
DLRCLK
LINE_IN_DSDATA
ABCLK
ALRCLK
SPI_IN
SPI_OUT
SPI_CLK
SPI_LATCH
HEADPHONE_OUT_L_P
HEADPHONE_OUT_L_N
HEADPHONE_OUT_R_P
HEADPHONE_OUT_R_N
LINE_IN_R_N
LINE_IN_R_P
LINE_IN_L_P
LINE_IN_L_N
File: audio_codec.sch
Sheet: headphone_out
HEADPHONE_SD
HEADPHONE_SDA
HEADPHONE_SCL
HEADPHONE_OUT_L_P
HEADPHONE_OUT_L_N
HEADPHONE_OUT_R_P
HEADPHONE_OUT_R_N
File: headphone_out.sch
Sheet: line_in
LINE_IN_R_N
LINE_IN_R_P
LINE_IN_L_P
LINE_IN_L_N
File: line_in.sch
Sheet: gpio
GPIO[0..31]
File: gpio.sch
Sheet: power
File: power.sch
Top Level Block Diagram
Sheet: /
File: audio_beamforming_daughter_board.sch
Title: Audio Beamforming Daughter Card
Size: USLetter
Date: 2018-04-12
Rev: 1
KiCad E.D.A. kicad 5.1.5
Id: 1/15

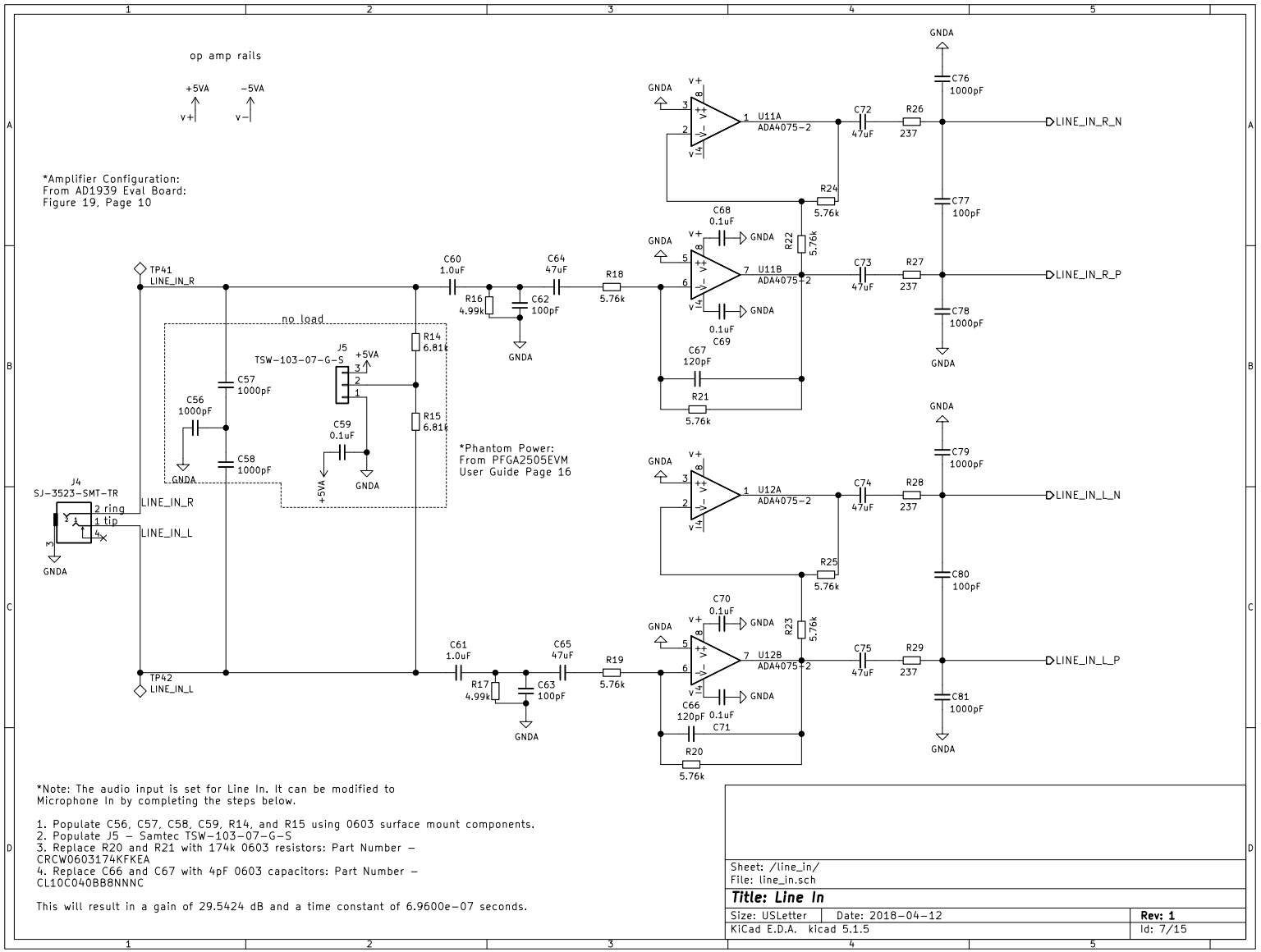

J1A
ASP-134488-01
J1B
ASP-134488-01
J1C
ASP-134488-01
J1D
ASP-134488-01
J1E
ASP-134488-01
Power rail current limits from FMC
12VD : 1 A
3V3D : 3 A
1V8D : 4 A
*Note: ABCLK is connected to pin G5, which can be routed to a pll refclk.
*Note: MCLK0 is connected to pin R6, which can be routed to a pll refclk.
GPIO[0..31]
Sheet: /FMC Connector/
File: fmc_connector.sch
Title: FMC Connector
Size: USLetter
Date: 2018-04-12
Rev: 1
KiCad E.D.A. kicad 5.1.5
Id: 2/15

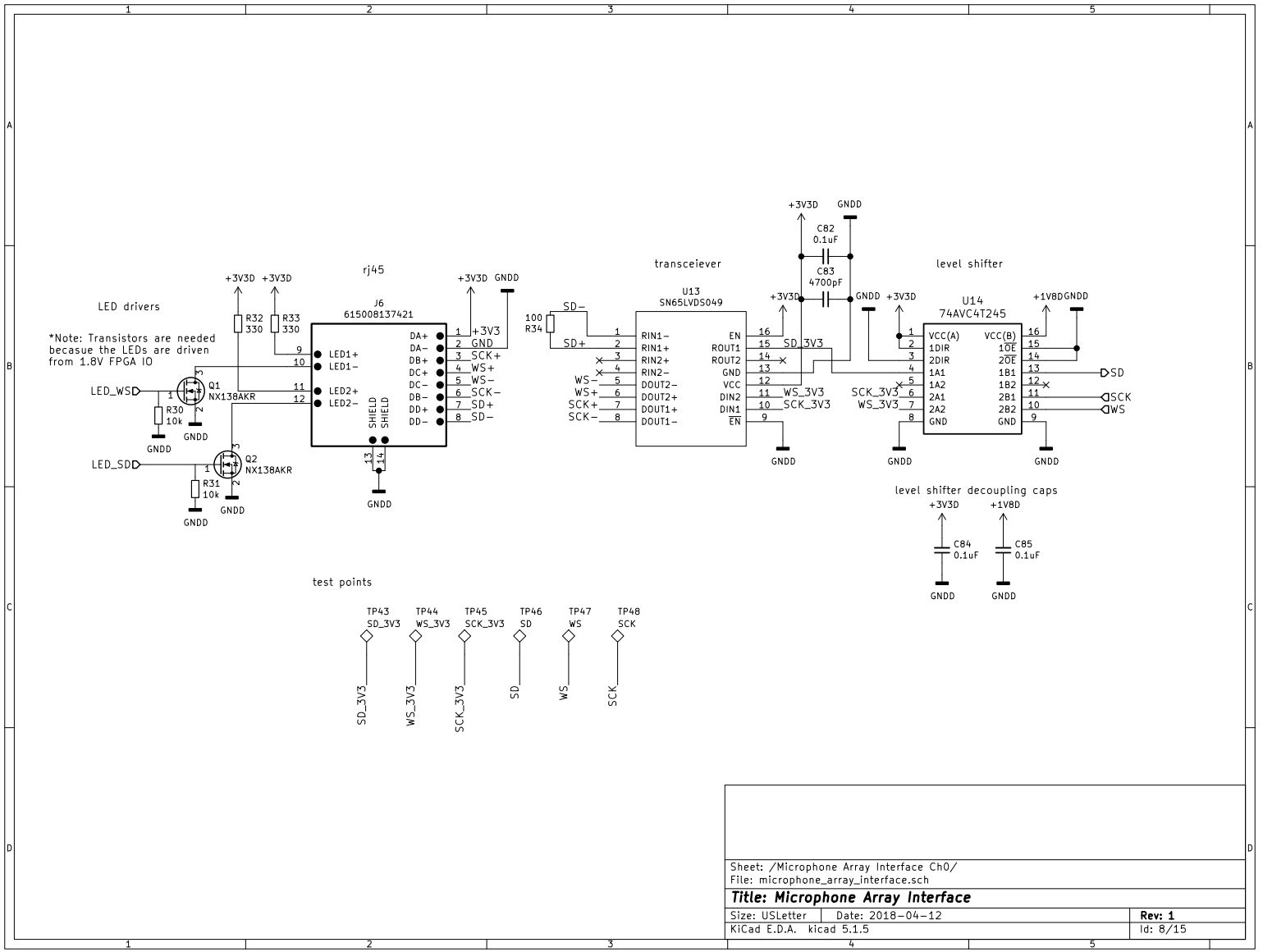

Power rail current limits
12VD : 1 A
5VD : 1 A
3V3D : 1.45 A
1V8D : 1.45 A
J2
TSW-120-07-F-D
GPIO0 1 2 GPIO1
GPIO2 3 4 GPIO3
GPIO4 5 6 GPIO5
GPIO6 7 8 GPIO7
GPIO8 9 10 GPIO9
GPIO10 11 12 GPIO11
GPIO12 13 14 GPIO13
GPIO14 15 16 GPIO15
GPIO16 17 18 GPIO17
GPIO18 19 20 GPIO19
GPIO20 21 22 GPIO21
GPIO22 23 24 GPIO23
GPIO24 25 26 GPIO25
+1V8D 27 28 GNDD
GPIO26 29 30 GPIO27
+3V3D 31 32 GNDD
GPIO28 33 34 GPIO29
+5VD 35 36 GNDD
GPIO30 37 38 GPIO31
+12VD 39 40 GNDD
no load
GPIO0
GPIO1
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
GPIO7
GPIO8
GPIO9
GPIO10
GPIO11
GPIO12
GPIO13
GPIO14
GPIO15
GPIO16
GPIO17
GPIO18
GPIO19
GPIO20
GPIO21
GPIO22
GPIO23
GPIO24
GPIO25
GPIO26
GPIO27
GPIO28
GPIO29
GPIO30
GPIO31
GPIO[0..31]
Sheet: /gpio/
File: gpio.sch
Title: GPIO
Size: USLetter
Date: 2018-04-12
Rev: 1
KiCad E.D.A. kicad 5.1.5
Id: 3/15


Power indicators and decoupling
+1V8D
+3V3D
+5VD
+12VD
R1 6.2
R2 107
R3 220
R5 680
C1 10uF
C2 100uF
C5 10uF
C6 100uF
C7 10uF
C8 100uF
C10 10uF
C11 100uF
D1 SML-LX0603SRW-TR
D2 SML-LX0603SRW-TR
D3 SML-LX0603SRW-TR
D4 SML-LX0603SRW-TR
GNDD
*1V7 LED forward voltage; running at 15 mA
−5VA regulator
*TANT limits input voltage transients
*RC filter reduces high frequency voltage spikes
U3
LTC1983ES6-5#TRMPBF
VIN
SHDN
C+
VOUT
GND
C−
R6 3.9
C14 10uF
C19 22uF
C22 10uF
C16 0.1uF
−5VA
GNDA
*Large capacitance, low ESR ceramic cap reduces high and low frequency ripple
*Lower Cfly cap reduces output ripple
5VD to 5VA
L2
BLM18BA220SN1D
500 mA
22 Ohm @ 100 MHz
C17 22uF
C20 10nF
C23 0.1uF
+12VD to +5VD
U1
TPS62153RGTT
PVIN
AVIN
EN
SS/TR
FSW
DEF
SW
VOS
PG
FB
AGND
PGND
L1 2.2uH
C3 10nF
C4 3300pF
C9 22uF
R4 100k
3V3A regulator
U2
ADP3303ARZ-3.3-RL7
OUT[2]
NR
GND
IN[2]
ERR
SD
C12 0.1uF
C13 47uF
C15 10nF
C18 0.1uF
C21 47uF
test points
TP1 GNDA
TP2 GNDD
TP3 GNDA
TP4 GNDD
TP5 +1V8D
TP6 −5VA
TP7 +5VA
TP8 +5VD
TP9 +3V3A
TP10 +3V3D
TP11 +12VD
GND
PWR_FLAG
Sheet: /power/
File: power.sch
Title: Power
Size: USLetter
Date: 2018-04-12
Rev: 1
KiCad E.D.A. kicad 5.1.5
Id: 4/15


decoupling caps
level shifters
crystal oscillator
ABM3B-12.288MHZ-10-1-U-T
MCLKI
MCLK0_3V3
U5
74AVC4T245
U6
74AVC4T245
U7
74AVC4T245
U4
74AVC4T245
U8
AD1939YSTZ
audio codec
RST_CODEC
RST_CODEC_3V3
MCLK0
LINE_IN_DSDATA
LINE_IN_DSDATA_3V3
ABCLK
ALRCLK
SPI_OUT
SPI_OUT_3V3
HEADPHONE_OUT_DSDATA
HEADPHONE_OUT_DSDATA_3V3
DBCLK
DLRCLK
SPI_IN
SPI_IN_3V3
SPI_CLK
SPI_CLK_3V3
SPI_LATCH
SPI_LATCH_3V3
HEADPHONE_OUT_R_N
HEADPHONE_OUT_R_P
HEADPHONE_OUT_L_N
HEADPHONE_OUT_L_P
LINE_IN_R_N
LINE_IN_R_P
LINE_IN_L_N
LINE_IN_L_P
Sheet: /audio_codec/
File: audio_codec.sch
Title: Audio Codec
Size: USLetter
Date: 2018-04-12
KiCad E.D.A. kicad 5.1.5
Rev: 1
Id: 5/15


level shifter
+3V3D
+1V8D
R71
10k
HEADPHONE_SD_3V3
Q17
NX138AKR
R70
10k
HEADPHONE_SD
+5VA
GNDA
C52
1.0uF
C53
1.0uF
C54
1.0uF
GND
VDD
GND
CPP
CPN
CPVSS
21
20
19
18
17
16
HEADPHONE_OUT_L_ND
HEADPHONE_OUT_L_PD
HEADPHONE_OUT_R_PD
HEADPHONE_OUT_R_ND
C46
C47
C48
C49
0.47uF
LEFTINM
LEFTINP
GND
RIGHTINP
RIGHTINM
headphone amp
CPVSS
HPLEFT
GND
VDD
HPRIGHT
HEADPHONE_OUT_L
HEADPHONE_OUT_R
C55
1.0uF
tip 1
ring 2
SJ-3523-SMT-TR
J3
U10
TPA6130A2
SD
SDA
SCL
R12
10k
R13
10k
+3V3D
test points
TP35
TP36
TP37
TP38
TP39
TP40
HEADPHONE_SD
HEADPHONE_SD_3V3
HEADPHONE_SDA
HEADPHONE_SCL
HEADPHONE_OUT_R
HEADPHONE_OUT_L
*Note: The SDA and SCL signals are coming from IO on the
FMC connector that are already level-shifted to 3V3 on the
Reflex Arria10 SoC SoM.
See pages 20-21 of the Achilles Arria 10 SoC SoM Reference Manual.
Sheet: /headphone_out/
File: headphone_out.sch
Title: Headphone Out
Size: USLetter
Date: 2018-04-12
Rev: 1
KiCad E.D.A. kicad 5.1.5
Id: 6/15


op amp rails
+5VA
−5VA
v+
v−
*Amplifier Configuration:
From AD1939 Eval Board:
Figure 19, Page 10
TP41
LINE_IN_R
TP42
LINE_IN_L
C60
1.0uF
C64
47uF
R18
5.76k
R16
4.99k
C62
100pF
GNDA
no load
J5
TSW-103-07-G-S
+5VA
R14
6.81k
R15
6.81k
C57
1000pF
C56
1000pF
C58
1000pF
C59
0.1uF
*Phantom Power:
From PFGA2505EVM
User Guide Page 16
J4
SJ-3523-SMT-TR
2 ring
1 tip
C61
1.0uF
C65
47uF
R19
5.76k
R17
4.99k
C63
100pF
U11A
ADA4075-2
U11B
ADA4075-2
U12A
ADA4075-2
U12B
ADA4075-2
C68
0.1uF
C69
0.1uF
C67
120pF
R21
5.76k
R22
5.76k
C70
0.1uF
C71
0.1uF
C66
120pF
R20
5.76k
R23
5.76k
R24
5.76k
R25
5.76k
C72
47uF
C73
47uF
C74
47uF
C75
47uF
R26
237
R27
237
R28
237
R29
237
C76
1000pF
C77
100pF
C78
1000pF
C79
1000pF
C80
100pF
C81
1000pF
LINE_IN_R_N
LINE_IN_R_P
LINE_IN_L_N
LINE_IN_L_P
*Note: The audio input is set for Line In. It can be modified to
Microphone In by completing the steps below.
1. Populate C56, C57, C58, C59, R14, and R15 using 0603 surface mount components.
2. Populate J5 – Samtec TSW-103-07-G-S
3. Replace R20 and R21 with 174k 0603 resistors: Part Number –
CRCW0603174KFKEA
4. Replace C66 and C67 with 4pF 0603 capacitors: Part Number –
CL10C040BB8NNNC
This will result in a gain of 29.5424 dB and a time constant of 6.9600e-07 seconds.
Sheet: /line_in/
File: line_in.sch
Title: Line In
Size: USLetter
Date: 2018-04-12
Rev: 1
KiCad E.D.A. kicad 5.1.5
Id: 7/15

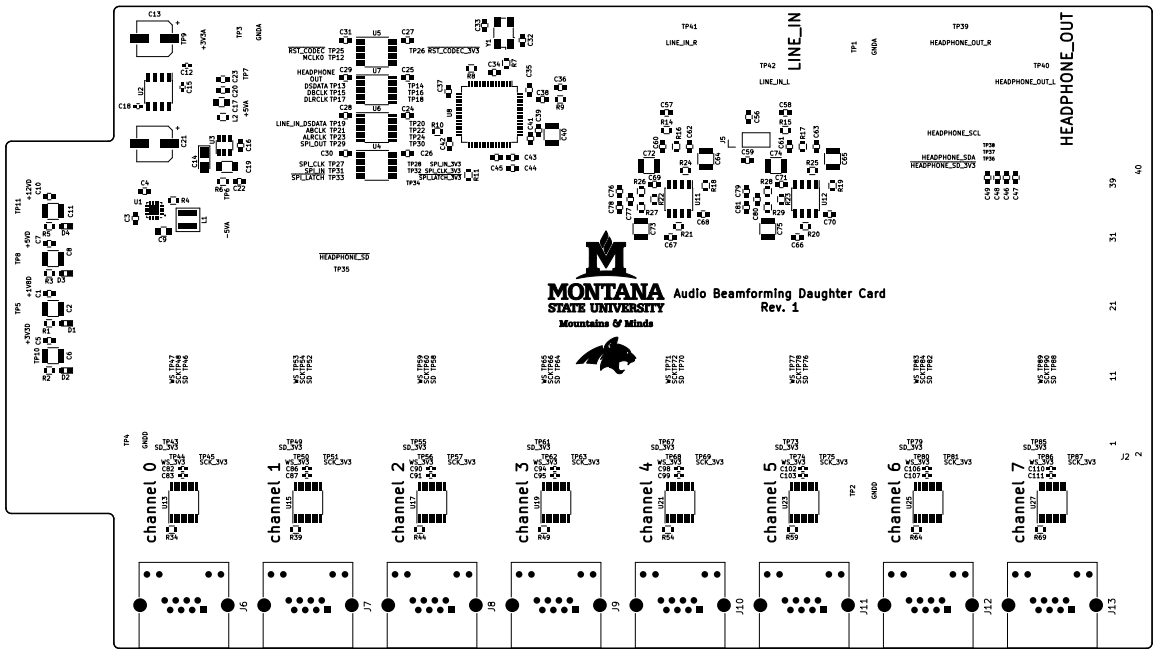


Sheet: /Microphone Array Interface Ch0/
File: microphone_array_interface.sch

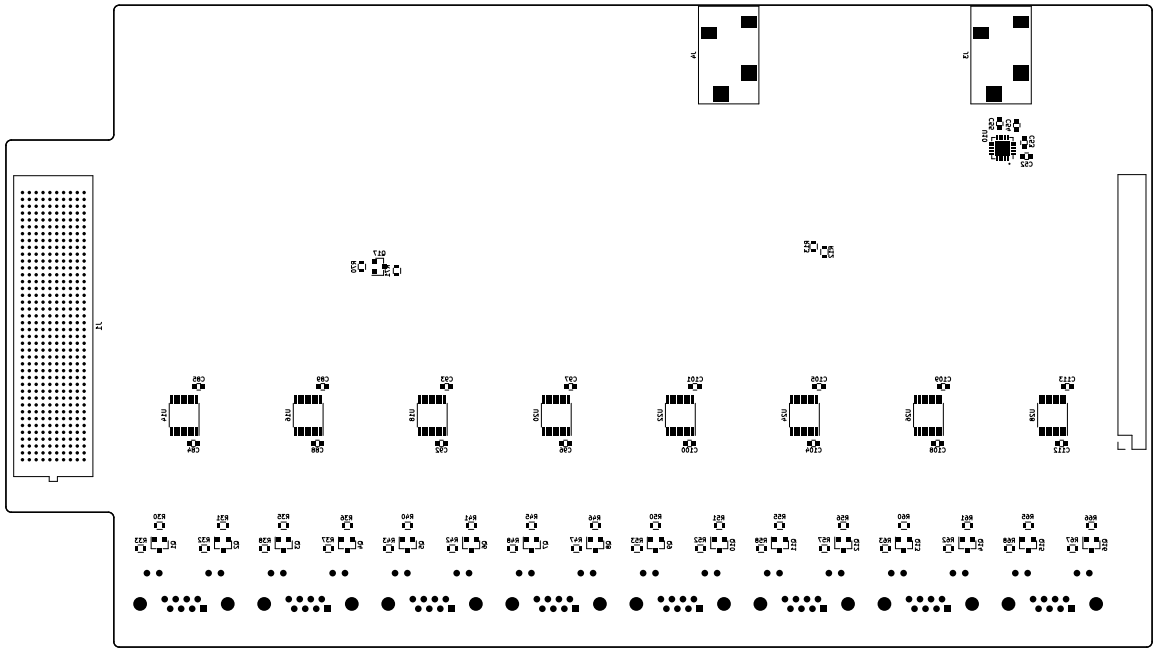
Title: Microphone Array Interface

Size: USLetter	Date: 2018-04-12	**Rev: 1**
KiCad E.D.A. kicad 5.1.5		Id: 8/15

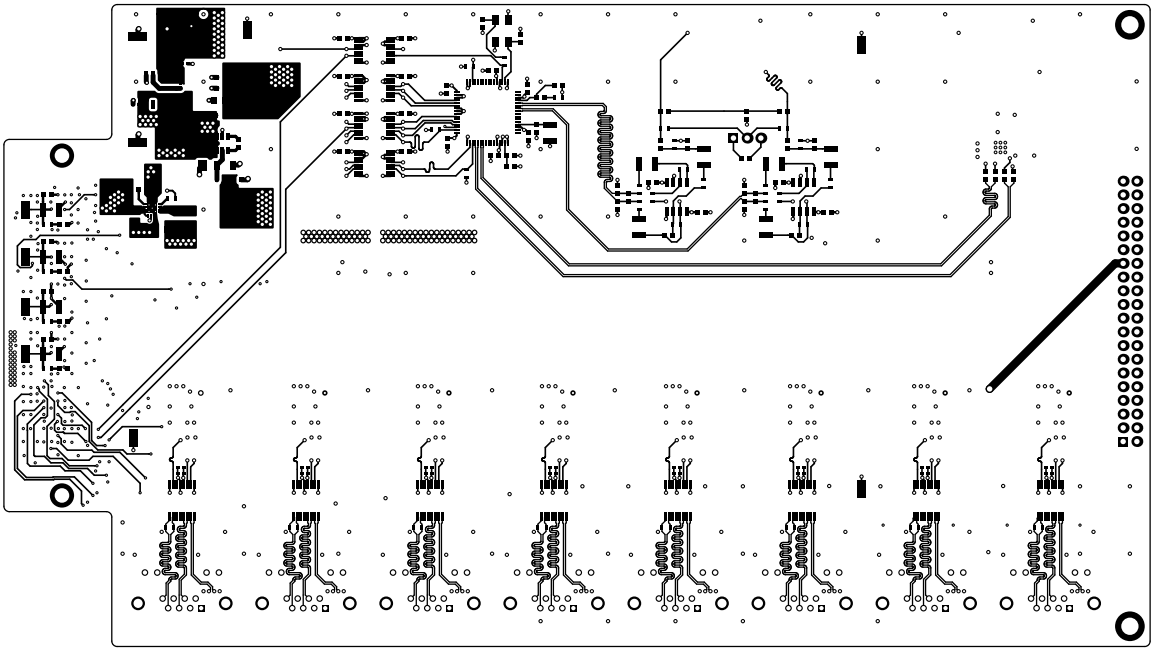
top view



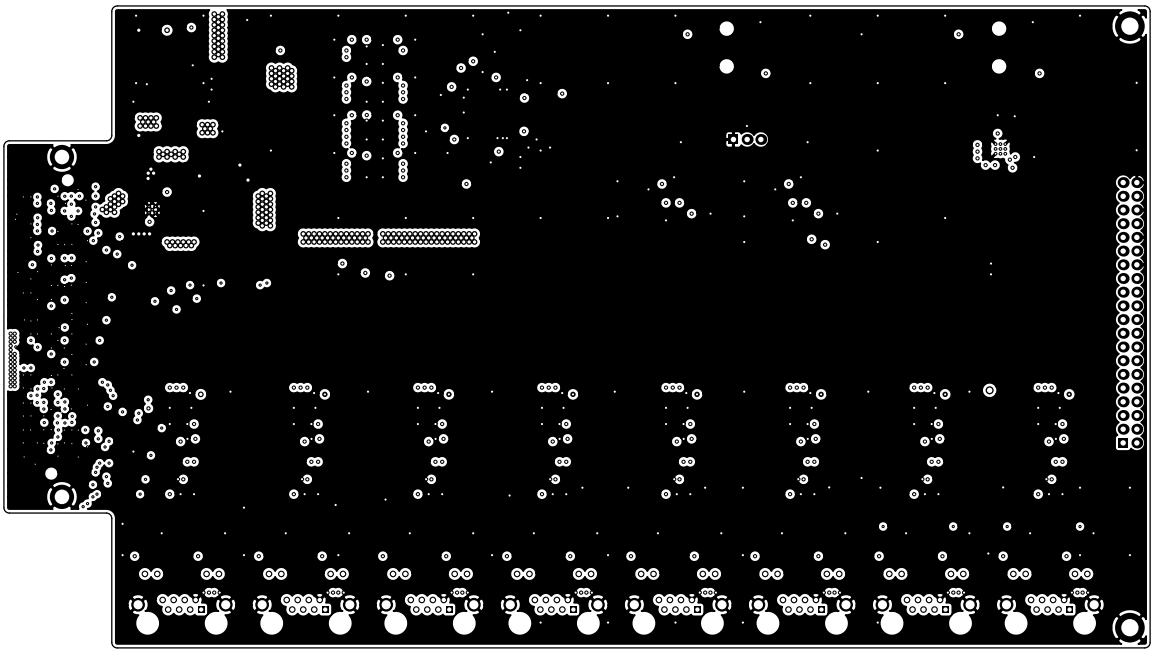
bottom view



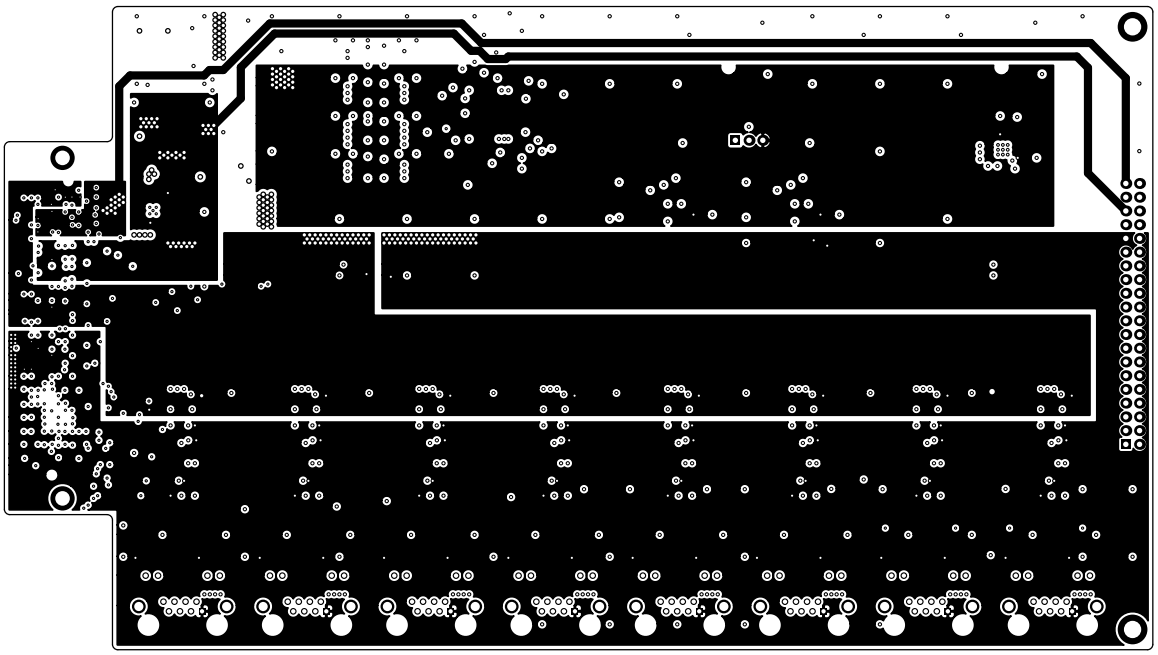
top layer



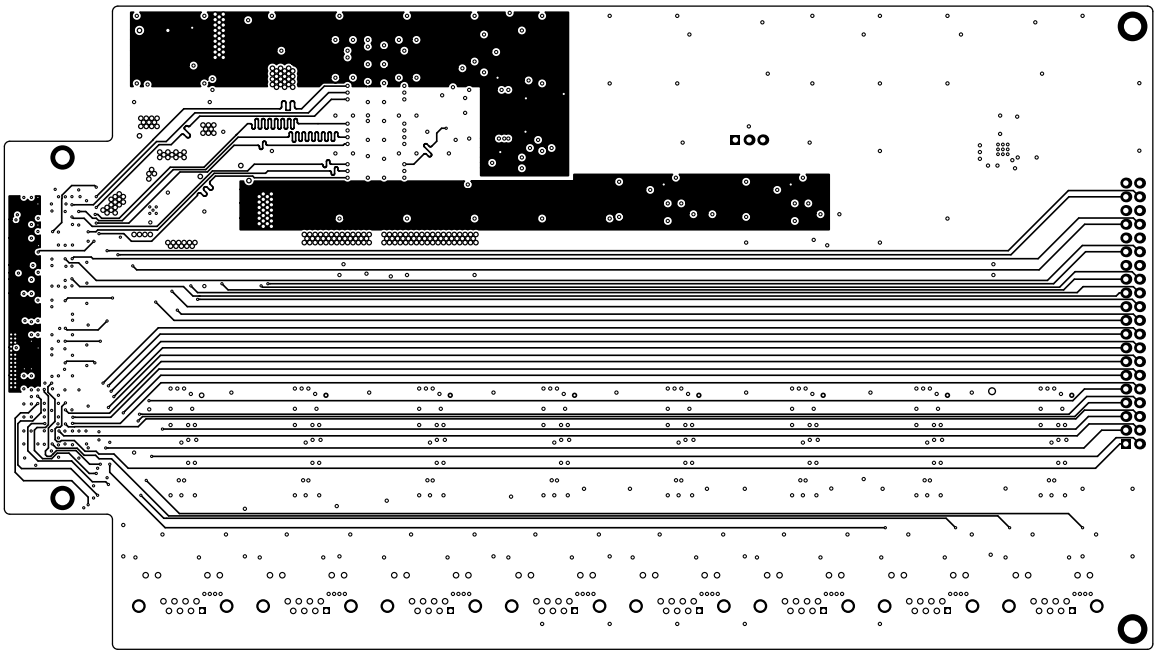
ground layer 1



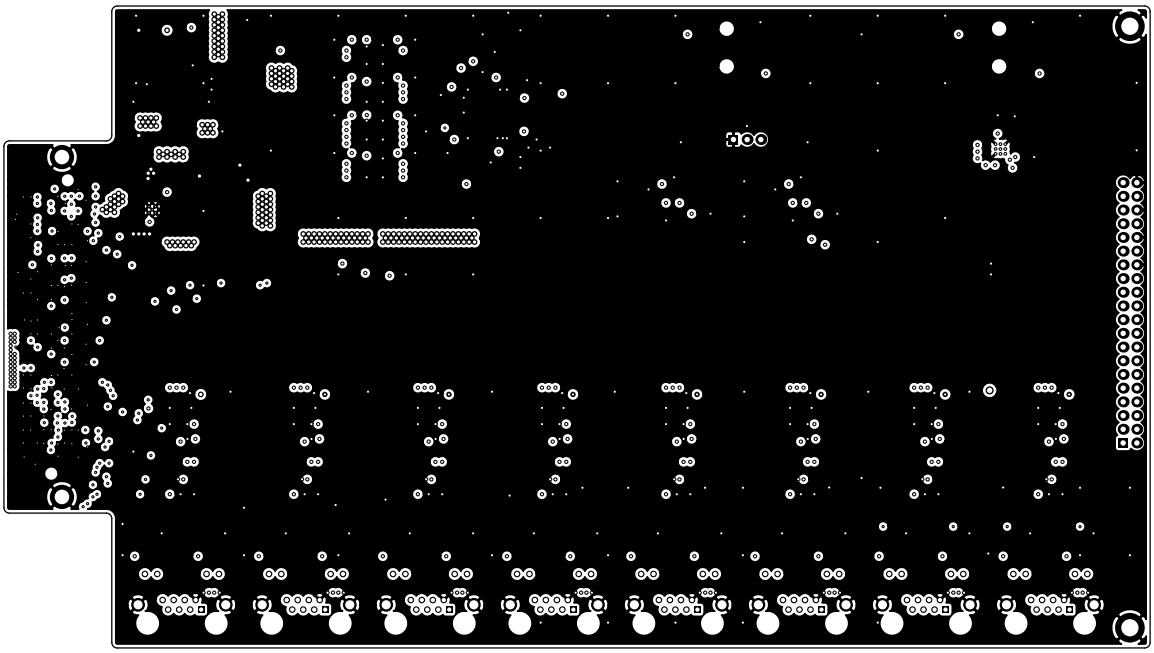
power layer



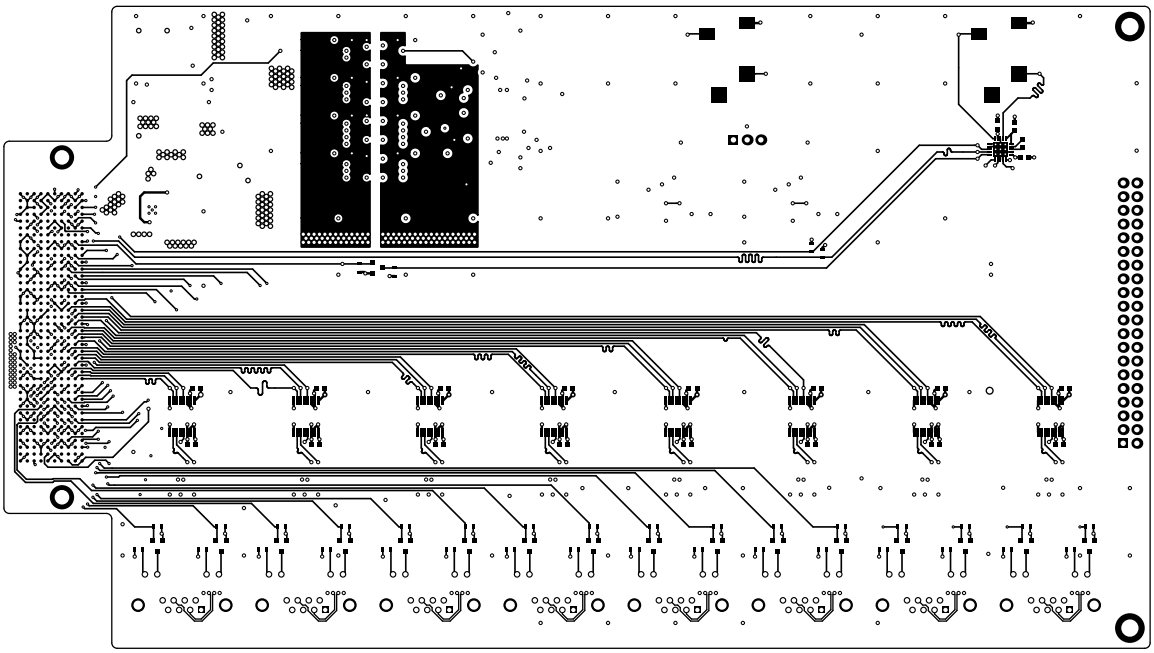
internal signal layer



ground layer 2



bottom layer



APPENDIX K

BEAMFORMING FPGA DATAPLANE CODE

mic_array_param.vhd

```
1  ------------------------------------------------------------------------------
   ↪  ----
2  --
3  --! @file       mic_array_param.vhd
4  --! @brief      Mic array constants
5  --! @details    Miscellaneous constants/parameters related to the microphone
   ↪  arrays
6  --! @author     Trevor Vannoy
7  --! @date       January 2019
8  --! @copyright  Copyright (C) 2019 Trevor Vannoy
9  --!
10 --! Software Released Under the MIT License
11 --
12 --  Permission is hereby granted, free of charge, to any person obtaining a copy
13 --  of this software and associated documentation files (the "Software"), to deal
14 --  IN the Software without restriction, including without limitation the rights
15 --  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
16 --  copies of the Software, and to permit persons to whom the Software is
   ↪  furnished
17 --  to do so, subject to the following conditions:
18 --
19 --  The above copyright notice and this permission notice shall be included IN all
20 --  copies or substantial portions of the Software.
21 --
22 --  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   ↪  IMPLIED,
23 --  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
24 --  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
   ↪  COPYRIGHT
25 --  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
   ↪  ACTION
26 --  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
27 --  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
28 --
29 --  Trevor Vannoy
30 --  Electrical and Computer Engineering
31 --  Montana State University
32 --  610 Cobleigh Hall
33 --  Bozeman, MT 59717
34 --
35 ------------------------------------------------------------------------------
   ↪  ---------
36 library ieee;                --! Use standard library.
37 use ieee.math_real.all;      --! Use math library for log2
38
39 package mic_array_param is
40
41   ----------------------------------------------------------------------------
     ↪  -----------
```

```
42    -- Constant and Type Declarations
43    ------------------------------------------------------------------------------
      ↪  -----------
44    --! the width of the 24-bit word being sent from the microphones
45    constant mic_data_width     : integer := 24;
46    --! the width of the TDM slot in the microphone array data stream
47    constant tdm_slot_width     : integer := 32;
48
49    --! width of the output data. We make it 32.28 to normalize the incoming 24-bit
      ↪  data to
50    --! +/- 1 and allow for a little bit of overflow.
51    constant data_width         : integer := 32;
52    --! number of fractional bits in the output data
53    constant data_fraction      : integer := 28;
54    --! number of integer bits in the output data
55    constant data_integer       : integer := 4;
56
57    --! number of channel in the data stream; also the number of microphones in the
      ↪  array
58    constant num_channels       : integer := 16;
59    constant ch_width           : integer :=
      ↪  integer(ceil(log2(real(num_channels))));
60
61    --! length of the delay line used to synchronize incoming data between
      ↪  microphone arrays
62    constant delay_line_length  : integer := 16;
63
64  end package;
```

mic_array_avalon.vhd

```
1   ------------------------------------------------------------------------------
    ↪  ----
2   --
3   --! @file       mic_array_avalon.vhd
4   --! @brief      Microphone array Avalon wrapper
5   --! @details    Avalon streaming and memory-mapped interface wrapper for the
6   --!             microphone array subsystem; outputs data/channel/valid for
7   --!             the mic array TDM serial data, and allows register control of
8   --!             measured propagation delay.
9   --! @author     Trevor Vannoy
10  --! @date       January 2019
11  --! @copyright  Copyright (C) 2019 Trevor Vannoy
12  --!
13  --! Software Released Under the MIT License
14  --
15  --  Permission is hereby granted, free of charge, to any person obtaining a copy
16  --  of this software and associated documentation files (the "Software"), to deal
17  --  IN the Software without restriction, including without limitation the rights
18  --  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
```

```
19 --  copies of the Software, and to permit persons to whom the Software is
   ↪  furnished
20 --  to do so, subject to the following conditions:
21 --
22 --  The above copyright notice and this permission notice shall be included IN all
23 --  copies or substantial portions of the Software.
24 --
25 --  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   ↪  IMPLIED,
26 --  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
27 --  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
   ↪  COPYRIGHT
28 --  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
   ↪  ACTION
29 --  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
30 --  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
31 --
32 --  Trevor Vannoy
33 --  Electrical and Computer Engineering
34 --  Montana State University
35 --  610 Cobleigh Hall
36 --  Bozeman, MT 59717
37 --
38 ------------------------------------------------------------------------------┘
   ↪  ---------
39 library ieee;                 --! Use standard library.
40 library work;                 --! Use local library.
41 use ieee.std_logic_1164.all;  --! Use standard logic elements.
42 use ieee.numeric_std.all;     --! Use numeric standard.
43 use work.mic_array_param.all; --! Use mic array related constant parameters
44
45 ------------------------------------------------------------------------------
46 --
47 --! @brief    mic_array_avalon
48 --! @details  Avalon streaming and memory-mapped interface wrapper for the
49 --!           microphone array subsystem; outputs data/channel/valid for
50 --!           the mic array TDM serial data, and allows register control of
51 --!           measured propagation delay.
52 -- TODO: update documentation!!!!
53 --! @param  sys_clk           Fabric system clock. Syncrhonous to sck, but faster.
54 --! @param  sck_master        Master SCK clock domain for all mic arrays
55 --! @param  sck_rcv           Recieve clock for incoming mic array data
56 --! @param  rst               Asynchronous active-high reset
57 --! @param  avs_s1_address    Avalon bus address
58 --! @param  avs_s1_write      Avalon bus write enable flag
59 --! @param  avs_s1_writedata  Avalon bus write data
60 --! @param  avs_s1_read       Avalon bus read enable flag
61 --! @param  avs_s1_readdata   Avalon bus read data
62 --! @param  ast_data          Streaming microphone array data
63 --! @param  ast_channel       Microphone array data stream TDM channel
64 --! @param  ast_valid         Avalon streaming valid flag, asserted at
```

```
65  --!                           the beginning of each TDM slot / word
66  --! @param  led_sd            RJ45 LED indicator signal for SD
67  --! @param  led_ws            RJ45 LED indicator signal for WS
68  --! @param  sd                Mic array SD data signal
69  --! @param  ws                Mic array WS sampling pulse
70  --
71  ------------------------------------------------------------------------------
72  entity mic_array_avalon is
73    port (
74      ---------------------------------------------------------------
75      -- Clock and Reset Signals
76      ---------------------------------------------------------------
77      sys_clk           : in  std_logic;
78      sck_master        : in  std_logic;
79      sck_rcv           : in  std_logic;
80      rst               : in  std_logic;
81      ---------------------------------------------------------------
82      -- Avalon Memory Mapped Slave Signals
83      ---------------------------------------------------------------
84      avs_s1_address    : in  std_logic_vector(1 downto 0);
85      avs_s1_write      : in  std_logic;
86      avs_s1_writedata  : in  std_logic_vector(31 downto 0);
87      avs_s1_read       : in  std_logic;
88      avs_s1_readdata   : out std_logic_vector(31 downto 0);
89      ---------------------------------------------------------------
90      -- Avalon Streaming Source Signals
91      ---------------------------------------------------------------
92      ast_data          : out std_logic_vector(data_width - 1 downto 0);
93      ast_channel       : out std_logic_vector(ch_width - 1 downto 0);
94      ast_valid         : out std_logic;
95      ---------------------------------------------------------------
96      -- External Conduit Signals
97      ---------------------------------------------------------------
98      led_sd            : out std_logic;
99      led_ws            : out std_logic;
100     sd                : in  std_logic;
101     ws                : out std_logic
102   );
103 end entity;
104
105
106 architecture mic_array_avalon_arch of mic_array_avalon is
107
108   ------------------------------------------------------------------------------
    ↪  -----------
109   -- Constant and Type Declarations
110   ------------------------------------------------------------------------------
    ↪  -----------
111
112   ------------------------------------------------------------------------------
    ↪  -----------
```

```
113   -- Signal Declarations
114   ------------------------------------------------------------------------------
      ↪  -----------
115   signal frame_delay_reg  : std_logic_vector(avs_s1_readdata'high downto 0) := (0
      ↪  => '1', others => '0');
116   signal data_old         : std_logic_vector(data_width-1 downto 0) := (others =>
      ↪  '0');
117   signal data             : std_logic_vector(data_width-1 downto 0) := (others =>
      ↪  '0');
118
119   ------------------------------------------------------------------------------
      ↪  -----------
120   -- Component Declerations
121   ------------------------------------------------------------------------------
      ↪  -----------
122   component mic_array is
123     port (
124       sys_clk     : in  std_logic;
125       sck_master  : in  std_logic;
126       sck_rcv     : in  std_logic;
127       rst         : in  std_logic;
128       led_sd      : out std_logic;
129       led_ws      : out std_logic;
130       sd          : in  std_logic;
131       ws          : out std_logic;
132       frame_delay : in  integer;
133       data        : out std_logic_vector(data_width - 1 downto 0);
134       channel     : out std_logic_vector(ch_width - 1 downto 0);
135       valid       : out std_logic
136     );
137   end component;
138
139 begin
140
141   mic_array0 : component mic_array
142     port map (
143       sys_clk     => sys_clk,
144       sck_master  => sck_master,
145       sck_rcv     => sck_rcv,
146       rst         => rst,
147       led_sd      => led_sd,
148       led_ws      => led_ws,
149       sd          => sd,
150       ws          => ws,
151       frame_delay => to_integer(unsigned(frame_delay_reg)),
152       data        => data,
153       channel     => ast_channel
154       -- valid       => ast_valid
155     );
156
```

```
157   -- TODO: add a "start" register so we can control when to start clocking the
      ↪  mic array
158
159   -- read/write to registers
160   process(sys_clk)
161   begin
162     if rising_edge(sys_clk) then
163       -- read the registers
164       if avs_s1_read = '1' then
165         case avs_s1_address is
166           when "00"   => avs_s1_readdata <=  frame_delay_reg;
167           when others => avs_s1_readdata <= (others => '0');
168         end case;
169       -- write the registers
170       elsif avs_s1_write = '1' then
171         case avs_s1_address is
172           when "00"   => frame_delay_reg <= avs_s1_writedata;
173           when others => null;
174         end case;
175       end if;
176     end if;
177   end process;
178
179   delay_data : process(sys_clk)
180   begin
181     if rising_edge(sys_clk) then
182       data_old <= data;
183     end if;
184   end process;
185
186   ast_valid <= '1' when data_old /= data else '0';
187   ast_data <= data;
188
189 end architecture;
```

mic_array.vhd

```
1  ------------------------------------------------------------------------------
   ↪  ---------
2  --
3  --! @file       mic_array.vhd
4  --! @brief      Microphone array subsystem
5  --! @details    Wrapper for all of the components needed to successfully capture
   ↪  data
6  --!             from a microphone array and get it ready for further processing.
7  --! @author     Trevor Vannoy
8  --! @date       August 2018
9  --! @copyright  Copyright (C) 2018-2019 Trevor Vannoy
10 --!
11 --! Software Released Under the MIT License
```

```
12 --
13 --  Permission is hereby granted, free of charge, to any person obtaining a copy
14 --  of this software and associated documentation files (the "Software"), to deal
15 --  IN the Software without restriction, including without limitation the rights
16 --  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
17 --  copies of the Software, and to permit persons to whom the Software is
    ↪  furnished
18 --  to do so, subject to the following conditions:
19 --
20 --  The above copyright notice and this permission notice shall be included IN all
21 --  copies or substantial portions of the Software.
22 --
23 --  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    ↪  IMPLIED,
24 --  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
25 --  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    ↪  COPYRIGHT
26 --  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
    ↪  ACTION
27 --  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
28 --  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
29 --
30 --  Trevor Vannoy
31 --  Electrical and Computer Engineering
32 --  Montana State University
33 --  610 Cobleigh Hall
34 --  Bozeman, MT 59717
35 --
36 ------------------------------------------------------------------------------┘
    ↪  ---------
37 library ieee;                 --! Use standard library.
38 library work;                 --! Use local library.
39 use ieee.std_logic_1164.all;  --! Use standard logic elements.
40 use ieee.numeric_std.all;     --! Use numeric standard.
41 use ieee.math_real.all;       --! Use math library for log2
42 use work.mic_array_param.all; --! Use mic array related constant parameters
43
44 ------------------------------------------------------------------------------
45 -- TODO: update documentation!
46 --
47 --! @brief    mic_array
48 --! @details  Wrapper for all of the components needed successfully capture data
49 --!           from a microphone array and get it ready for further processing.
50 --! @param    sys_clk       Fabric system clock. This is synchronous to sck, but
    ↪  faster.
51 --! @param    sck_master    Master SCK clock domain for all mic arrays
52 --! @param    sck_rcv       Recieve clock for incoming mic array data
53 --! @param    rst           Asynchronous active-high reset
54 --! @param    led_sd        RJ45 LED indicator signal for SD
55 --! @param    led_ws        RJ45 LED indicator signal for WS
56 --! @param    sd            Mic array SD data signal
```

```
57  --! @param    ws            Mic array WS sampling pulse
58  --! @param    frame_delay   TDM data frame propagation delay in SCK cycles
59  --
60  ----------------------------------------------------------------------------
61  entity mic_array is
62    port (
63      sys_clk     : in  std_logic;
64      sck_master  : in  std_logic;
65      sck_rcv     : in  std_logic;
66      rst         : in  std_logic;
67      led_sd      : out std_logic;
68      led_ws      : out std_logic;
69      sd          : in  std_logic;
70      ws          : out std_logic;
71      frame_delay : in  integer;
72      data        : out std_logic_vector(data_width - 1 downto 0);
73      channel     : out std_logic_vector(ch_width - 1 downto 0);
74      valid       : out std_logic
75    );
76  end entity;
77
78
79  architecture mic_array_arch of mic_array is
80
81    ------------------------------------------------------------------------┐
    ↪  -----------
82    -- Constant and Type Declarations
83    ------------------------------------------------------------------------┐
    ↪  -----------
84
85    ------------------------------------------------------------------------┐
    ↪  -----------
86    -- Signal Declarations
87    ------------------------------------------------------------------------┐
    ↪  -----------
88    signal sd_valid         : boolean := false;
89    signal frame_start      : std_logic;
90    signal frame_start_tmp  : std_logic;
91    signal data_tmp         : std_logic_vector(data_width - 1 downto 0);
92    signal channel_tmp      : std_logic_vector(ch_width - 1 downto 0);
93    signal valid_tmp        : std_logic;
94    signal word_clk         : std_logic;
95    signal word_clk_tmp     : std_logic;
96    signal sd_delayed       : std_logic;
97    signal ws_tmp           : std_logic;
98
99    ------------------------------------------------------------------------┐
    ↪  -----------
100   -- Component Declerations
101   ------------------------------------------------------------------------┐
    ↪  -----------
```

```
102   -- performs startup sequence for mic array and generates WS signal
103   component mic_array_startup is
104   port (
105     sck         : in  std_logic;
106     rst         : in  std_logic;
107     data_valid  : out boolean := false;
108     ws           : out std_logic
109   );
110   end component;
111
112   -- send out a pulse at the beginning of every received TDM frame
113   component frame_start_counter is
114   port (
115     sck         : in  std_logic;
116     rst         : in  std_logic;
117     frame_delay : in  integer;
118     frame_start : out std_logic
119     );
120   end component;
121
122   -- deserialize incoming mic array data
123   component mic_array_deserializer is
124   port (
125     clk         : in  std_logic;
126     rst         : in  std_logic;
127     frame_start : in  std_logic;
128     din         : in  std_logic;
129     dout        : out std_logic_vector(data_width-1 downto 0) := (others => '0');
130     word_clock  : out std_logic;
131     channel     : out std_logic_vector(ch_width - 1 downto 0);
132     valid       : out std_logic
133     );
134   end component;
135
136   -- generic data synchronizer
137   component synchronizer is
138     generic (
139       data_width : integer := 1
140     );
141     port (
142       clk   : in  std_logic;
143       rst   : in  std_logic;
144       din   : in  std_logic_vector(data_width - 1 downto 0);
145       dout  : out std_logic_vector(data_width - 1 downto 0)
146     );
147   end component;
148
149   -- delay incoming data bits so all arrays are synchronous to each other
150   component tapped_delay_line is
151     generic (
152       length : integer := 16
```

```
153     );
154     port (
155       clk   : in  std_logic;
156       rst   : in  std_logic;
157       din   : in  std_logic;
158       sel   : in  integer range 0 to length - 1;
159       dout  : out std_logic := '0'
160     );
161   end component;
162 
163 begin
164 
165   -- drive the RJ45 LEDs so we know the hardware is doing something.
166   led_sd  <= not sd; -- sd is normally high; inverting it makes sure the status
      ↪  LED is off when there isn't data
167   led_ws  <= ws_tmp;
168 
169   -- drive the mic array
170   ws      <= ws_tmp;
171 
172   mic_array_startup0 : mic_array_startup
173     port map (
174       sck        => sck_master,
175       rst        => rst,
176       data_valid => sd_valid,
177       ws          => ws_tmp
178     );
179 
180   -- with the way the tapped_delay_line is set up, the beginning of the data frame
181   -- will always start at delay_line_length cycles after sending out the WS pulse.
182   frame_start_counter0 : frame_start_counter
183     port map (
184       sck          => sck_rcv,
185       rst          => rst,
186       frame_delay  => delay_line_length,
187       frame_start  => frame_start_tmp
188     );
189 
190   deserializer : mic_array_deserializer
191     port map (
192       clk         => sck_rcv,
193       rst         => rst,
194       frame_start => frame_start_tmp,
195       din         => sd_delayed,
196       dout        => data_tmp,
197       word_clock  => word_clk_tmp,
198       channel     => channel_tmp,
199       valid       => valid_tmp
200     );
201 
```

```
202   -- the tap number is selected so the effective delay on the data line equals
      ↪  the length
203   -- of the delay line; this way, each array's data frames will be synchronous
      ↪  with each other.
204   tapped_delay_line0 : tapped_delay_line
205     port map (
206       clk   => sck_rcv,
207       rst   => rst,
208       din   => sd,
209       sel   => delay_line_length  - 1 - frame_delay,
210       dout  => sd_delayed
211     );
212
213   -- synchronize mic data with the fabric system clock domain
214   synchronize_data : synchronizer
215     generic map (
216       data_width => data_width
217     )
218     port map (
219       clk   => sys_clk,
220       rst   => rst,
221       din   => data_tmp,
222       dout  => data
223     );
224
225   -- synchronize TDM channel number with the fabric system clock domain
226   synchronize_channel : synchronizer
227     generic map (
228       data_width => ch_width
229     )
230     port map (
231       clk   => sys_clk,
232       rst   => rst,
233       din   => channel_tmp,
234       dout  => channel
235     );
236
237
238   -- synchronize data word clock with the fabric system clock domain
239   synchronize_word_clk : synchronizer
240     port map (
241       clk     => sys_clk,
242       rst     => rst,
243       din(0)  => word_clk_tmp,
244       dout(0) => word_clk
245     );
246
247 end architecture;
```

mic_array_deserializer.vhd

```
1  ------------------------------------------------------------------------------
   ↪  ----
2  --
3  --! @file       mic_array_deserializer.vhd
4  --! @brief      Deserialze mic array data
5  --! @details    Deserialize mic array data by using a shift register and some
6  --!             bit and word counters to separate the TDM channels/words.
7  --!             The incoming mic array data words are 24-bit, which are
8  --!             MSB-aligned in a 32-bit slot (i.e. the first 24 bits we received
9  --!             is the data we are interested in).
10 --! @author     Trevor Vannoy
11 --! @date       August 2018
12 --! @copyright  Copyright (C) 2018 Trevor Vannoy
13 --!
14 --! Software Released Under the MIT License
15 --
16 --  Permission is hereby granted, free of charge, to any person obtaining a copy
17 --  of this software and associated documentation files (the "Software"), to deal
18 --  IN the Software without restriction, including without limitation the rights
19 --  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
20 --  copies of the Software, and to permit persons to whom the Software is
   ↪  furnished
21 --  to do so, subject to the following conditions:
22 --
23 --  The above copyright notice and this permission notice shall be included IN all
24 --  copies or substantial portions of the Software.
25 --
26 --  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   ↪  IMPLIED,
27 --  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
28 --  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
   ↪  COPYRIGHT
29 --  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
   ↪  ACTION
30 --  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
31 --  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
32 --
33 --  Trevor Vannoy
34 --  Electrical and Computer Engineering
35 --  Montana State University
36 --  610 Cobleigh Hall
37 --  Bozeman, MT 59717
38 --
39 ------------------------------------------------------------------------------
   ↪  ---------
40
41 library ieee;                 --! Use standard library.
42 library work;                 --! Use local library.
43 use ieee.std_logic_1164.all;  --! Use standard logic elements.
44 use ieee.numeric_std.all;     --! Use numeric standard.
45 use ieee.math_real.all;       --! Use math library for log2
```

```
46 use work.mic_array_param.all; --! Use mic array related constant parameters
47
48 ----------------------------------------------------------------------------
49 --
50 --! @brief    mic_array_deserializer
51 --! @details  Deserialize mic array data by using a shift register and some
52 --!           bit and word counters to separate the TDM channels/words.
53 --! @param    clk           Data sampling clock
54 --! @param    rst           Asynchronous active-high reset
55 --! @param    frame_start   Pulse indicating start of data frame
56 --! @param    din           Serial input data
57 --! @param    dout          Parallel output data
58 --! @param    word_clock    Clock at the frequency of the incoming words
59 --! @param    channel       TDM channel number
60 --! @param    valid         Flag indicating when output data is valid for
61 --!                         use with streaming interfaces
62 --
63 ----------------------------------------------------------------------------
64 entity mic_array_deserializer is
65   port (
66     clk         : in  std_logic;
67     rst         : in  std_logic;
68     frame_start : in  std_logic;
69     din         : in  std_logic;
70     dout        : out std_logic_vector(data_width-1 downto 0) := (others => '0');
71     word_clock  : out std_logic;
72     channel     : out std_logic_vector(integer(ceil(log2(real(num_channels))))-1
       ↪  downto 0);
73     valid       : out std_logic := '0'
74   );
75 end entity;
76
77 architecture mic_array_deserializer_arch of mic_array_deserializer is
78
79   ----------------------------------------------------------------------------
   ↪  -----------
80   -- Constant and Type Declarations
81   ----------------------------------------------------------------------------
   ↪  -----------
82   constant num_pad_bits : integer := tdm_slot_width - data_width;
83
84   ----------------------------------------------------------------------------
   ↪  -----------
85   -- Signal Declarations
86   ----------------------------------------------------------------------------
   ↪  -----------
87   signal dout_tmp    : std_logic_vector(mic_data_width-1 downto 0) := (others =>
   ↪  '0');
88   signal bit_count    : integer := 0;
89   signal channel_temp : integer := -1;
90   signal word_done    : boolean := false;
```

```
91
92  begin
93    -- shift in the serial data
94    shift_register : process(clk, rst)
95    begin
96      if rst = '1' then
97        dout_tmp <= (others => '0');
98      elsif rising_edge(clk) then
99        -- shift left so the new bit gets shifted into lsb
100       dout_tmp <= dout_tmp(dout_tmp'high-1 downto 0) & din;
101     end if;
102   end process;
103
104   -- latch the deserialized word once it is all the way into the the shift
      ↪  register
105   -- additionally, set valid flag for one clock cycle for use with streaming
      ↪  interfaces
106   output_reg : process(clk, rst)
107   begin
108     if rst = '1' then
109       dout <= (others => '0');
110     elsif rising_edge(clk) then
111       if word_done then
112         -- convert the incoming 24-bit signed word into a 32.28 word by adding 4
113         -- fractional bits and sign-extending.
114         dout(data_fraction-1 downto 0) <= dout_tmp & "0000";
115         dout(data_width-1 downto data_fraction) <= (others =>
            ↪  dout_tmp(dout_tmp'high));
116         valid <= '1';
117       else
118         valid <= '0';
119       end if;
120     end if;
121   end process;
122
123   -- count the number of incoming bits so we know where the words start/end
124   bit_counter : process(clk, rst)
125   begin
126     if rst = '1' then
127       bit_count <= 0;
128       word_done <= false;
129     elsif rising_edge(clk) then
130       -- count bits until we reach the next TDM slot, then restart the count
131       if frame_start = '1' then
132         bit_count <= 1;
133       elsif bit_count < tdm_slot_width - 1 then
134         bit_count <= bit_count + 1;
135       else
136         bit_count <= 0;
137       end if;
138       -- check if the word is done
```

```
139       if bit_count = mic_data_width - 1 then
140         word_done <= true;
141       else
142         word_done <= false;
143       end if;
144     end if;
145   end process;
146 
147   -- count how many words have come through so we know what channel we're on
148   word_counter : process(clk, rst)
149   begin
150     if rst = '1' then
151       channel_temp <= -1;
152     elsif rising_edge(clk) and word_done then
153       if channel_temp < num_channels - 1 then
154         channel_temp <= channel_temp + 1;
155       else
156         channel_temp <= 0;
157       end if;
158     end if;
159   end process;
160 
161   channel <= std_logic_vector(to_unsigned(channel_temp, channel'length));
162   word_clock <= '1' when word_done else '0'; -- TODO: remove word_clock signal? I
      ↪ don't really need it now that I've captured data with MATLAB.
163 
164 end architecture;
```

mic_array_startup.vhd

```
1   ------------------------------------------------------------------------------
    ↪ ----
2   --
3   --! @file       mic_arary_startup.vhd
4   --! @brief      Mic array startup sequence
5   --! @details    Generate WS for mic array after the startup delay specified
6   --!             in the ICS-52000 datasheet
7   --! @author     Trevor Vannoy
8   --! @date       August 2018
9   --! @copyright  Copyright (C) 2018 Trevor Vannoy
10  --!
11  --! Software Released Under the MIT License
12  --
13  --  Permission is hereby granted, free of charge, to any person obtaining a copy
14  --  of this software and associated documentation files (the "Software"), to deal
15  --  IN the Software without restriction, including without limitation the rights
16  --  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
17  --  copies of the Software, and to permit persons to whom the Software is
    ↪ furnished
18  --  to do so, subject to the following conditions:
```

```
19  --
20  --  The above copyright notice and this permission notice shall be included IN all
21  --  copies or substantial portions of the Software.
22  --
23  --  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    ↪  IMPLIED,
24  --  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
25  --  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    ↪  COPYRIGHT
26  --  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
    ↪  ACTION
27  --  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
28  --  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
29  --
30  --  Trevor Vannoy
31  --  Electrical and Computer Engineering
32  --  Montana State University
33  --  610 Cobleigh Hall
34  --  Bozeman, MT 59717
35  --
36  ------------------------------------------------------------------------------┘
    ↪  ---------
37
38  library ieee;                   --! Use standard library.
39  use ieee.std_logic_1164.all;    --! Use standard logic elements.
40  use ieee.numeric_std.all;       --! Use numeric standard.
41
42  ------------------------------------------------------------------------------
43  --
44  --! @brief    mic_array_startup
45  --! @details  Generate WS for mic array after the startup delay specified
46  --!           in the ICS-52000 datasheet, and set data_valid once the mic
47  --!           array data is valid (also specified in the datasheet)
48  --! @param    sck         Mic array clock
49  --! @param    rst         Asynchronous active-high reset
50  --! @param    data_valid  Data valid flag
51  --! @param    ws          WS mic array sampling clock
52  --
53  ------------------------------------------------------------------------------
54  -- TODO: better entity name?
55  entity mic_array_startup is
56    port (
57      sck         : in  std_logic;
58      rst         : in  std_logic;
59      data_valid  : out boolean := false;
60      ws          : out std_logic
61    );
62  end entity;
63
64  architecture mic_array_startup_arch of mic_array_startup is
65
```

```
66    ------------------------------------------------------------------------------
      ↪  -----------
67    -- Constant and Type Declarations
68    ------------------------------------------------------------------------------
      ↪  -----------
69    --! clock frequency in Hz
70    constant clk_freq                       : integer := 24576000;
71    --! number of clock cycles before sending the frame sync (WS) pulse; wait 20 ms
      ↪  (1/50 s) to send WS
72    constant startup_sck_cycles             : integer := clk_freq/50;
73    --! number of clock cycles after startup before the output data is valid;
      ↪  specified in ICS-52000 datasheet
74    constant sck_cycles_until_data_valid    : integer := 262144;
75    --! number of clock cycles per sampling frame
76    constant cycles_per_frame               : integer := 512;
77
78    ------------------------------------------------------------------------------
      ↪  -----------
79    -- Signal Declarations
80    ------------------------------------------------------------------------------
      ↪  -----------
81    signal ws_enable            : boolean := false;
82    signal ws_counter           : integer := 0;
83    signal startup_counter      : integer := 0;
84    signal data_valid_counter   : integer := 0;
85
86
87  begin
88    -- wait startup_sck_cycles before enabling ws.
89    -- the ICS5200 datasheet specifices to wait at least 10 ms before sending out
      ↪  ws pulses,
90    ws_enable_counter : process(sck, rst)
91    begin
92      if rst = '1' then
93        startup_counter <= 0;
94        ws_enable <= false;
95      elsif rising_edge(sck) then
96        if startup_counter < startup_sck_cycles - 1 then
97          startup_counter <= startup_counter + 1;
98        else
99          ws_enable <= true;
100       end if;
101     end if;
102   end process;
103
104   -- generate the ws pulses with a 50% duty cycle
105   generate_ws : process(sck, rst)
106   begin
107     if rst = '1' then
108       ws_counter <= 0;
109       ws <= '1';
```

```
110     elsif rising_edge(sck) and ws_enable then
111       if ws_counter < cycles_per_frame/2 then
112         ws <= '1';
113         ws_counter <= ws_counter + 1;
114       elsif ws_counter < cycles_per_frame - 1 then
115         ws <= '0';
116         ws_counter <= ws_counter + 1;
117       else
118         ws <= '0';
119         ws_counter <= 0;
120       end if;
121     end if;
122   end process;
123
124   -- set the data valid flag after sck_cycles_until_data_valid clock cycles
125   set_data_valid_flag : process(sck, rst)
126   begin
127     if rst = '1' then
128       data_valid_counter <= 0;
129       data_valid <= false;
130     elsif rising_edge(sck) and ws_enable then
131       if data_valid_counter < sck_cycles_until_data_valid - 1 then
132         data_valid_counter <= data_valid_counter + 1;
133       else
134         data_valid <= true;
135       end if;
136     end if;
137   end process;
138
139 end architecture;
```

frame_start_counter.vhd

```
1  ------------------------------------------------------------------------------
   ↪  ----
2  --
3  --! @file       frame_start_counter.vhd
4  --! @brief      Indicate start of data frame
5  --! @details    Indicates the start of a TDM data frame with a pulse delayed
6  --!             by a specified number of clock cycles
7  --! @author     Trevor Vannoy
8  --! @date       August 2018
9  --! @copyright  Copyright (C) 2018 Trevor Vannoy
10 --!
11 --! Software Released Under the MIT License
12 --
13 --  Permission is hereby granted, free of charge, to any person obtaining a copy
14 --  of this software and associated documentation files (the "Software"), to deal
15 --  IN the Software without restriction, including without limitation the rights
16 --  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
```

```
17 --  copies of the Software, and to permit persons to whom the Software is
   ↪  furnished
18 --  to do so, subject to the following conditions:
19 --
20 --  The above copyright notice and this permission notice shall be included IN all
21 --  copies or substantial portions of the Software.
22 --
23 --  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   ↪  IMPLIED,
24 --  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
25 --  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
   ↪  COPYRIGHT
26 --  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
   ↪  ACTION
27 --  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
28 --  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
29 --
30 --  Trevor Vannoy
31 --  Electrical and Computer Engineering
32 --  Montana State University
33 --  610 Cobleigh Hall
34 --  Bozeman, MT 59717
35 --
36 ----------------------------------------------------------------------------------
   ↪  ---------
37
38 library ieee;                  --! Use standard library.
39 use ieee.std_logic_1164.all;   --! Use standard logic elements.
40 use ieee.numeric_std.all;      --! Use numeric standard.
41
42 ------------------------------------------------------------------------------
43 --
44 --! @brief    frame_start_counter
45 --! @details  Indicates the start of a TDM data frame with a pulse delayed
46 --!           by frame_delay clock cycles
47 --! @param    sck           Phase-shifted sampling clock for incoming data
48 --! @param    rst           Asynchronous active-high reset
49 --! @param    frame_delay   Number of clock cycles the received data
50 --!                         frame is delayed by relative to the original
51 --!                         sampling pulse
52 --! @param    frame_start   Pulse indicating start of data frame
53 --
54 ------------------------------------------------------------------------------
55 -- TODO: better entity name?
56 entity frame_start_counter is
57   port (
58     sck         : in  std_logic;
59     rst         : in  std_logic;
60     frame_delay : in  integer;
61     frame_start : out std_logic
62   );
```

```
63  end entity;
64
65  architecture frame_start_counter_arch of frame_start_counter is
66    ------------------------------------------------------------------------------
    ↪  -----------
67    -- Constant and Type Declarations
68    ------------------------------------------------------------------------------
    ↪  -----------
69      constant frame_length   : integer := 512;
70
71    ------------------------------------------------------------------------------
    ↪  -----------
72    -- Signal Declarations
73    ------------------------------------------------------------------------------
    ↪  -----------
74      signal count            : integer := 0;
75
76  begin
77
78    -- wait frame_delay cycles then output the frame_start pulse for one clock cycle
79    generate_frame_start_pulse : process(sck, rst)
80    begin
81      if rst = '1' then
82        count <= 0;
83        frame_start <= '0';
84      elsif rising_edge(sck) then
85        if count < frame_length - 1 then
86          count <= count + 1;
87        else
88          count <= 0;
89        end if;
90        if count = frame_delay then
91          frame_start <= '1';
92        else
93          frame_start <= '0';
94        end if;
95      end if;
96    end process;
97
98  end architecture;
```

synchronizer.vhd

```
1   ------------------------------------------------------------------------------
    ↪  ----
2   --
3   --! @file       syncrhonizer.vhd
4   --! @brief      A 2-stage synchronizer
5   --! @details    Synchronizes input data to a desired clock domain
6   --!             with a 2-stage shift register
```

```
7  --! @author     Trevor Vannoy
8  --! @date       August 2018
9  --! @copyright  Copyright (C) 2018 Trevor Vannoy
10 --!
11 --! Software Released Under the MIT License
12 --
13 --  Permission is hereby granted, free of charge, to any person obtaining a copy
14 --  of this software and associated documentation files (the "Software"), to deal
15 --  IN the Software without restriction, including without limitation the rights
16 --  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
17 --  copies of the Software, and to permit persons to whom the Software is
    ↪  furnished
18 --  to do so, subject to the following conditions:
19 --
20 --  The above copyright notice and this permission notice shall be included IN all
21 --  copies or substantial portions of the Software.
22 --
23 --  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
    ↪  IMPLIED,
24 --  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
25 --  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
    ↪  COPYRIGHT
26 --  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
    ↪  ACTION
27 --  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
28 --  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
29 --
30 --  Trevor Vannoy
31 --  Electrical and Computer Engineering
32 --  Montana State University
33 --  610 Cobleigh Hall
34 --  Bozeman, MT 59717
35 --
36 ------------------------------------------------------------------------------┘
    ↪  ---------
37
38 library ieee;                  --! Use standard library.
39 use ieee.std_logic_1164.all;   --! Use standard logic elements.
40 use ieee.numeric_std.all;      --! Use numeric standard.
41
42 ------------------------------------------------------------------------------
43 --
44 --! @brief      synchronizer
45 --! @details    Synchronizes din to clk using a 2-stage shift register;
46 --!             output latency of 2 clock cycles
47 --! @param      clk   Clock domain to synchroniz data with
48 --! @param      rst   Asynchronous active-high reset
49 --! @param      din   Input data to synchronize
50 --! @param      dout  Synchronized output data
51 --
52 ------------------------------------------------------------------------------
```

```
53  entity synchronizer is
54    generic (
55      data_width : integer := 1
56    );
57    port (
58      clk   : in  std_logic;
59      rst   : in  std_logic;
60      din   : in  std_logic_vector(data_width - 1 downto 0);
61      dout  : out std_logic_vector(data_width - 1 downto 0)
62    );
63  end entity;
64
65  architecture syncrhonizer_arch of synchronizer is
66
67    ------------------------------------------------------------------------------
      ↪  -----------
68    -- Constant and Type Declarations
69    ------------------------------------------------------------------------------
      ↪  -----------
70    type delay_line is array (1 downto 0) of std_logic_vector(data_width - 1 downto
      ↪  0);
71
72    ------------------------------------------------------------------------------
      ↪  -----------
73    -- Signal Declarations
74    ------------------------------------------------------------------------------
      ↪  -----------
75    signal shift_reg : delay_line := ((others => '0'), (others => '0'));
76
77  begin
78    -- run data through a 2-stage shift register
79    sync : process(clk, rst)
80    begin
81      if rst = '1' then
82        shift_reg <= ((others => '0'), (others => '0'));
83      elsif rising_edge(clk) then
84        shift_reg <= shift_reg(0) & din;
85      end if;
86    end process;
87
88    -- take the output at the second stage
89    dout <= shift_reg(1);
90
91  end architecture;
```

tapped_delay_line.vhd

```
1   ------------------------------------------------------------------------------
    ↪  ----
2   --
```

```
3  --! @file       tapped_delay_line.vhd
4  --! @brief      Generic tapped delay line
5  --! @details    Arbitrary-length delay line with a mux to select a bit from any
   ↪  stage
6  --! @author     Trevor Vannoy
7  --! @date       September 2018
8  --! @copyright  Copyright (C) 2018 Trevor Vannoy
9  --!
10 --! Software Released Under the MIT License
11 --
12 --  Permission is hereby granted, free of charge, to any person obtaining a copy
13 --  of this software and associated documentation files (the "Software"), to deal
14 --  IN the Software without restriction, including without limitation the rights
15 --  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
16 --  copies of the Software, and to permit persons to whom the Software is
   ↪  furnished
17 --  to do so, subject to the following conditions:
18 --
19 --  The above copyright notice and this permission notice shall be included IN all
20 --  copies or substantial portions of the Software.
21 --
22 --  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   ↪  IMPLIED,
23 --  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
24 --  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
   ↪  COPYRIGHT
25 --  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
   ↪  ACTION
26 --  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
27 --  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
28 --
29 --  Trevor Vannoy
30 --  Electrical and Computer Engineering
31 --  Montana State University
32 --  610 Cobleigh Hall
33 --  Bozeman, MT 59717
34 --
35 ------------------------------------------------------------------------------┘
   ↪  ---------
36
37 library ieee;                  --! Use standard library.
38 use ieee.std_logic_1164.all;   --! Use standard logic elements.
39 use ieee.numeric_std.all;      --! Use numeric standard.
40
41 ------------------------------------------------------------------------------
42 --
43 --! @brief    tapped_delay_line
44 --! @details  Arbitrary-length delay line with a mux to select a bit from any
   ↪  stage
45 --! @param    clk   Input clock
46 --! @param    rst   Asynchronous active-high reset
```

```vhdl
--! @param    din   Input data bit
--! @param    sel   Mux select line
--! @param    dout  Output data bit selected by mux
--
--------------------------------------------------------------------------------
entity tapped_delay_line is
  generic (
    length : integer := 16
  );
  port (
    clk   : in  std_logic;
    rst   : in  std_logic;
    din   : in  std_logic;
    sel   : in  integer range 0 to length - 1;
    dout  : out std_logic := '0'
  );
end entity;

architecture tapped_delay_line_arch of tapped_delay_line is

  ------------------------------------------------------------------------------
  -- Constant and Type Declarations
  ------------------------------------------------------------------------------

  ------------------------------------------------------------------------------
  -- Signal Declarations
  ------------------------------------------------------------------------------
  signal dout_vec : std_logic_vector(length - 1 downto 0) := (others => '0');

begin
  -- shift in the serial data
  shift_register : process(clk, rst)
  begin
    if rst = '1' then
      dout_vec <= (others => '0');
    elsif rising_edge(clk) then
      -- shift left so the new bit gets shifted into lsb
      dout_vec <= dout_vec(length - 2 downto 0) & din;
    end if;
  end process;

  -- multiplexer
  dout <= dout_vec(sel);

end architecture;
```

mono2stereo_adapter.vhd

```
1  ------------------------------------------------------------------------------
   ↪  ----
2  --
3  --! @file       mono2stereo_adapter.vhd
4  --! @brief      Convert a stream of single channel data to two-channel data.
5  --! @details    Given a stream of signle channel data (e.g. mono audio) in
6  --!             channel-data-valid format, convert the data into two-channel
7  --!             data (e.g. stereo audio). This is done by changing the
8  --!             channel number and asserting the valid pulse every T/2,
9  --!             where T is number of clock cycles per sampling period.
10 --! @author     Trevor Vannoy
11 --! @date       March 2019
12 --! @copyright  Copyright (C) 2019 Trevor Vannoy
13 --!
14 --! Software Released Under the MIT License
15 --
16 --  Permission is hereby granted, free of charge, to any person obtaining a copy
17 --  of this software and associated documentation files (the "Software"), to deal
18 --  IN the Software without restriction, including without limitation the rights
19 --  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
20 --  copies of the Software, and to permit persons to whom the Software is
   ↪  furnished
21 --  to do so, subject to the following conditions:
22 --
23 --  The above copyright notice and this permission notice shall be included IN all
24 --  copies or substantial portions of the Software.
25 --
26 --  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   ↪  IMPLIED,
27 --  INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
28 --  PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR
   ↪  COPYRIGHT
29 --  HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
   ↪  ACTION
30 --  OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
31 --  SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
32 --
33 --  Trevor Vannoy
34 --  Electrical and Computer Engineering
35 --  Montana State University
36 --  610 Cobleigh Hall
37 --  Bozeman, MT 59717
38 --
39 ------------------------------------------------------------------------------
   ↪  ---------
40
41 library ieee;                 --! Use standard library.
42 use ieee.std_logic_1164.all;  --! Use standard logic elements.
43 use ieee.numeric_std.all;     --! Use numeric standard.
```

```
44
45  ------------------------------------------------------------------------------
46  --
47  --! @brief    mono2stereo_adapter
48  --! @details  Given a stream of single channel data (e.g. mono audio) in
49  --!           channel-data-valid format, convert the data into two-channel
50  --!           data (e.g. stereo audio). This is done by changing the
51  --!           channel number and asserting the valid pulse every T/2,
52  --!           where T is number of clock cycles per sampling period.
53  --! @param    clk           system clock
54  --! @param    rst           system reset
55  --! @param    data_in       incoming TDM mono data stream
56  --! @param    channel_in    incoming TDM channel number
57  --! @param    valid_in      valid flag asserted when incoming data is valid
58  --! @param    data_out      outgoing TDM stereo data stream
59  --! @param    channel_out   outgoing TDM channel number
60  --! @param    valid_out     valid flag asserted when outgoing data is valid
61  --
62  ------------------------------------------------------------------------------
63  entity mono2stereo_adapter is
64    generic (
65      sample_rate : natural := 48000;
66      clk_freq    : natural := 24576000;
67      data_width  : natural := 32
68    );
69    port (
70      clk         : in  std_logic;
71      rst         : in  std_logic;
72      data_in     : in  std_logic_vector(data_width-1 downto 0);
73      channel_in  : in  std_logic;
74      valid_in    : in  std_logic;
75      data_out    : out std_logic_vector(data_width-1 downto 0) := (others => '0');
76      channel_out : out std_logic := '0';
77      valid_out   : out std_logic := '0'
78    );
79  end entity;
80
81  architecture mono2stereo_adapter_arch of mono2stereo_adapter is
82
83    ------------------------------------------------------------------------------
    ↪  -----------
84    -- Constant and Type Declarations
85    ------------------------------------------------------------------------------
    ↪  -----------
86    constant cycles_per_sample : natural := clk_freq/sample_rate;
87    -- attribute keep : boolean;
88    -- attribute keep of cycles_per_sample: constant is true;
89    -- signal cnt1 : integer;
90    -- signal cnt2 : integer;
91
```

```
92    ------------------------------------------------------------------------------
      ↪  -----------
93    -- Signal Declarations
94    ------------------------------------------------------------------------------
      ↪  -----------
95
96  begin
97
98    channel_generator : process(clk)
99      variable i : natural range 0 to cycles_per_sample - 1 := 0;
100   begin
101     if rising_edge(clk) then
102       -- reset the counter when new valid data comes in; this ensures that the
          ↪  counter
103       -- starts counting at the beginning of each frame
104       if valid_in = '1' then
105         i := 0;
106       else
107         -- alternate the channel number between 0 and 1 every half period
108         if i < cycles_per_sample/2 then
109           i := i + 1;
110           channel_out <= '0';
111         elsif i < cycles_per_sample-1 then
112           i := i + 1;
113           channel_out <= '1';
114         else
115           i := 0;
116         end if;
117       end if;
118     end if;
119   end process;
120
121   valid_generator : process(clk)
122     variable i : natural range 0 to cycles_per_sample - 1 := 0;
123   begin
124     if rising_edge(clk) then
125       -- reset the counter when new valid data comes in; this ensures that the
          ↪  counter
126       -- starts counting at the beginning of each frame
127       if valid_in = '1' then
128         i := 0;
129       else
130         -- assert valid signal for 1 clock cycle every half period
131         if i = 0 or i = cycles_per_sample/2 then
132           i := i + 1;
133           valid_out <= '1';
134         elsif i = cycles_per_sample-1 then
135           i := 0;
136         else
137           i := i + 1;
138           valid_out <= '0';
```

```
139           end if;
140         end if;
141       end if;
142     end process;
143 
144 -- the incoming data stream may be changing when valid isn't asserted, but we
145 -- want to repeat the valid data in both channel slots. To do this, we'll
146 -- just latch the data every time the input valid signal is asserted.
147 latch_data : process(clk)
148 begin
149   if rising_edge(clk) then
150     if valid_in = '1'  then
151       data_out <= data_in;
152     end if;
153   end if;
154 end process;
155 
156 end architecture;
```