Wideband Full-Stokes parameterized spectrometer for Xilinx RFSoC Platform

FINAL REPORT

Team 41

Advisers

Professor Phillip Jones

Team Members/Roles

Brian Bradford

Vishal Joel

Jared Danner

Louis Hamilton

Nick Knuth

Team Email

sdmay19-41@iastate.edu

Team Website

sdmay19-41.sd.ece.iastate.edu

Revised: 04/29/2019/#1.0

Table of Contents

List of figures/tables/symbols/definitions

CASPER - Collaboration for Astronomy Signal Processing and Electronics Research

Migen - Fragmented Hardware Description Language based in python to automate the VLSI design process.

Xilinx - FPGA manufacturer. *The FPGA used in this project is a Xilinx product.

DSP - Digital Signal Processing.

SDR - Software Defined Radio.

FPGA - Field Programmable Gate Array.

SoC - System on a Chip

0. Executive Summary

We proposed the development of a wideband full stokes parameterized spectrometer (polarimeter) for radio astronomy applications, targeting the Xilinx Zynq RFSoC platform.
Spectrometers are instruments that measure and record the spectral content of signals, such as radio waves received from astronomical sources [8]. Analysis of the output of these instruments provides valuable information to radio astronomers, to better understand the celestial radio sources. The greater the bandwidth of the spectrometer, the more information available to the astronomer, driving the need for new wider bandwidth instruments. However, the progress has been limited by the cost, performance and complexity of development such systems.
CASPER [1], the "Collaboration for Astronomy Signal Processing and Electronic Research" have addressed this problem by developing platform independent hardware and open source software to take advantage of developments in Field Programmable Gate Array (FPGA) and Analog to Digital Convertor (ADC) technology and "quickly" target new platforms such as the Xilinx RFSoC evaluation board (ZCU111).

This project focused on the porting of the CASPER spectrometer instrument and dependent libraries to the Xilinx ZCU111 evaluation platform. We also trialed improving the board support package infrastructure by exemplifying a viable migration from the current CASPER toolflow to instead use Migen, an open source python library for generating and building gateware projects [2]. The toolchain was extended to support the Xilinx PYNQ infrastructure [3], which could be interfaced to existing CASPER Python libraries.

1. Requirements Specification

1.1 High-Level Requirements

The high-level requirements of our project are a successful port of a CASPER spectrometer application, a trial implementation of Migen to test its viability, and extend the current toolflow to use PYNQ. The initial requirement was the main focus of our project as we were testing the ZCU111 and the RFSoC platform's viability in CASPER's toolflow. This included porting existing CASPER IP to see whether it worked on Xilinx's Ultrascale+ architecture and the benefits of the RFSoC as a single cohesive system for the entire spectrometer application. The trial implementation in Migen was also important to test as a new open-source tool to host the CASPER toolflow. Currently, the toolflow uses MATLAB/Simulink, a closed-source application which inhibits the ability for everyone to access CASPER's work. Migen is a newly developed open-source tool for generating HDL source code and Vivado project files, that if successful would increase the accessibility of CASPER's tools. The final requirement of extending PYNQ is to provide better functionality with Xilinx-specific Zynq boards and to give better software control of the hardware design. It allows for software to read the internal hardware devices and registers to configure and receive status data as needed.

1.2 Use-Cases

CASPER [1] is a community of hundreds of scientists and engineers around the world, who collaborate on the development of radio astronomy instrumentation. The CASPER community has reached out to developers to integrate Xilinx's new RFSoC platform into their current tools. Once supported, anyone making use of the open-source tools will be able to quickly develop digital instrumentation for their specific applications.

Should time allow, we planned to pursue the next iteration of the UC Berkeley SETI Research Center's Search for Extraterrestrial Radio Emissions from Nearby Developed Intelligent Populations (SERENDIP) program [4]. This installment would have been the next generation instrument system for the Search for Extraterrestrial Intelligence (SETI) coined SERENDIP VII. The system would have been an open-source, ultra-high resolution, wide-bandwidth dual-pol spectrometer to be used on the world's largest radio telescopes.

1.3 Functional Requirements

- CASPER spectrometer support
    - ZCU111 board must be made to support the CASPER spectrometer and the DSP libraries that it is dependent on.
- Migen
    - Must be able to auto-generate HDL from CASPER modules, create a corresponding Vivado project and generate a bitstream targeting the ZCU111
- Spectrometer
    - Parameterized inputs allow users to quickly configure the application for different use cases

The Xilinx board made to successfully support the CASPER spectrometer is main objective of this project. The plan to move to a newer toolflow that does not rely on MATLAB/Simulink, is dependent on this requirement. Part of moving to the newer toolflow includes writing support for DSP libraries.

Migen must automatically generate HDL code reflecting implemented CASPERized modules. It should create a Vivado project for a user to manually inspect/easily modify what it has created and automatically generate a bitstream targeting the ZCU111. It should also output a mapping of its instantiated registers for PYNQ.

Parameterized input allows for a more user-friendly interaction between the user and application. The applications should be able to be configured so that certain parameters can be enabled or disabled depending on what the user wants to do with the application. This means that the user should be able to configure data such as visualization.

1.4 Non-Functional Requirements

- Backwards compatibility
    - The ability to still use MATLAB/Simulink and existing CASPER Toolflow along with Migen to program the board.
- Open source
    - Anyone can use our code in their application.
- IEEE Standards
    - 211-1997^{TM} IEEE Standard Definitions of Terms for Radio Wave Propagation
    - 1241-2010^{TM} IEEE Standard of Terminology and Test Methods for Analog-to Digital Converters
    - 802.3^{TM} IEEE Standards for Ethernet

CASPER operates entirely under the GNU General Public License V2.0. This grants the use of the source code to anyone for commercial or private use. This also means anyone has the right to modify and/or distribute the code, however CASPER accepts no liabilities or warranties on said source code.

Open source allows other developers to continue unofficial work on the current project, or use the knowledge/code from our project as a code base for other similar projects.

IEEE has several standards related to our project, since we intend this project to be used outside of our own personal use; it's important for our project to meet IEEE standards for easier use in the public domain. The IEEE Std 211-1997TM standard outlines terms and definitions that should be used when discussing Radio Wave Propagation [5]. Since, the spectrometer instrument we intend to build will interpret electromagnetic waves, any terms used to describe this instrument will adhere to this standard. The IEEE Std 1241-2010TM standard outlines terms, definitions and test methods involving Analog-to Digital Converters (ADCs) [6]. The ADC on the boards will need testing, we'll need to ensure it is tested up to industry standards. Any references to the ADC will used this standard as a guide. The IEEE Std 802.3TM standard outlines proper protocols involved in setting up a various speed of an Ethernet connection [7]. Our board support package requires a 10 Gbps Ethernet interface, so following the IEEE protocols for this portion of our project will help guide us to a working interface.

2. System Design & Development

2.1 Design Plan

Our design plan involved designing an ADC, spectrometer, and ethernet module on the ZCU111, in order to properly port an existing CASPER spectrometer. The board infrastructure was designed using Migen and PYNQ. The ADC, spectrometer, and ethernet modules were all derivations of existing CASPER cores and applications. These cores were ported and upgraded to work with the Zynq Ultrascale+ architecture on the ZCU111 board. Migen was used as an experimental tool to test whether it would be a viable option to replace the existing CASPER toolflow. PYNQ was ported to the ZCU111 with interfaces to the three modules in order to provide software access each modules configuration and status registers. Finally, the GUI in Jupyter Notebooks provided an interface for the entire spectrometer application, which would have allowed us to configure various parameters in our spectrometer and display data from throughout the system. More detail regarding the design's various modules, interfaces and the overall system's architecture can be found in the following sections.

2.2 Design Objectives

As stated throughout this report, the overarching goal of our project was to port an existing CASPER spectrometer with dependent modules (ADC & 10Gbit Ethernet) along with additional features such as a graphical interface to view captured and filtered data.

Expected tasks:

1. Porting of the CASPER spectrometer instrument and dependent libraries to the Xilinx ZCU111 evaluation platform.
    a. General board support
    b. Ethernet interface
    c. ADC interface
2. Migrating the current CASPER Toolflow to use Migen, an open source python library for generating and building gateware projects.
3. Support for Xilinx PYNQ infrastructure, which will interface to existing CASPER Python libraries.
4. Documentation of project for future users: source code, gateware, and firmware descriptions.

Additional task:

1. Wideband full stokes parameterized spectrometer (polarimeter) for radio astronomy applications, targeting the Xilinx Zynq RFSoC platform.

2.3 System Constraints

- CASPER spectrometer support
  - ZCU111 board must be made to support the CASPER spectrometer and the libraries it is dependent on.
- Migen
  - Must be able to auto-generate HDL from CASPER modules, create a corresponding Vivado project and generate a bitstream targeting the ZCU111
- Spectrometer
  - Parameterized inputs allow users to quickly configure the application for different use cases
- GUI (GNURadio)
  - The board must be able to communicate with the GUI using an interface.
- Backwards compatibility
  - The ability to still use MATLAB/Simulink and existing CASPER Toolflow along with Migen to program the board.
- Open source
  - Anyone can use our code in their application.
- IEEE
  - 211-1997 IEEE Standard Definitions of Terms for Radio Wave Propagation
  - 1241-2010 IEEE Standard of Terminology and Test Methods for Analog-to Digital Converters
  - 802.3 IEEE Standards for Ethernet
- Time
  - Roughly 6-8 hours expected per week, per team member from beginning of December 2018 to end of April 2019

2.4 Design Trade-Offs

Trade-offs were made when during the design process we chose to use Xilinx specific IP cores in our ethernet and ADC modules, focused on the use of Migen to interface between each subcomponent, and using Jupyter Notebook as our GUI. By choosing to use Xilinx IP and not solely using existing CASPER IP for our block designs we reduce the boards that our project can be ported to. This trade-off decreases the time to develop because the Xilinx IP can just be added without much modification, but reduces the effectiveness of our overall project in CASPER's open-source environment. By using Migen we trade-off existing known solutions such as Vivado to interface and compile each module together for a new test environment. Migen potentially reduces development time if successful because our project can be defined and written in Python which is easier to understand than VHDL. A trade-off of this is unknown compilation and import issues that could arise from using Migen. By using Jupyter Notebook we maintain the open-source goal of the project, but it isn't the most performant solution.

2.5 Architectural and Block Diagrams

Figure 1 outlines a simple, high-level conceptual view of how our spectrometer could be used in a real-world setting.

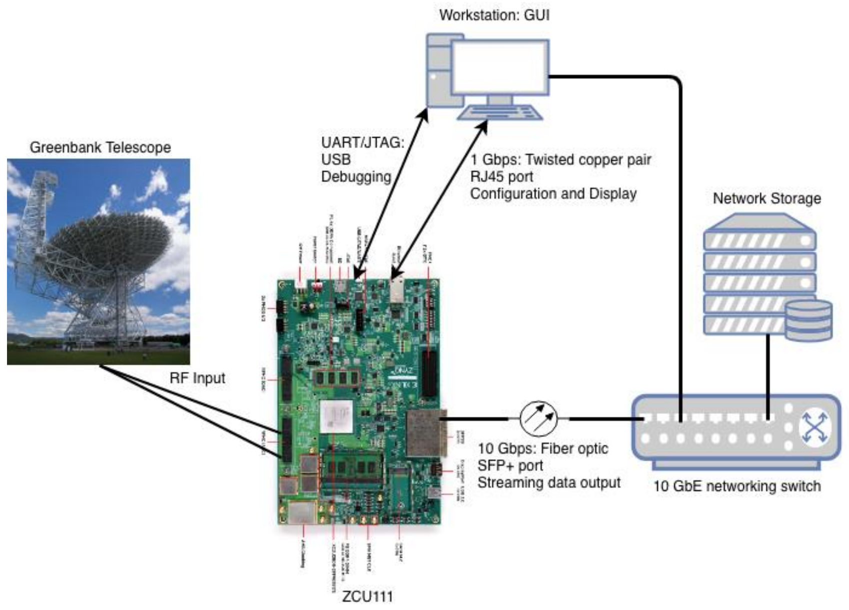


Figure 1. Conceptual Sketch

Figure 2 represents our final architecture for the entire system, which was described in detail in section 2.1 Design Plan.

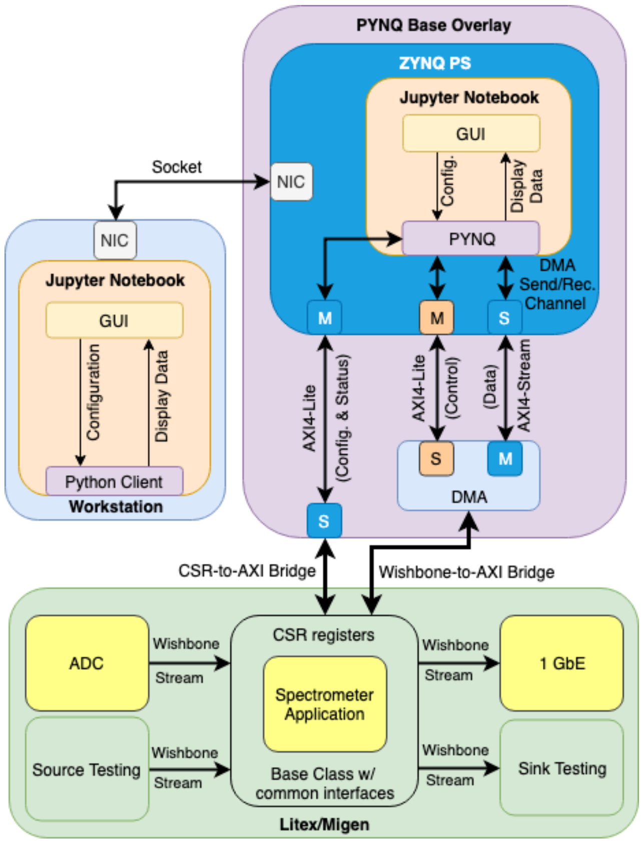


Figure 2. System Architecture

Figure 3 is a functional block diagram that outlines a high-level view of how data flows throughout the system. The ADC and 1 GbE "Yellow Blocks" represent actual peripherals on the ZCU111 RFSoC. The "Green Blocks" represent CASPER DSP library blocks that are implemented in the existing Simulink/MATLAB-based Toolflow.

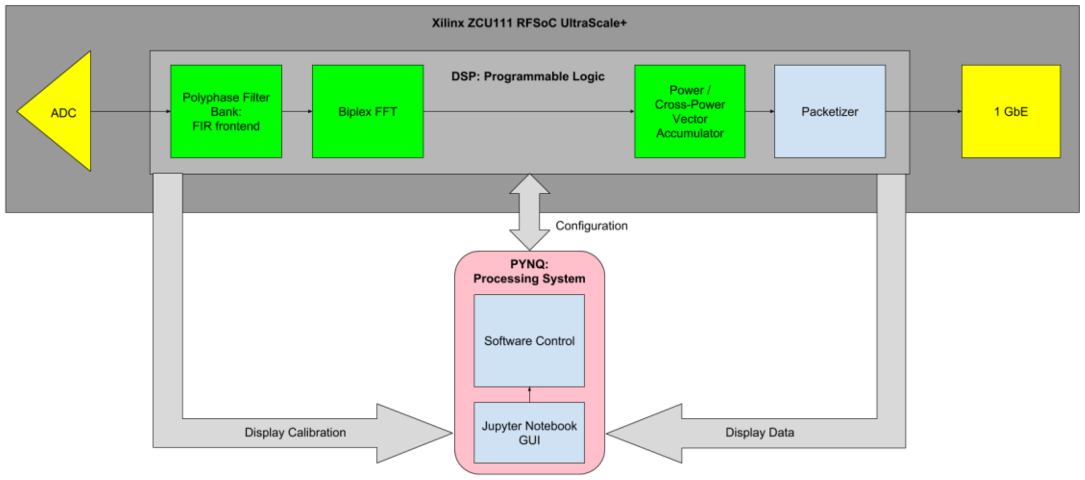


Figure 3. Functional Block Diagram

2.6 Modules, Constraints, and Interfaces

ADC Module
As a quick introduction, the ADC module that our team developed utilized all eight ADCs onboard the ZCU111 to sample RF signal and output I & Q (in-phase and quadrature) data streams to the spectrometer application described below.

To explain in more detail, our ADC design, built in Vivado 2018.2, was built to deliver a sampling bandwidth of 0 to 100 MHz. The ADC module was designed to sample this 100 MHz bandwidth at a sampling frequency of 256 MHz to provide slight over sampling of the desired spectrum. This oversampling allows for filters (spectrometer application) to better handle the I & Q data streams being outputted from the ADC module by allowing a bit of headway in bandwidth. Our team's module, as shown in Figure 5 below, when compared to the original TRD (Figure 5) has had its Direct Memory Components (DMA) and all related IP cores removed. All of this architecture originally was used to interface between Xilinx's TRD & Data Converter GUI. As mentioned above, our spectrometer application inputs I & Q data streams instead of pulling captured sample data from memory. Our team also had no use for the onboard DACs and therefore removed those components as well to reduce the footprint during implementation.

The below Vivado block diagram (Figure 4) is from the original TRD that interfaces with the Xilinx provided Data Converter GUI. This GUI allowed our team to use its pre-built features to quickly customize and test the ZCU111's ADCs.

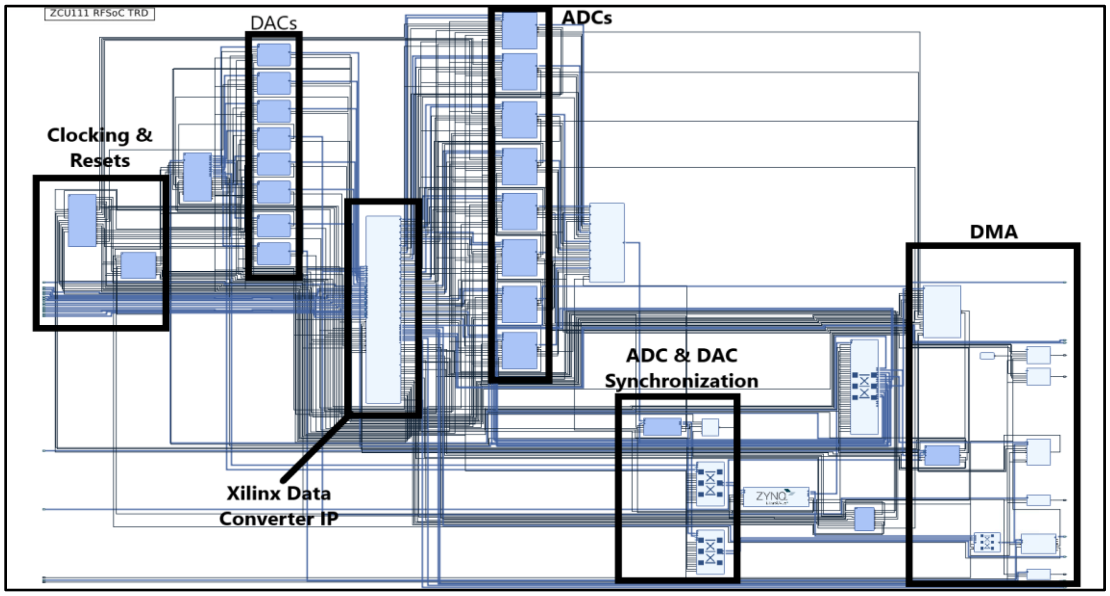


Figure 4. Xilinx ADC Targeted Reference Design

From this given TRD, our team stripped out unneeded components such as DMA & DAC interfacing, to reduce the footprint needed to implement said design onto the FPGA. The goal of our ADC Vivado design was to sample a 100 MHz spectrum produced by a noise generator outputting a 2 GHz signal. This generated signal will be captured at a sampling frequency of 256 MHz. Our design, featured below as Figure 5, shows our purpose-built ADC design. This design utilized all 8 ADCs by passing I & Q data streams straight to the spectrometer application.

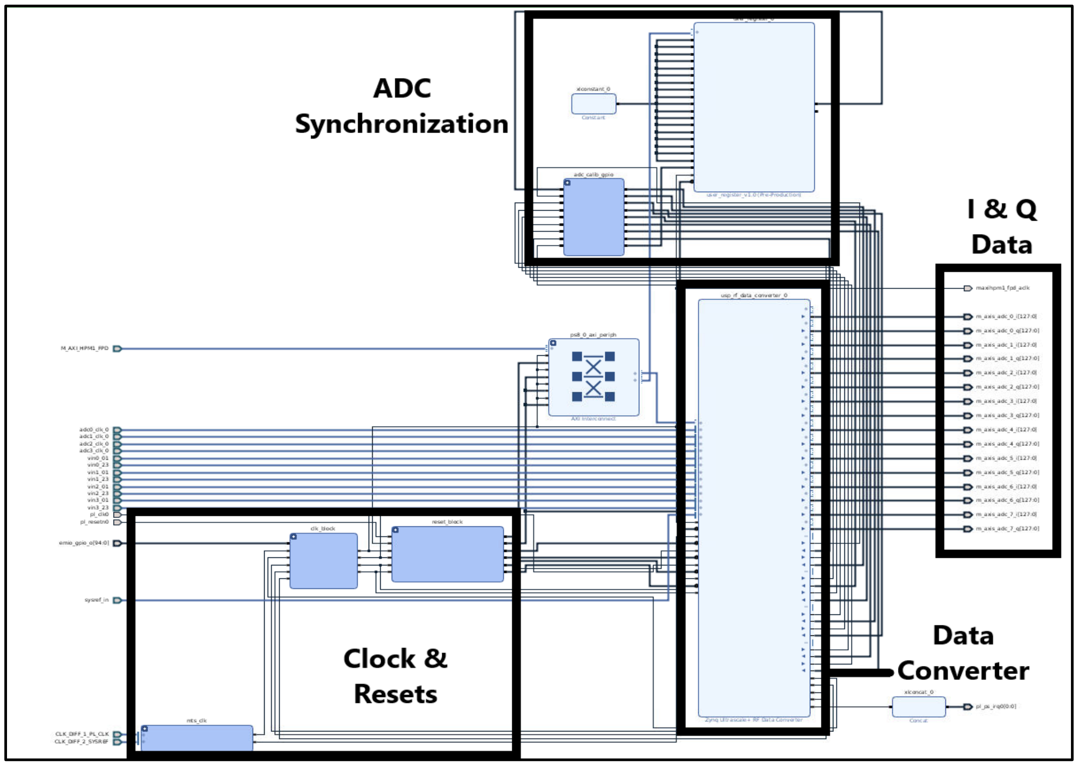
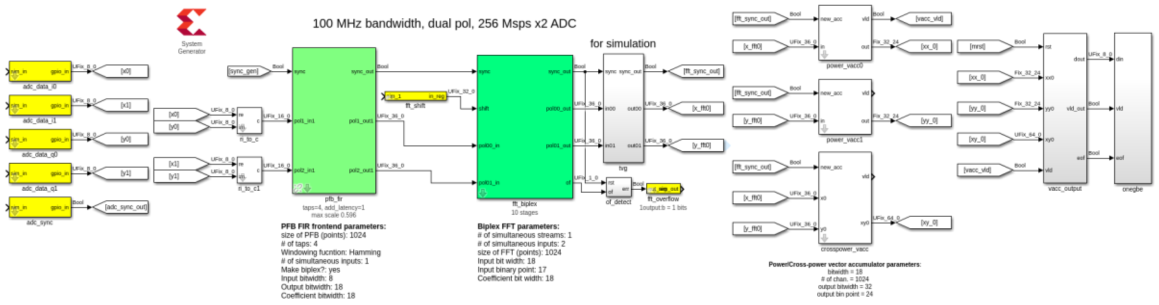


Figure 5. Sdmay19-41 ADC Vivado Design

Spectrometer Module

The spectrometer application takes digitized in-phase and quadrature (I/Q) sinusoidal components from two ADC's operating at 256 MHz. These are converted from real and imaginary data streams to one complex data stream for each ADC. These streams are then fed into a Polyphase Filterbank Finite Impulse Response filter (PFB FIR filter), which divides the signal into parallel 'taps' then applies finite impulse response filters (FIR) with a Hamming windowing function. The output of this block is still a time-domain signal. The Biplex FFT accepts two independent parallel inputs or 'pols', and transforms the time-domain signals to frequency domain bins/channels for each pol. Together, the PFB FIR filter followed by an FFT, makes up a Polyphase Filterbank [8]. The power_vacc and crosspower_vacc blocks take the power of each complex pol or a combination of pols (cross-power) and integrates over a time related to the length of the vector and the size of its output, which outputs the full Stokes parameters representation of the polarization state of an electromagnetic wave [8]. The vacc_output block packetizes the spectrum and creates an 8-bit data stream for the 1 GbE block. The spectrometer design, its corresponding DSP blocks and their parameters can be seen in more detail in Figure 6 below.



PFB FIR frontend parameters:
size of PFB (points): 1024
# of taps: 4
Windowing fucntion: Hamming
# of simultaneous inputs: 1
Make biplex?: yes
Input bitwidth: 8
Output bitwidth: 18
Coefficient bitwidth: 18

Biplex FFT parameters:
# of simultaneous streams: 1
# of simultaneous inputs: 2
size of FFT (points): 1024
Input bit width: 18
Input binary point: 17
Coefficient bit width: 18

Power/Cross-power vector accumulator parameters:
bitwidth = 18
# of chan. = 1024
output bitwidth = 32
output bin point = 24

Figure 6. Spectrometer Application design

1 GbE Module

The 1 GbE module is the main data transfer module for the entire project. It will output the resulting 8-bit data stream from the spectrometer application to any device connected to board. The main components of this module are the Zynq PS, UDP module, MAC module, and 1G/2.5G PCS/PMA. The Zynq PS is the processing system for the ZCU111. It provides the reset and system clock signals for the rest of the components in the module. The UDP module is exactly that. It is responsible for packaging the data following the Transport layer protocol. This module is also responsible for receiving data and unpacking it. The MAC module is responsible for handling the addressing of the board and implementing the rest of the Data-Link layer protocol. It must transfer data appropriately from the UDP module to the PCS/PMA module and from the PCS/PMA module to the UDP module. The PCS/PMA module is responsible for handling the physical layer of the ethernet stack. Figure 7 below is our implementation of this module. Each component is highlighted with a number next to it in the order discussed above.

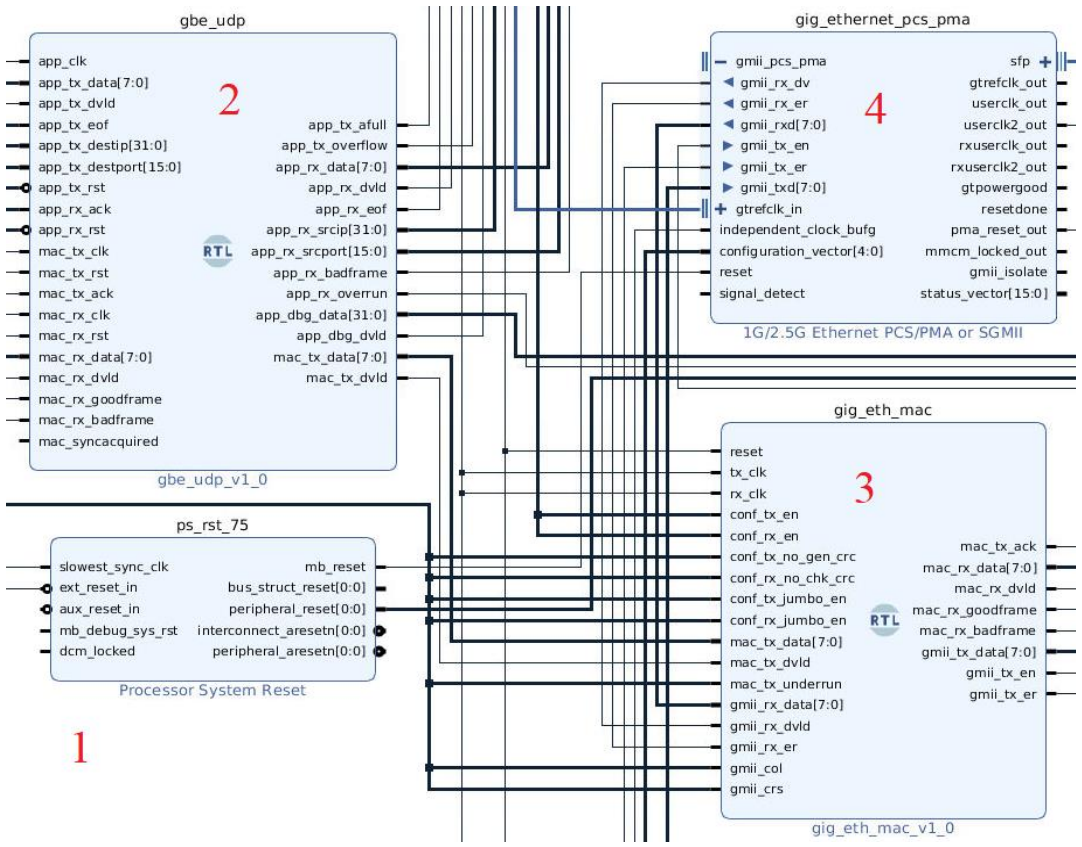


Figure 7. Ethernet Module

GUI Module

The GUI consists of three main parts:

- A Python server (On the board itself)
- An interactive GUI (On the board itself)
- A non-interactive GUI (can be any computer on the same network as the board)

The Python server will start by accessing the board's PYNQ Overlay, and use PYNQ's "recv"/"send" channel to communicate with the DMA's AXI4-Stream interface. The server will then communicate with interactive GUI, waiting for its connection and requesting a configuration file (JSON), if none is received the board will work with its default settings, else the server will use the AXI4-Lite interface to send the configuration data. Once the board is configured (or not), the server will sample from the board using the "recv" channel that connects with the AXI4-Stream interface and then send the data to the interactive display for plotting, while waiting for new configuration files. The non-interactive display will wait for the server to start and will only receive data and plot it on the display. The data will move between the server and the clients (displays) by using sockets.

Interfaces
The ADC, Spectrometer, and 1 GbE modules were all designed to inherit from a base Migen module. This base module featured a common Wishbone streaming interface on both the input and outputs. This would allow for ease of swapping out actual modules with Source/Sink testing modules that would allow each individual component to be properly unit testing by feeding in a corresponding source and verifying the expected output. This interface would also be used to stream data in and out of DMA from the programmable logic with a Wishbone-to-AXI4-Stream bridge. From here, the PYNQ software could easily generate test signals to write to DMA and confirm that data at various parts in the design are correct by reading from DMA.

The base Migen module that the ADC, Spectrometer, and 1 GbE modules inherited also featured a CSR bus and an CSR-to-AXI4-Lite bridge. The CSR bus is a low-bandwidth, resource-sensitive bus implemented by Migen/Litex, which was designed for accessing the configuration and status registers of cores from software. Each module could instantiate its own configuration and status registers, which would be automatically added to a bank; this bank would be connected to the CSR bus and bridged to AXI4-Lite interfaces exposed by the PYNQ base overlay. This allowed PYNQ to write/read to the configuration/status registers for each respective module, since any registers and address space attached to an AXI4-Lite interface would be mapped into memory. The Migen/Litex build environment that we created would output a CSR mapping of the registers in a python dictionary, which could be imported into the PYNQ base overlay. He wrote functions that would create a PYNQ generic IP class from each modules mapping. This would expose a corresponding module's registers with friendly names in order to read/write from/to them easily, which replaced referencing them via the interfaces base address and the registers corresponding offset. This matched the PYNQ functionality of how Xilinx's own IP cores could be managed.

These interfaces can be seen in relation to the rest of the system's architecture in Figure 2 above.

3. Implementation

3.1 Implementation Diagram, Technologies, and Software

ADC Module
The ADC subcomponent that our team worked on was derived from Xilinx's Targeted Reference Design (referred to below as TRD). This TRD was designed to work with Xilinx's Data Converter IP which allows quick interaction & configuration access to the ADCs & DACs onboard the RFSoC ZCU111. The TRD was used in tandem with a Xilinx provided Graphical User Interface (GUI) to verify custom system parameters via loopback tests. The data converter GUI is dependent upon LabView by National Instruments. All HDL, block diagram, and logic development work were conducted in Vivado 2018.2. External hardware that was used during the development process included Xilinx's RFSoC ZCU111, a 100 MHZ low pass filter, a combination of 1.2 & 1.4 GHz band pass filters, noise generator, & spectrum analyzer. Software used during development included PYNQ and VHDL.

Spectrometer Application
The spectrometer application was derived from a number of different sources. Our industry consultant, Jack Hickish, provided us with a Polyphase Filterbank-based (PFB) Simulink spectrometer application that he originally designed for the Allen Telescope Array in Northern California using the existing CASPER Toolflow [11]. In order to determine the proper DSP settings for our spectrometer, we modified an interactive Jupyter Notebook [12] originally created by Danny Price as a supplemental tutorial for his paper: Spectrometers and Polyphase Filterbanks in Radio Astronomy [8]. Once our spectrometer design was completed, Xilinx/MATLAB System Generator generated the corresponding HDL code for the spectrometer application. This generated HDL code was wrapped and instantiated inside of a Migen module.

Ethernet Module
The technologies used in implementation of the ethernet subcomponent of our design are the existing CASPER IP for ethernet design. These VHDL source files were able to be imported into the software used in implementation. They included a MAC protocol block, a UDP protocol block, an instance of a MicroBlaze core, and certain loopback testing blocks to aid in troubleshooting. The software used in implementation of the ethernet subcomponent include Vivado for synthesis and implementation of the sources for our ZCU111 FPGA and common Linux networking utilities used for troubleshooting and testing. Vivado was key in both importing the existing CASPER IP and interfacing it with design components updated for the ZCU111. The Linux network utilities (tcpdump, ping, etc.) allowed for monitoring of the ZCU111's network traffic and ensuring that packets are sent and received properly.

GUI Module
One of the technologies used for the implementing the GUI module was a PYNQ Z1, using a Xilinx provided FIR filter IP core, Zynq 7000 PS IP and Direct Memory Access IP. The board was synthesized using Vivado with these IPs. The next technology used Jupyter Notebooks, which comes with PYNQ on the board. Libraries used in Jupyter Notebooks were matplotlib, NumPy, socket, Thread, ipywidgets and pickle.

3.2 Rationale for Choices

ADC Module
As stated in the "ADC Module" portion of 3.1, the Xilinx provided Targeted Reference Design (TRD) was chosen by our team to be the platform we build from due to its prior validation and encouragement of use by Xilinx. By providing the team with a fully functional baseline implementation of an interface to the ADCs & DACs, this platform allowed our team to verify custom system parameters to insure they would produce the desired sampling bandwidth and data output.

Vivado 2018.2 was chosen as our development version for a few reasons. 1) 2018.2 was the latest version of Vivado at the time our project started and 2) 2018.2 included hardware support for the RFSoC ZCU111.

Spectrometer application
This spectrometer application mentioned in the previous section was able to be modified to meet the constraints of the 100 MHz bandwidth spectrometer we were targeting for this project. The Jupyter Notebook also described in the previous section allowed us to quickly create a test signal, feed it through a PFB spectrometer, and in real-time tweak the parameters for the PFB Finite Impulse Response frontend, and Fast-Fourier Transform (FFT) backend of the spectrometer application. These parameters included the number of taps, number of channels, type of windowing-function and integration period described in the original Jupyter Notebook [12]. After these parameters were chosen, we consulted again with Jack Hickish as well as Alan Wilson-Langman to verify and implement the spectrometer using CASPER's Simulink DSP libraries. The System Generated HDL code was wrapped and instantiated inside of a Migen module along with various configuration/status registers in order to control the spectrometer from software via PYNQ.

Ethernet Module
The choice made to use existing CASPER IP for ethernet was made so that the project would be more portable and fit better into CASPER's open-source environment. It was also done in an attempt to make development of the ethernet subcomponent easier as rather than rewriting VHDL for each part of the entire ethernet stack only minor changes should have had to be made. Using Vivado was an obvious choice because the ZCU111 is a Xilinx FPGA and Vivado is the Xilinx design software. It is also the software used as a backend for CASPER's current toolflow. The Linux network utilities were used instead of a third-party application like Wireshark because they are easier to setup and run and came installed on the computer used.

GUI Module
The GUI application's main focus is to offer visualization through and from the board as well as easier access to configuring parameters through the PYNQ registers. The program isn't integrated into the main spectrometer application but uses a server/client process to communicate with the board. This was initially done to make user interaction with the spectrometer application easier. Rather than manually setting registers, or typing in commands the GUI will allow all of this to be done via the ipywidgets module from Jupyter Notebooks.

3.3 Standards and Best Practices

ADC Module
During development and testing, the IEEE Std 1241-2010TM: IEEE Standard of Terminology and Test Methods for Analog-to Digital Converters was followed. The procedures and guidelines stated in Std 1241-2010 ensures that our team followed proven metrics for testing and verifying the ZCU111's ADCs.

Ethernet Module
The only applicable standard used in development of the ethernet subcomponent is IEEE 802.3 [7], which describes the protocols for setting up and operating various speeds and bandwidths of ethernet. This protocol was used as a guide to ensure we were completing all of the prerequisite components for an ethernet connection to the FPGA. It also was used to ensure our configurations and settings for our ethernet are correct. For the VHDL coding done it was all modifications to existing VHDL code and simply followed the existing formatting and styles of each file.

GUI module
The application follows a PEP 8 standard - which is a coding style guideline for Python Code. PEP 8 accounts for standardizing the use of comments, naming conventions, whitespace and programming recommendation. The application uses this standard to avoid consistency issues between two developers of different coding backgrounds. This standard also improves on layout and the readability of the code.

4. Testing, Validation, and Evaluation

4.1 TEST PLAN

ADC Testing Methodology:
The ADC sub-component will be tested in the following ways:

1. The Xilinx provided TRD has been verified via the Xilinx Data Converter GUI application. Through this GUI, a loopback test was conducted via the DACs (noise generation at a certain specific frequency) & ADCs (sampling and FFT).
2. Our ADC design will be verified via two methods:
    a. The Xilinx provided GUI to confirm the parameters of our target design.
    b. A 2 GHz noise generator in tandem with 100 MHz low pass filter feeding a sinusoid into the ADCs which will stream out I & Q data that, when feed into a spectrum analyzer, should produce the inputted sinusoid.

Spectrometer Testing Methodology:
The spectrometer sub-component was planned to be tested in the following ways:

1. Create a Jupyter Notebook to simulate and verify the DSP parameters necessary for 100 MHz spectrometer design
2. Simulate in Simulink by generating white-noise, injecting a signal and feeding the samples to the fully implemented spectrometer design made with CASPER's Simulink/MATLAB DSP libraries
3. Read/write the input/output data for the spectrometer component from/to DMA using PYNQ:
    a. Generate a sampled signal using NumPy in PYNQ, write to DMA, read from DMA in hardware and feed to spectrometer HDL
    b. Write the spectrum to DMA in hardware, read spectral data from DMA in PYNQ and display plot to verify

Due to time constraints, the first two methods of testing were only performed. This was because the third required a working DMA IP core inside of our ZCU111 PYNQ base overlay. Louis was able to demonstrate a functioning DMA IP core based on a FIR filter example application on the PYNQ-Z1 board but ran out of time to port this core to our base overlay.

1 GbE Testing Methodology:
The testing plan for the ethernet subcomponent was as follows: loopback testing to ensure that a basic implementation of the ethernet operates with transmit and receive, a MicroBlaze core is added and tested to complete the functionality of CASPER's existing implementations of ethernet, and the subcomponent would then be integrated with the rest of the project and tested to ensure functionality is maintained. All of the testing is done manually through the use of network tools in the Linux development environment and iterations of the subcomponent with debugging registers and LED's set up. The testing covers all functionality of the ethernet subcomponent including sample data testing.

GUI Methodology:
The testing plan for the GUI subcomponent was as follows:

1. Use a Jupyter Notebook from FPGA developer [13], with access to the DMA's IP core, which itself was implemented with a Xilinx FIR IP core, writing a NumPy generated signal to the DMA and reading from the same DMA.
2. Write to the GPIO control registers using the GUI control surfaces, and verify the GUI can write to control registers.
3. Create a Jupyter Notebook to create threads to test communication between a client process and a server process.

4.2 Unit Testing

ADC Unit Testing:
The unit tests planned for the ADC module involved a slightly modified ADC design that included DMA IP cores. This would enable the usage of a PYNQ overlay built to interface / configure / and feed data into & out of the ADCs memory registers. This PYNQ interface would allow for developers to have quick access to a predefined memory map that would hold offsets to specific internal ADC parameters. Using this defined memory map, our team would be able to capture data via the customized ADC cores, stream that data into DMA, and view the captured values via our team's GUI.

These tests were unable to be completed due to an incomplete DMA infrastructure in the PYNQ base overlay. This lack of completed technology hindered our interface into the system. The ADC parameter's memory offsets were collected but not placed into our team's main PYNQ overlay.

Spectrometer Unit Testing:
The spectrometer module was planned to be unit tested by utilizing the common Wishbone streaming interfaces with a Wishbone-to-AXI4-Stream bridge to the DMA core in the PYNQ base overlay. PYNQ could generate a sampled sinusoid using NumPy, which emulates the digitized samples that would come from the ADC core and write that data to DMA. The spectrometer module would read this data in from DMA and behave normally and write its respective data back to the DMA. PYNQ would then read this data out of DMA and display a plot of the data for us to carefully consider and ensure that the resulting information is in fact correct and what we expected. We were not actually able to perform this unit test as the DMA infrastructure in the PYNQ base overlay was not added. Louis was able to demonstrate a working DMA demo via PYNQ with a different hardware design but ran out of time to implement this infrastructure in our hardware design. The Migen/Litex infrastructure to support this sort of testing was implemented and ready to be integrated.

1 GbE Unit Testing:
The initial testing done was to test whether an implementation of just the ethernet stack can transmit and receive data in a loopback setup. This was done using the CASPER IP to setup the ethernet subcomponent in a loopback manner and test whether it actually works by pinging the ZCU111 from our development environment. Further testing setup included pulling the clock signals to a counter and flashing an LED to ensure they are working and looking at the status registers for the various components to make sure they are also working. The next step in testing would be to add an instance of the MicroBlaze core and test that the previous functionality is maintained along with adding ICMP functionality, ARP requests, and DHCP. This would be done in the same fashion as the earlier tests with manual checking of network traffic and status LED's and registers. The final testing of the subcomponent would be to integrate it with the other subcomponents in Vivado and Migen. This would be done by wrapping the subcomponent in a common interface to include as a block in a new Vivado project with each subcomponent included similarly. Each would be connected and tested to ensure previous functionality is maintained along with the new functionality of taking data from the spectrometer and transmitting it to the development environment.

GUI Unit Testing:
The initial testing was done using a Jupyter Notebook project found on FPGAdeveloper., this project implemented two FIR Filters, one using python libraries and another implemented using Xilinx IP core. NumPy was then used to generate signals that will be passed to both filters and the times compared. The next step was to test each process, server and client. The server component was tested by using the FIR hardware implementation with NumPy, generating signals passing them to one server thread and the other a client thread waiting to verify that data. The Client process was tested by verify control surfaces, set certain values-based input. The next step was to use these control surfaces with a client process to connect to a server process, which would then write to GPIO control registers.

4.3 Interface Testing

The AXI4-Lite/CSR interfaces were tested by instantiating various configuration and status registers in the Spectrometer module as well as a 'test module'. This test module included a wide range of different possibilities in order for us to spot any mistakes in the functionality of the interfaces/bridges. These two modules registers were automatically added to CSR banks which hooked up to each modules respective CSR bus. Each modules CSR bus was bridged to a corresponding AXI4-Lite interface exposed by the PYNQ base overlay. We demonstrated that each of these interfaces and register accesses worked by reading and writing to each module's registers in PYNQ, also by hooking a configuration register up to the board GPIO LEDs and a status register up to the board's GPIO dip switches and push buttons. For a couple weeks, we had to debug problems with these interfaces, which mostly came from a lack of documentation, lack of examples on Migen/Litex's CSR-to-AXI4-Lite bridge, and a misunderstanding of the underlying functionality of Xilinx's AXI Interconnect IP. In the end, he still had problems where the interface could only properly read/write to every 4th, 32-bit register, probably resulting from CSR's word addressing and AXI4's byte addressing. He was able to mitigate this issue by making a workaround that instantiating a dummy register of size' 128-desired_reg_size' after the instantiation of the desired register. His CSR mapping dictionary simply ignored these dummy registers before exporting.

The GUI's socket interface was tested by creating two threads in Jupyter Notebooks that would simply attempt to send and receive data between one another and verify a client process was able to connect to a server process. From there the next test created two threads, with a client process sending data to a server process, with access to PYNQ, and wrote to GPIO LEDs using configuration registers mentioned above.

4.4 System Integration Testing

We were able to confirm that the automated Migen/Litex build environment with common base Migen modules and a functioning PYNQ Base Overlay he created functioned properly by the testing described in the Interface Testing section above. This demonstrated a number of system components and their proper integration: a functioning Migen base module with configuration interface, functioning interface bridges to the PYNQ base overlay, a functioning PYNQ image and PYNQ base overlay (excluding DMA IP core), and PYNQ properly running a Jupyter Notebook server on the processing system.

Had the ADC core and 1 GbE cores been completed, they would have been wrapped in a Migen module and connected via the common Wishbone streaming interface. Had the DMA IP core been successfully implemented in our PYNQ base overlay, we would have integrated the ADC and spectrometer modules (after proper unit-testing), generated a sinusoid in the lab using a signal generator, and fed this signal to the ADC's. We would have streamed the output of the spectrometer module directly to DMA and used PYNQ to display a plot of the spectrum (frequency vs power [dB]). We would have been able to verify a spike in power at the exact frequency of the lab-generated sinusoid. We would have integrated only the spectrometer and 1 GbE modules together and generated a sampled sinusoid using NumPy in PYNQ and streamed the data over the Wishbone interface as an input into the spectrometer module. We would have then used python on a lab computer connected to the same network as the ZCU111 to deconstruct the UDP packets and display a plot of the spectrum (frequency vs power [dB]). We would have been able to verify a spike in power at the exact frequency of the PYNQ/NumPy generated sinusoid. We would then have moved on to the following procedure described in the next section.

4.5 Validation and Verification

The functionality of the entire integrated system could have been validated by first feeding a lab-generated sinusoid to the ADC's and using python on lab computer connected to the same network as the ZCU111 to deconstruct the UDP packets and display a plot of the spectrum (frequency vs power [dB]). We would have been able to verify a spike in power at the exact frequency of the lab-generated sinusoid. The final, ultimate test would have been to use our noise-generator to generate white-noise and inject a lab-generated sinusoid into this signal. Repeating the procedure described above, if we would have been able to see a clear spike in power at the exact frequency of the lab-generated sinusoid, then the spectrometer would have been shown to function properly according to our client, CASPER.

4.6 Evaluation

The ADC portion of this senior design project had a few different sub tasks that needed to be verified and evaluated. The first thing that needed to be verified was the Xilinx provided Targeted Reference Design to ensure that the ZCU111's ADC & DAC components were operational. This evaluation consisted of performing a loopback test by generating an output signal at a certain frequency (shown below as Figure 8), running that signal through a series of bandpass filters, and reading in that signal through the ADCs. After many trials at different frequencies, our team was confident that our RFSoC ZCU111's RF oriented components were functional. These tests were run with the Xilinx Data Converter GUI Tool. After this series of tests, this TRD & Data Converter GUI were used to test our ADC design's target parameter values to verify that these values produced the expected and desired results.

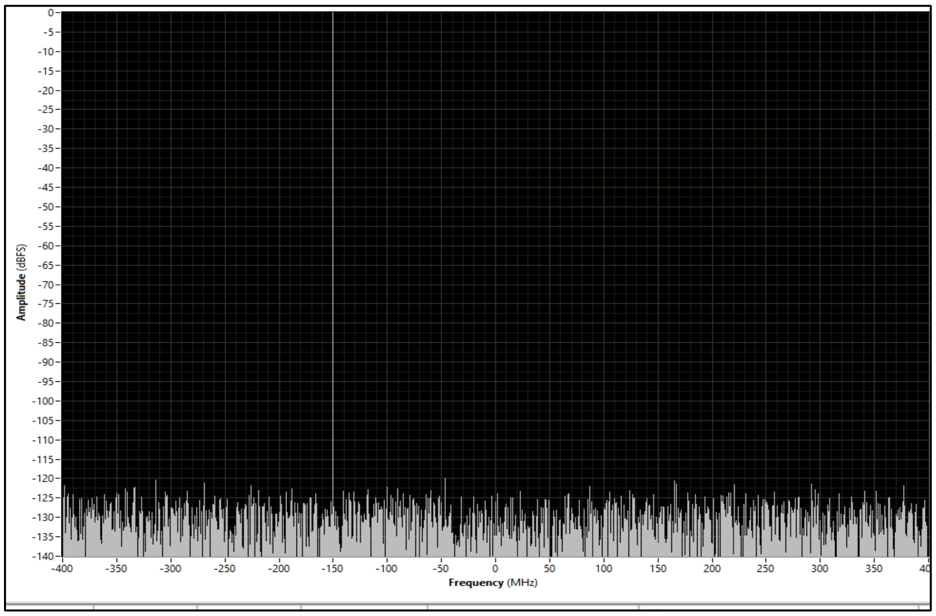


Figure 8: TRD Loopback Test (DAC Generation)

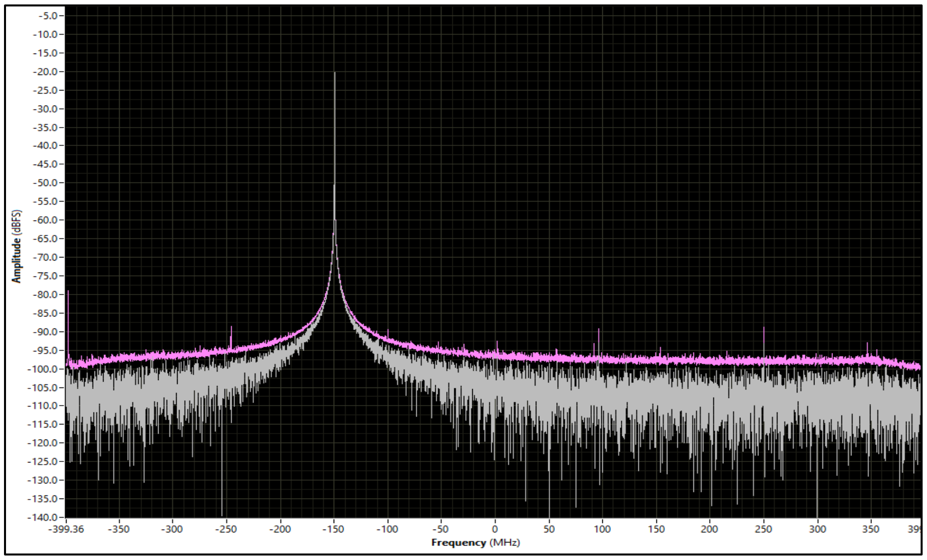


Figure 9: TRD Loopback Test (ADC Capture w/ FFT)

The hardest part of the ethernet module was deciding what implementation to go with and actually getting it successfully implemented. We were able to finally import the proper CASPER IP needed and implement a basic design. At this point in the project we were able to confirm that the project is able to be successfully implemented on the board. We are still currently testing the loopback interface as timing issues between each of the clocking sources involved are numerous.

We were able to confirm that the automated Migen/Litex build environment with common base Migen modules and a functioning PYNQ Base Overlay he created functioned properly by the testing described in the Interface Testing section above. This demonstrated a number of system components and their proper integration: a functioning Migen base module with configuration interface, functioning interface bridges to the PYNQ base overlay, a functioning PYNQ image and PYNQ base overlay (excluding DMA IP core), and PYNQ properly running a Jupyter Notebook server on the processing system.

The GUI was able to read data from a DMA IP core on the PYNQ-Z1, and plot the data for display. The Server process was tested using threading, for verification of communication with a client process. Once the communication was established between the board and the work station, simple plots were made reading from the PYNQ-Z1's DMA IP core, then said data was also sent over socket to the workstation client process for verification and plotting.

5. Project and Risk Management

5.1 Task Decomposition & Roles and Responsibilities

PYNQ

1. Create and test a ZCU106, then ZCU111 PYNQ image
2. Port generic ZCU104 base overlay to ZCU106 and verify functionality
3. Port generic ZCU106 base overlay to ZCU111 and verify functionality
4. Design base overlay for ZCU111 to be adapted from generic
5. Test configuration and status register access from Migen-generated module code via CSR-to-AXI4-Lite bridge
6. Create PYNQ function that will inherit Default IP class to have common attributes as Xilinx IP in PYNQ, this function would parse CSR register map and define a register map in PYNQ
7. Implement ZCU111 base overlay for application that exposes modules/registers with friendly names
8. Create and test a PYNQ-Z1 base overlay.

Build Environment (Migen/Litex)

1. ZCU106/111 platform support
    a. Create Migen platform file for mapping of inputs/outputs to physical FPGA pins
    b. Create simple test application that blinks LED's verifying board can be targeted
2. Base Migen module for CASPER
    a. Create a base Migen module that all other CASPER modules will inherit from with common Wishbone stream interfaces for data and configuration/status interface
    b. Make a test module that instantiates wide-range of configuration and status registers
3. Bridging Migen design with PYNQ base overlay

    a. Create TCL script that will instantiate PYNQ base overlay wrapper inside of a new Migen-generate Vivado project
    b. Design an interface architecture that allows access to registers created in Migen from PYNQ
    c. Design interface architecture that allows ease of unit testing each Migen module via PYNQ (DMA)
    d. Implement CSR-to-AXI4-Lite bridge for register access
    e. Create configuration/status register map python dictionary for PYNQ

ADC

1. Verify TRD via the Xilinx provided Data Converter GUI Tool
    a. Perform loopback test using onboard ADCs & DACs
2. Remove unused logic from Xilinx TRD
    a. Correctly extract DMA IP cores
    b. Remove DAC synchronization logic
3. Split out I & Q data from TRD
    a. Split I & Q data streams from each ADC core
    b. Remove unused logic (IQ merging, multiplexed output, etc.)
    c. Verify correct bus width for I & Q streams to ensure they properly connect to the spectrometer application
4. Collect memory offsets for important ADC system parameters to be used within PYNQ

1/10 GbE core

1. Porting 10GbE application
    a. Port existing 10GbE application from CASPER
    b. Upgrade IP to new board
2. Investigation into alternatives to 10GbE
    a. Example Xilinx implementations of 10GbE core
    b. Determine whether slower ethernet such as 1GbE is acceptable
3. Porting 1GbE applications
    a. Use existing 1GbE applications to base new project off of
    b. Import CASPER IP in place of Xilinx IP that is restricted
4. Testing 1GbE implementation
    a. Loopback testing
    b. MicroBlaze core implementation testing
    c. Sample data testing
5. Wrap 1GbE in common interface
    a. Use interface developed in Migen

Spectrometer application

1. Adapt Jupyter Notebook [12] that simulates PFB-based spectrometer in order to determine DSP parameters for targeted spectrometer
2. Consult with Jack Hickish/Alan Wilson-Langman and modify example spectrometer [11] application for our targeted spectrometer
3. Rework vacc_output (packetizer) block to match 1 GbE data stream
4. Generate HDL code via Xilinx/MATLAB System Generator
5. Create Migen Spectrometer module that instantiates system generated HDL code
6. Expose all input/outputs needing to interface to ADC or 1 GbE module
7. Instantiate all relevant configuration and status registers related to spectrometer application

GUI

1. Use Jupyter Notebook to simulate a sine wave graph (using dummy data) to verify visualization through Jupyter notebooks.
2. Adapt Jupyter Notebook's ipywidgets to configure GUI parameters.
3. Use Python json module to adapt a configuration file that can be initialized/modified by the user.
4. Integrate the graph view with the PYNQ's python libraries so that it reads and displays data from accessing PYNQ's registers.
5. A client process to server process communication using sockets to transmit data from board to workstation.

Roles and Responsibilities

Brian Bradford - FPGA/DSP Engineer

- Build Environment (Migen/Litex)
- PYNQ tasks: 1-7
- Spectrometer application

Louis Hamilton - FPGA Engineer and GUI developer

- GUI design & implementation
- PYNQ task: 8

Jared Danner - FPGA Engineer

- ADC Design & Implementation
- ZCU111 Migen Platform File

Nick Knuth - FPGA Engineer

- 1/10 GbE core

Vishal Joel - GUI Developer and FPGA Engineer

- GUI design & implementation

5.2 Project Schedule – Gantt Chart (Proposed vs Actual)

Figure 9 shows our proposed Gantt chart at the end of the first semester of senior design. Figure 10 is the actual Gantt chart at the end of this semester of senior design. The largest reason for the discrepancies between the two is unforeseen issues that arose during implementation and testing. These issues are highlighted in detail in the following section to explain the large extensions in time to each component.

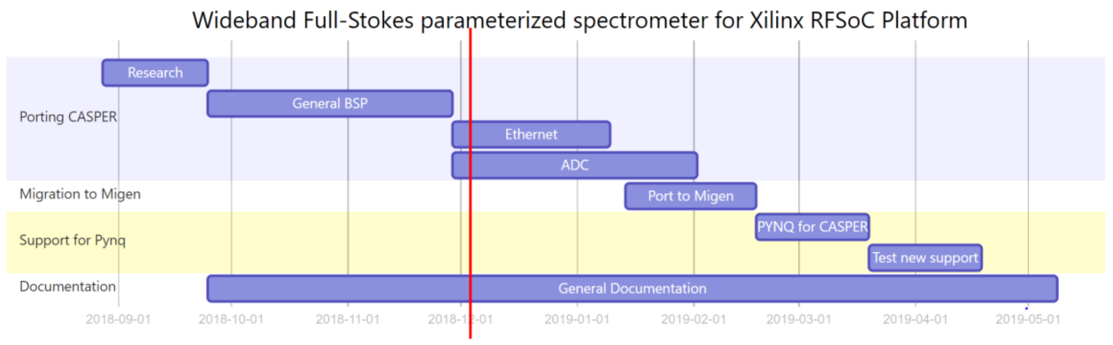


Figure 9. Proposed Gantt Chart

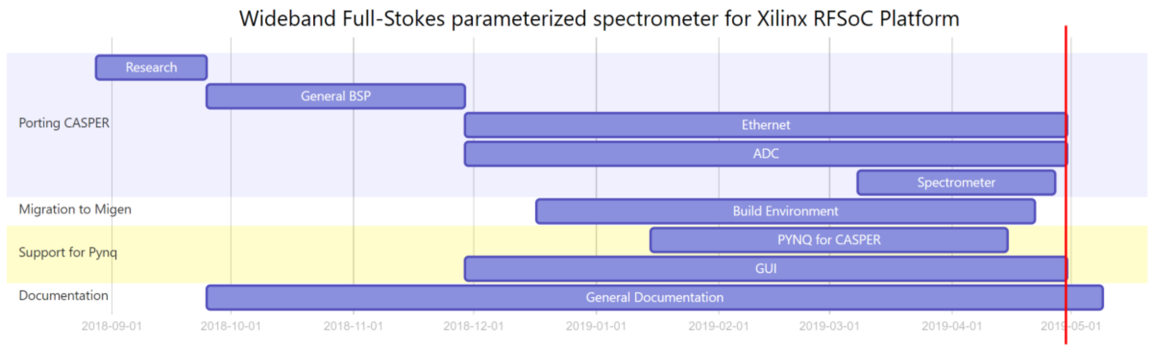


Figure 10. Actual Gantt Chart8

5.3 Risks and Mitigation (Potential vs Actual)

For the ADC portion of the project, our team's initial goal was to create a functional ADC component that, when utilizing all eight ADC cores, could sample at rates high enough to measure the Hydrogen-Emission line. This was not achieved due to lack of knowledge of how complex these systems were and how significant prior experience was to the completion of the component. Although this hindered the success of the ADC design, the team did spend a large amount of time researching other similar designs from industry/universities to gain a better understanding of what we needed to complete to achieve our goals.

The ethernet module portion of the project incurred many unforeseen issues. The first issue to come up was our own fault for assuming the porting of the existing CASPER implementations for 10GbE ethernet would be simple. Once we were finally able to specify the correct board settings in Vivado after importing the CASPER project Vivado immediately signaled outdated IP. Upon upgrading the IP to their current versions, it still threw an issue. The CASPER implementation used a Xilinx IP core that was unsupported on the ZCU111's Ultrascale+ architecture. This was a major issue because we do not have the license for the 10GbE IP core that was necessary for the existing implementation. We made the decision at this point to pivot to 1GbE. We first used an example implementation that Xilinx had and attempted to get that to work. This ran into the same issues before where we do not have the license required to implement the design. From this point we began to try to merge what we could from the Xilinx example project and older existing CASPER implementations. This began with looking at a SNAP board project that was somewhat similar to our goal. We attempted to use this, but the MAC and UDP blocks from this project were too different to our desired implementation due to larger differences in architecture. At this point we started to use a VCU118 implementation. This implementation was much more similar to the Ultrascale+ architecture and allowed me to import the MAC and UDP blocks and begin testing. This came with its own unforeseen timing issues due to difference in standards from the ZCU111, VCU118, and CASPER's toolflow in MATLAB.

Initially the plan was use to GNURadio for the GUI application, however due to time and functional constraints discussed in 2.3, the team decided to use Jupyter notebook directly from the PYNQ image. This is because PYNQ was later installed onto the FPGA board, which comes with support for Jupyter notebook out of the box as well as other Python modules that allow visualization and configuring data. Therefore, instead of going through an extra process of getting GNURadio installed and communicating with the board, the team agreed on using Jupyter Notebook and other python modules directly.

5.4 Lessons Learned

Overall, our team has learned that FPGA Design and implementation can take a significant amount of time due to many different variables that can result in unforeseen problems, therefore it is extremely difficult to plan out realistic timelines.

When reflecting on work done for our senior design project, there are many valuable takeaways spanning from gained technical knowledge to newly established industry relationships. From our time working on the Vivado designs, we have gained immense knowledge of the internal operations and effects that a proper component can have on both itself and other components. Going beyond the hardware, our team became very proficient with the Vivado block diagram interface (version 2018.2) and how to modify block designs at both the high and low levels.

We also gained experience with working on new technology that may be beneficial to future projects. Using new software tools is a valuable experience to have especially when transitioning to new working environments. We learned a lot about the challenges of adapting to new software and the risks that come with working on new implementations of that software. We also learned a lot about the need for risk management when using new technologies.

6. Conclusions

6.1 Closing Remarks

Given the time constraints, the fact that FPGA design and implementation often present unforeseen problems, and that a majority of our team would be considered rookie FPGA Engineers at best: our team was not able to fully implement and test a 100 MHz spectrometer on the Xilinx ZCU111 platform. We demonstrated that CASPER could potentially migrate to a fully open-source toolflow in the future by creating an automated build environment using Migen/Litex and PYNQ. We also provided the leg-work for the creation of a CASPER ZCU111 ADC and 1 GbE core that can easily be taken over by more FPGA-experienced CASPER members to finish their implementation.

6.2 Future Work

All relevant code and documentation for this project will be truly open-source and made available in a repository on Brian Bradford's GitHub. He plans to email CASPER's mail list and inform its members about its location the what work was accomplished. There are around half a dozen or more CASPER members that are actively trying to build instruments with the Xilinx ZCU111 RFSoC platform, they will most likely take the work done on the ADC and 1 GbE core (supplemented with this report) and fully implement them within CASPER's current toolflow for the rest of the community to use. Brian Bradford plans to give a Migen/Litex and PYNQ infrastructure talk at this year's CASPER workshop in August to inform members about the work done and the potential to use Migen/Litex and PYNQ in a future CASPER toolflow.

7. References

1. Casper-dsp.org. (2018). *The Collaboration for Astronomy Signal Processing and Electronics Research*. [online] Available at: http://casper-dsp.org/ [Accessed 23 Oct. 2018].

2. M-labs.hk. (2018). *Migen | M-Labs*. [online] Available at: https://m-labs.hk/migen/index.html [Accessed 23 Oct. 2018].

3. Readthedocs.org. (2018). *PYNQ | Read the Docs*. [online] Available at: https://readthedocs.org/projects/pynq/ [Accessed 23 Oct. 2018].I

4. Ieeexplore.ieee.org. (2018). *Commissioning and testing of SERENDIP VI instrumentation USNC-URSI national radio science meeting - IEEE Conference Publication*. [online] Available at: https://ieeexplore.ieee.org/document/7436240/ [Accessed 23 Oct. 2018].

5. IEEE Std 211-1997^{TM} IEEE Standard Definitions of Terms for Radio Wave Propagation

6. IEEE Std 1241-2010^{TM} IEEE Standard of Terminology and Test Methods for Analog-to Digital Converters

7. IEEE Std 802.3^{TM} Standards for Ethernet

8. Price, D. (2018). Spectrometers and Polyphase Filterbanks in Radio Astronomy. [online] Arxiv.org. Available at: https://arxiv.org/abs/1607.03579 [Accessed 23 Oct. 2018].

9. Xilinx.com. (2018). [online] Available at: https://www.xilinx.com/support/documentation/application_notes/xapp1305-ps-pl-based-ethernet-solution.pdf [Accessed 23 Oct. 2018].

10. Xilinx.com. (2018). *Zynq UltraScale+ RFSoC Data Converter*. [online] Available at: https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_0/pg269-rf-data-converter.pdf [Accessed 23 Oct. 2018].

11. Hickish, J (2018). *Allen Telescope Array: SNAP firmware.* [online] Available at: https://github.com/realtimeradio/ata_snap [Accessed 8 Jan. 2019].

12. Price, D (2017). *Polyphase filterbanks: an interactive introduction.* [online] Available at: https://github.com/telegraphic/pfb_introduction [Accessed 12 Feb. 2019].

13. fpgadeveloper.com (2018). [online] Available at: http://www.fpgadeveloper.com/2018/03/how-to-accelerate-a-python-function-with-pynq.html [Accessed 1 Feb. 2019]

8. Team Information

Members:
Brian Bradford - FPGA/DSP Engineer: bjbford@iastate.edu
Louis Hamilton - FPGA Engineer and GUI Developer: louis@iastate.edu
Jared Danner - FPGA Engineer: jdanner@iastate.edu
Nick Knuth - FPGA Engineer njknuth@iastate.edu
Vishal Joel - GUI Developer and FPGA Engineer: joel16@iastate.edu

Key Industry Consultants:
Dan Werthimer: Chief Scientist, SETI@Home, UC Berkeley Space Sciences Lab: danw@ssl.berkeley.edu
Jack Hickish: Assistant Researcher, UC Berkeley Radio Astronomy Lab: jackhickish@ssl.berkeley.edu
Alan Wilson-Langman: Senior Project Engineer, Vermeer Corporation: awilson-langman@vermeer.com

Advisor:
Dr. Phillip Jones: Associate Professor, Department of Electrical and Computer Engineering, Iowa State University: phjones@iastate.edu