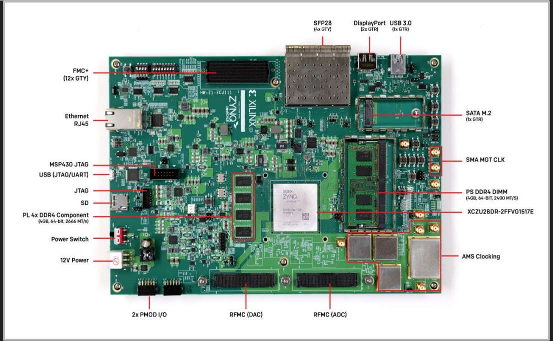
SDMay19-41

CASPER

Brian Bradford
Jared Danner
Nick Knuth
Vishal Joel
Louis Hamilton



High-Level Requirements

- Porting of the CASPER spectrometer application to the ZCU111 and RFSoC
- Integration of Migen into the CASPER tooflow
- Extending of PYNQ to work with the tooflow

Use Case

- 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 their open-source tools will be able to quickly develop digital instrumentation for their specific applications.

Functional & Non-functional Requirements

- **Functional**
  - CASPER spectrometer support
    - ZCU111 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
- **Non-functional**
  - 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™ 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

Design Plan

- Designing an ADC, spectrometer, ethernet module to port CASPER's spectrometer instrument to the ZCU111.

- The board infrastructure was designed using Migen and PYNQ.
  - Migen was used as an experimental tool to test whether it would be a viable option to replace the existing CASPER toolflow.
  - PYNQ interfaced with the modules in order to provide register software access.

- Create a GUI in Jupyter Notebooks to provide an interface for the entire spectrometer application, which allows for
  - Quick configuration of various spectrometer parameters
  - Effectively display data from various sections of the system

Design Objectives

- Porting of the CASPER spectrometer instrument and dependent libraries to the Xilinx ZCU111 evaluation platform.
  - General board support
  - Ethernet interface
  - ADC interface

- Migrating the current CASPER Toolflow to use Migen, an open source python library for generating and building gateware projects.

- Support for Xilinx PYNQ infrastructure, which will interface to existing CASPER Python libraries.

- Documentation of project for future users: source code, gateware, and firmware descriptions.

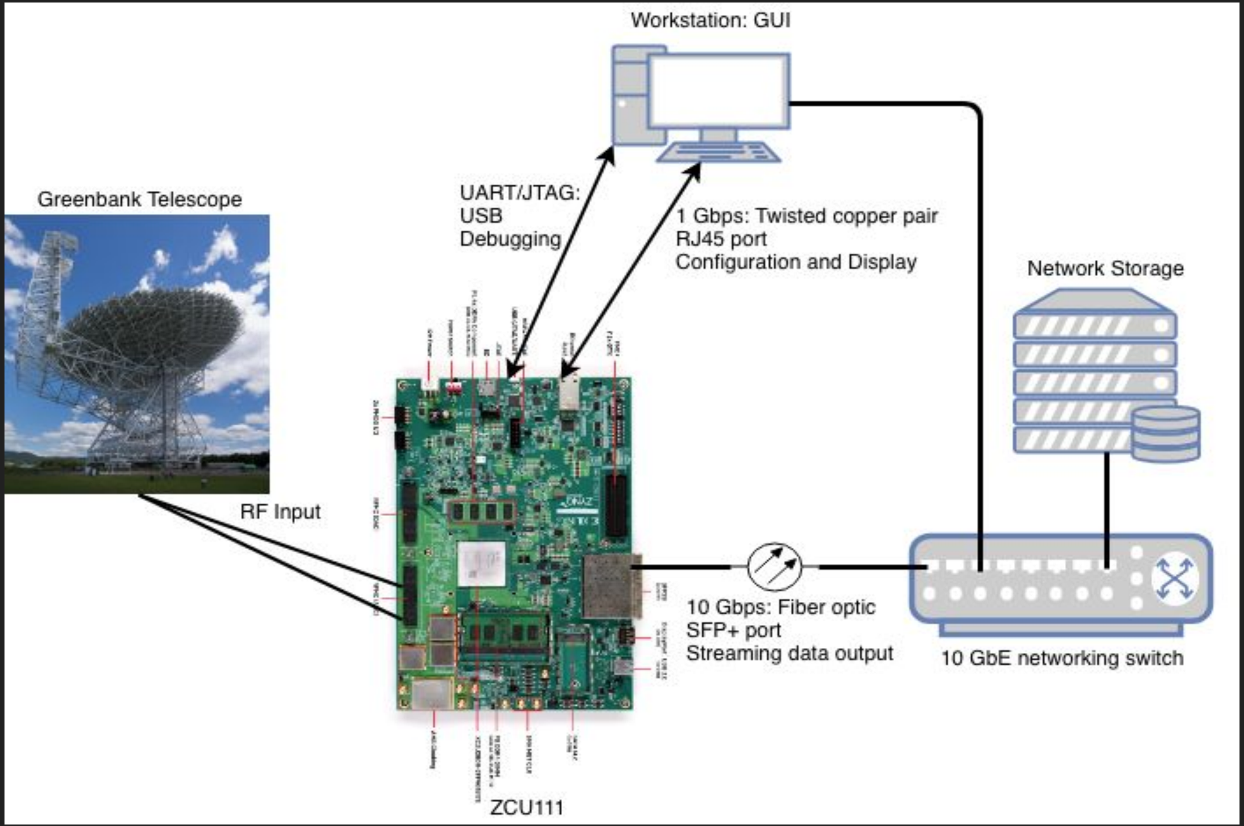
System Constraints

- 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
- 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

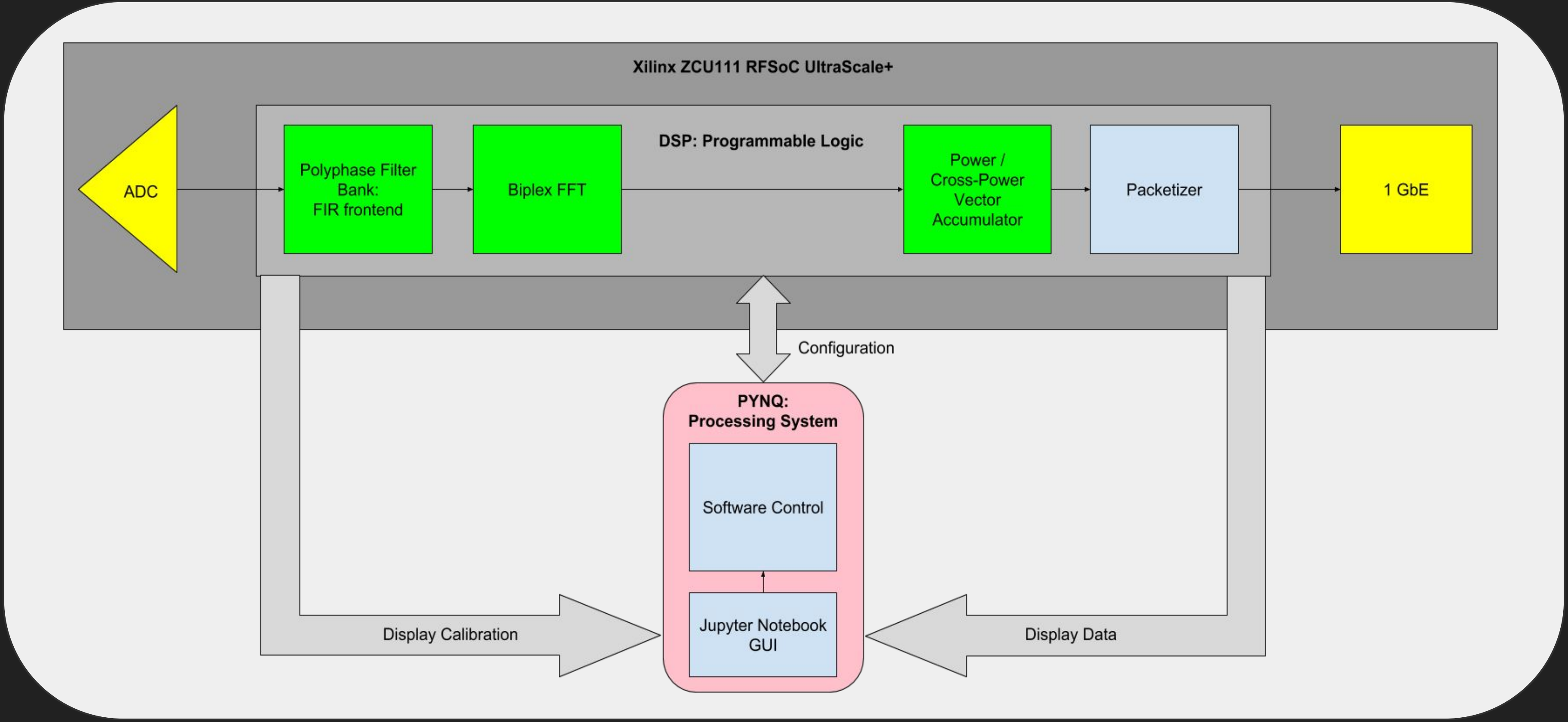
Design Trade-Offs

- Trade-Offs
  - Xilinx specific IP cores in our ethernet and ADC modules
  - Interfacing
    - Migen to interface between each subcomponent
    - Jupyter Notebook as our GUI.
- Resulting Actions
  - Xilinx IP
    - Decreased time to develop because the Xilinx IP due to modification
    - Reduces the effectiveness of our overall project in CASPER’s open-source environment.
  - Migen
    - Swapping out a proven & known project compilation for an unproven technology
    - Potentially reduces development time if successful & increases if not

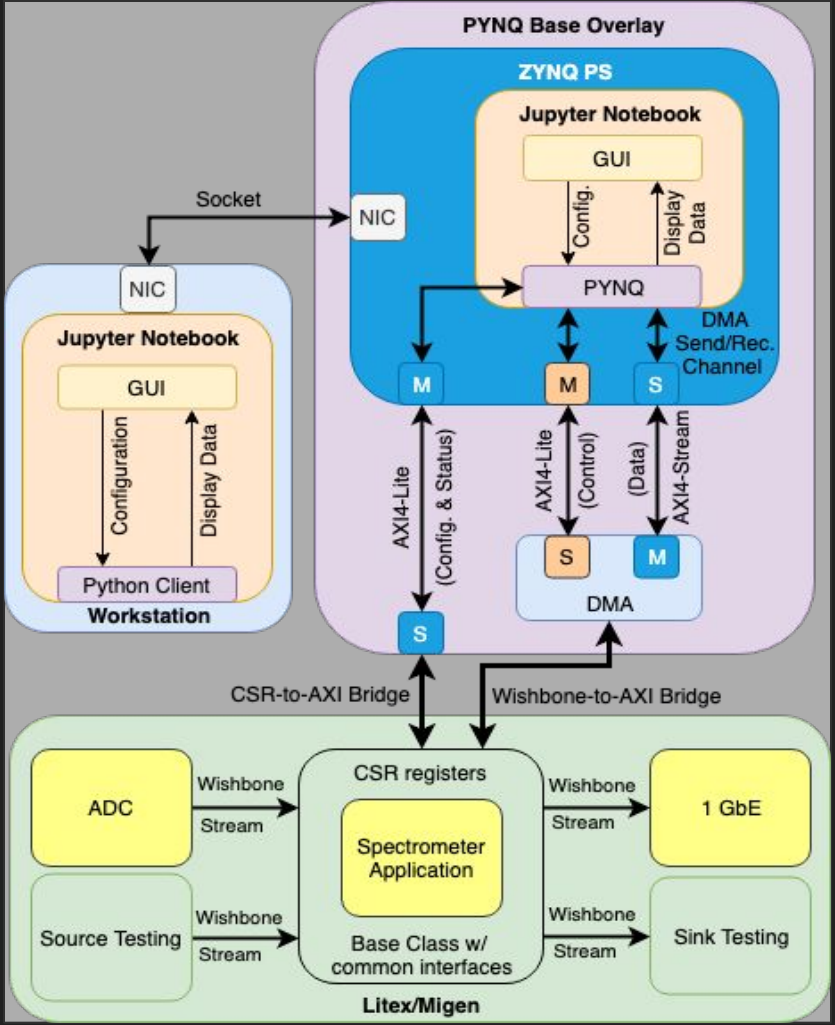
Conceptual Sketch - CASPER spectrometer



Functional Block Diagram

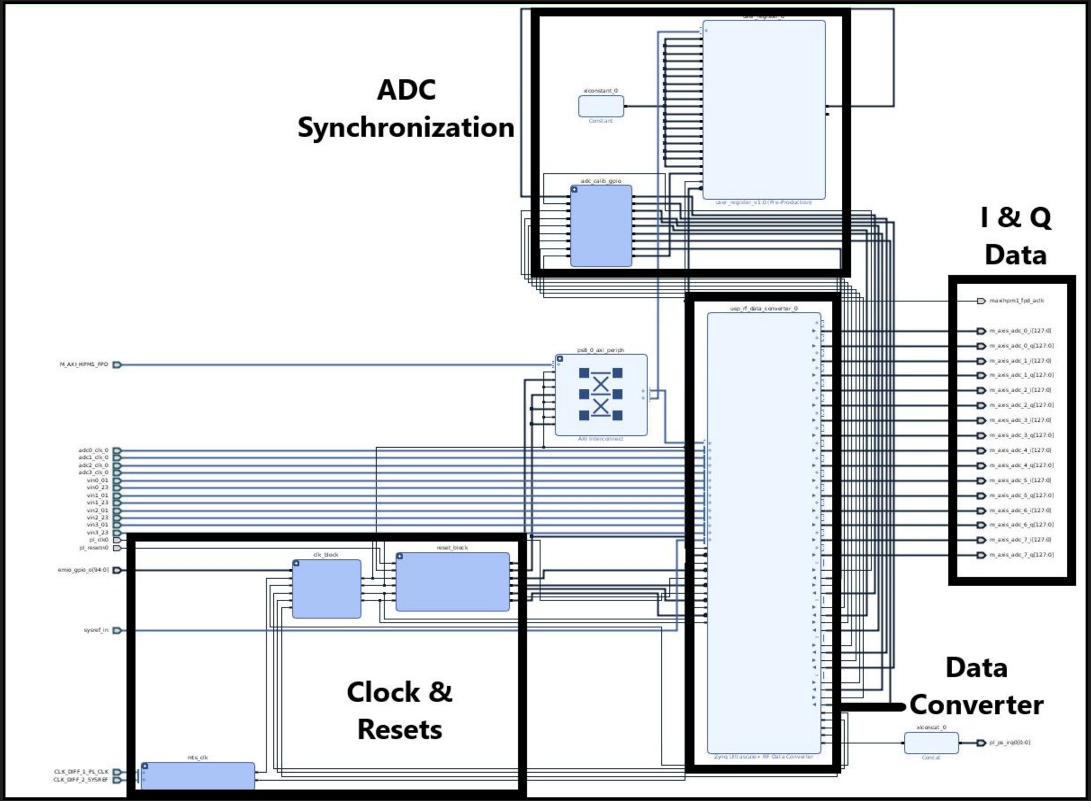


System Architecture



ADC - Design & Implementation

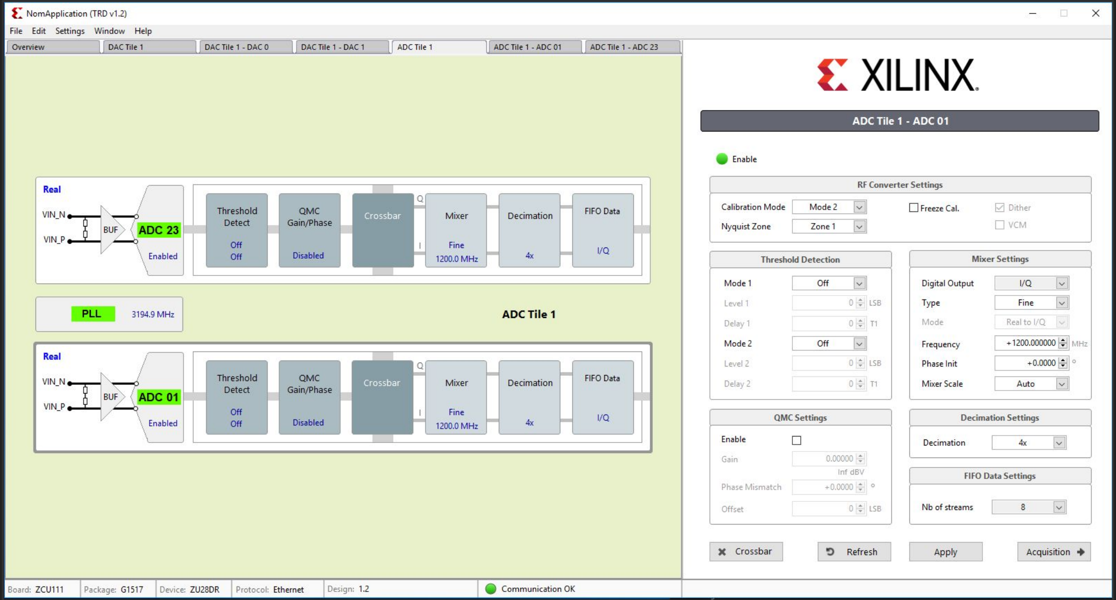
- Purpose & Goal
  - Initial desired spectrum: 0 - 100 MHz
  - Provide In-phase & Quadrature (I&Q) data to spectrometer
  - Synchronous sampling for all ADC cores



Sdmay19-41 ADC Design

ADC - Design & Implementation

- System Parameters
  - Sampling Frequency: 256 MHz
  - Multi-Tile Synchronized ADC Sampling
  - 8-bit I&Q data streams from each ADC core
  - Driven externally by ZYNQ PS



RF Converter Settings

Calibration Mode	Mode 2	Freeze Cal. ☐	Dither ☑
Nyquist Zone	Zone 1		VCM ☐

Threshold Detection

Mode 1	Off	
Level 1	0	LSB
Delay 1	0	T1
Mode 2	Off	
Level 2	0	LSB
Delay 2	0	T1

Mixer Settings

Digital Output	I/Q	
Type	Fine	
Mode	Real to I/Q	
Frequency	+1200.000000	MHz
Phase Init	+0.0000	°
Mixer Scale	Auto	

QMC Settings

Enable	☐	
Gain	0.00000 Inf dBV	
Phase Mismatch	+0.0000	°
Offset	0	LSB

Decimation Settings

Decimation	4x

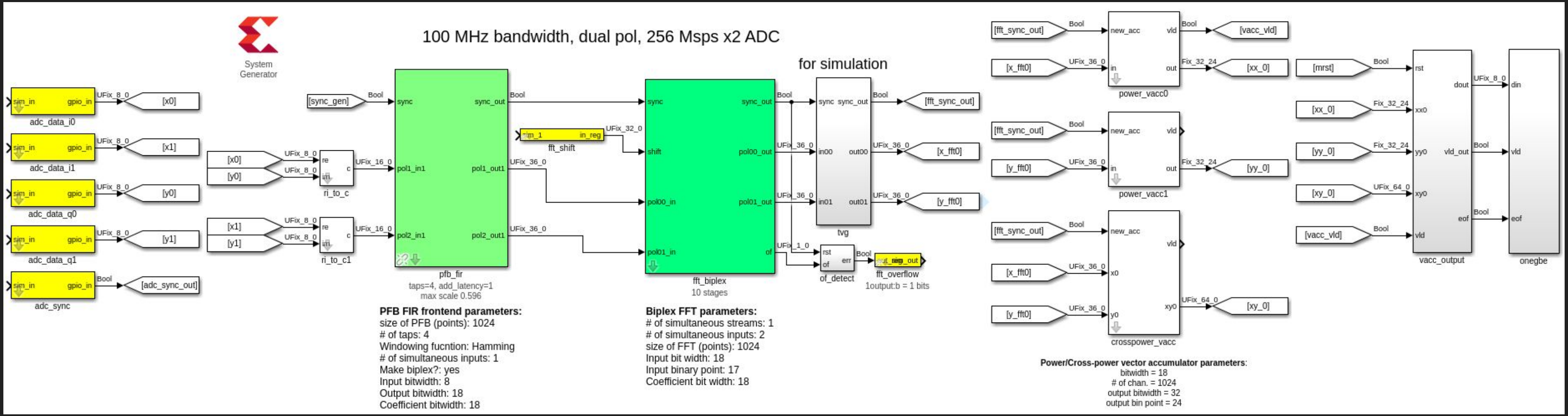
FIFO Data Settings

Nb of streams	8

Board: ZCU111 | Package: G1517 | Device: ZU28DR | Protocol: Ethernet | Design: 1.2 | Communication OK

Xilinx GUI ADC Configuration

Spectrometer - Design & Implementation



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

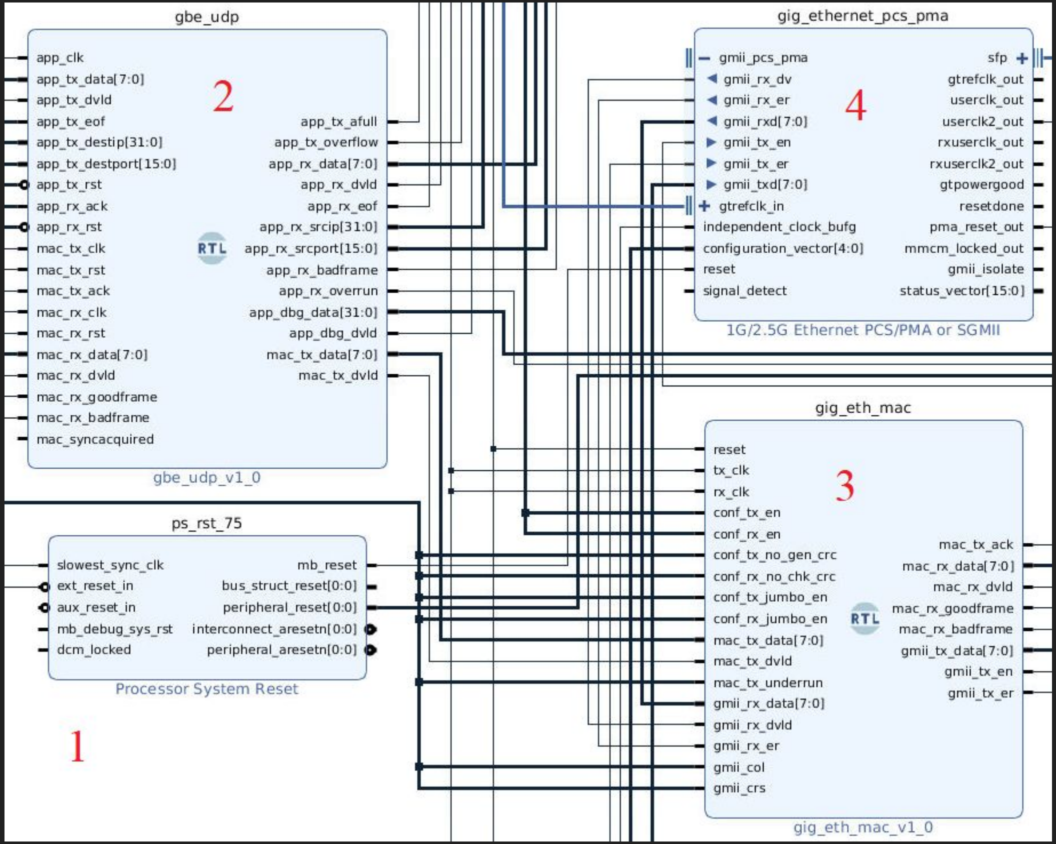
Biplex FFT parameters:
# of simultaneous streams: 1
# of simultaneous inputs: 2
size of FFT (points): 1024
Input bit width: 18
Input binary point: 17
Coefficient bit width: 18

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

- Derived from Polyphase Filterbank-based (PFB) Simulink application for Allen Telescope Array in Northern California
- Used existing CASPER Toolflow's Simulink DSP libraries
- Created interactive Jupyter Notebook to simulate various DSP parameters
- Used Xilinx/MATLAB System Generator to generate corresponding HDL code
- Wrapped and instantiated inside of a Migen module with configuration and status registers
- Consulted with Jack Hickish regarding DSP parameters

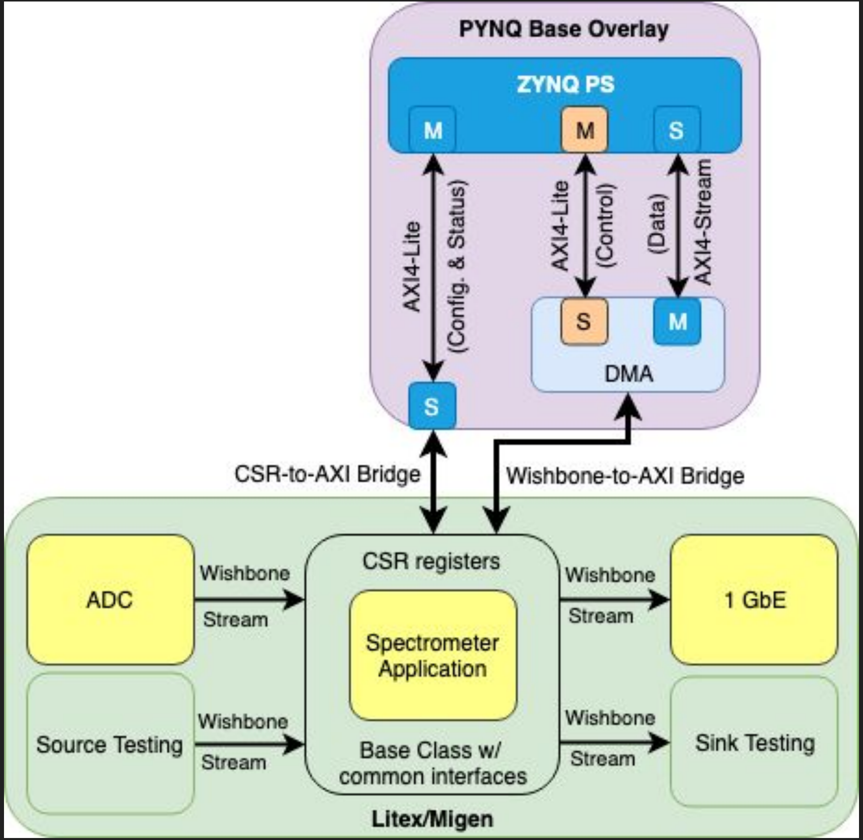
Ethernet - Design & Implementation

- 10 GbE to 1GbE
- Merging CASPER IP with Xilinx IP
- 8-Bit data stream with 1 Gigabit speed
- Transmit only



Build Environment - Design & Implementation

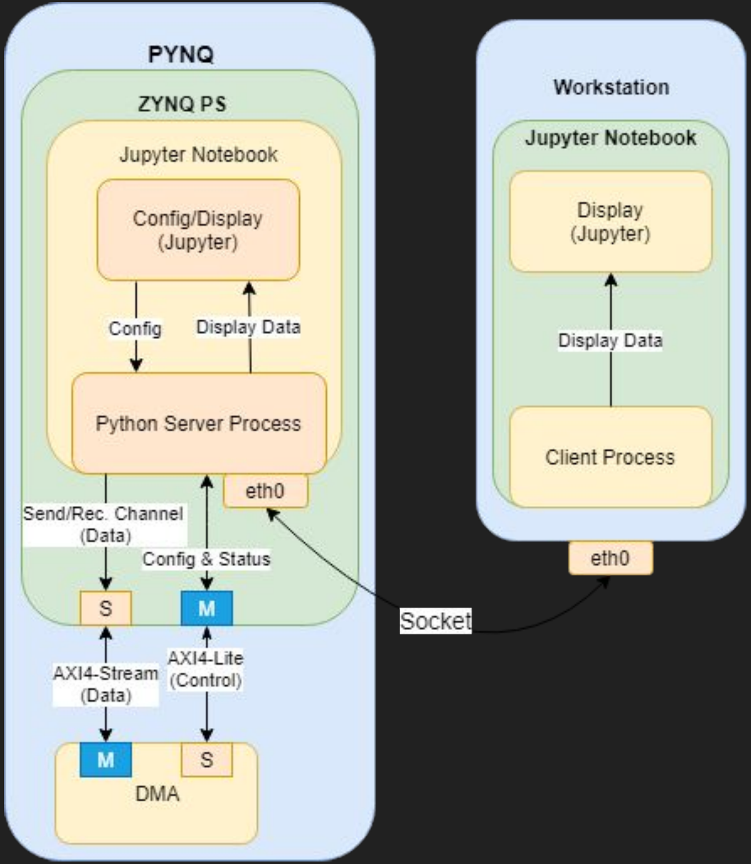
- Litex/Migen build environment:
  - Open-source FPGA design/SoC builder used to create cores and full FPGA designs.
  - Fully-automated build process
  - CSR Banks/buses
  - CSR-to-AXI4-Lite bridge
- Migen Base Class
  - Common CSR bus for config/status
  - Wishbone Stream input & output
- PYNQ base overlay
  - Exposes 3x AXI4-Lite interfaces
  - Zynq Processing System
  - DMA IP core with AXI4-Stream



*Please chat with me more about this at poster presentation

GUI - Design & Implementation

- Requirements
  - Sample data from the DMA IP core, and plot the data on Jupyter Notebooks
  - Establish communication between Python Server and Client process using sockets
  - Send commands from the client process and respond with sampled data from the DMA IP to the client

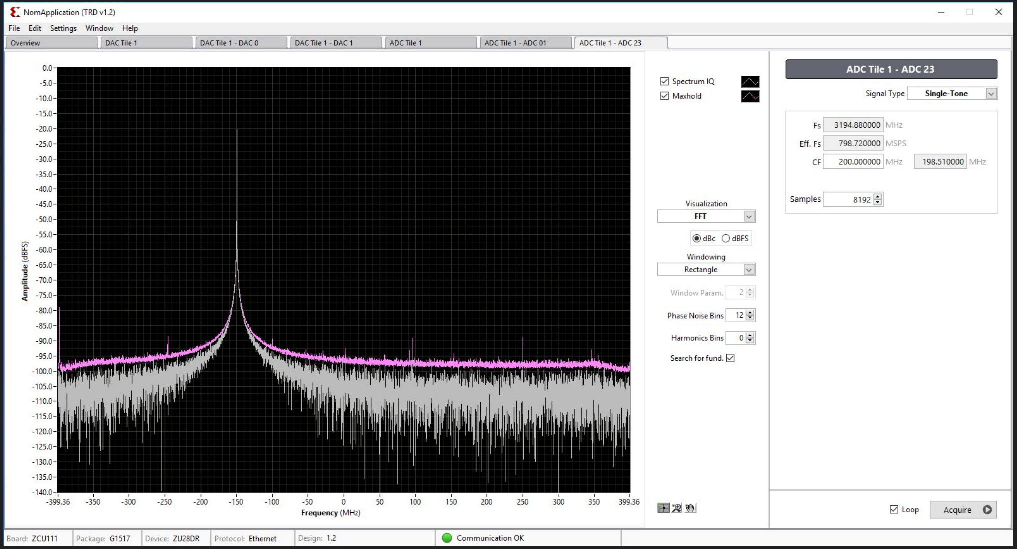


ADC - Testing & Validation

- Verified Xilinx TRD
    - Targeted Reference Design

- Validated target sampling parameters w/ TRD

- Collected parameter memory offsets to be tested w/ PYNQ



ADC Tile 1 - ADC 23

Signal Type: Single-Tone
Fs: 3194.880000 MHz
Eff. Fs: 798.720000 MSPS
CF: 200.000000 MHz 198.510000 MHz
Samples: 8192

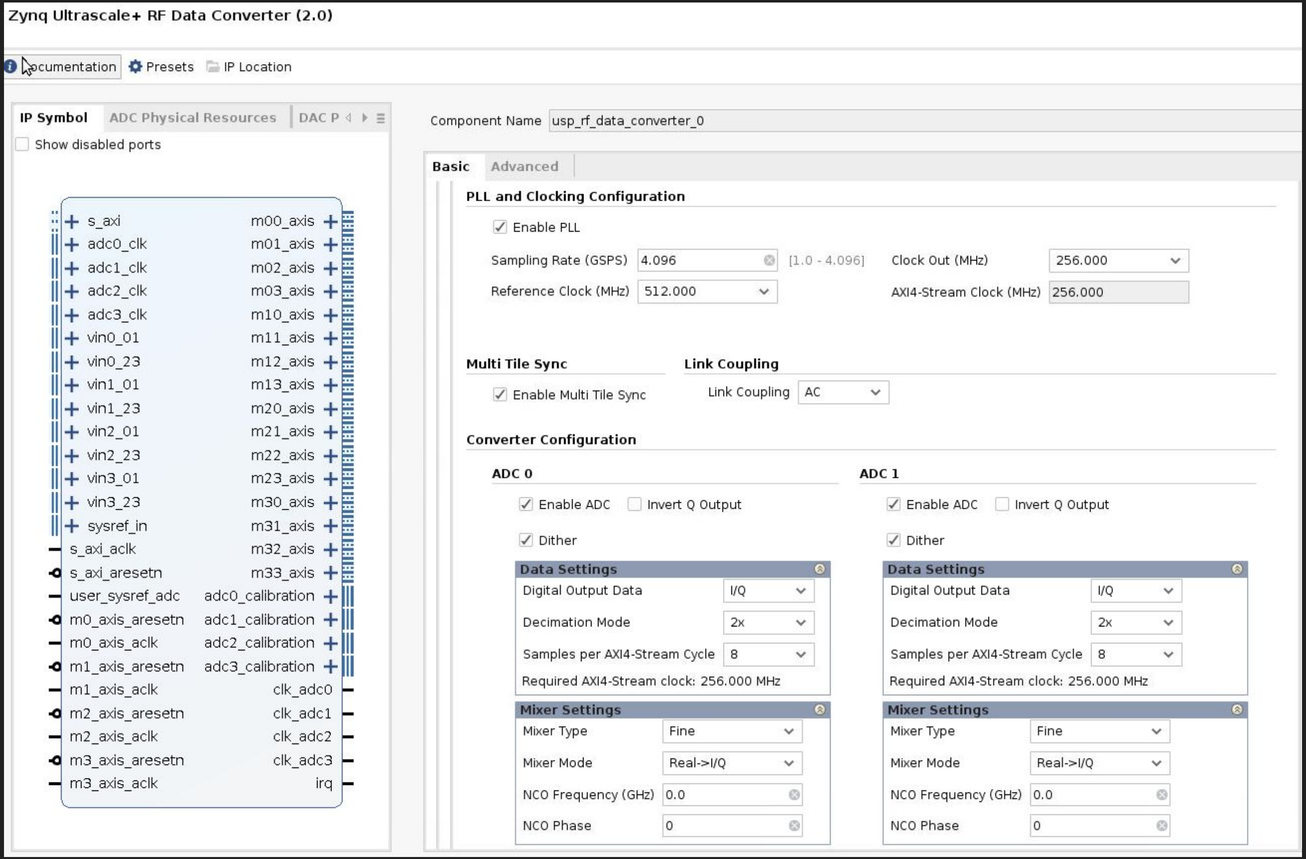
✓ Spectrum IQ
✓ Maxhold
Visualization: FFT
[✓ dBc] dBFS
Windowing: Rectangle
Window Param. 2
Phase Noise Bins 12
Harmonics Bins 0
Search for fund. ✓
✓ Loop

Board: ZCU111 Package: G1517 Device: ZU28DR Protocol: Ethernet Design: 1.2 Communication OK

TRD ADC Data Capture w/ FFT

ADC - Testing & Validation

Xilinx Data Converter IP Parameters



Zynq Ultrascale+ RF Data Converter (2.0)

Component Name: usp_rf_data_converter_0

PLL and Clocking Configuration

✓ Enable PLL

Sampling Rate (GSPS)	4.096	[1.0 - 4.096]	Clock Out (MHz)	256.000
Reference Clock (MHz)	512.000		AXI4-Stream Clock (MHz)	256.000

Multi Tile Sync

✓ Enable Multi Tile Sync

Link Coupling

Link Coupling: AC

Converter Configuration

	ADC 0	ADC 1
Enable ADC	✓	✓
Invert Q Output		
Dither	✓	✓
Data Settings		
Digital Output Data	I/Q	I/Q
Decimation Mode	2x	2x
Samples per AXI4-Stream Cycle	8	8
Required AXI4-Stream clock:	256.000 MHz	256.000 MHz
Mixer Settings		
Mixer Type	Fine	Fine
Mixer Mode	Real->I/Q	Real->I/Q
NCO Frequency (GHz)	0.0	0.0
NCO Phase	0	0

ADC - Risks & Mitigation

- Inability to finish due to:
  - Limited time
  - Lack of expertise
  - Pivot from 8 core ADC design to 2 Core ADC design
- Mitigation of risks
  - Reach out to industry experts for help & guidance

Spectrometer - Testing & Validation

1. Created a Jupyter Notebook to simulate and verify the DSP parameters necessary for 100 MHz spectrometer design

2. Simulated 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

* Was unable to complete #3 as DMA infrastructure was not completed in time

Spectrometer - Testing & Validation

1.

```python
def pfb_spectrometer(x, n_taps, n_chan, n_int, window_fn="hamming"):
    M = n_taps
    P = n_chan

    # Generate window coefficients
    win_coeffs = generate_win_coeffs(M, P, window_fn)

    # Apply frontend, take FFT, then take power (i.e. square)
    x_fir = pfb_fir_frontend(x, win_coeffs, M, P)
    x_pfb = fft(x_fir, P)
    x_psd = np.abs(x_pfb)**2

    # Trim array so we can do time integration
    x_psd = x_psd[:np.round(x_psd.shape[0]//n_int)*n_int]

    # Integrate over time, by reshaping and summing over axis (efficient)
    x_psd = x_psd.reshape(x_psd.shape[0]//n_int, n_int, x_psd.shape[1])
    x_psd = x_psd.mean(axis=1)

    return x_psd
```

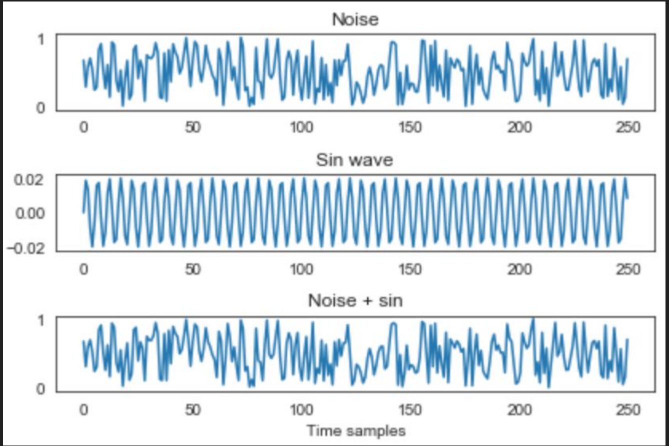
2.

```python
M     = 4          # Number of taps
P     = 1024       # Number of 'branches', also fft length
W     = 1000       # Number of windows of length M*P in input time stream
n_int = 2          # Number of time integrations on output data

# Generate a test data steam
samples = np.arange(M*P*W)
noise   = np.random.random(M*P*W)
# Hydrogen line
freq = 1.42e9
amp  = 0.02
cw_signal = amp * np.sin(samples * freq)
data = noise + cw_signal
```

3.

```python
plt.subplot(3,1,1)
plt.title("Noise")
plt.plot(noise[:250])
plt.subplot(3,1,2)
plt.title("Sin wave")
plt.plot(cw_signal[:250])
plt.subplot(3,1,3)
plt.title("Noise + sin")
plt.plot(data[:250])
plt.xlabel("Time samples")
plt.tight_layout()
```

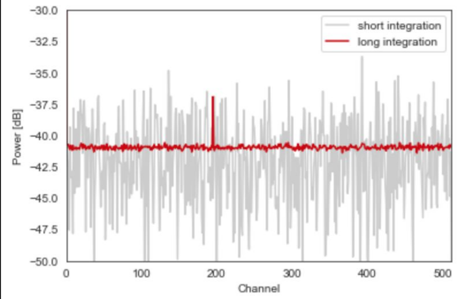
4.



5.

```python
X_psd2 = pfb_spectrometer(data, n_taps=M, n_chan=P, n_int=1000, window_fn="hamming")

plt.plot(db(X_psd[0]), c='#cccccc', label='short integration')
plt.plot(db(X_psd2[1]), c='#cc0000', label='long integration')
plt.ylim(-50, -30)
plt.xlim(0, P/2)
plt.xlabel("Channel")
plt.ylabel("Power [dB]")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x123cd7780>
```



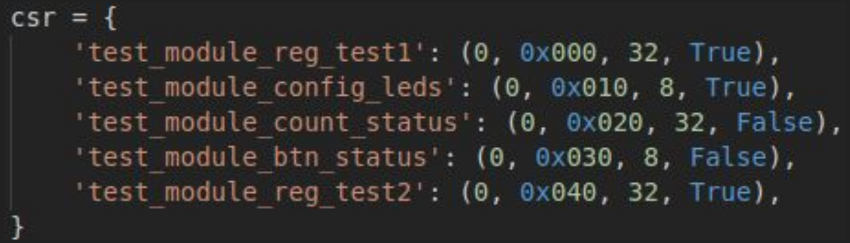
Ethernet - Testing & Validation

- Implementation Choice
- Loopback Testing
- MicroBlaze Testing
- Sample Data Testing
- Current Progress

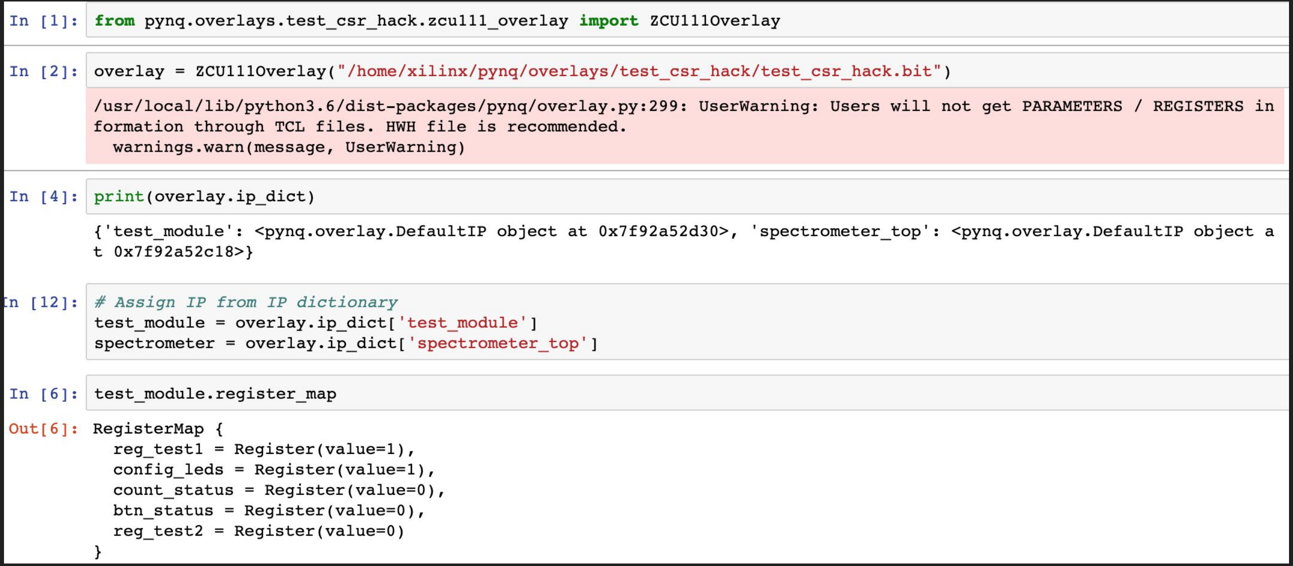
Ethernet - Risks & Mitigation

- 10 GbE import
- Xilinx IP Licensing
- Using other CASPER IP
- Clocking and IP documentation

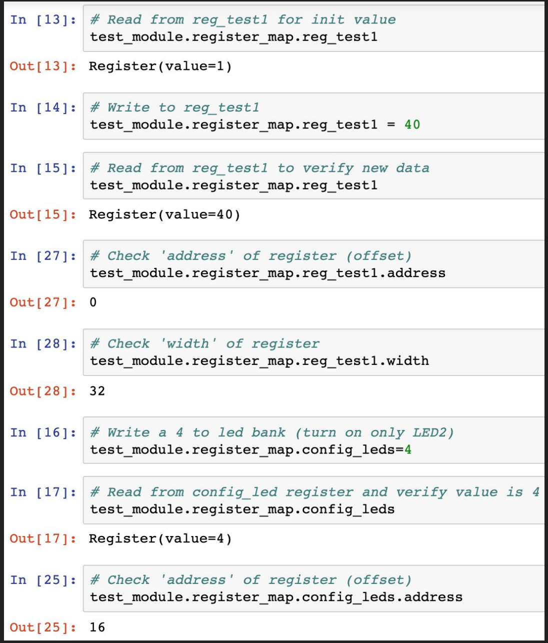
Build Environment - Testing & Validation



```
csr = {
    'test_module_reg_test1': (0, 0x000, 32, True),
    'test_module_config_leds': (0, 0x010, 8, True),
    'test_module_count_status': (0, 0x020, 32, False),
    'test_module_btn_status': (0, 0x030, 8, False),
    'test_module_reg_test2': (0, 0x040, 32, True),
}
```



```
In [1]: from pynq.overlays.test_csr_hack.zcu111_overlay import ZCU111Overlay

In [2]: overlay = ZCU111Overlay("/home/xilinx/pynq/overlays/test_csr_hack/test_csr_hack.bit")

        /usr/local/lib/python3.6/dist-packages/pynq/overlay.py:299: UserWarning: Users will not get PARAMETERS / REGISTERS in
        formation through TCL files. HWH file is recommended.
          warnings.warn(message, UserWarning)

In [4]: print(overlay.ip_dict)

        {'test_module': <pynq.overlay.DefaultIP object at 0x7f92a52d30>, 'spectrometer_top': <pynq.overlay.DefaultIP object a
        t 0x7f92a52c18>}

In [12]: # Assign IP from IP dictionary
        test_module = overlay.ip_dict['test_module']
        spectrometer = overlay.ip_dict['spectrometer_top']

In [6]: test_module.register_map

Out[6]: RegisterMap {
          reg_test1 = Register(value=1),
          config_leds = Register(value=1),
          count_status = Register(value=0),
          btn_status = Register(value=0),
          reg_test2 = Register(value=0)
        }
```



```
In [13]: # Read from reg_test1 for init value
         test_module.register_map.reg_test1

Out[13]: Register(value=1)

In [14]: # Write to reg_test1
         test_module.register_map.reg_test1 = 40

In [15]: # Read from reg_test1 to verify new data
         test_module.register_map.reg_test1

Out[15]: Register(value=40)

In [27]: # Check 'address' of register (offset)
         test_module.register_map.reg_test1.address

Out[27]: 0

In [28]: # Check 'width' of register
         test_module.register_map.reg_test1.width

Out[28]: 32

In [16]: # Write a 4 to led bank (turn on only LED2)
         test_module.register_map.config_leds=4

In [17]: # Read from config_led register and verify value is 4
         test_module.register_map.config_leds

Out[17]: Register(value=4)

In [25]: # Check 'address' of register (offset)
         test_module.register_map.config_leds.address

Out[25]: 16
```

* Please refer to our demo during the poster presentation this afternoon for more information

GUI - Testing & Validation

- Server and Client
  - Generating signals by passing them to one server thread and the other to a client thread waiting to verify that data is being transmitted.
  - The client process was tested by verifying data that was set with certain values based on user input.
  - Control surfaces with a client process to connect to a server process, which would then write to GPIO control registers.
- Generating signals
  - Numpy was then used to generate signals that will be passed to two FIR filters and the times compared.

GUI - Risks and Mitigation

- GNURadio (Later switched to Jupyter Notebook)
    - PYNQ readily supports Jupyter Notebook
    - Notebook modules that allow data visualization
    - Easier to configure data using ipywidgets and json for storing configurations
    - Ability to read PYNQ registers through PYNQ modules

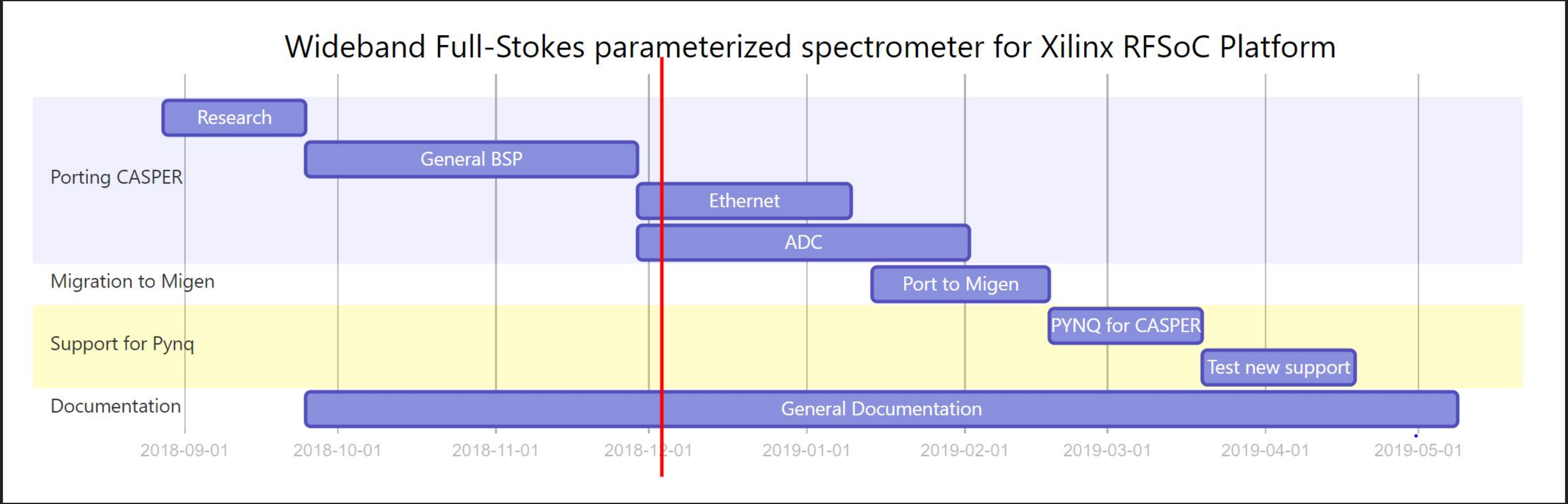
- Expertise

- Time

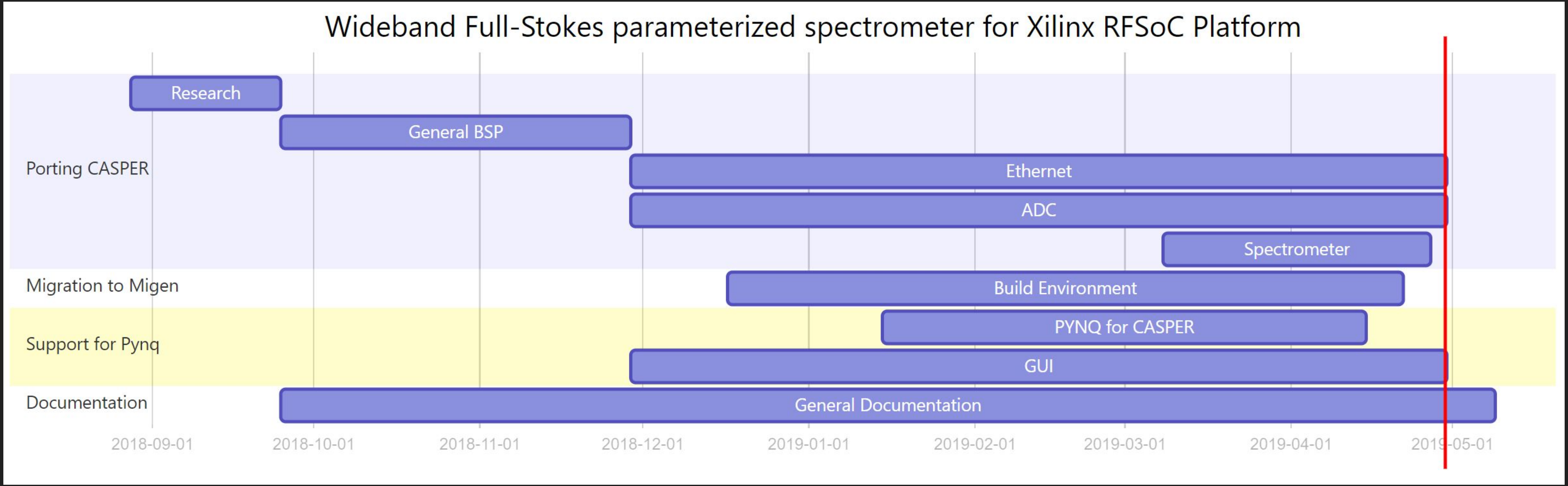
Roles & Responsibilities

- *Brian Bradford* - FPGA/DSP Engineer
  - Build Environment (Migen/Litex)
  - PYNQ
  - Spectrometer application
- *Louis Hamilton* - FPGA Engineer and GUI developer
  - GUI design & implementation
  - PYNQ DMA
- *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

Project Schedule - Proposed



Project Schedule - Actual



Lessons Learned

- Time required for FPGA development
- Technical knowledge necessary and gained
- Experience with unfamiliar technology
- Overcoming unforeseen risks and challenges
- Adapting to major changes in design

Conclusions & Future Work

- Were not able to fully implement and test 100 MHz spectrometer on ZCU111
  - Mostly rookie FPGA Engineers
  - Time constraints
  - Unforeseen risks regarding each module (explained earlier)
- Demonstrated potential to migrate to a fully open-source toolflow
  - Automated build environment: Migen and PYNQ
- Substantial leg-work done for creation of CASPERized:
  - ADC core
  - 1 GbE core
- Code to be stored in Brian Bradford’s GitHub & will notify CASPER community about location and progress
- Half a dozen or more members interested in expanding on our work
- Brian to give talk regarding project at CASPER 2019 conference in Boston

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

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]