Design Document

Ultra-Fast Failure Test Unit

Capstone Project #45

Boran Gungor
Jude Lu
Lukas Lund
Romina Adib
Xingwei Su

Contents

Contents 1

List of Abbreviations 2

1 Summary 2

2 System Overview 3

3 Zynq Ultrascale+ RFSoc Data Converter 5

4 PetaLinux 5

4.2 Compiling Custom Software Using PetaLinux 5

4.3 Creating Custom Boot Images with PetaLinux 6

4.4 Common Issues with Custom PetaLinux Images 7

5 TCP Client Application 7

5.1 Command Interface 8

5.2 Data Interface 8

6 Signal Processing 8

6.1 Aliasing 8

6.1 Low-pass Filter 9

6.2 Memory Impact on Noise 10

6.3 Hardware Setup 12

7 Data Post-Analysis 13

7.1 Converting from LSB to Voltage 13

7.2 Post Sample Data Analysis GUI 14

8 Auxiliary Design Choices 16

Data Handler Module 16

8.1 Overview of IP Blocks Used in the Data Handler Module 17

8.1.1 FIFO 17

8.1.2 DMA Module 18

8.1.3 AXI Interconnect 18

References 20

List of Abbreviations

ADC	Analog to Digital Converter
ADI	Analog Devices Incorporated
AXI	Advanced eXtensible Interface
DDR	Double Data Rate
DUT	Device Under Test
FFT	Fast Fourier Transform
FIFO	First In First Out
FMC	FPGA Mezzanine Card
FPD	Full Power Domain
FPGA	Field Programmable Gate Array
GSPS	Giga Samples Per Second
GUI	Graphical User Interface
HPC	High Pin Count
IP	Intellectual Property
LPC	Low Pin Count
LPD	Low Power Domain
LVCMOS	Low Voltage Complementary Metal Oxide Semiconductor
LVDS	Low Voltage Differential Signaling
PS	Processing System
PL	Programmable Logic
RF	Radio Frequency
SMA	SubMiniature version A
SNR	Signal to Noise Ratio
SoC	System on Chip

1 Summary

This document outlines the design of the Ultra-Fast Failure Test Unit that we have developed for the Ivanov Group. In this document we will provide an overview of our design, including a decomposition of the design into major sub-components. Finally, we will justify each design choice by linking them to specific requirements, constraints and goals as specified in our Requirements Document.

2 System Overview

Researchers want to induce aging in transistors, this is done by configuring a test environment for the DUT (Figure 2). Temperature, magnetic field and supply voltage are varied to accomplish this goal.

We monitor the aging of the DUT by measuring the output voltage, labeled as V_{out} in Figure 2 (a) and (b).

Sampling rate, frequency of the signal to be measured, and number of samples to collect are parameters which can be configured using our custom program. These parameters are sent to initiate the ADC accordingly before sampling of the signal begins. Our program also initiates the ZCU111 memory, setting how much and what type of memory will be used.

Once the parameters have been input by the client, and the hardware has been configured, the system begins sampling the analog voltage signal from the output from the DUT. This signal is fed to an ADC to be converted to the digital domain. The digital data is then stored in memory. The ADC is connected to the DUT via SMA cables.

Once testing has been completed, the data in memory is exported as a .csv file. The exported data can be opened using our GUI which allows the user to plot, observe and analyze the data.

The following figure is a visual representation of how our system works. Anything in greyscale was designed by our client and is not included as part of our design choices. Yellow blocks represent hardware we used to complete the project, and green blocks are modules designed and configured inside the hardware. The purple blocks represent the custom software written by members of our team.

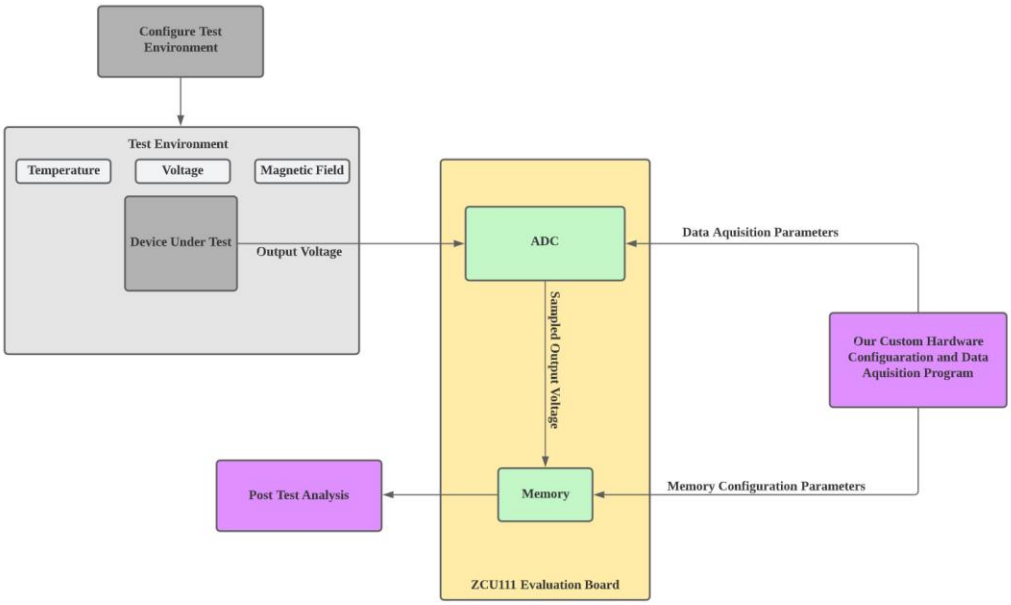
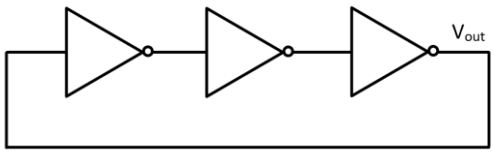
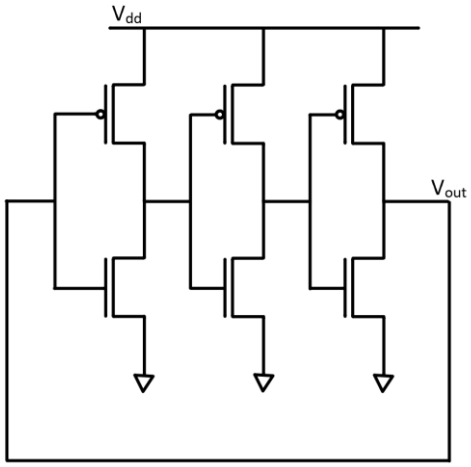


Figure 1: High-level functional system diagram



(a) Symbol Schematic



(b) Transistor-level Schematic

Figure 2: Schematics representing a ring oscillator, similar to the DUT

3 Zynq Ultrascale+ RFSoc Data Converter

The RFDC evaluation tool (we will refer to it as rftool throughout this document) developed by Xilinx allows a user to initiate and customize the ADC and DAC. Some parameters which can be customized include:

- Memory type (i.e., BRAM vs DDR) for storage of sampled data
- Number of ADC/DAC channels to be used for sampling
- Clock driving the ADC/DAC
- Sampling rate and signal frequency

In the early stages of our project, we used a Xilinx GUI to access and familiarize ourselves with the rftool. We proceeded to design a program which replaces the Xilinx GUI, allowing for configuration of only the necessary parameters. Our program is configured as a custom application in Petalinux, details of which are included in the following sections. Our system optimizes usability and efficiency for the client by eliminating the need for downloading memory extensive programs with functionality outside the scope of testing needs.

4 PetaLinux

PetaLinux is an embedded Linux operating system developed specifically for Xilinx processing systems. It allows engineers to synchronize software platforms with hardware designs. The primary advantage of using PetaLinux is that it eliminates the need for low level device driver development. More specific to our project, without the use of PetaLinux we would have needed to write driver code for the ADC to sample incoming signal data. This would have taken significant engineering time and effort which is not necessary to develop the best possible system for the client.

One of the major advantages of using PetaLinux to communicate with the FPGA system is that it enables us to write custom software to run on the Ultrascale+ processor, generate a bootable Linux image with our custom software installed, then boot the system which is already configured to receive commands from our TCP client application described in lthe next section.

Running custom software on the ZCU111 allows us to alter the source code of the rftool and customize its interaction with our client application. This was useful for research and debugging purposes while learning how to interact with the hardware and debugging our design. This is covered in more detail in Verification and Validation, but the final version of our application is able to connect to the default RFDC PetaLinux image and issue commands to the rftool.

4.2 Compiling Custom Software Using PetaLinux

The PetaLinux tool and its documentation make creating new projects and applications simple. The first step is to create a new project from a board support package file (.bsp) which is

generated from the hardware design in Vivado. Since Xilinx provides an RFDC .bsp file pre-configured for the ZCU111 hardware, we were able to bypass the custom hardware design phase and focus more of our engineering effort on designing the most elegant solution for our client.

To do this, we began by creating a new PetaLinux project from the RF Data Converter .bsp using the following command:

```
petalinux-create -t project -s <path/to/rfdc.bsp>
```

Once the .bsp has been extracted, we can descend into the folder that has just been created by the command above, and create a custom application, if desired. We will refer to the root directory of the project as `$PETALINUX_PROJ_HOME` for simplicity.

Again, using the `petalinux-create` command, we can create a new PetaLinux application based on a C template.

```
petalinux-create -t apps --name <name-of-app> --template c \
--enable
```

The subdirectories for each custom app we create – as well as all the built-in apps from the .bsp we imported – can be found in `$PETALINUX_PROJ_HOME/project-spec /meta-user/recipes-apps/`. Once we are ready to build a specific app, we can use the following command:

```
petalinux-build -c <name-of-app>
```

4.3 Creating Custom Boot Images with PetaLinux

Once the project is finished building, it is time to create our boot image.

```
petalinux-package --boot --u-boot --format BIN
```

We can then find the necessary boot files in `$PETALINUX_PROJ_HOME/images/linux/`. The following files need to be copied over to the micro-SD card we wish to boot from:

- `BOOT.BIN`
- `image.ub`

Our custom PetaLinux image with custom software pre-installed is now ready to be booted. To automatically setup the ethernet port on the ZCU111 and start the rftool software every time the system is booted, we can also load the following `autostart.sh` file to the SD card.

```
#!/bin/sh
ifconfig -a | grep eth0
RESULT=$?
if [ $RESULT -eq 0 ]; then
      ifconfig eth0 192.168.1.3
      rftool
fi
```

We can now set up a wired connection at IP address `192.168.1.3`.

4.4 Common Issues with Custom PetaLinux Images

As of RFDC version 2021.1, there are a number of problems that we had to resolve before successfully booting our custom images. The first of which is that we needed to reconfigure the default PetaLinux image. To do this, we run the petalinux-config command with no arguments:

```
petalinux-config
```

A simple ncurses interface will come up. Use the arrow keys to select Image Packaging Configuration > INITRAMFS/INITRD and rename this field to “petalinux-image-minimal”.

Another frustrating bug is that the default application responsible for reading any autostart.sh files loaded onto the SD card seems to have been written on a Windows machine, and is thus encoded with DOS line endings (CRLF) instead of the standard UNIX encoding (LF). To solve this issue and successfully run the autostart script, the source code for the `trd-autostart` app must be opened in any modern editor and converted from CRLF encoding to LF, and then the app should be rebuilt. The `autostart.sh` file described in the previous section should now execute without issue.

5 TCP Client Application

The RF Data Converter GUI sends commands to the rftool running on the ZCU111 via a serial TCP connection. By studying the rftool source code we were able to determine exactly what type of connections, and to which ports, it expects. We have designed a simple-to-use client application – which can run on any PC operated by researchers – to connect these serial ports and send the appropriate rftool commands to configure the ADC and collect data.

This design choice provides all the benefits and functionality of Xilinx’s RF Data Converter tool in the most efficient, user-friendly way which will enable researchers to easily automate their testing procedures. By eliminating the need to use Xilinx’s general-purpose user interface, users will be able to run a simple program with optional input parameters defining ADC

configurations. This will allow for easy integration of our tool into automation scripts that can be used for accelerating research and improving user experience.

The rftool running on the ZCU111 listens for connection requests on two TCP server ports: 8081 (command interface) and 8082 (data interface). Our client application connects to both these ports to send and receive commands and data.

5.1 Command Interface

The rftool command interface is defined in the rftool source code found within the RFDC .bsp package. It defines how the rftool listens for commands from the TCP port, the expected commands and command formats, and the expected arguments for each command. Our application sends the necessary configuration commands based on user input to set up the ADC.

To initialize the command interface, an empty command consisting of only a new line character must be issued from the client side. After this step, commands can be sent to configure the ADC.

5.2 Data Interface

The data port of the rftool is used to transmit data between the server and client. When the ADC is configured, the user input defining sampling rate, and data storage options is passed via this data interface. Once the ADC has been configured, and memory is read by issuing a memory read command over the command interface, the sampled data is returned over the data interface. Data is returned as a string of characters sent over the TCP connection, and thus needs to be converted to floating point. Each sample is then written to a .csv file for post-processing.

6 Signal Processing

6.1 Aliasing

Aliasing is a phenomenon that occurs when the sampling rate is too low to accurately recover the input signal. In order to reconstruct a sinusoidal input signal, the sampling rate must be greater than double the frequency of the sinusoid. This minimum sampling rate is known as the Nyquist Rate. If a high frequency signal is sampled at a rate below the Nyquist Rate, the reconstructed signal will appear as a sinusoid with a lower frequency, and will no longer resemble the original signal.

In figure 3, each green point represents one sample of the sinusoidal input signal shown. Due to an insufficient sampling rate, the recovered signal has a much lower frequency than the actual input.

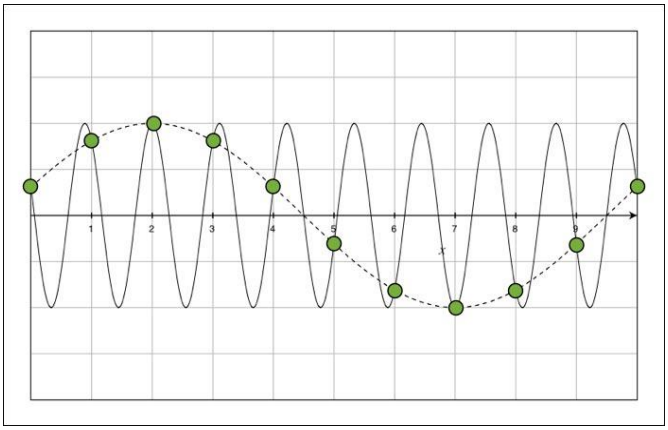


Figure 3: An illustration of an aliased signal

Due to the effects of aliasing, noise above the Nyquist Frequency (equal to one half the sampling rate) will be mixed into the baseband frequency of the signal, increasing the amount of noise distorting the input. For this reason, we have added an anti-aliasing low-pass filter at the input to reduce aliasing high frequency noise.

6.1 Low-pass Filter

When sampling an analog signal, a wide range of frequency content will be observed, due to noise at frequencies above and below the transmitted frequency. Thus, it is important for the integrity of the recovered signal that a filter is applied to eliminate as much noise as possible. Since we are sampling data at 1 GHz, we have applied a low pass filter at the input which filters out frequencies above 1.3 GHz. As shown in Figure 4, the addition of this low-pass filter significantly improves the quality of the recovered signal.

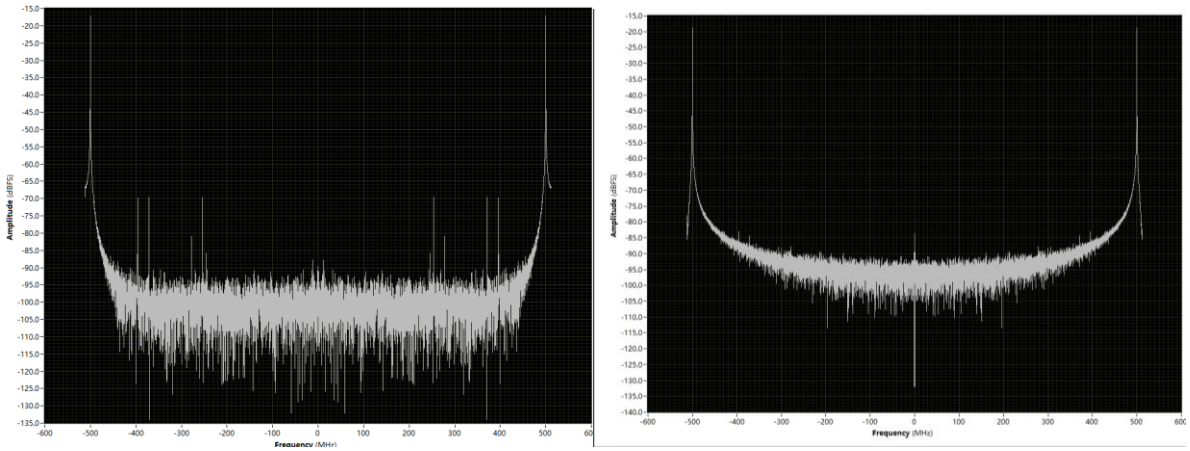


Figure 4: Comparison between no filter (left) and 10-2500 MHz low-pass filter (right)

6.2 Memory Impact on Noise

During our testing of the Xilinx ADC and DAC UI we discovered that the type of memory used to store sampled data makes a significant difference in the amount of noise present in the constructed signal. Using the DDR4 memory produced a far noisier signal than using the BRAM.

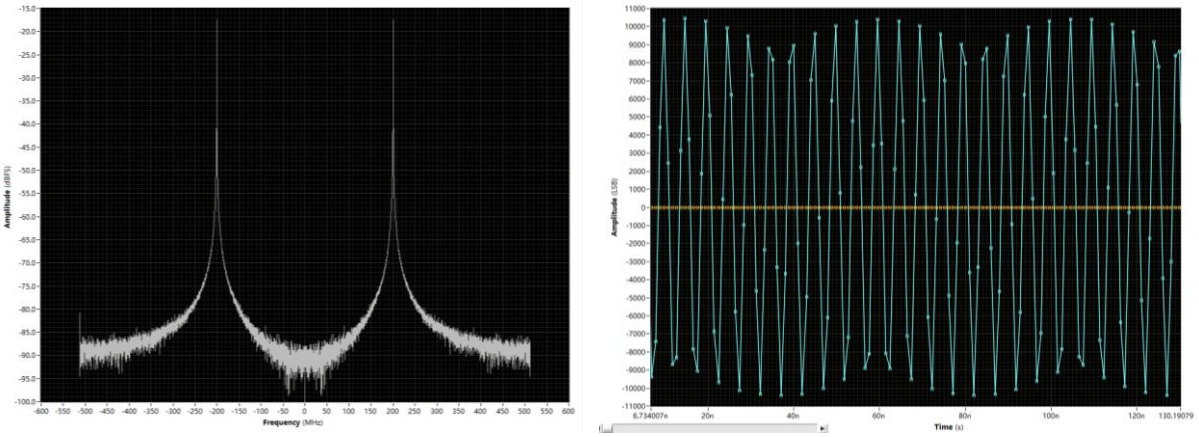


Figure 5: BRAM Memory Write at 200MHz

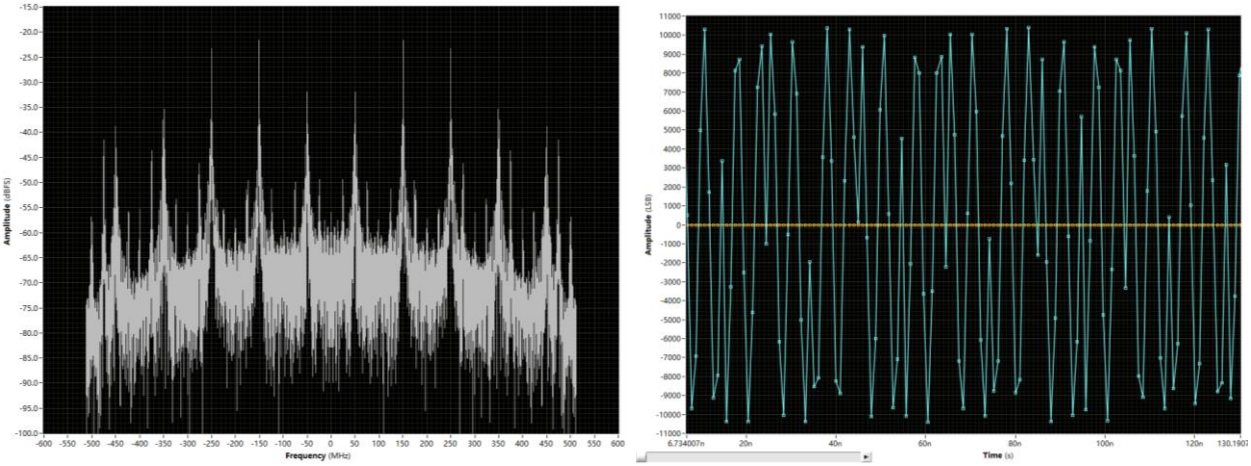


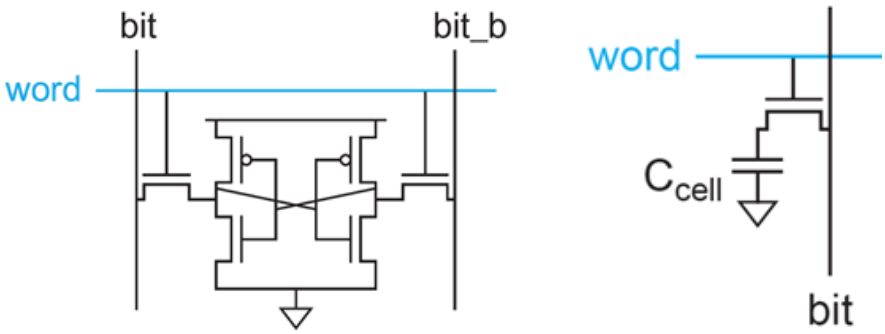
Figure 6: DDR4 Memory Write at 200MHz

We are sampling our data at a rate of 1 GSPS which requires at least 1.5 GB of memory to save the sampled data. However, we are not able to directly write the data into storage (our SSD) since it does not meet the writing speed of 1.5 GB per second. To avoid data loss, we need a fast memory that can hold the data temporarily. The evaluation tool gives us the option to choose between BRAM (Block RAM) and DDR which is a type of DRAM.

DRAM refers to dynamic random-access memory. Figure 7 (b) indicates the structure of a DRAM cell. DRAM consists of two main components. One access transistor and one capacitor. Data is stored in the capacitor as electrical charge, but it leaks over time. Once a capacitor is charged, it will eventually start to slowly discharge, its content will slowly erase over time. To deal with leakage, DRAM is refreshed regularly to avoid losing any stored data. This means the contents of DRAM memory are read and written back. This process introduces latency and therefore normal read and write operations to DDR4 memory are slower than the same operations for an SRAM.

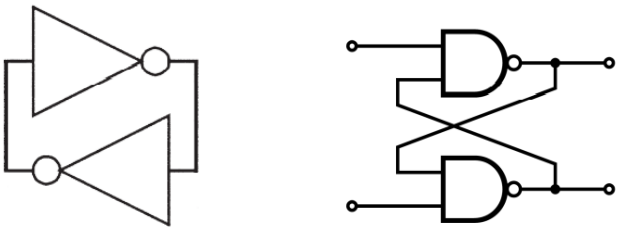
SRAM refers to static random-access memory. An SRAM stores the data using six transistors in the form of two back-to-back NOT gates as illustrated in Figure 8. Once data is written to an SRAM cell, it will not change, the data stays there for all time. BRAM is similar to SRAM, it is also static. This type of memory is much faster as there is no need for refresh cycles to maintain charge.

Therefore, the faster memory access speed of the BRAM decreases the amount of noise sampled by the ADC.



(a) (b)

Figure 7: SRAM (a) and DRAM (b) transistor level schematics



(a) (b)

Figure 8: SRAM (a) and DRAM (b) gate level schematics

6.3 Hardware Setup

There are two important hardware connections which are made. First, the ZCU111 board is connected to the data converter development board with an FMC port. Second, on the data converter development board, the DAC is connected to a low-pass filter, then to the ADC using an SMA cable. Figure 9 shows this setup. The DAC port is on the top left and the ADC port is on the bottom left, both are circled in red.

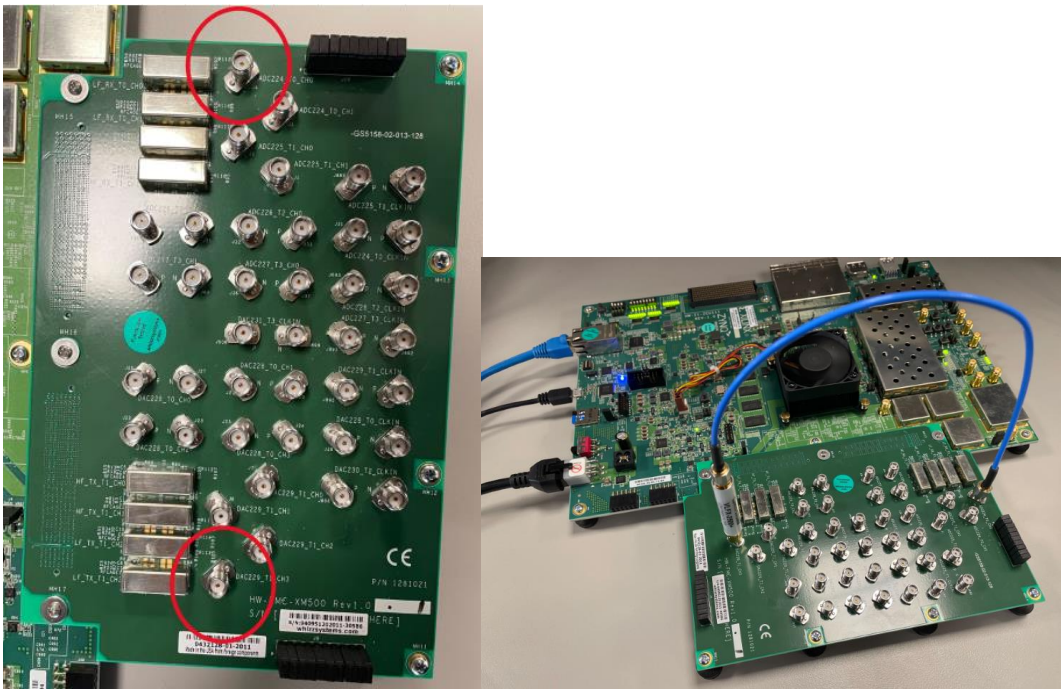


Figure 9: Hardware setup for analog signal sampling

7 Data Post-Analysis

7.1 Converting from LSB to Voltage

The RF data converter tool we use in this project represents incoming data as the number of counts in voltage. A count in voltage is equivalent to the LSB, which is the smallest voltage level that the ADC can convert to the digital domain [y]. Please see the figure below for a visual representation of this concept.

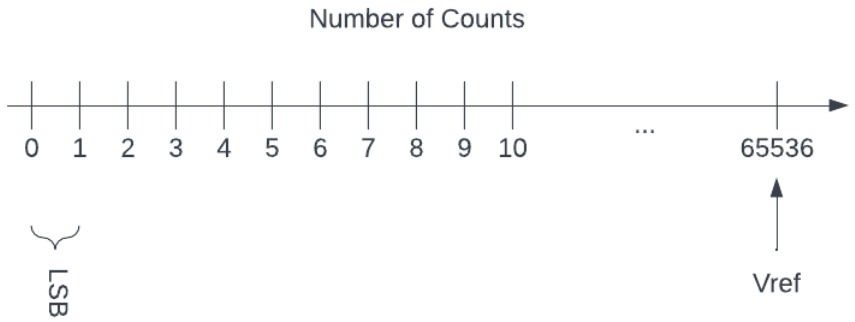


Figure 10: Visual representation of the relation between LSB and voltage

We use the following equation to convert from LSB to the desired voltage during post-processing of data:

$$LSB = \frac{V_{ref}}{2^n}$$

$$V_{in} = Number\ of\ Counts \cdot LSB$$

We are measuring a 0-2.5V signal therefore:

$$V_{ref} = 2.5V$$

Though our ADC has a 12-bit width, the tool aligns the data to 16-bit samples during sample collection [x]. We must adjust accordingly for this detail, therefore:

$$n = 16 \text{ and } V_{ref} = 2.5V = 2^{16} = 65536$$

We can calculate our LSB as:

$$LSB = \frac{2.5V}{2^{16}} = 38\mu V$$

Our client expects the final data to be presented in units of voltage. To successfully accomplish this, we multiply every value measured by the ADC by the calculated LSB.

7.2 Post Sample Data Analysis GUI

To help users analyze sampled data, we developed a data analysis GUI using Python.

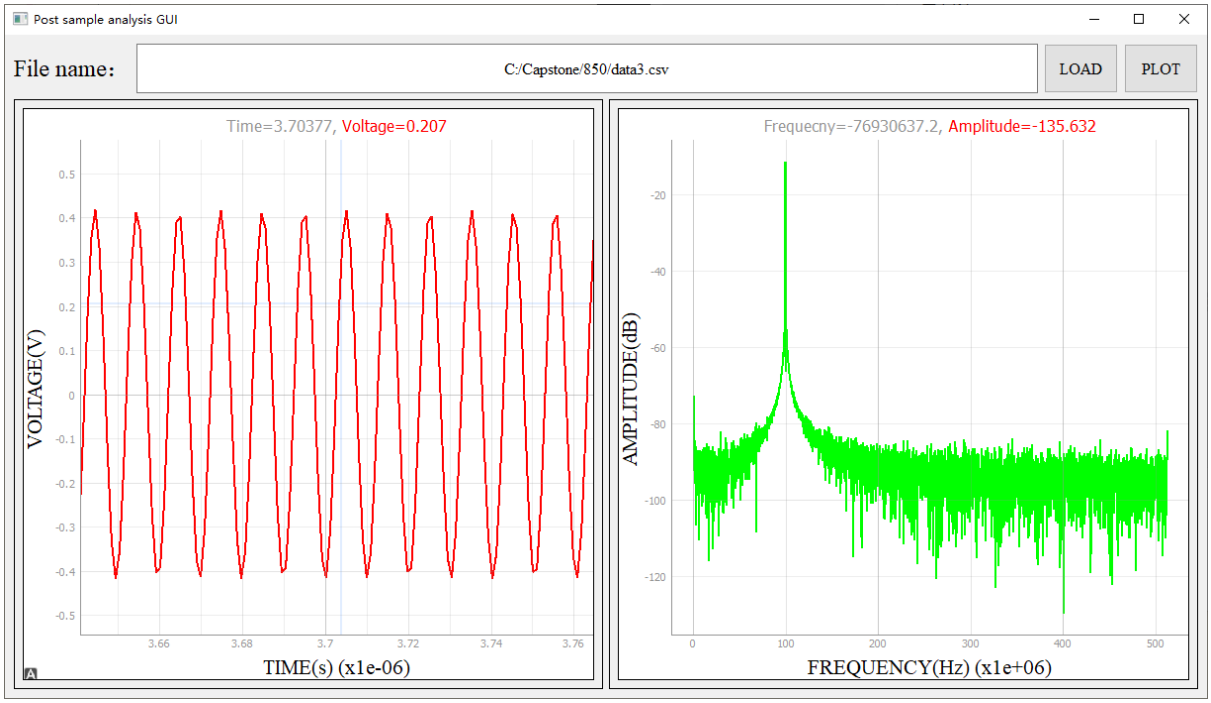


Figure 11: Plotting sampled data in time and frequency domains using custom GUI

Data Selection

Before plotting, a .csv file must be chosen for analysis. We applied the static functions of the QFileDialog class in the Pyqt package to enable navigation of the file system to select and open the .csv file exported from the board.

Functionality and Features

The main window layout is developed using a Python package called Pyqt. The interface includes two main windows. The window on the left shows the acquired data in the time domain, and the window on the right shows the data in the frequency domain. Viewing both waveforms simultaneously allows the user to analyze and compare the two domains.

A cursor appears when the user hovers over any point on the plot. Coordinates of the cursor can be viewed at the top of the window. This is a critical feature which allows for accurate and precise viewing of the sampled data.

To better observe the signal we have included free dragging and zooming functions to the plot windows. Data from any point of time during sampling can be viewed by manually entering the interval of interest. The GUI will switch to the corresponding plot area.

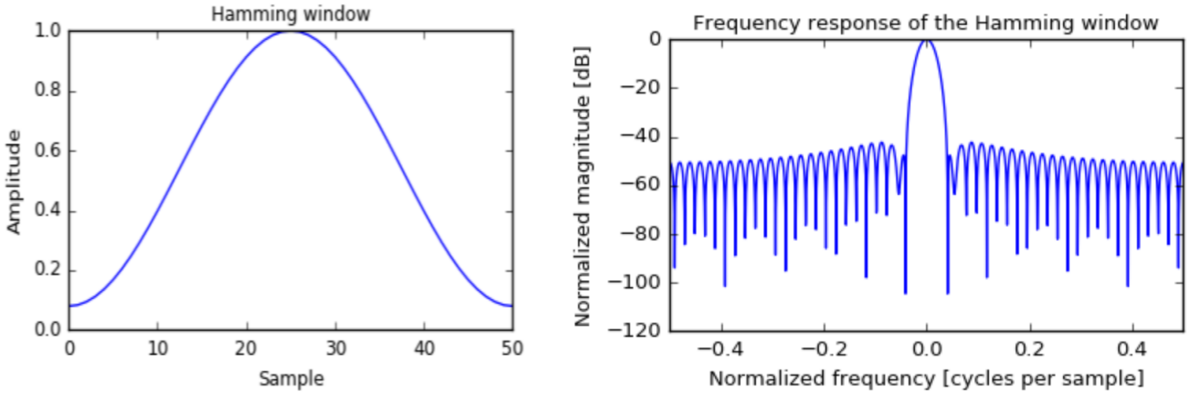
Further instructions on how to use all these features can be found in our user guide document.

Data Processing

Once the data has been read from the file, we proceed to processing. We use NumPy, a math library available in Python, to convert the LSB values into their corresponding voltage values.

Taking an FFT of the sampled data is completed with the use of the SciPy package. For any given time interval, a limited number of data samples can be transformed from the time-domain to the frequency domain using scipy.fft. In the case of periodic signals, if the time interval is not a multiple of the period, then data loss or signal distortion may occur when taking an FFT.

This problem is addressed by using a Hamming Window Function. This function choses which data samples get transformed and displayed in the FFT plot using a taper by using a raised cosine with non-zero endpoints, as visible in Figure 12 (a). More specifically, it ensures that any frequencies outside of the desired range get tapered off to nearly zero. As data is plotted, the window function moves forward by half-intervals, ensuring that any lost data is represented again.



(a) (b)

Figure 12: Hamming Window

8 Auxiliary Design Choices

The following section involves design development Alinx AXKU040 FPGA. These design choices are not included in our final solution.

Unlike the ZCU111 the AXKU040 does not include its own ADC. We ran into a roadblock in our endeavor to purchase an ADC to connect to the AXKU040 board. Despite extensive research on physical connectors and communication protocols, the AD9094 ADC evaluation board we ordered did not physically connect to the Alinx board. Though both boards had the necessary HP FMC connectors, neighbouring ports prevented complete physical interlocking.

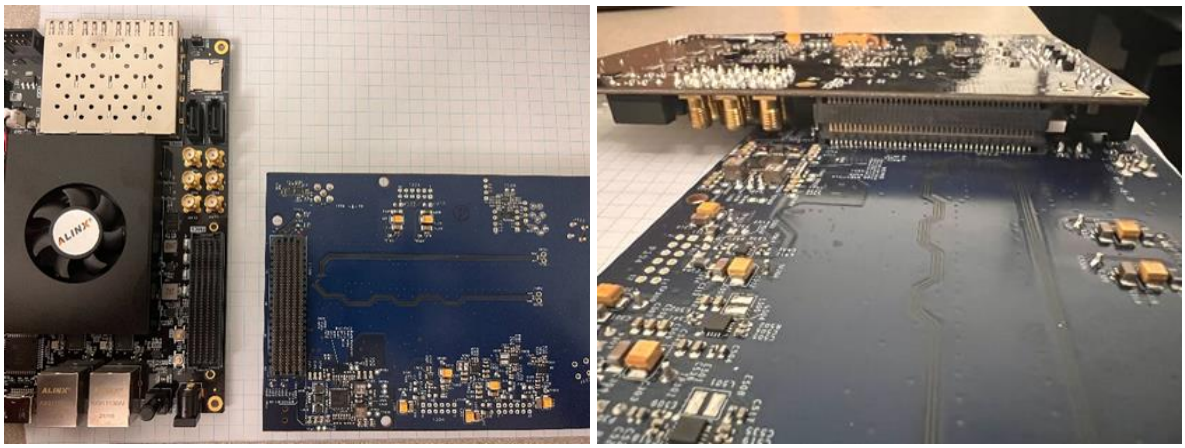


Figure 13: Alinx AXKU040 FPGA Development board incompatibility with AD9094 ADC Evaluation Board

As part of our risk assessments early in the project timeline, our team set a contingency plan of switching focus to the ZCU111 if we encountered setbacks such as this one. We are confident that the choice to shift our efforts to development for the ZCU111 board was the appropriate choice and allowed us to provide our client with a working solution within the allocated project timeframe.

Data Handler Module

This module takes input data from an ADC, and feeds it to a FIFO. The FIFO transfers data directly into the DDR4 memory using a DMA module. DMA allows memory to be controlled without the use of the processing unit, which frees up the CPU to configure and control the ADC port. Finally, data is exported from memory into an external form of storage.

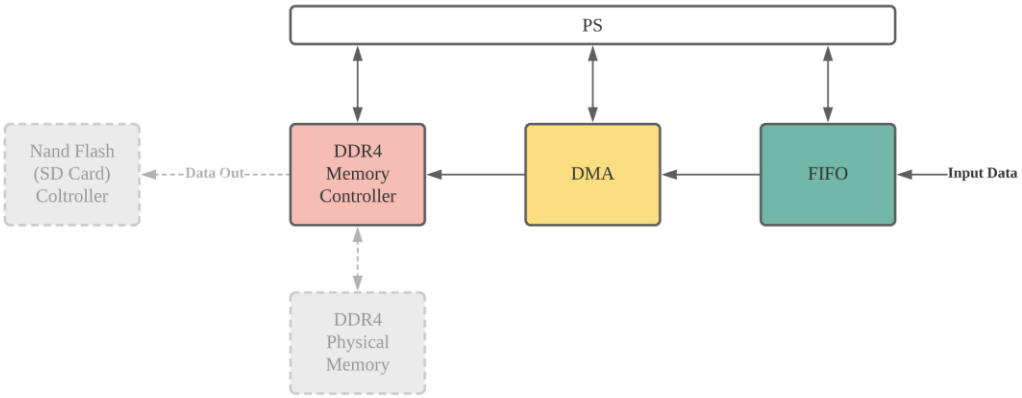


Figure 14: Digital System Block Diagram

8.1 Overview of IP Blocks Used in the Data Handler Module

8.1.1 FIFO

First In, First Out is a data processing technique commonly used in RX data buffers. The premise of a FIFO is just as the name suggests: the data packets present in the FIFO buffer are processed in the same order in which they arrived. FIFO buffers have two output signals which indicate to the memory controller when the FIFO is full and can no longer receive more data, and when the FIFO is empty and can no longer be read.

Using a FIFO data buffer provides us with the following necessary functionality: allows us to slow down incoming data before writing it to memory and allows us to accumulate a wide enough data bus to fill an entire memory block.

The first of these functionalities is important because we are sampling data at a faster rate than we can write to memory. Our ADC has an 8-bit width and is sampling data at 1 GSPS, therefore our data transfer rate is 8 Gbps. The DDR4 data transfer rate is 2400 Mbps. Given that memory writes are far slower than the incoming sampled data rate, proves the necessity of a FIFO.

The second of these functionalities must be considered because the DDR4 memory width is 64-bits and the ADC outputs 8-bit samples. We need to gather 8 samples in order to write to a single address in memory.

The FIFO IP module in Vivado allows the user to set two main parameters, data width, and FIFO depth. A width of 64-bits was selected: we chose to equate the FIFO width to the memory block size for simplicity. This will have to be changed to 8-bits once we integrate the analog part of our design. The maximum FIFO depth of 2048 was chosen to maximize the amount of buffer space available for receiving data. Once the FIFO is full with 2048 samples, we begin memory writes to contiguous addresses until the FIFO is empty.

8.1.2 DMA Module

A DMA IP module is a vital component of a computer system to allow data to be transferred directly into the memory without assigning extra work to the CPU.

The DMA module is vital for allowing data to be transferred directly into memory without making use of the CPU. This can reduce system latency by increasing the rate at which data can be emptied from the FIFO. Memory writes can occur simultaneously to the ADC control, without over-exerting the CPU.

MM2S (Memory-Mapped to Stream) and S2MM (Stream to Memory-Mapped) connections have access to the processor system's memory and these connections are used to write to and read from the memory.

Summary of DMA pins used:

- S_AXI_LITE pin is used to configure the high-speed streaming data bus
- S_AXIS_S2MM pin is used to receive data from the FIFO
- M_AXIS_MM2S pin is used to transfer data to the DDR4 memory IP block.
- axi_resetn pin is the reset signal connection

Using the CPU to control data transfers would be less efficient than using a DMA, whose primary purpose is to reduce hardware resources required for memory accesses.

8.1.3 AXI Interconnect

The AXI interconnect module is a protocol which allows the processor to communicate with the DMA and FIFO blocks using a master/slave configuration[1]. There are three important master/slave interface pins of note, which are briefly outlined below.

Summary of master/slave interconnect pins:

- S00_AXI connects to the FPD master control pin from the processor.
- S01_AXI connects to the LPD master control pin from the processor.
- M00_AXI connects to the slave input of the data FIFO.

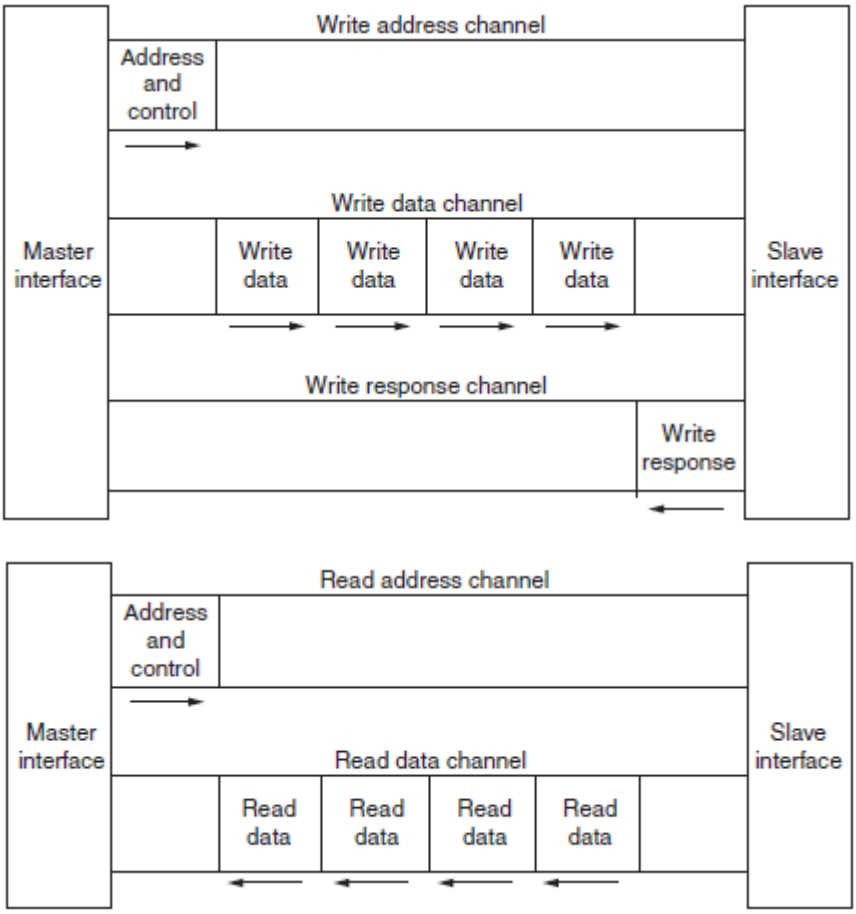


Figure 15: AXI Protocol Channel Architecture of Reads And Writes

The AXI protocol is as follows:

Master and slave perform a “handshake” to confirm valid signals are being transmitted. In our design, the DDR4 and FIFO blocks are considered “slaves”. In the case of continuous data transfer, the protocol supports burst-type communications. AXI is a point-to-point specification, rather than a bus specification. This means it only describes the signals and timing between interfaces. This is suitable for our design, as we are continuously receiving sampled data from our ADC.

AXI is also capable of interconnecting multiple blocks with the processing system. The “master” keeps track of incoming handshake signals and can resolve requests without waiting for a previous task to be completed. This decreases overall system latency as it allows for parallelism.

References

Alinx (2020). KINTEX UltraScale Development Platform User Manual. Alinx, Retrieved from: https://drive.google.com/file/d/1N_XBmYSrAyLu8X3UpVVEuoqvzugi2y1M/view?usp=sharing

Analog Devices (2020). AD9094 Datasheet (Rev.0). Analog Devices. Retrieved from: https://www.analog.com/media/en/technical-documentation/data-sheets/ad9094.pdf

Analog Device (2013). JESD204B Survival Guide. Analog Device. Retrieved from: https://www.analog.com/media/en/technical-documentation/technical-articles/JESD204B-Survival-Guide.pdf

Arm Developer (n.d.). AXI Protocol Overview. An Introduction to AMBA AXI. Retrieved from: https://developer.arm.com/documentation/102202/0200/AXI-protocol-overview=

DataQ Instruments (2021). How Much ADC Resolution Do You Really Need. DataQ Instruments. Retrieved from: https://www.dataq.com/data-acquisition/general-education-tutorials/how-much-adc-resolution-do-you-really-need.html

Harris, D. M., & Harris, S. L. (2015). Digital Design and computer architecture. Morgan Kaufmann.

Perpetual Industries. (n.d.). Signal Processing: Aliasing. Perpetual Industries. Retrieved from: https://xyobalancer.com/signal-processing-aliasing/

Pini, A. (2020). The Basics of Anti-Aliasing Low-Pass Filter (and Why They Need to be Matched to the ADC). Digi-Key. Retrieved from: https://www.digikey.ca/en/articles/the-basics-of-anti-aliasing-low-pass-filters

Texas Instruments (1998). An Overview of LVDS Technology. National Semiconductor N.971. Retrieved from: https://www.ti.com/lit/an/snla165/snla165.pdf

Xilinx (2018). ZCU111 Evaluation Board User Guide. UG1271 (V1.2). Xilinx. Retrieved from: https://www.xilinx.com/support/documentation/boards_and_kits/zcu111/ug1271-zcu111-eval-bd.pdf

Tan, J. (2021). Embedded Linux: A Beginner's Guide. Seeed Studio. Retrieved from: https://www.seeedstudio.com/blog/2021/01/20/beginners-guide-to-embedded-linux/

Xilinx (2016). Integrated Logic Analyzer v6.2 Product Guide. Vivado Design Suite PG172. Retrieved from: https://www.xilinx.com/content/dam/xilinx/support/documentation/ip_documentation/ila/v6_2/pg172-ila.pdf

Xilinx (2012). AXI Reference Guide. UG761 (V14.3). Xilinx. Retrieved from: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug761_axi_reference_guide.pdf

Xilinx. (n.d.). PetaLinux Tools. Xilinx. Retrieved from: https://www.xilinx.com/products/design-tools/embedded-software/petalinux-sdk.html#tools

Xilinx. (2018). Zynq UltraScale+ RFSoC RF Data Converter Evaluation Tool (ZCU111) User Guide. UG1287 (V2018.2). Xilinx. Retrieved from: https://www.xilinx.com/support/documentation/boards_and_kits/zcu111/2018_2/ug1287-zcu111-rfsoc-eval-tool.pdf

[x] RF Data Converter Product Guide:

https://www.xilinx.com/support/documentation/ip_documentation/usp_rf_data_converter/v2_4/pg269-rf-data-converter.pdf

[y] ADC LSB

https://masteringelectronicsdesign.com/an-adc-and-dac-least-significant-bit-lsb/