UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

Master's degree in Applications and Technologies for Unmanned Aircraft Systems

MASTER THESIS

TITLE: Contribution to the development of microwave remote sensing for UAV systems.

MASTER'S DEGREE: Master's degree in Applications and Technologies for Unmanned Aircraft Systems (Drones) (MED)

AUTHOR: Tuvshinbayar Alagbaatar

PROFESSIONAL ADVISOR: Jové Casulleras, Roger
ACADEMIC ADVISOR: Gonzalez Arbesu, Jose Maria

DATE: July, 13rd 2022

Abstract

Current telecommunication systems have transitioned from analogical systems to digital ones. Such transformation has been enabled by the use of Software Defined Radios (SDR). Nowadays, such change is moving to the Remote Sensing applications.

This Master thesis has been developed within the industrial framework of the company Microwave Sensors and Electronics Ltd, known by its trademark "Balamis".

Balamis develops microwave sensors. In particular: L-band radiometers and multi-frequency radars. Following the telecommunication trend, the Company is currently implementing its new systems based on Software Defined Radios with the objective to provide remote sensing data.

The use of Software Defined Radio technology on-board an L-band radiometer allows the detection of the interferences. Previous Balamis internal works partially funded by CDTI (part of the Ministry of Industry) showed how drone electronics create interferences that mask L-band radiometry measurements. Consequently, a contribution towards the detection of the interferences that affect the L-band radiometry measurements is the first objective of this Master Thesis: Designing and implementing a “FFT system in FPGA” using the most recent architecture of XILINX 7-series FPGA. Digilent’s Arty z7 evaluation board, its XILINX’s previously generated in-system peripherals, such as FFT, DMA, XADC Intellectual Property (IP) cores, and Zynq processing system are used for the development. In the software development, bare-metal C and PetaLinux embedded operating system are used.

The second objective of this Master Thesis is the temperature stabilization with “Thermal control”. FPGA-based systems like SDRs tend to have a high power consumption which has to be dissipated. However, temperature stability is a requirement beyond the cooling need of the FPGAs. L-band radiometer accuracy depends on the thermal stability of the system itself. Consequently, a “thermal control system” has been developed as the second part of this thesis. The thermal control is developed using a microcontroller (STM32), ADC, H-bridge driver for controlling Peltier cell.

The report starts with technical characteristics, descriptions and configurations of in-system or physical components used for both implementations. Then followed by some explanation of software. Methodologies of designing the systems are described in the Chapter 3 with all the steps walked through for the system design. At the end, before the conclusion, the result of whole project is illustrated.

CONTENTS

LIST OF FIGURES

INTRODUCTION

Microwave technology is very sensitive to Radio Frequency Interferences (RFI). Works previously done within this Master by Marc Jou [1] showed the impossibility to retrieve measurements using L-band radiometers on-board drones. After detecting such issue, Balamis first tried to solve it by hardware: a new antenna design and the extensive use of shielding on the drone were tried without success.

Balamis started the development of its first digital radiometer based on the use of Software Defined Radio architecture on 2017, partially funded with the support of CDTI. The resulting minimum viable digital radiometer was ready by June 2019, but it did not include any RFI mitigation capability. Developments done my Master student Ahmad Daoud [2] demonstrated the identification of RFI using Fast Fourier Transform (FFT) over RAW data but could not provide any efficient implementation of its mitigation on-board the L-band radiometer.

The proposed solution is the implementation of the FFT and the RFI filters using Field Programmable Gate of Array (FPGA) for the input signals, and its concurrent performance. Filtering an analog signal by introducing in-system FFT of ZYNQ7000 FPGA is implemented in this project.

Additionally, the power consumption of FPGA, and the need to dissipate it, forces the development of a temperature control system with cooling capabilities. It is done to improve the previous heating-only thermal control of Balamis radiometer. Such more advanced thermal control will be also used for the Interferometric Ground-based Synthetic Aperture Radar that Balamis is developing.

Solving these two goals are therefore the purpose of this Master Thesis.

CHAPTER 1 THEORETICAL BACKGROUND

1.1 Digital signal processing

Filtering, convolution, correlation, and Fourier transforms are examples of Digital Signal Processing (DSP) operations. These operations necessitate addition and multiplication, so the unit know as a multiply-accumulate (MAC) must be included in DSP processing system. DSP processors are used for running DSP applications. In the DSP processors, multiplying and summing are performed one after another. A DSP processing unit can only accomplish an only one summation-multiplication operation in a one cycle when digital filters are used in filtering. Consequently, the output of each filter is performed in a one cycle, but more iterations are needed to implement the filter specific requirements in a higher order filter.

FPGAs' parallel processing and multi-threading are well-known. The summing and multiplying processes run parallel in FPGAs. In contrast to DSP processing units, an FPGA can execute numerous such operations in a single cycle. FPGAs thus require fewer iterations than DSP processors.

As application demand grows, so does algorithmic complexity. To implement these new algorithms, signal processing system with hight efficiency is necessitated. Traditional DSP processors with fixed hardware platforms could really keep up. As a result, as algorithmic intensity increases, so does the deficit [3]. The difference between a formal DSP processing unit implementation and an FPGA integration is visualized in Fig 1.1.

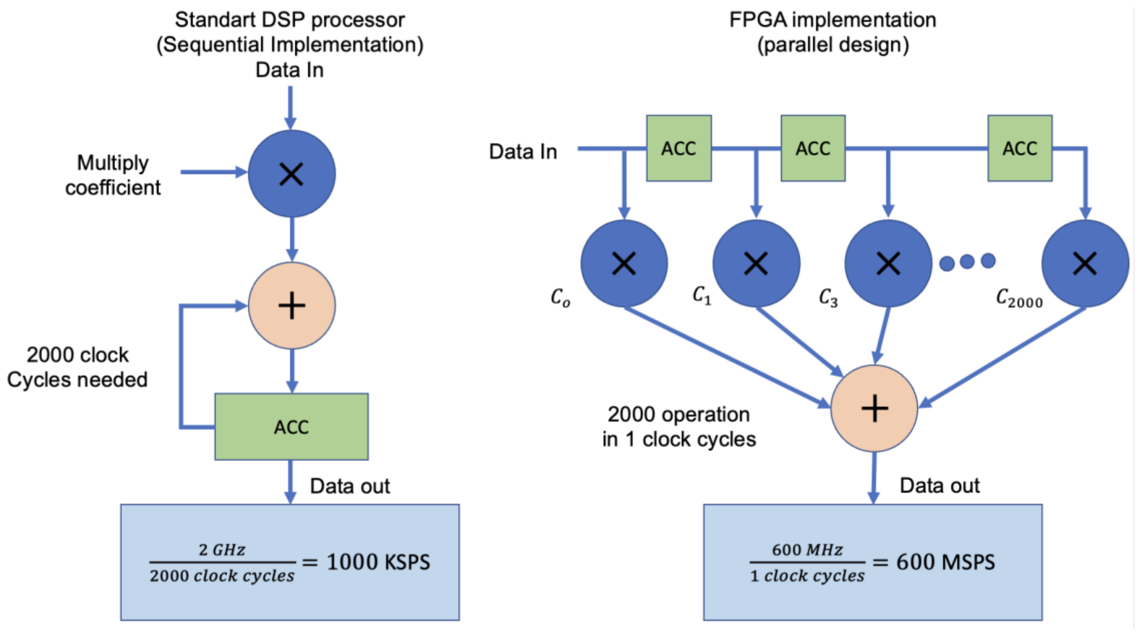


Figure 1.1 Conventional DSP and FPGA

For a variety of reasons, FPGAs are well appropriate to filling this technology gap. These chips provide incredibly high signal processing capacity, thanks to their parallel processing.

- Because of the flexible architecture, they ensure very low risk.
- They allow for design transition in order to accommodate modifying norms.
- Engineers can use FPGAs to create a unique and customized technical solution.
- They are reasonably priced.
- They have a low power to function ratio [4].

A DSP or FPGA manages computational models in accordance with the digital filter of a system. The filter which takes digital inputs and produces digital outputs is a digital filter. During filtering, a DSP unit or a FPGA retrieves a sampled analog signal from an analog to digital converter (ADC), completes computational operation based on the filtering form, and outputs to a digital to analog converter (DAC).

1.2 Sampling Process

Analog-to-digital conversion, also known as digitization of analog signals, consists of sampling which is time digitization and quantization which is digitization in magnitude. An analog signal is represented as a series of values by the sampling component. This procedure is carried out with the help of a "sample-and-hold" circuit, which keeps the sampled value of the signal until the next sample [5]. A fundamental DSP procedure is shown in Figure 1.2.

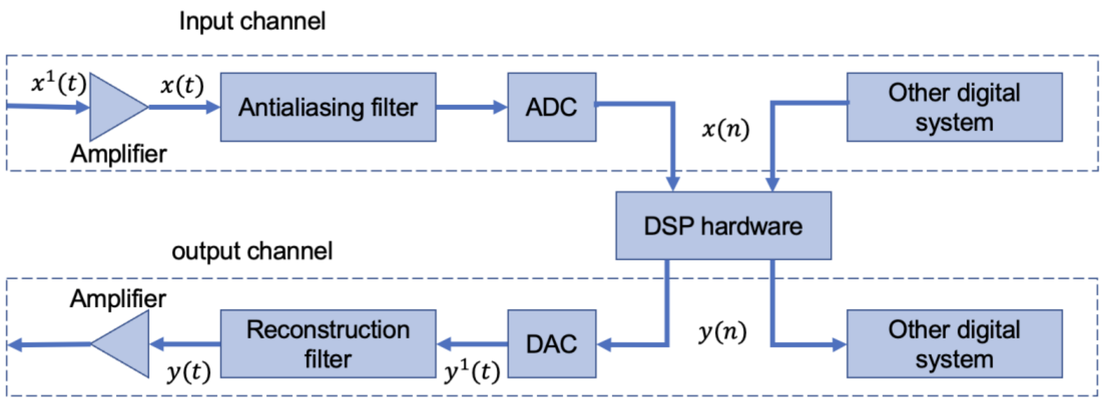


Figure 1.2 Basic DSP operation

The analog to digital conversion is performed in the following manner: the analog input signal is sampled by the ADC in uniformly pierced samples. In this way, an analog signal is transformed into a discrete time signal.

$$x[n] = x(nT) \qquad -\infty < n < \infty \qquad (1.1)$$

where T is the sampling time and n is a positive integer.

Every quantized 2^k level belongs to one sample's magnitude, where k is the number of bits of ADC. These samples are also binary encoded. An ADC is similar to a switch that is opened and closed in every T seconds. As a result, the sampling frequency is represented as follows:

$$f_s = \frac{1}{T} \qquad (1.2)$$

where f_s is in hertz. At discrete time nT, being n = 0,1,2,..., the discrete-time signal $x(nT)$ can be represented by a positive integer [1]. This is depicted in Figure 1.3.

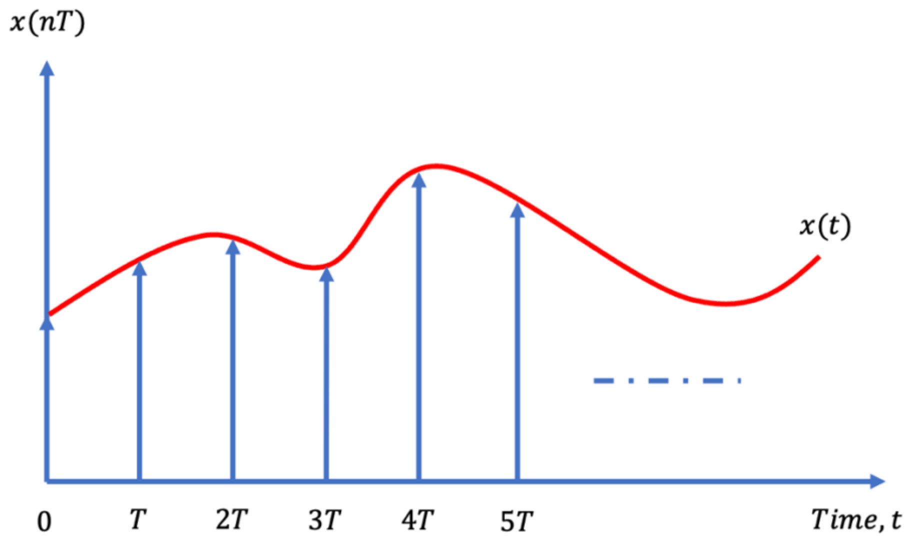


Figure 1.3 Analog signal sampling

To define precisely an analog signal $x(t)$ as a discrete-time signal $x(nT)$, the sampling frequency f_s must be at least twice as large as the frequency part of the analog signal f_M. It is written as follows:

$$f_s \geq 2f_M \qquad (1.3)$$

Where f_M is usually referred to it as the "Nyquist frequency," and $2f_M$, that must be exceeded by the sampling frequency, is referred to as the "Nyquist rate" [2].

$$\Delta = \frac{x_{max} - x_{min}}{2^b - 1} \qquad (1.4)$$

where x represents the input signal and b stands for the number of bits that reflect the signal. Δ recognizes the quantizer phase size specified by an ADC [3].

According to Equation 1.4, increasing the number of bits results in a significant improvement in signal resolution. An analog amplitude of 1V can represent this situation. There are only sixteen steps to indicate that 1V if a four-bit digital signal is utilized to depict it. Steps are fitted, however, if the analog signal is defined by a 24-bit digital signal. If the quantization is homogeneous, every step has such a fitted size, and one step equals:

$$\frac{1v}{2^{24}-1} = 0.0596\mu V \qquad (1.5)$$

In case of digital signal with 4-bit is used, then every quantization is:

$$\frac{1v}{2^{4}-1} = 125mV \qquad (1.6)$$

1.3 Digital Filtering

Digital filtering is accomplished through the application of mathematical operations to sampled data (digitized data). Electronic circuits are used to perform filtering operations for analog filtering. Coefficients are used to perform digital filtering on sampled signals [6]. There are numerous filters, but in this thesis, one of the digital filters known as the FFT system in FPGA is used.

1.4 FFT system in FPGA

In the field of DSP, the "Fast Fourier Transform" (FFT) is a major method of measurement. It decomposes a signal into individual spectral information and provides frequency components of the signal as a result. FFTs are used in machine or system for specific purposes, such as fault analysis, quality control, and condition monitoring.

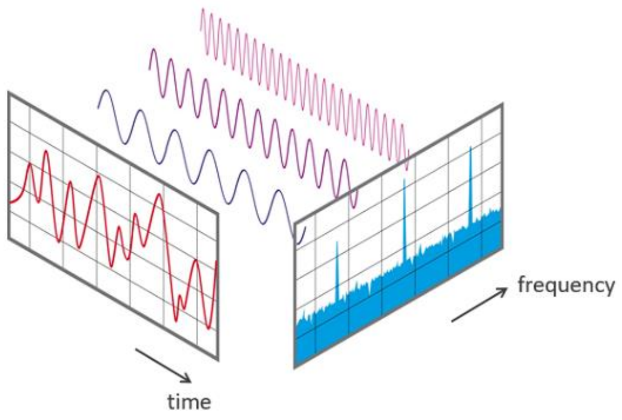


Figure 1.4 Signal components in the time and frequency domain [7]

In its most basic form, The FFT itself is the popular algorithm for accomplishing the "Discrete Fourier Transformation" (DFT). Over time, a signal is processed and sampled, and its frequency components are separated. These elements are single sine wave with distinct frequencies and amplitudes and phases. This transformation is depicted in the Figure 1.4 above. In the example, the signal contains three distinct dominant frequencies over the measured time period.

These spikes in the figure are belonged to useful frequency components of the original signal whereas the smaller spikes are of noise. Based on this characteristic, unwanted noise elements of the signal can be eliminated by setting a threshold relative to the amplitude of frequency components.

1.5 PID controller

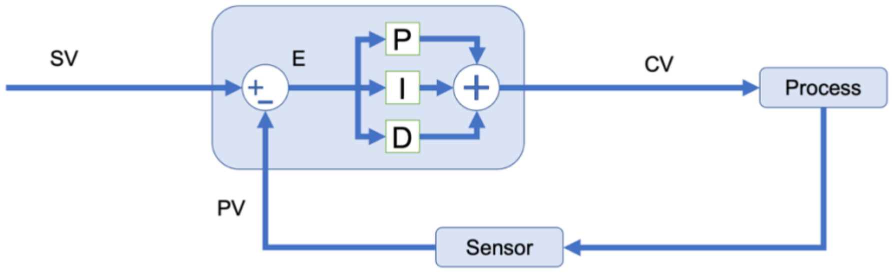


Figure 1.5 Block diagram of PID controller

The controller determines the P, I, and D actions and multiplies each parameter by the error, or E, which is equal to

$$E = SV - PV \qquad (1.6)$$

Where, SV - Set Variable, PV – Process. The Control Variable (CV) is then created by adding the results of all parameter calculations. In Figure 1.5:

P: The proportional gain, not a quantity but rather a "ratio", is the phrase "proportional" which is also called the "P" Constant. The parameter is also known as K_p, Gain. This parameter controls how quickly the system reacts. A higher setting for this tuning parameter may result in more sensitive, less stable loops for controllers which use the term "Gain." Conversely, lowering this tuning parameter has the same effect on the loop on controllers with proportional band units [8].

I: The integral term, sometimes known as the "I" Constant or "Reset," can also have varied measurements. K_i is another name for this parameter. This variable controls how quickly the steady state error is eliminated [8].

"D" or derivative the usual constant units are seconds or minutes. Predicting change is the goal of the "Derivative" constant. The Process Variable's rate of change is acted upon by the Derivative action. How far into the future you want to estimate the pace of change is essentially indicated by the value of this parameter. This setting can aid in making the loop operate better and respond more quickly [8].

CHAPTER 2 HARDWARE AND SOFTWARE PLATFORM

2.1 Initial Consideration and Platform Selection

2.1.1 Hardware selection of FFT system in FPGA

In both situations, studying for or working with FPGAs, it is always critical to select correct chip or evaluation board for one's desired implementation. FPGA itself allows number of possibilities to design "In-system design" for engineers and scientists because of its adaptive logic modules (ALMs) and logic elements (LEs) which are connected with each other through programmable interconnects.

In our case, a system for implementing the Fast Fourier Transformation (FFT) based filtering in Digital Signal Processing (DSP) domain has to be developed. The system also must include an access for Direct Random Access Memory (DRAM) to be able to write and read the results. I choose an evaluation board of XILINX's FPGA, "Arty Z7-20" with Zynq 7000 processor and external DDR3. The reasons why I choose Arty Z7 are follows.

- The FPGA in Arty Z7 board is big enough for large projects
- It features enough built-in peripherals, including FFT IP and XADC for signal processing, USB for a webcam and HDMI input and output.
- Sufficient I/O headers and Peripheral Module Interface (PMOD)

2.1.2 Hardware selection of Thermal Control

To implement any system which controls devices (such as motors or Peltier cells, a thermal sensor, an ADC, a driver) and a control unit are needed. In our case, I have chosen a microcontroller STM32 by STMicrocontrollers as my control unit. There are many other options for the microcontroller, such as AVR series by Atmel, Nios series by Altera and MSC series by Intel so on.

2.1.3 Software selection of FFT system in FPGA

Since FPGA developed by XILINX is used in this project, it is easy and straight forward to select VIVADO as my FPGA development environment and XILINX SDK as a software development environment. In fact, according to the recommendation by XILINX where they suggest that Ubuntu Linux 16.04.5, 16.04.6, 18.04.1, 18.04.2, 18.04.3, and 18.04.4 LTS (64-bit) versions are compatible with VIVADO tool [8]. I used VIVADO in Ubuntu 18.04 which is 5 years old open-source software and has less bugs to implement FPGA designs.

The Logic Analyzer (a piece of tool that records and shows numerous signals from a given circuit or system) and High-Level Synthesis (HLS) (a piece of software that converts high-level C code into Verilog logic gates. It enables users to create hardware accelerators for sophisticated ML algorithms on FPGA even if they are not knowledgeable with logic design) features of VIVADO are now available for everyone to access for all WebPACK targets (a bunch of software for a specific environment), including the Arty Z7, start from VIVADO 2015.4 release [8]. The Logic Analyzer assistance in logic debugging, and a C code can be compiled directly into HDL by the HLS tool. Arty Z7 follows suitability for embedded Linux systems as any Zynq platform does. Digilent offers a PetaLinux OS (Embedded Linux operation system) platform to support users for use of Linux OS in their system directly [8].

As a specification of the project, the output of FFT must be stored in any kind of storages such as a SD card. By doing so, these outputs would be available for further signal processing, like filtering. To fulfil this specification, an embedded operation system PetaLinux is chosen because one can develop stand-alone applications for certain system and can use numerous libraries to compile or build these applications. Alongside the reason mentioned above, thanks to the drivers of XILINX's IP block for PetaLinux, it makes engineers' life easier to work with the IP blocks.

2.1.4 Software selection of Thermal Control

In the field of designing the PCBs, there are plenty of choices available as software including Altium, KiCAD, EasyEDA, Proteus and so on. Among them, I have chosen KiCAD software by Jean-Pierre Charras to design my PCB layout design of “Thermal Control” because it open source, used as professionally and its output is accepted by PCB manufacturers [9]. To write a program for controlling the whole system, Microsoft Visual Studio with PlatformIO IDE is used.

2.2 Hardware Platform Description

2.2.1 Hardware Platform Description of FFT system in FPGA

As specification of the project, the FPGA technology has to be utilized in this project. There are plenty of supplier of FPGA, such as XILINX Inc, Altera Corporation, Lattice Semiconductor, Achronix, QuickLogic Corporation, Microchip Technology and Microsemi Corporation. The FPGA of XILINX Inc (XILINX) is chosen because this company provides plenty of supporting materials, like datasheets and instructions for each IP, to its users.

2.2.1.1 XILINX's Arty Z7 20 evaluation board

The Arty Z7 is an evaluation platform that is ready to use and is based on Xilinx's Zynq-7000 All Programmable System-on-Chip (AP SoC). The Zynq-7000 incudes XILINX's 7-series FPGA logic and a dual-core ARM Cortex-A9 processing system which is running at 650 MHz. This combination enables one to cover a complex multifunctional processing system with a customized set of softly pre-designed peripherals and processing system for the desired application [10].

The toolsets, such as Vivado, Petalinux and SDSoC, offer a straightforward way between defining one's custom peripheral set and bringing its functions and features are limited to the Linux OS or processor's bare-metal platform. For designers who is looking for an engineering experience of a more classical logic gate design, the ARM processors can be ignored, and the Zynq's FPGA can be programmed in the same way just like any other Xilinx FPGA. Variety of documentations and datasheet of the Arty Z7 that really can be handy to play with the toolkit are provided by Digilent.

The Arty Z7 is fully compatible with the Vivado Design Suite from Xilinx. This toolset integrates FPGA logic circuit design and integrated ARM processor programming into a simple, user-friendly design process. It is being used to implement solutions for any complex system, from an entire operating system running multiple applications at the same time to a simple bare-metal C code controlling some inputs or outputs.

To those who don't want to use the ARM processor in their target system, FPGA part of Zynq, without the processor, can be used for standalone FPGA system.

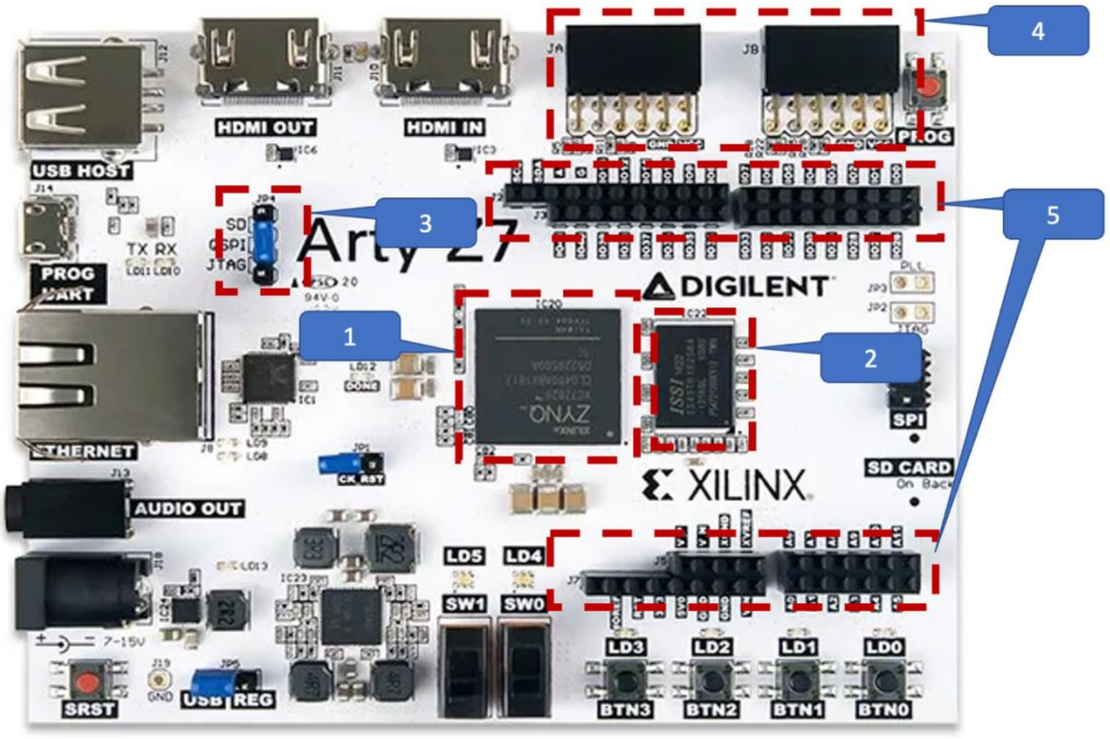


Figure 2.1 Arty z7 evaluation board (top)

In figure 2.1, 1 – Zync 7000, 2 – External DDR, 3 – JP4 (Booting option), 4 – PMOD header, 5 – Arduino extension connector.

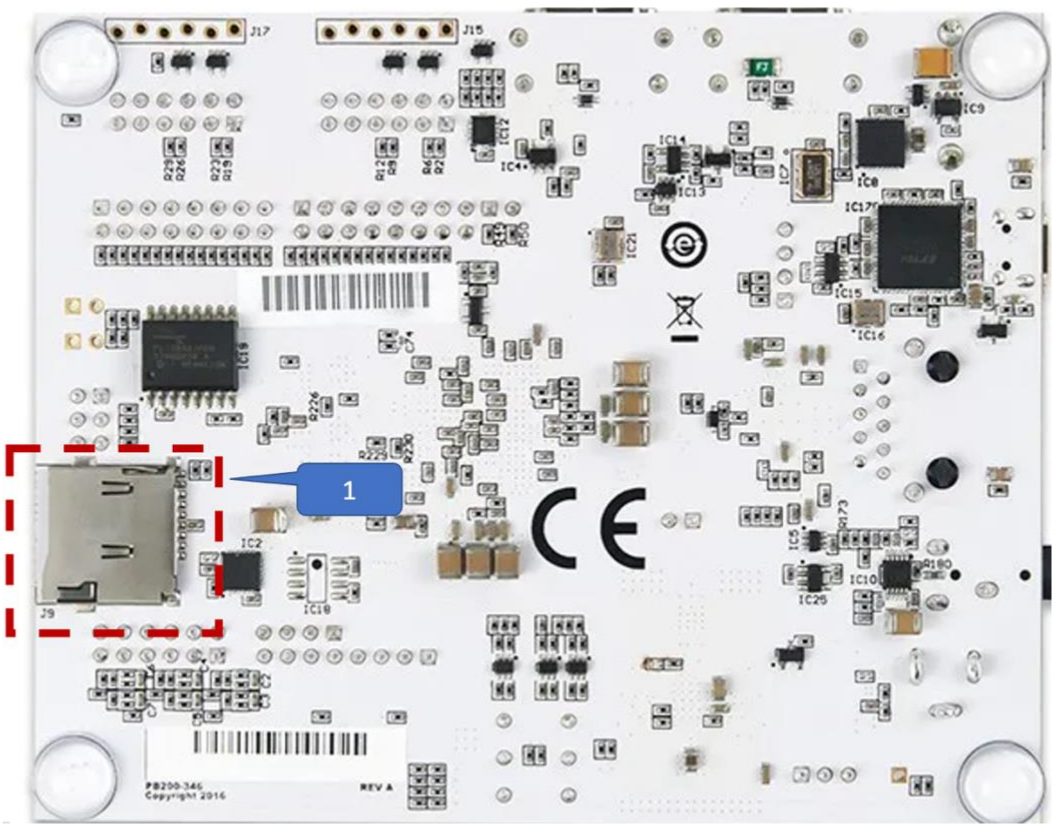


Figure 2.2 Arty z7 evaluation board (bottom)

In Figure 2.2, 1- SD card holder.
There are two main subsystems in the Zynq APSoC which are The Processing System (PS) and the Programmable Logic (PL). The overall picture of the Zynq APSoC architecture is illustrated in Figure 2.3 where the PS is highlighted in light green and the PL is highlighted in yellow [11].

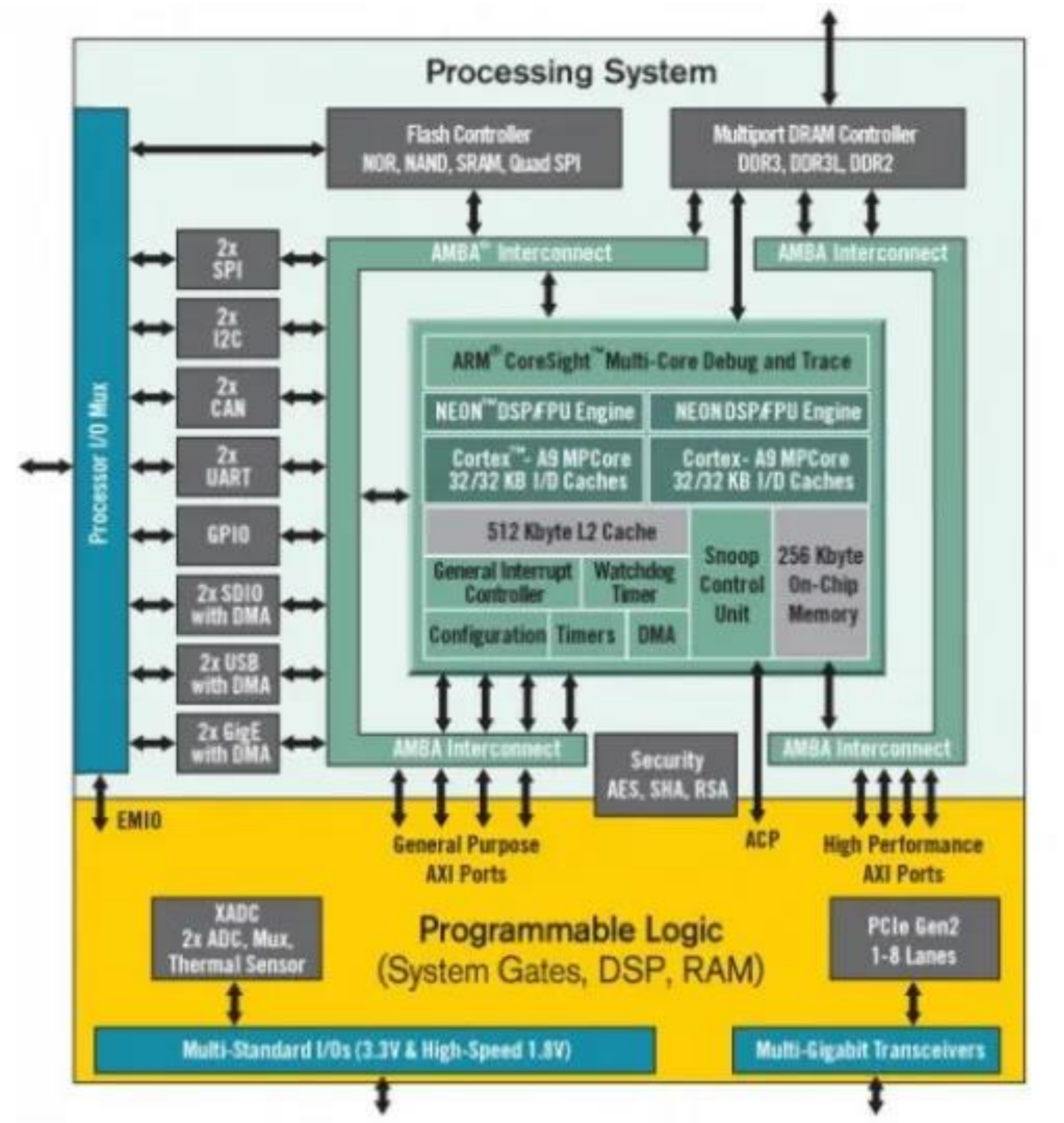

Processing System
Flash Controller NOR, NAND, SRAM, Quad SPI
Multiport DRAM Controller DDR3, DDR3L, DDR2
Processor I/O Mux
2x SPI
2x I2C
2x CAN
2x UART
GPIO
2x SDIO with DMA
2x USB with DMA
2x GigE with DMA
AMBA® Interconnect
AMBA Interconnect
ARM® CoreSight™ Multi-Core Debug and Trace
NEON™ DSP/FPU Engine
NEON DSP/FPU Engine
Cortex™- A9 MPCore 32/32 KB I/D Caches
Cortex- A9 MPCore 32/32 KB I/D Caches
512 Kbyte L2 Cache
General Interrupt Controller
Watchdog Timer
Configuration
Timers
DMA
Snoop Control Unit
256 Kbyte On-Chip Memory
AMBA Interconnect
Security AES, SHA, RSA
AMBA Interconnect
EMIO
General Purpose AXI Ports
ACP
High Performance AXI Ports
XADC 2x ADC, Mux, Thermal Sensor
Programmable Logic (System Gates, DSP, RAM)
PCIe Gen2 1-8 Lanes
Multi-Standard I/Os (3.3V & High-Speed 1.8V)
Multi-Gigabit Transceivers

Figure 2.3 Block diagram of Zynq APSoC [11]

The PL is remarkably similar to a Xilinx 7-series Artix FPGA, with the exception of a few specialized pins and buses connect PL to the PS. The PL lacks the same settings for the hardware which a typical 7-series FPGA includes, and one can configure them by using the ARM processor or with its JTAG port.

The PS is made up of several devices, such as the Application Processing Unit (APU) with two Cortex-A9 processors, the Advanced Microcontroller Bus Architecture (AMBA) Interconnect, the DDR3 Memory Controller, and numerous peripheral controlling units, (known as Multiplexed I/O, or MIO 54 pins). Peripheral controlling units with inputs and outputs that are not connected to MIO pins can route their I/O via the PL using the Extended-MIO (EMIO) interface [11].

The controllers of peripheral are slaves to the PS through the Interconnect IPs and include accessible control registers that can be found in the PL's memory. The PL must be hooked up as a slave to the interconnect, and the designer can use the FPGA fabric for implementing multiple cores, each with addressable control registers. In addition, interrupts can be sent by cores implemented in the PL to processors. This is how DMA completes the access to DDR3 memory.

Both of the USB port, what we call JTAG and another kind of powering, such as an external battery or a power adapter, can power the Arty Z7. The power source is determined by jumper JP5 (located near the power switch). According to the specifications, a USB 2.0 port can power it at a maximum current of 0.5A. This can be sufficient for simpler designs or applications. Some applications, such as those that requires number of peripheral systems or other additional USB devices, may necessitate more powerful course than the one delivered by the USB port. Power consumption will rise in this case until limitation of the USB host. This limit varies significantly between chip makers and is determined by a number of factors. When the current limit is reached, or when the voltage rails drop below their nominal value, Power-on Reset signal forces Zynq to reset, and power consumption to inactive. Furthermore, in some systems, it is required operating without having connection of a USB port of a control PC. In these situations, an external power source can be used, such as a battery.

APSoC devices, unlike Xilinx FPGA devices, are developed with a use of the PL, which serves as a master port to the PL fabric and to all those other types of on-chip devices, such as DMA IP, FFT IP, ADC IP .. etc, in the FPGA. As a result, the boot process of Zynq can be seen it as a microcontroller rather than an FPGA. The PL loads and executes a Zynq Boot (file) Image, which contains a First Stage Bootloader (FSBL), a configuration for the PL, known as a bitstream, and a custom application. There are three stages the booting process is divided:

Stage 1: When the Arty Z7 is powered on or the Zynq is reset (such as through software or the reset button, SRST), one of its processor cores, such as CPU0, begins executing an inner part of read-only predesigned software known as the BootROM. If the Zynq is only recently turned on, the BootROM initially loads the states of mode pin in terms of the mode register. In eather case, the BootROM is being operated as a result of a reset event mode or the mode pins are still not asserted, and the previous state of the mode register is used. This implies that any change in the jumper (which changes programming mode) requires a power cycle to be registered by the Arty Z7.

Stage 2: The FSBL first completes configuring the PS settings, like the control of DDR, during this stage. The bitstream is then read and used to configure the PL if it is present in the Zynq Boot Image. Finally, the Zynq Boot Image is used to load the user application into memory, and execution is passed to it.

Vivado and the Xilinx Software Development Kit are used to create the Zynq Boot Image (Xilinx SDK). We will discuss little bit about this process in the following chapters. The Arty Z7 supports three different boot modes: microSD, Quad SPI Flash, and JTAG. After powering on, the Mode jumper (JP4) is used to select the boot mode, that also modify the condition of the Zynq settings pins.

Stage 3: The accomplishment of the FSBL-loaded software application is the final stage. This could range from a "Hello World" application to a Second Stage Boot loader used for booting an OS, like Linux [5].

2.2.1.2 Zynq 7000 processor

The digital revolution now allows inventors to sum up different system components into extremely small devices, furthermore into a single complex chip, introducing the new concept namely System-on-Chip (SoC) to the field of electronics. Because of usage of XILINX's FPGA in this project, Zynq-7000 is explained as an instance of a SoC. It incorporates number of on-chip devices, such as two hard PSs, PL, numerous peripherals like ADC block or FFT block into a single silicon chip.

Prior to the discovery of the Zynq, PSs were accompanied with a FPGA, that had complicated interaction between the PL and the PS. The newer model of Xilix's all-programmable System-on-Chip (SoC) families, the Zynq architecture, merges a dual-core ARM Cortex-A9 processor with a traditional FPGA [10]. The Advanced eXtensible Interface (AXI) standard-based interface is responsible for communication be between the various elements, such as ADC, DAC, DMA IP, in Zynq-7000. This interface allows for high bandwidth and low latency connections.

Users were previously utilizing a soft-core processor, like XILINX's Microblaze before developing the ARM processor inside the Zynq device. The flexibility of cpu cores instances inside a layout was and continues to be the primary benefit of Microblaze. In Zynq, however, the addition of a hard processor results in greater improvements in performance. Moreover, by minimizing the system to a single chip, the total cost and physical dimensions of the device are reduced.

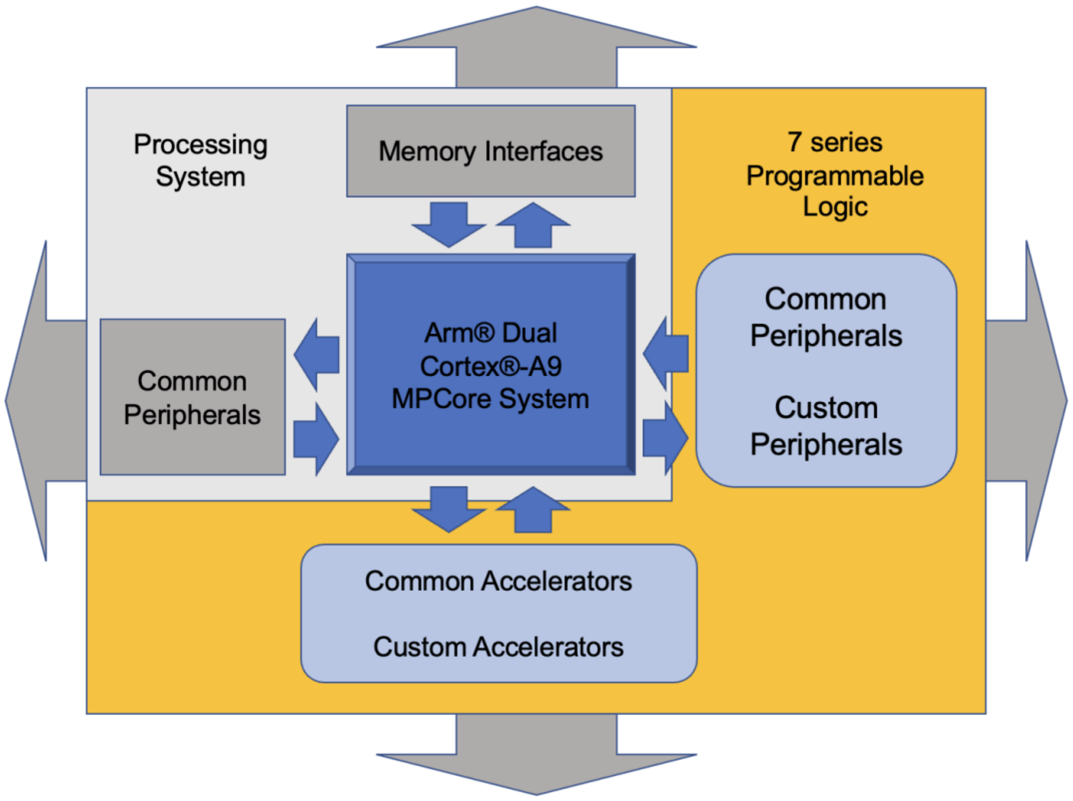


Figure 2.4 Structure of the APU

The APU has two ARM cortex-A9 processor units, each of them consists of a NEON unit, a floating-point unit (FPU), a memory management unit (MMU), and L1 caches. Snoop control and L2 caches are also included in the APU. Figure 2.4 depicts the structure of the APU [10].

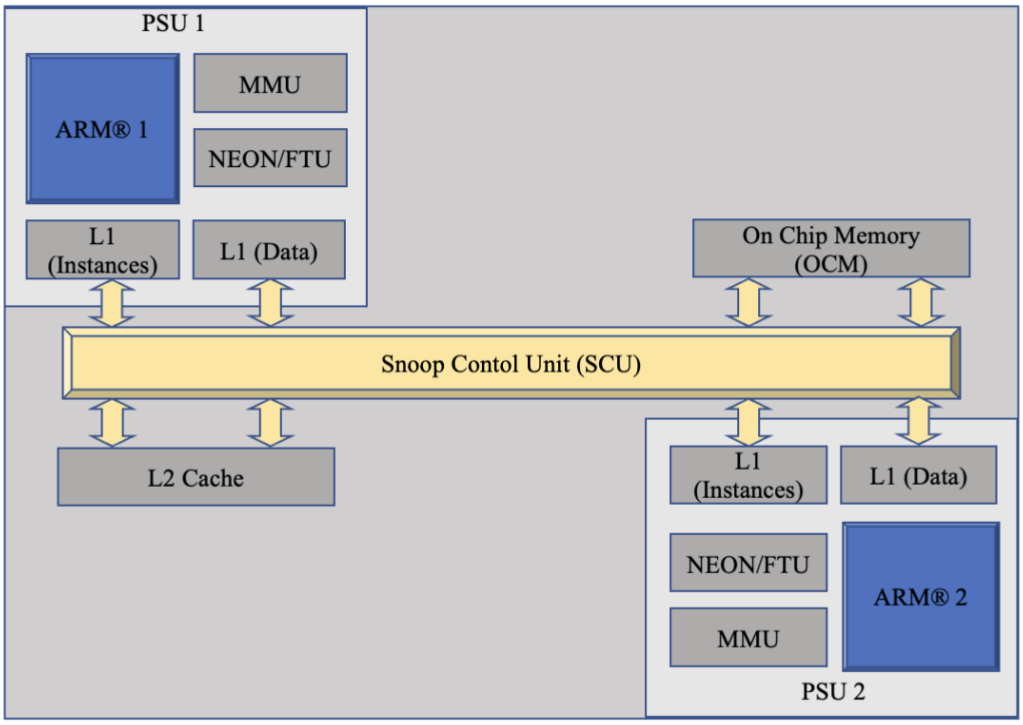


Figure 2.5 Zync-700 device Application Processing Unit (APU)

- NEON: This unit supports Single Instruction Multiple Data (SIMD), one that dramatically enhances DSP and media computational methods on the main ARM processor.
- FPU: This unit gives floating point operation acceleration.
- Level 1 cache: For storing instructions and data, each processor has its own instruction and data caches.
- MMU: It translates data from virtual memory addresses to physical memory addresses.
- Snoop Control Unit (SCU): One of the SCU's primary functions is to interface between processors, L1 and L2 caches.
- L2 cache: This cache is shared by the processor cores and allows them to access the most current variable update.

The Zynq SoC's programmable logic, like that of other FPGAs, is formed of configurable logic blocks (CLBs) with 2 slices. Four look-up tables (LUTs), eight flip-flops (FFs), and a switch matrix are included in each slice. There are also Block RAMs and DSP slices. The PL structure is depicted in Figure 2.6.

- Slice: The resources used to build the combinatorial and sequential circuits are divided into individual slices.
- Look-up Table (LUT): LUTs are used in RAM, ROM, or shift registers to implement a logical design with up to six inputs.
- Flip-flop (FF): A 1-bit register with reset feature is implemented using this sequenced component.
- Switch Matrix: It links the different components within and between the CLBs as well as other components of the PL.

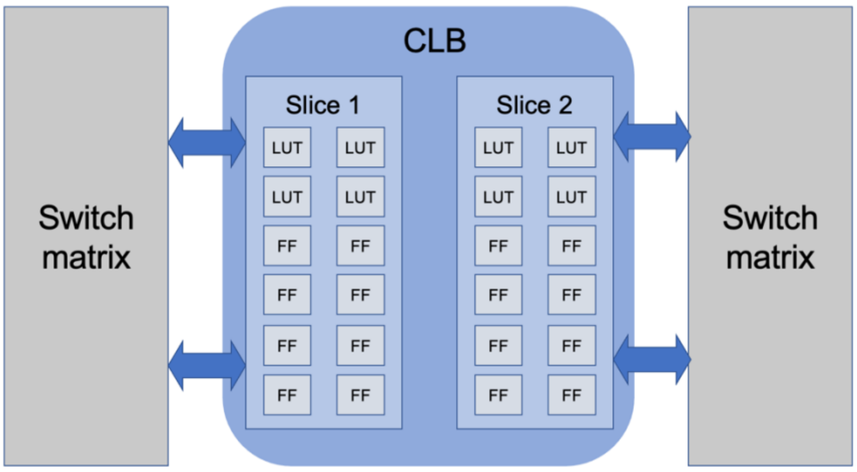


Figure 2.6 Structure of Programmable Logic [10]

The Zynq designing method is equivalent to that of a classic FPGA. Specifying the needs and requirements for the system is the first step. During the system design phase, a process called task partitioning assigns the various tasks (functions) to be implemented in either a PL or PS. This stage is very important. Because of that, missions are delegated for implementation in the most appropriate technique: hardware or software, will determine the desired system's performance.

Following that, testing, and evaluation of hardware in PL and software in PS must be completed. The task for the PL is to choose or decide the functional blocks needed to accomplish the design features, and perhaps to consolidate them as IPs and hook up them suitably. Likewise, the software activity entails writing lines of code that will be executed by the PS. As a consequence, both the wares are merged, and testing will be required to finish the design. Figure 2.7 depicts the Zynq SoC design flow.

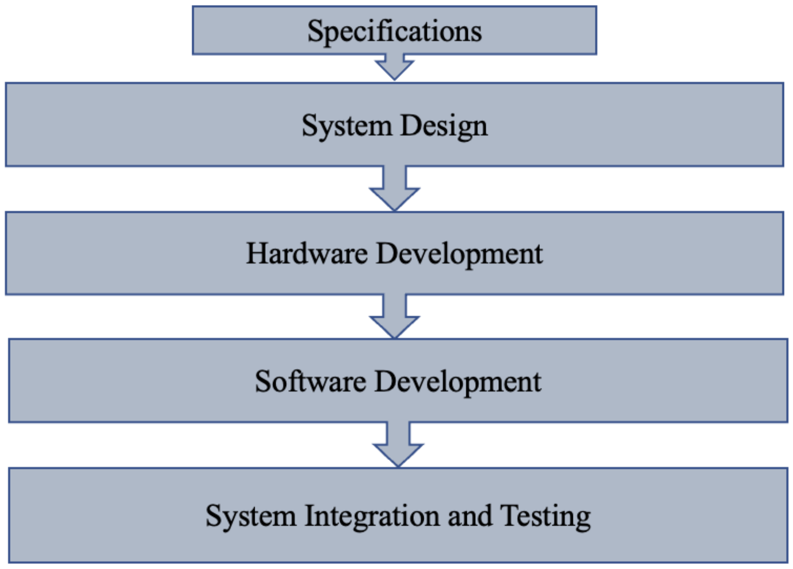


Figure 2.7 Zynq Design Flow

2.2.1.3 DMA IP of Zynq 7000

In fact, the Zynq processing system contains its own DMA channels. But these DMA channels are only for memory-to-memory (M2M) transfers and not for stream-to-memory (S2MM) or memory-to-stream transfers (MM2S). Therefore, it is obvious that if one wants to perform the memory-to-memory transfers, the peripherals must already have a DMA capable interface as Zynq has DMA channels.

The M2M transfers (read or write) data between memory locations. The S2MM transfers take a serial streaming data then sends it to a specific memory location. The MM2S transfers read data from memory and convert it to a serial format. The AXI DMA IP block enables any peripheral with an AXI stream interface, such as FFT IP, to access the system's main memory via S2MM or MM2S transfers. For FPGA developers, DMA IP is used more often for S2MM/MM2S transfers, so I'm using the AXI DMA IP for this project.

2.2.1.4 FFT IP of Zynq 7000

The most popular and efficient computational algorithm for computing the Discrete Fourier Transform (DFT) of a sequence is the Fast Fourier Transform (FFT). The FFT is a powerful tool for analysing and designing a signal in digital signal processing domain. The FFT is, in general, a widely used transform in a variety of digital signal processing applications.

A hardware implementation of FFT can be difficult due to its complex algorithm. As a result, customizable FFT IP cores have been created by various FPGA vendors. These IPs can be introduced to a system design with a few clicks. However, before using such IP cores with confidence, the designer must first understand some relevant information about the FFT's hardware implementation in a system design. There are numerous FFT IP cores available by vendors such as Intel, Lattice Semiconductor, and Dillon Engineering. Since I am using XILINX's FPGA, it is obvious to take a look at Xilinx's FFT IP core in this projec.

2.2.1.5 XADC IP of Zynq 7000

A dual 12-bit ADC with 1 Mega sample per second (MSPS) and on-chip sensors, such as internal temperature are included in the Xilinx analog-to-digital converter (XADC). This ADC is designed for general-purpose, high-precision applications [12]. Figure 2.8 depicts the XADC block diagram.

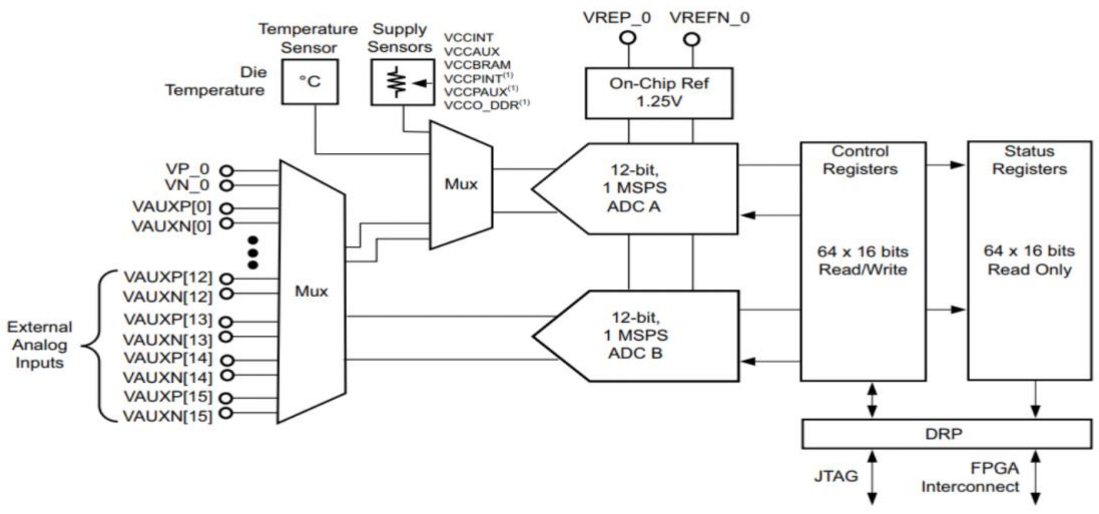


Figure 2.8 XADC block diagram [12]

2.2.1.5.1 Unipolar mode of XADC

The analog signal's amplitude range in this mode is 0-1 V. When you apply 0 V to the XADC, it outputs "000h," and when you apply 1 V to the input, it outputs "FFFh" [12]. Figure 2.9 depicts the transfer function for unipolar mode, whereas Figure 2.10 depicts the unipolar input amplitude range.

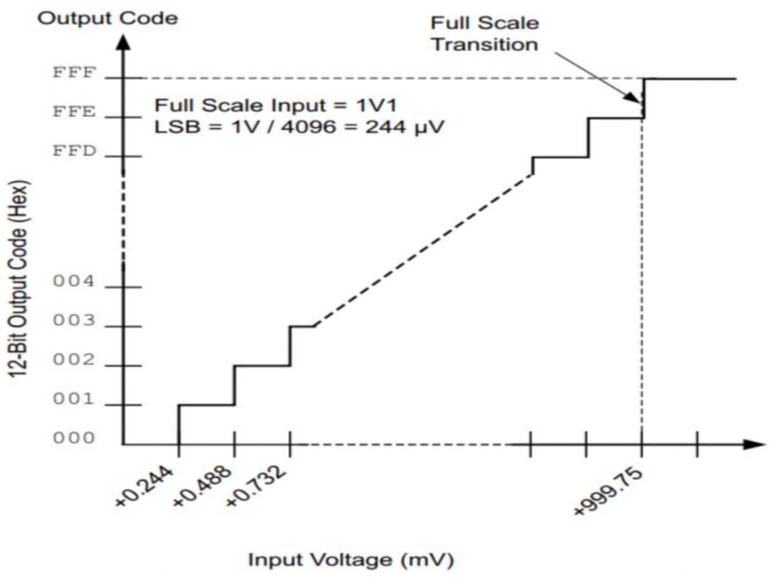


Figure 2.9 Transfer function for unipolar mode [12]

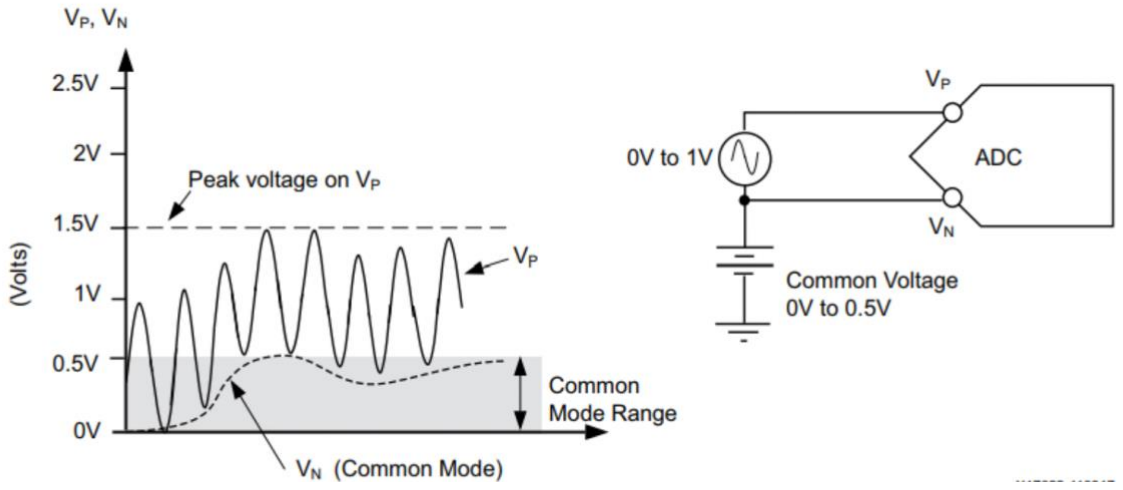


Figure 2.10 Unipolar input range [12]

2.2.1.5.2 Bipolar mode of XADC

The input amplitude range of the analog signal in this mode is between -0.5V and +0.5V. In fact, this mode may be helpful for learning more about the analog input signal while analyzing differential signal types. 0 V is the same as "000h." The value "800h" is assigned when the input voltage is -0.5 V. The input is identified as "7FFh" when 0.5 V is applied. As may be observed, the comparative Vn and Vp are shown using the complementary of two [12]. Figure 2.12 shows the bipolar transfer function, while Figure 2.13 shows the bipolar input amplitude range.

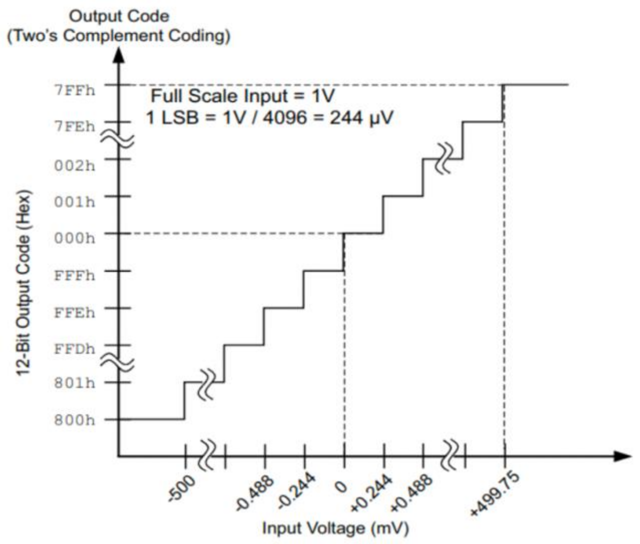


Figure 2.11 Transfer function for bipolar mode [12]

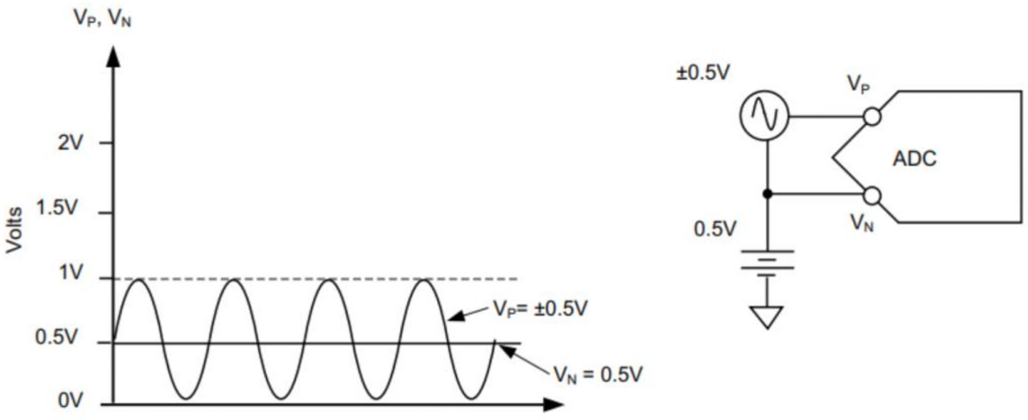


Figure 2.12 Bipolar input range [12]

2.2.1.6 Interconnect IP of Zynq 7000 processor

One or more AXI memory-mapped Master devices and one or more memory-mapped Slave devices are connected by this IP in general. The AXI interfaces are compatible with the AXI4-Lite control register connection subset of the ARM AMBA AXI version 4 specifications. AXI4-Stream transfers are not supported by the Interconnect IP; only memory-mapped transfers are intended. The AXI Interconnect IP can be utilized as a core from the Embedded Development ToolKit (EDK) or as a standalone IP from the CORE Generator TM IP catalog included in the Vivado IP catalogue [13].

2.2.2 Hardware Platform Description of Thermal Control

2.2.2.1 Microcontroller

A microcontroller (MCU) is a microchip that is embedded in the majority of electronic home appliances that people use every day. In an embedded system, it is a small integrated controller that handles a specific operation. A typical microcontroller is a single chip with a processor, memory, and I/O peripherals. A microchip contains timers, communication interfaces, digital and analog I/O pins, as well as timers and analog I/O pins. A processor is in charge of organizing instructions and arranging input/output. The manufacturer has defined the inclusive input/output devices and hardware performance. The basic layout of a microcontroller is as follows.

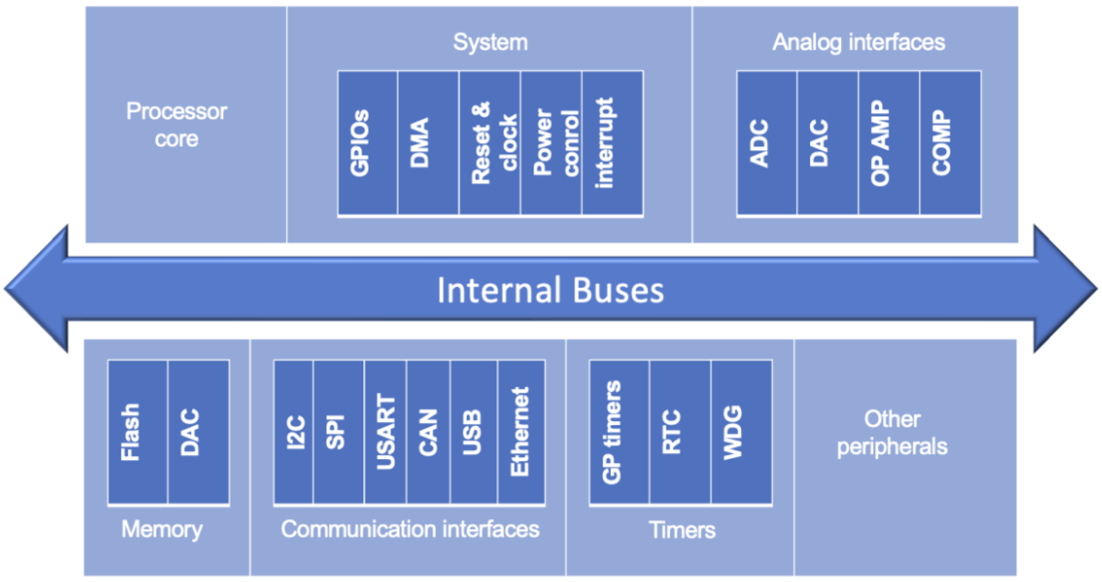


Figure 2.13 A microcontroller's basic layout

As part of the master's thesis, for the thermal control system, STM32 MIU has been chosen due to its characteristics and availability of developing the system in future work.

ARM Cortex-M processor based STM32 MCUs are manufactured by STMicroelectronics. The 32-bit RISC ARM processor cores are intended for low-cost, low-power microcontrollers.
The RISC ARM (Reduced Instruction Set Computer) instruction set divides commands into simple and straightforward instructions designed to achieve specific goals. The STM32 family includes many devices that combine:

- Complete integration
- • The large environment and professional design tools, including free ones, make development easier.
- For some families, extremely high performance is provided by the Arm Cortex-M core and the ST ART AcceleratorTM.
- Capabilities in real time.
- Processing of digital signals.
- Low-voltage and low-power operation Connectivity.
- Graphical elements [14].

Practically, STM32 microcontroller can used in many fields. Such as:

- Control systems for industry
- Human-machine interface devices
- Metering in the cloud
- Motor control

- Medical equipment
- Security and buildings (alarms, access control, etc.)
- Consumer goods (PC peripherals, GPS, gaming, etc.)
- The Internet of Things (IoT) [14]

2.2.2.2 ADC

An ADC converts a continuous analog signal into a digital signal with discrete time and amplitude. Because the conversion requires quantization of such input, some error or noise is introduced. Furthermore, an ADC performs the conversion on a regular basis, rather than repeatedly, by sampling and limiting the reasonable bandwidth of the input signal.

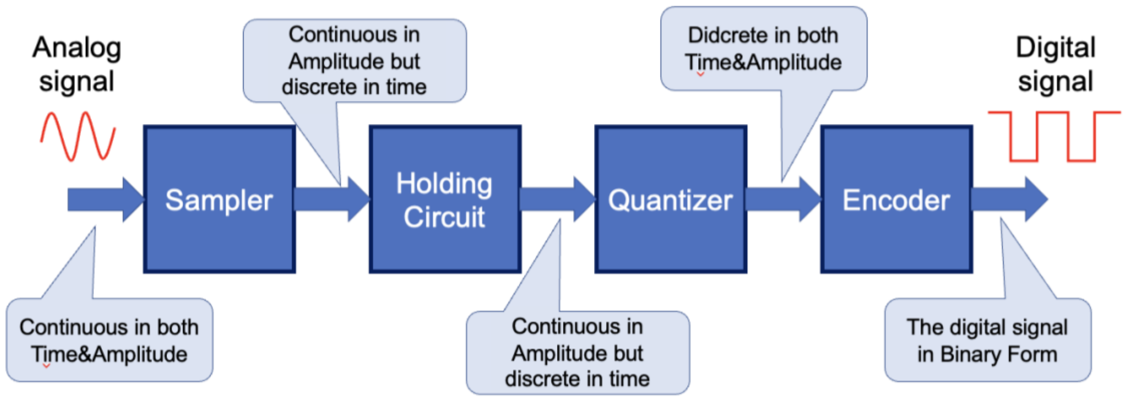

Analog signal
Continuous in Amplitude but discrete in time
Didcrete in both Time&Amplitude
Digital signal
Sampler
Holding Circuit
Quantizer
Encoder
Continuous in both Time&Amplitude
Continuous in Amplitude but discrete in time
The digital signal in Binary Form

Figure 2.14 ADC block diagram

An ADC's signal-to-noise ratio (SNR) and bandwidth serve as general indicators of its performance. An ADC's sampling rate essentially determines its bandwidth. An ADC's SNR is affected by the resolution, linearity, accuracy (how closely the quantization levels match the actual analog signal), as well as aliasing and jitter. An ADC's effective number of bits (ENOB), or the number of bits it typically returns for each measurement that are not noise, is frequently used to describe its SNR. An ideal ADC's resolution and ENOB are essentially comparable. The needed bandwidth and SNR of the digitized signal are taken into account while choosing the ADCs. According to the Nyquist-Shannon sampling theorem, if an ADC runs at a sample rate larger than twice the signal bandwidth, complete reconstruction is feasible. Quantization error is a constraint on even the SNR of a perfect ADC. The effects of the ADC can be disregarded, resulting in a nearly flawless digital reproduction of the analog signal, though, if the SNR of the ADC is higher than that of the input signal.

2.2.2.3 Bridge driver

What engineers use for DC motors to move forward or backward is, generally, a type of electronic circuit named H-bridge, also known as a bridge driver. In our thermal control circuit, changing the polarity of a voltage applied to a load by using H-bridge decides which side of the Peltier cell cools down or warms up.

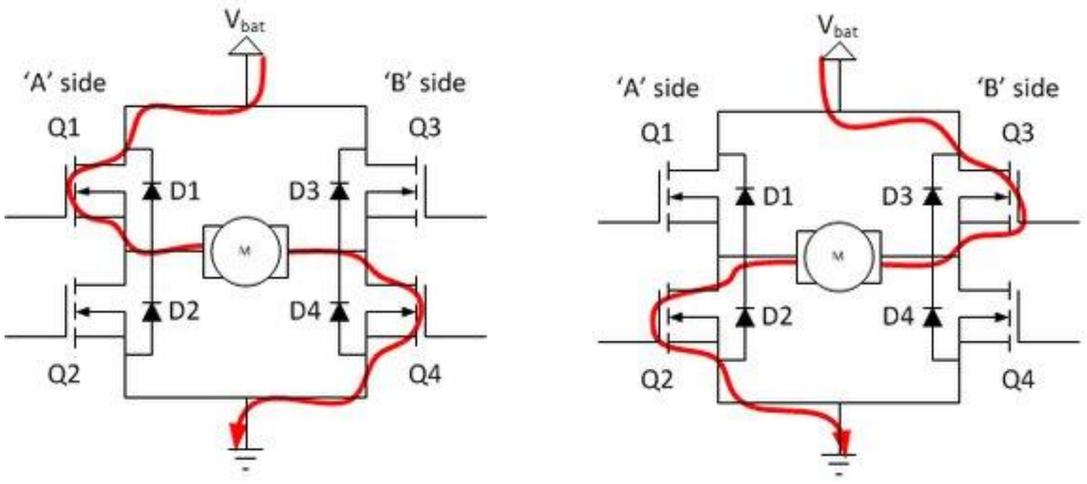


Figure 2.15 Schematics of H-Bridge driver

The polarity of the power of the device (in our case, peltier sell) can be changed by properly arranging the switches. For example, transistors Q1 and Q4 are open in the left circuit of the Figure 2.15 while Q2 and Q3 are open in the right circuit of the Figure 2.15. As illustrated in both images, therefore, the direction of electric current can be changed, and the polarity of a load can be swapped. There are "short" cases that can cause the power source and switches to short connect when switching. As a result, not every combination of switching conditions is safe. The short cases can occur when certain combinations (Q1&Q2 and Q3&Q4) of the transistors are opened.

2.2.2.4 Peltier cell

A Peltier cell is an electrothermal device that can generate cold or hot temperature when it is connected to an electrical current source. This functions as a solid-state heat pump, allowing heat to be transferred from a cold focus to a hot one, generating an opposition to the temperature gradient. When a Peltier cell is activated, one side begins to heat up while the other side cools. There are much more efficient heat generation systems, which is why it is typically used for cooling. Developing a cooling system with a Peltier cell for FPGA based system of drones is simple because the Peltier cell itself is easy to control, and is efficient in terms of weight.

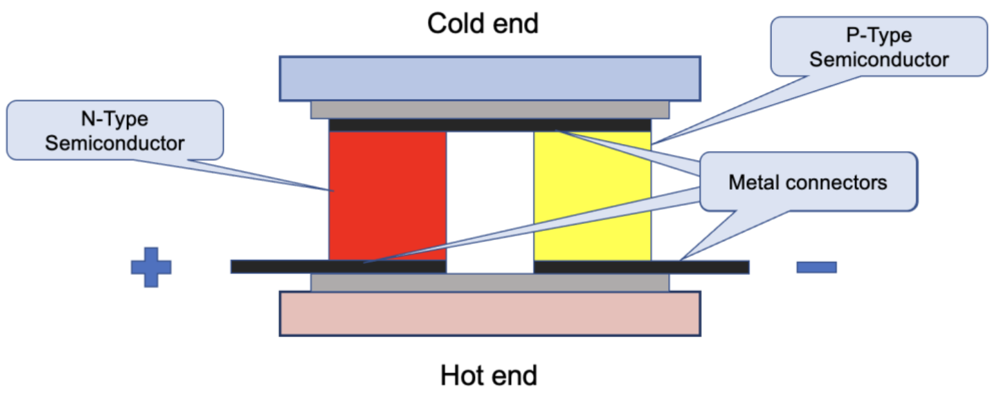


Figure 2.16 Peltier cell

Its operation for thermoelectric cooling is straightforward. It is essentially a sandwich of two metal plates filled with two semiconductors. Typically, these two semiconductors are:

- P-doped semiconductors: Boron, indium, and gallium are some examples.
- N-doped semiconductors: Phosphorus, arsenic, and antimony are examples [15].

One of the ceramic plates cools when a current is passed between them. All the heat lost by one of the ceramic plates is collected by the second, which heats up. Because the heat of the Peltier cell is not affected by the ambient temperature, we can have a liquid that is colder than the ambient temperature. Thermodynamics prevents any air sinks or liquid cooling from lowering the temperature below the ambient temperature. This system enables us to achieve better cooling of our components.

Despite its benefits, its application in the hardware world is quite limited. Because it necessitates high currents, this device is far less efficient than liquid cooling, and forget about air-cooled heatsinks. It is important to remember that the heat produced by the heater must also be drained away. To reduce excessive heat generation, we must install a heatsink.

It is important to remember that the temperature difference of the two sides of the Peltier cell is normally around $70°C$. The cell becomes inefficient at this point, and the cold face begins to heat up. Furthermore, both faces are very close, and heat starts to rise, affecting the cold plate of the cell [15].

A small 40x40cm plate can generate 60W of heat, which must be cooled, adding to the difficulties of requiring heat dissipation systems like heat pipes, heat sinks and fans.

CHAPTER 3 SYSTEM DESIGN

3.1 Overall design flow of FFT implementation

Despite having some experiences regarding embedded systems, FPGA itself is completely new for me. Therefore, I decided to develop my project step by step while familiarizing and studying FPGA. Completing small tutorials and looking at results of them or how the IPs work are my first approaches alongside the reading and understanding whole picture of FPGA. To understand the key features and workflow of all the IPs which I used for my project, I started with testing and configuring them. By doing so, I was able to realize what configuration I must do for each IP and what result or output I would expect from peripherals.

Initial steps I walked through this project are splitting up the whole project in several minor parts which are:

- FFT implementation and verification using FFT IP
- Stream to Memory Mapping transfer implementation and verification using DMA IP
- Real input (analog signal) sampling implementation and verification using XADC IP
- Booting Zynq processor using PetaLinux operation system and installing costume application on PetaLinux.

Finally, once all the steps mentioned above of the project are succussed, all the verified parts of the project's schematics, such as the XADC IP part, the FFT IP part and the DMA IP part, are merged. Because there were verifications for each previous part, the final result of the project was expected to be correct.

Those who want to implement any design must implement a hardware part where internal IPs are chosen, connected and configured, and a software part where an operation system platform and individual software of the system are developed. In the following figure, overall design flow of any system (in our case, every step I walked through including lab works and final FFT system in FPGA) is illustrated.

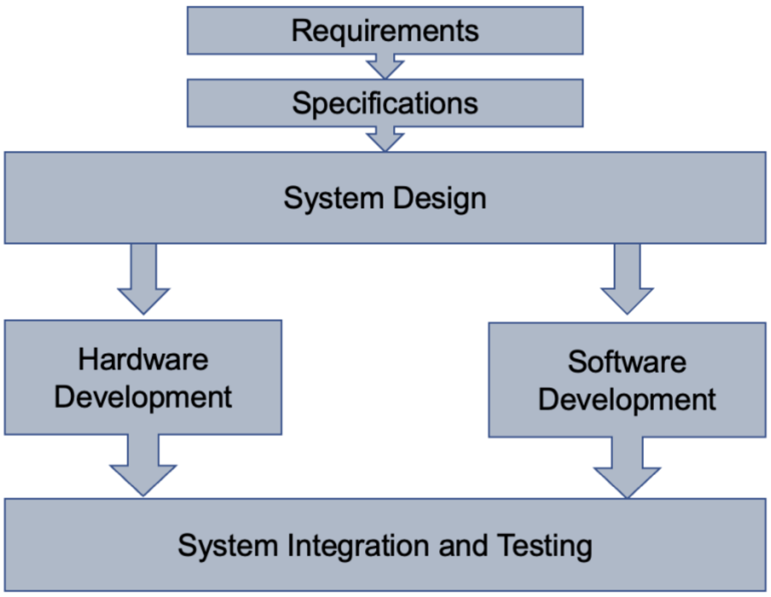


Figure 3.1 Design flow of FFT implementation

In the first 2 steps, one has to decide which kind of hardware or software solutions are needed for the certain design. After that, system designing has to be done. In this phase, if you use any FPGA for the implementation hardware and software parts are assumed separately. In our case, since XILINX's FPGA has been chosen, the hardware part where all the IPs are connected each other properly and configured is done in VIVADO tool, and the software development of the project has been done by using XILINX SDK (System Development Kit). Finally, in the system integration and testing step, the hardware and the software are joined together for whole system and tested together.

3.1.1 Functionality testing and configuration verification of used IPs

3.1.1.1 Understanding and testing DMA IP

As mentioned in 3.1, the first step is to test and verify DMA IP. Direct Memory Access, or DMA as it is more commonly known, is a crucial component of the design of embedded systems because it enables the main memory (DDR) of such systems to be accessed without the CPU being connected, allowing the CPU to carry out other tasks during the read/write cycle to memory. DMA enables the processor to start a transfer to read from or write to the primary internal or external memory. Once the transfer is complete, the CPU generates an interrupt. Until the interrupt skips its corresponding service procedure, this allows the CPU to work on other tasks.

Although the idea behind DMA is straightforward, it can be challenging to picture how to apply it to an FPGA with an embedded processor, such as the ARM-core processor in the Xilinx Zynq SoC. It only becomes worse when you add the extra complication of connecting it from the user space of an operating system like Linux.

The idea to test DMA and verify its function is to examine if the transferred data from DDR by passing DMA (Memory Mapped to Stream) is sent back and received correctly by passing the same DMA (Stream to Memory Mapped). In other words, it was the test to compare what has been sent by Memory Mapped to Stream and what has been received by Stream to Memory Mapped as seen in the figure below.

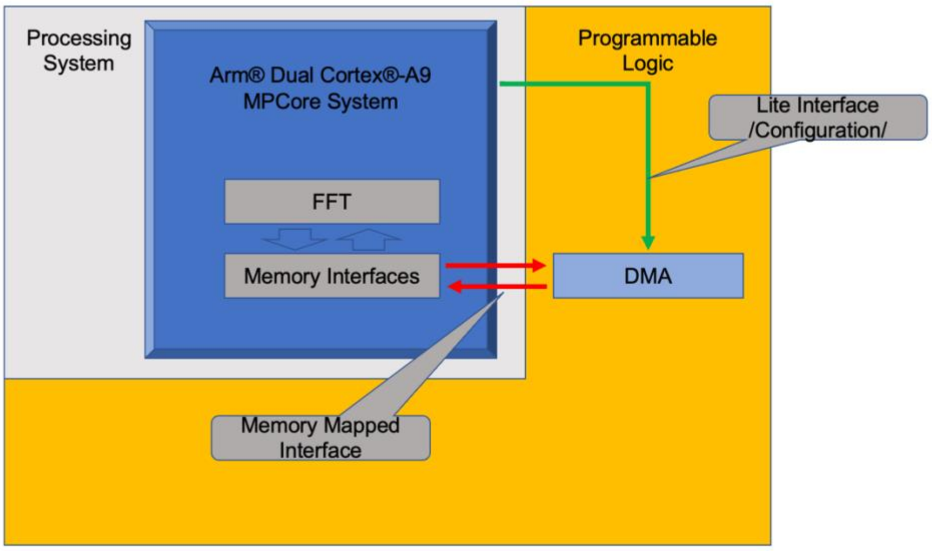


Figure 3.2 Block design of DMA IP verification lab work

This (what I call lab work) has been done initially with bare metal software which is coded in C language. Then as a specification of the project, the bare metal C code has been transferred to an application which is compatible with and installable in PetaLinux.

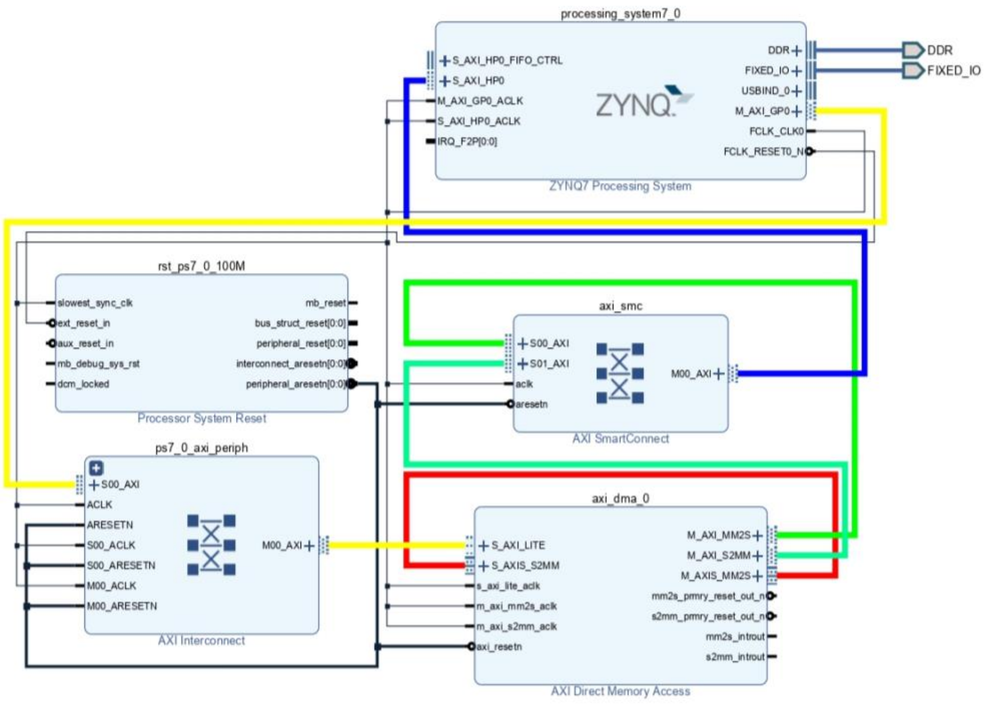
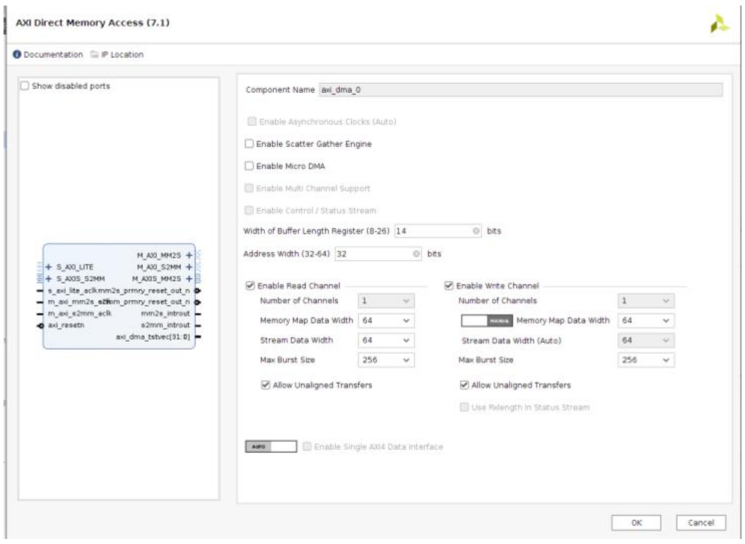


Figure 3.3 Hardware schematics of DMA IP.

Figure 3.3 shows the hardware schematics of DMA IP testing lab. What we have in the schematic alongside the ZYNQ processing system and the DMA are AXI Interconnect IP, AXI Smart Connect IP and Processor System Reset. These IPs are generated and connect automatically after executing "Run connection automation" when you add a DMA IP in your system. "Run connection automation" automatically connects every clock signal, reset signal and most of the necessary connections properly. For example, in Figure 3.3, all the connections except the red one are done by the command. Of course, one can choose the clock or reset source before executing it.


AXI Direct Memory Access (7.1)

Documentation   IP Location

☐ Show disabled ports

Component Name: axi_dma_0

☐ Enable Asynchronous Clocks (Auto)
☐ Enable Scatter Gather Engine
☐ Enable Micro DMA
☐ Enable Multi Channel Support
☐ Enable Control / Status Stream

Width of Buffer Length Register (8-26): 14 bits
Address Width (32-64): 32 bits

☑ Enable Read Channel
Number of Channels: 1
Memory Map Data Width: 64
Stream Data Width: 64
Max Burst Size: 256
☑ Allow Unaligned Transfers

☑ Enable Write Channel
Number of Channels: 1
Memory Map Data Width: 64
Stream Data Width (Auto): 64
Max Burst Size: 256
☑ Allow Unaligned Transfers
☐ Use Rxlength In Status Stream

☐ Enable Single AXI4 Data interface

Figure 3.4 Hardware configuration of DMA IP.

The configuration of DMA IP is illustrated in Figure 3.4. In our case, Scatter Gather Engine is disabled. Therefore, the control register of DMA IP is configured every transaction via its AXI Lite interface from PS. All the data widths of stream and memory map data are 64 bits because data transferred from PS to PL and from PL to PS are 64 bits float complex data type. Slave AXIS S2MM port and Master AXIS MM2S are connected that you can see in Figure 3.3. So, the dataset sent from PS through DMA IP's MM2S goes directly to DMA IP's S2MM port then passed to PS via AXI Smart Connect IP.

In this lab, thanks to the direct connection in red (in Figure 3.3), bunch of float complex type data was transferred from PS to DMA IP (MM2S transfer) and that data received to PS from DMA IP (S2MM transfer). As a result of this lab, data sent from PS via MM2S and data sent from PL via S2MM were exactly the same. Therefore, both S2MM and MM2S are verified, and configuration of the IP is confirmed.

3.1.1.2 Understanding and testing FFT IP

It is crucial to make sure the configuration and performance of IPs are correct. Second and the most important part is testing and verifying FFT IP of XILINX's FPGA. To do so, an artificially generated signal is used as input of FFT IP and output of FFT IP must be compared with calculated output of FFT in bare-metal C (in my FFT IP testing lab, 8 points FFT is used). The block diagram of this lab work looks like:

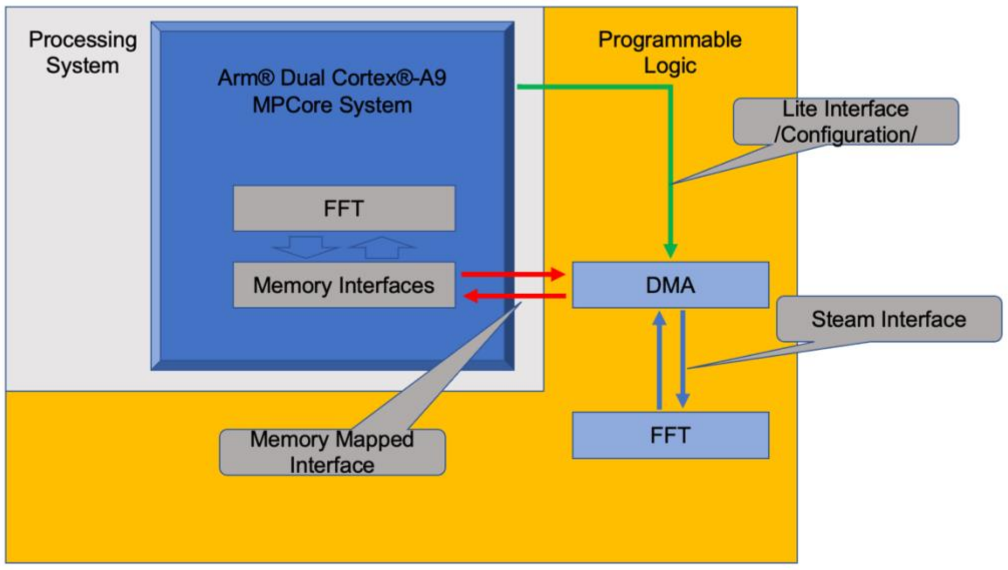


Figure 3.5 Block design of FFT IP verification lab work

As mentioned in previous chapter, the connection between DMA and FFT is steam interface while the connection between processing system and DMA is memory mapped interface. The idea here is to transfer the artificially generated signal to FFT as its input through MM2S by using DMA and FFT solution as its output received to the processing system via S2MM by passing DMA.

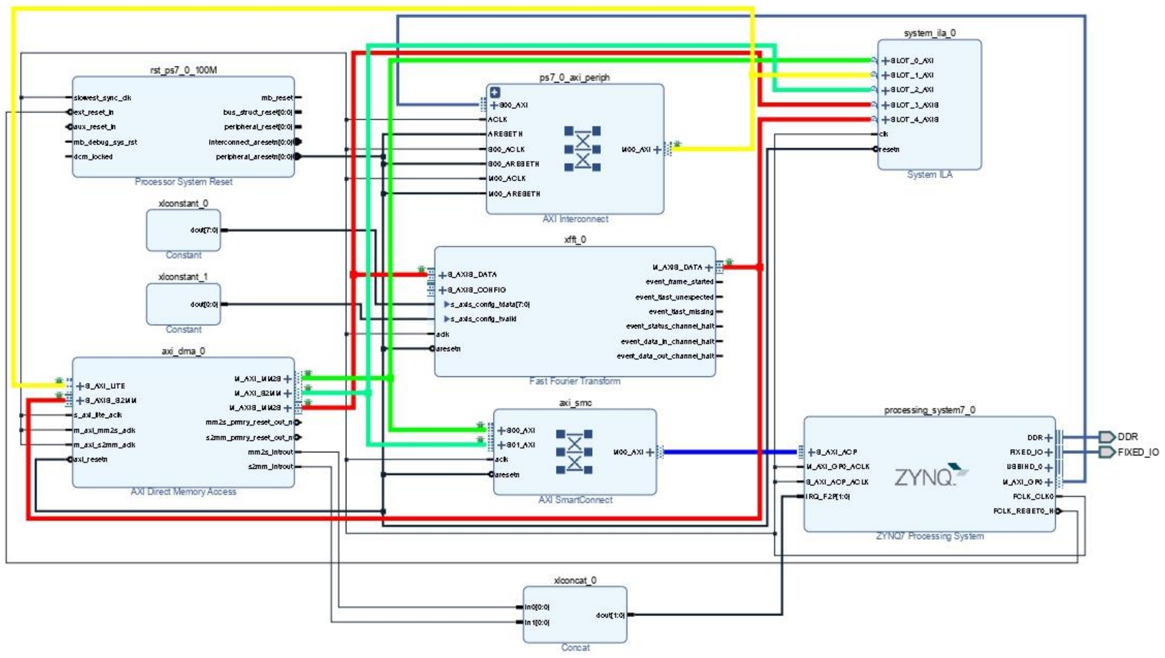
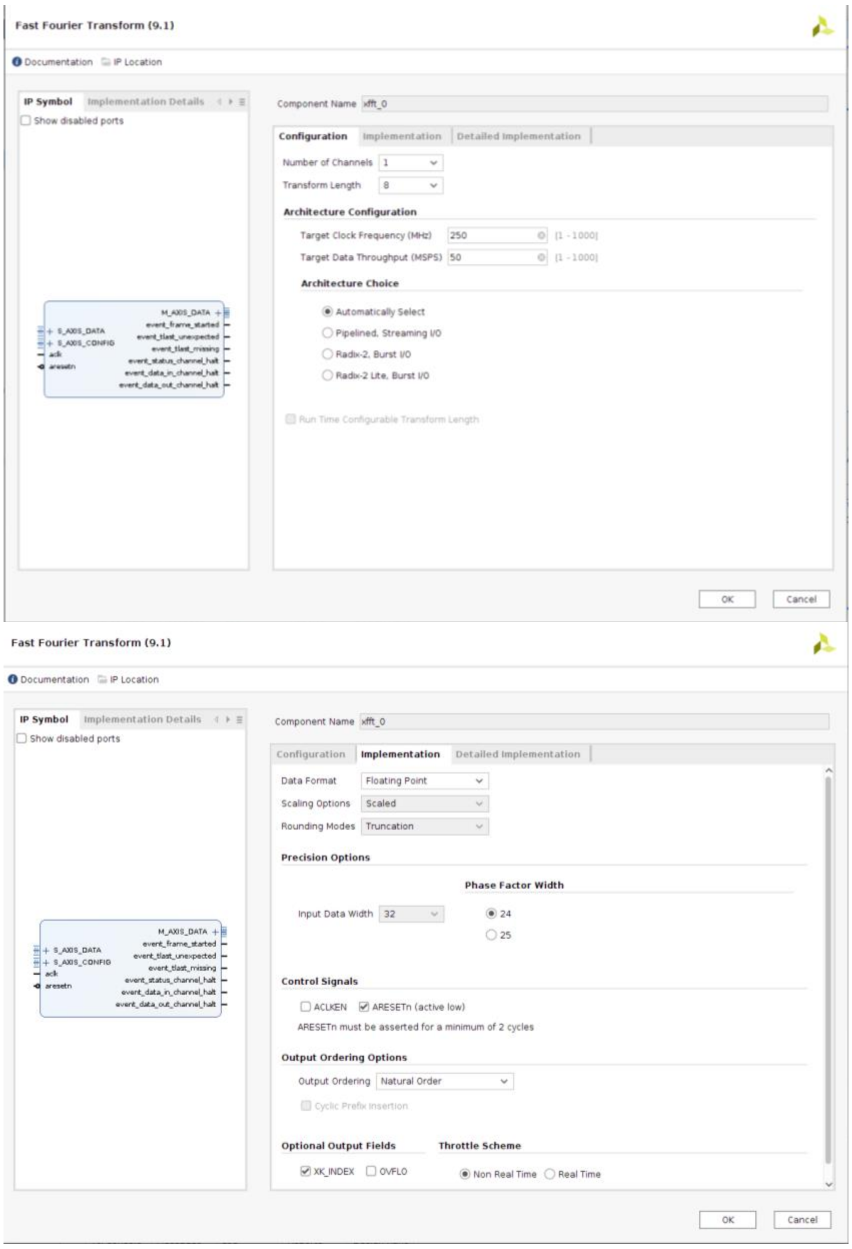


Figure 3.6 Hardware schematics of FFT with input of locally generated signal using DMA IP.

As shown in Figure 3.6, the input of FFT IP (S_AXIS_DATA port) is connected MM2S interface of DMA IP while the output of FFT IP (M_AXIS_DATA port) is connected S2MM interface of DMA IP. Thank to this connection in red, memory mapped complex signal which is generated in PS is turned into stream signal and input the FFT IP. The 8-point FFT of this stream signal is calculated in the FFT IP. Then the output of FFT IP which is stream data transferred to the PS via the DMA IP's S2MM interface. The goal of this lab work is to compare both results computed in the PS and the PL. The result of this lab work is shown in Figure 3.8.



Fast Fourier Transform (9.1)

Documentation | IP Location

Show disabled ports

Component Name: xfft_0

Configuration | Implementation | Detailed Implementation

Number of Channels: 1

Transform Length: 8

Architecture Configuration

Target Clock Frequency (MHz): 250 [1 - 1000]

Target Data Throughput (MSPS): 50 [1 - 1000]

Architecture Choice

[✓ Automatically Select]
Pipelined, Streaming I/O
Radix-2, Burst I/O
Radix-2 Lite, Burst I/O

Run Time Configurable Transform Length

Fast Fourier Transform (9.1)

Documentation | IP Location

Show disabled ports

Component Name: xfft_0

Configuration | **Implementation** | Detailed Implementation

Data Format: Floating Point

Scaling Options: Scaled

Rounding Modes: Truncation

Precision Options

Input Data Width: 32

Phase Factor Width: [✓ 24] 25

Control Signals

ACLKEN [✓ ARESETn (active low)]

ARESETn must be asserted for a minimum of 2 cycles

Output Ordering Options

Output Ordering: Natural Order

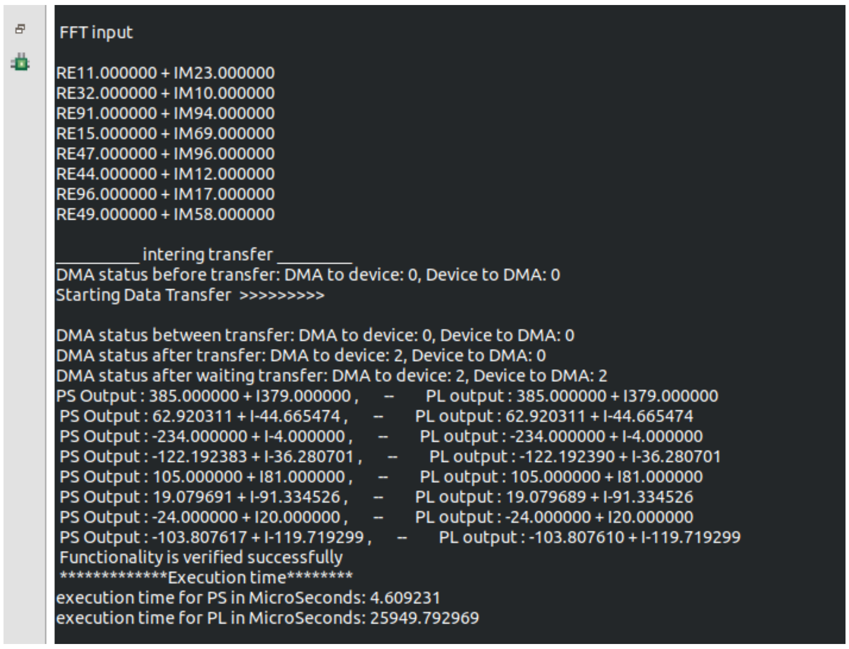
Cyclic Prefix Insertion

Optional Output Fields: [✓ XK_INDEX] OVFLO

Throttle Scheme: [✓ Non Real Time] Real Time

Figure 3.7 Hardware configuration of FFT with input of locally generated signal using DMA IP.

Figure 3.7 shows us the configuration of FFT IP where number of channels is 1, transfer length is 8 (this is because 8-point FFT is computed in the PS), architecture is in auto mode (the proper architecture is chosen automatically), data format is floating point (as the complex signal is transferred in data type of float complex), scaling option is scaled (scaling scheme register has to be set, in 3.6, Constant IP whose value is 0b1111111 in 7 bits is connected to s_axis_config_tdata port of the FFT IP) and rounding mode is truncation.



```
FFT input

RE11.000000 + IM23.000000
RE32.000000 + IM10.000000
RE91.000000 + IM94.000000
RE15.000000 + IM69.000000
RE47.000000 + IM96.000000
RE44.000000 + IM12.000000
RE96.000000 + IM17.000000
RE49.000000 + IM58.000000

_________ intering transfer ________
DMA status before transfer: DMA to device: 0, Device to DMA: 0
Starting Data Transfer  >>>>>>>>>

DMA status between transfer: DMA to device: 0, Device to DMA: 0
DMA status after transfer: DMA to device: 2, Device to DMA: 0
DMA status after waiting transfer: DMA to device: 2, Device to DMA: 2
PS Output : 385.000000 + I379.000000 ,    --    PL output : 385.000000 + I379.000000
 PS Output : 62.920311 + I-44.665474 ,    --    PL output : 62.920311 + I-44.665474
 PS Output : -234.000000 + I-4.000000 ,    --    PL output : -234.000000 + I-4.000000
 PS Output : -122.192383 + I-36.280701 ,    --    PL output : -122.192390 + I-36.280701
 PS Output : 105.000000 + I81.000000 ,    --    PL output : 105.000000 + I81.000000
 PS Output : 19.079691 + I-91.334526 ,    --    PL output : 19.079689 + I-91.334526
 PS Output : -24.000000 + I20.000000 ,    --    PL output : -24.000000 + I20.000000
 PS Output : -103.807617 + I-119.719299 ,    --    PL output : -103.807610 + I-119.719299
 Functionality is verified successfully
 *************Execution time********
execution time for PS in MicroSeconds: 4.609231
execution time for PL in MicroSeconds: 25949.792969
```

Figure 3.8 Output of the FFT IP and computed FFT

Since my code inside the processing system is able to calculate a correct FFT manually which is correct (in Figure 3.8, values defined as PS output), I was expecting the same result (there could be negligible difference) from the output of FFT IP as output of the software (in Figure 3.8, values defined as PL output). The code of this lab work is also transferred to the Linux application. Since both outputs of the PS FFT and PL FFT compared and functionality of the FFT IP is verified as the difference of both FFT output is less than 0.001, I can say this lab work has been done successfully. Therefore, I am sure that my configuration of the FFT is correct, and I expect correct FFT outputs in my FFT system in FPGA.

3.1.1.3 Understanding and testing XADC IP

The use of XADC in this project is simply as sampling a real signal. But to be sure, again, with its performance, verification of its output compared to its input is needed. Before input an analog signal, the output of ADC is verified by using DC voltage (0-1V) as the input. Arty Z7’s dedicated analogy input (in Figure 3.10, Vp_Vn and in ArtyZ7 evaluation board, V_n & V_p connector) for XADC is used.

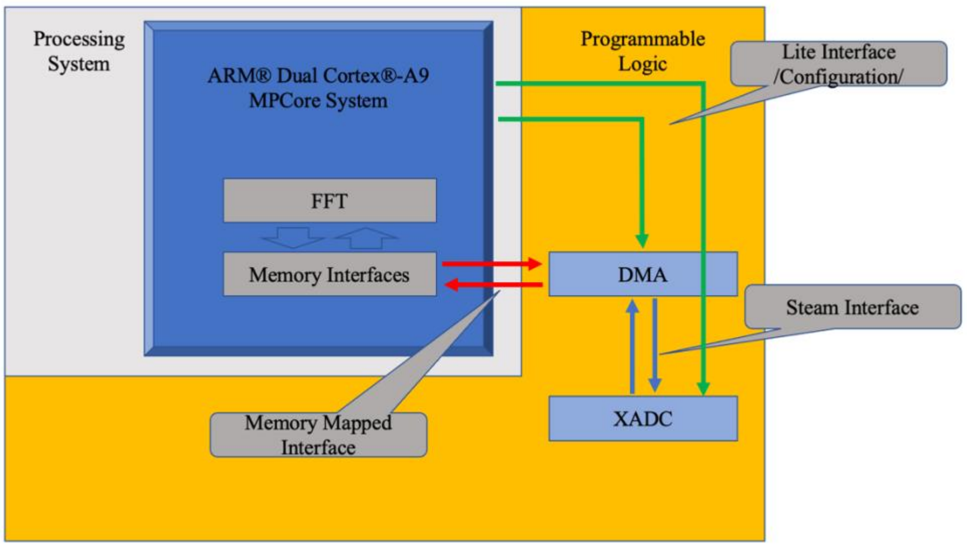


Figure 3.9 Block design of XADC IP verification lab work

As seen in the block diagram in Figure 3.9, either of the DMA IP or the XADC IP are configured via their Lite Interfaces. There are 2 different configuration interfaces in XADC IP (AXI4Lite & DRP). The AXI4Lite is used in this lab but the other one (DRP) is used in another version of the hardware schematic in Figure 3.10 which is bootable in PetaLinux OS.

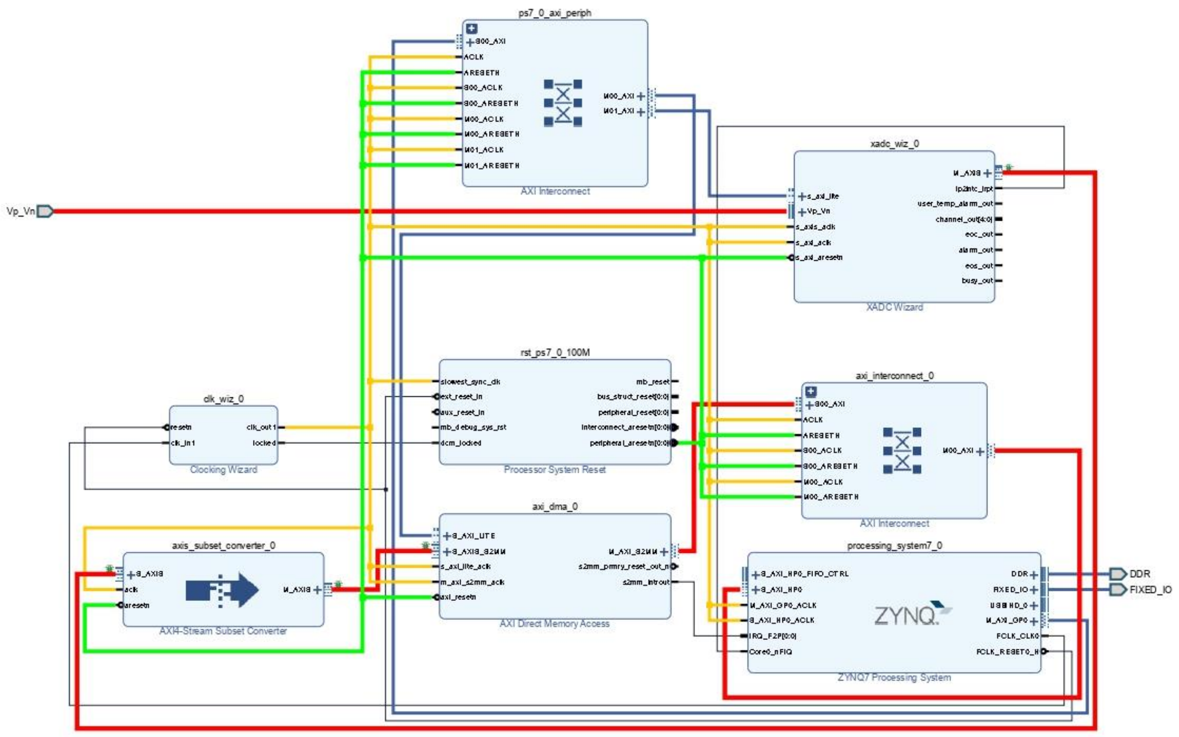


Figure 3.10 Hardware schematics of XADC IP

In the schematic in Figure 3.10, since we are using the DMA IP to steam signal to memory map data, most of the IPs, such as AXI Interconnect IP and System Reset IP, are connected automatically. What is to be careful about is, in this scheme, AXI4 stream output of XADC Wizard (XADC) does not include TLAST signal which is high only at the end of data frame. To transfer streaming output of the XADC Wizard IP to DMA IP, therefore, we must connect it with the AXI4-sream Subset Converter which generates TLAST.

As highlighted in red in Figure 3.10, the real analog signal comes through XADC IP’s dedicated analog input Vp_Vn. In other word, real signal from outside world (a sine signal generated by signal generator or constant voltage) is connected to the connectors named V_p and V_n (ground) whose maximum amplitude is 1V of Arty Z7 board. And the real signal is directly connected to the XADC IP where the signal is sampled, then it is connected to the DMA IP by passing by the AXI4-sream Subset Converter where the TLAST signal is added. Then the streaming sampled values are transferred as memory map data to PS. Finally, in the bare-metal C application, one can read digitalized value of real signal. By doing so, it is possible to verify the output of the XADC by comparing it with mathematically computed sampled value.

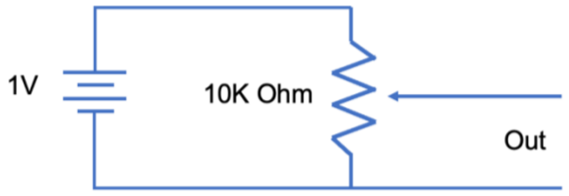
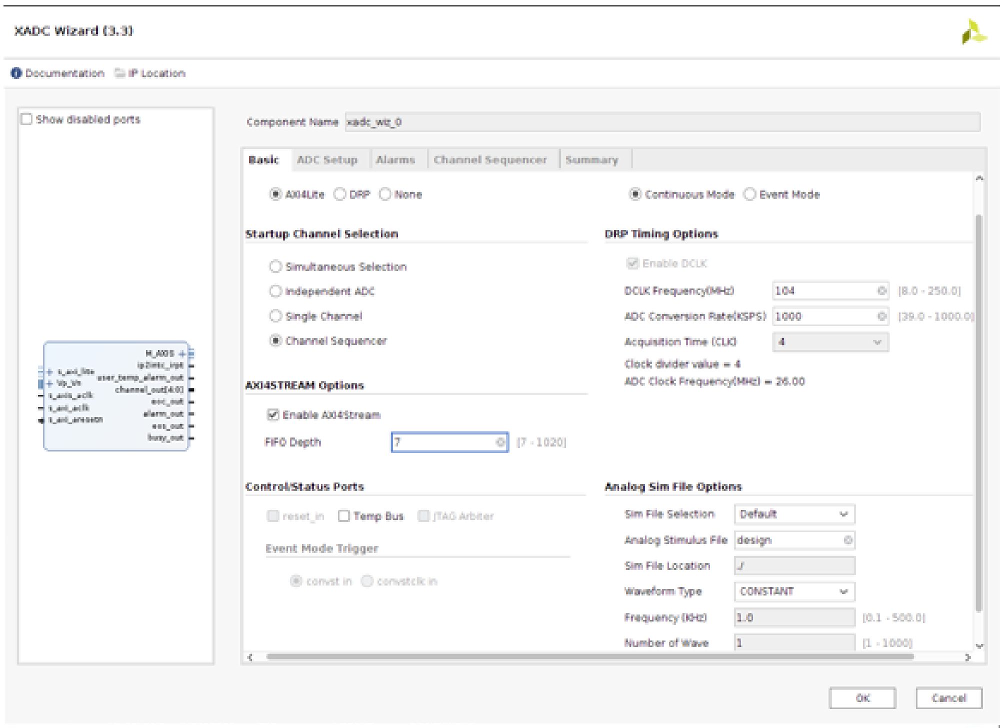


Figure 3.11 Schematic used in the testing of XADC

To be sure with the output of the XADC, its input is initially tested with direct current (DC). In this experiment, a 10K Ohm potentiometer is used as a voltage divider and powered with 1V DC as shown in Figure 3.11. The output of this circuit is directly connected to V_p and V_n headers of Arty Z7. The purpose of this experiment is to measure voltage value at the output of the scheme using a mustimeter and compare it with the values at the output of the XADC IP. By doing so, it is possible to validate the performance of the XADC. For example, the output must be 0x000 when the input is DC 0V, and the output must be 0xFFF when the input is DC 1V. Once this verification has been done, one can be sure that the configuration of the XADC is correct, and the functionality is verified.



Component Name xadc_wiz_0

[✓ AXI4Lite] DRP None [✓ Continuous Mode] Event Mode

Startup Channel Selection

Simultaneous Selection
Independent ADC
Single Channel
[✓ Channel Sequencer]

DRP Timing Options

[✓ Enable DCLK]

DCLK Frequency(MHz)	104	[8.0 - 250.0]
ADC Conversion Rate(KSPS)	1000	[39.0 - 1000.0]
Acquisition Time (CLK)	4	

Clock divider value = 4
ADC Clock Frequency(MHz) = 26.00

AXI4STREAM Options

[✓ Enable AXI4Stream]

FIFO Depth 7 [7 - 1020]

Control/Status Ports

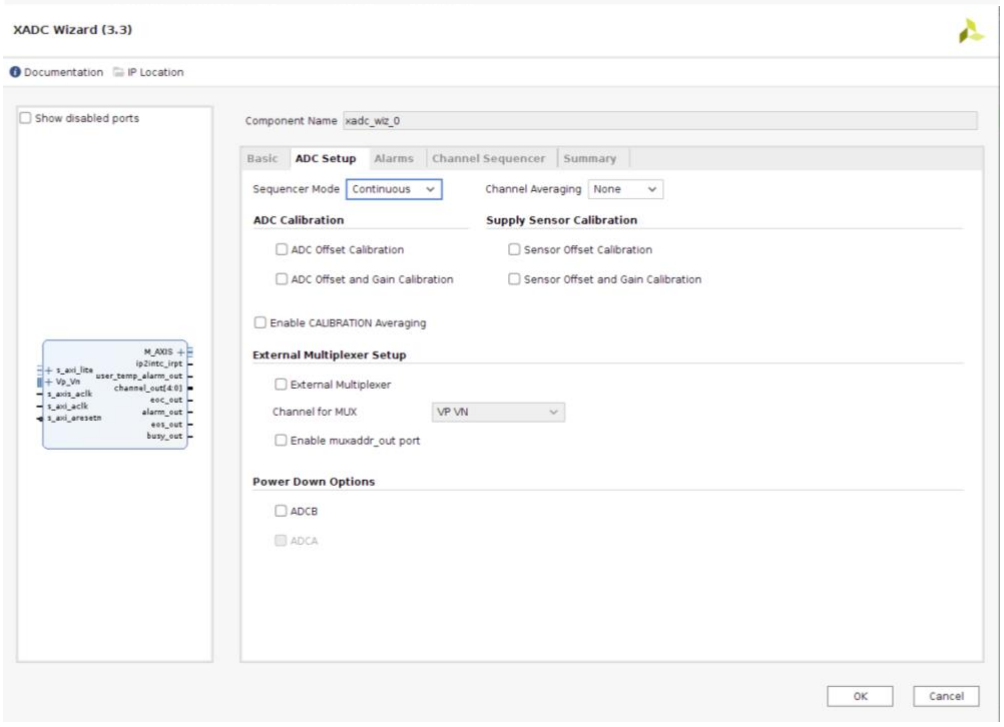
reset_in Temp Bus JTAG Arbiter

Event Mode Trigger

[✓ convst in] convstclk in

Analog Sim File Options

Sim File Selection	Default	
Analog Stimulus File	design	
Sim File Location	./	
Waveform Type	CONSTANT	
Frequency (KHz)	1.0	[0.1 - 500.0]
Number of Wave	1	[1 - 1000]



Component Name xadc_wiz_0

Sequencer Mode Continuous Channel Averaging None

ADC Calibration

ADC Offset Calibration
ADC Offset and Gain Calibration

Supply Sensor Calibration

Sensor Offset Calibration
Sensor Offset and Gain Calibration

Enable CALIBRATION Averaging

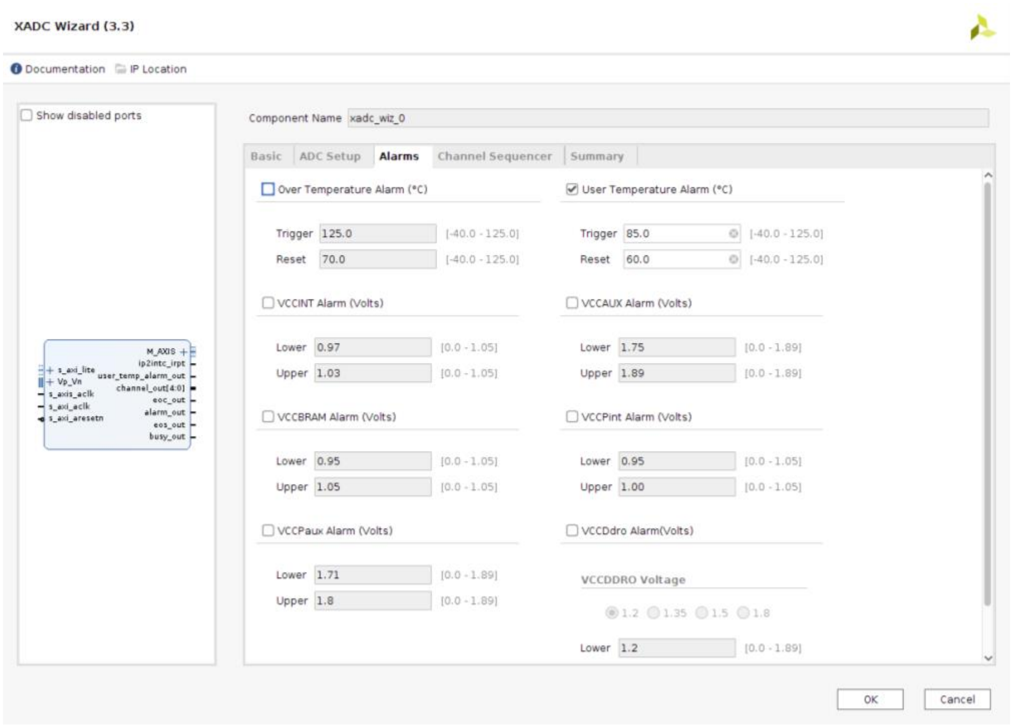
External Multiplexer Setup

External Multiplexer
Channel for MUX VP VN
Enable muxaddr_out port

Power Down Options

ADCB
ADCA

Figure 3.12 Hardware configuration of XADC IP


XADC Wizard (3.3)

Documentation | IP Location

☐ Show disabled ports

Component Name: xadc_wiz_0

Basic | ADC Setup | **Alarms** | Channel Sequencer | Summary

☐ Over Temperature Alarm (°C)

Trigger 125.0 [-40.0 - 125.0]
Reset 70.0 [-40.0 - 125.0]

☑ User Temperature Alarm (°C)

Trigger 85.0 [-40.0 - 125.0]
Reset 60.0 [-40.0 - 125.0]

☐ VCCINT Alarm (Volts)

Lower 0.97 [0.0 - 1.05]
Upper 1.03 [0.0 - 1.05]

☐ VCCAUX Alarm (Volts)

Lower 1.75 [0.0 - 1.89]
Upper 1.89 [0.0 - 1.89]

☐ VCCBRAM Alarm (Volts)

Lower 0.95 [0.0 - 1.05]
Upper 1.05 [0.0 - 1.05]

☐ VCCPint Alarm (Volts)

Lower 0.95 [0.0 - 1.05]
Upper 1.00 [0.0 - 1.05]

☐ VCCPaux Alarm (Volts)

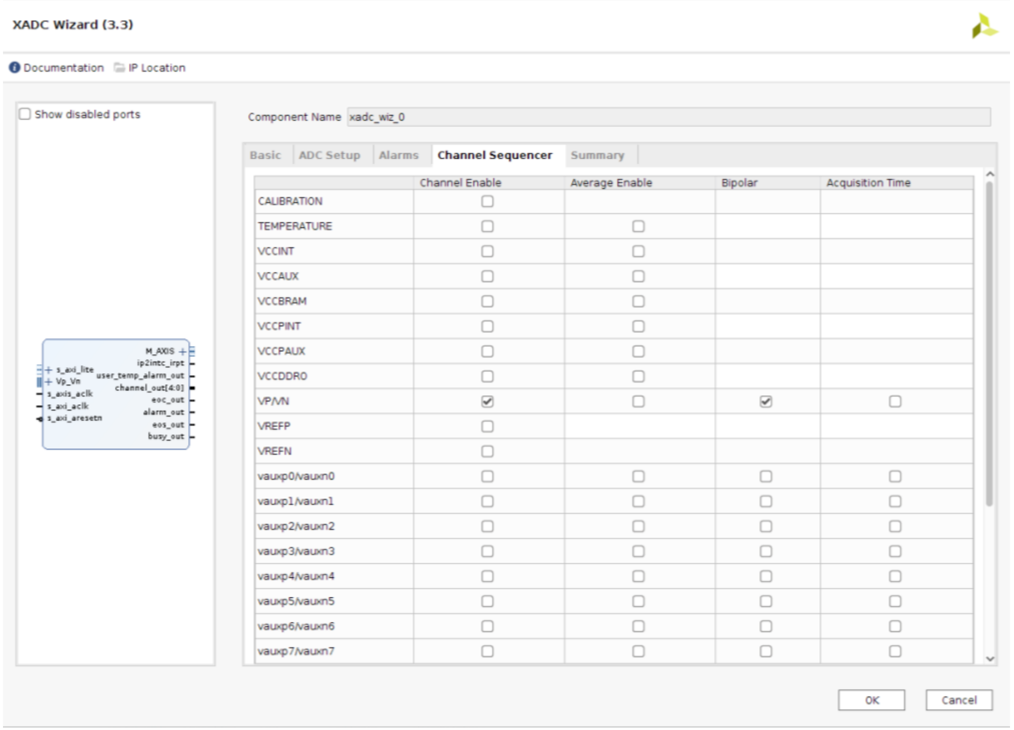
Lower 1.71 [0.0 - 1.89]
Upper 1.8 [0.0 - 1.89]

☐ VCCDdro Alarm(Volts)

VCCDDRO Voltage

[✓ 1.2] 1.35 1.5 1.8

Lower 1.2 [0.0 - 1.89]


XADC Wizard (3.3)

Documentation | IP Location

☐ Show disabled ports

Component Name: xadc_wiz_0

Basic | ADC Setup | Alarms | **Channel Sequencer** | Summary

	Channel Enable	Average Enable	Bipolar	Acquisition Time
CALIBRATION	☐			
TEMPERATURE	☐	☐		
VCCINT	☐	☐		
VCCAUX	☐	☐		
VCCBRAM	☐	☐		
VCCPINT	☐	☐		
VCCPAUX	☐	☐		
VCCDDRO	☐	☐		
VP/VN	✓	☐	✓	☐
VREFP	☐			
VREFN	☐			
vauxp0/vauxn0	☐	☐	☐	☐
vauxp1/vauxn1	☐	☐	☐	☐
vauxp2/vauxn2	☐	☐	☐	☐
vauxp3/vauxn3	☐	☐	☐	☐
vauxp4/vauxn4	☐	☐	☐	☐
vauxp5/vauxn5	☐	☐	☐	☐
vauxp6/vauxn6	☐	☐	☐	☐
vauxp7/vauxn7	☐	☐	☐	☐

Figure 3.13 Hardware configuration of XADC IP

In the group of screen shots in Figure 3.12 and Figure 3.13, all the configuration, needed for the XADC in case of a real-world input signal, is shown. Architecture is AXI4Lite (configuration by AXI4Lite interface), Channel selection is Channel sequencer (with this option, one can input 16 different inputs). Sequencer mode is continuous (XADC IP samples and streams the input continuously). All the alarms are disabled (in fact, there is an interrupt for every alarm). In the Channel Sequencer part, only VP_VN (the dedicated analog input) is chosen and set as bipolar. So that, the input of the XADC is set as Vp_Vn and amplitude between -

0.5V and +0.5V due to its bipolar type. For the timing option of the XADC, clock frequency is chosen as 104 MHz meaning its sampling rate is 1 MSPS. To supply 104 MHz, the Clocking Wizard IP is used as shown in Figure 3.10. Once the output of XADC is satisfied, we can directly use its configuration for a real signal input of FFT system in FPGA.

3.1.1.4 Booting Zynq processor using PetaLinux operation system and installing costume application on PetaLinux.

The Yocto Linux distribution is built upon by a collection of high-level commands known as PetaLinux. Embedded Linux solutions and Linux images for Xilinx processing systems can be modified, created, and deployed using PetaLinux tools. It is designed to increase design productivity, and it integrates with Xilinx hardware design tools (like Vivado) to make it simpler to construct Linux systems for Zynq-7000 SoCs, and MicroBlazeTM. Two steps of the PetaLinux design flow are mentioned below.

Step 1: Creation of the hardware platform
The procedures to follow when using Vivado 2019 to design unique hardware for your board are covered in this step. A Hardware Description File (HDF) must be produced from Vivado. The HDF file is generated automatically after the bit stream file generated from Vivado project. The PetaLinux tools must be aware of the board's underlying hardware system in order to create a specific PetaLinux project for it. Your board's hardware architecture is described in the HDF file. It is necessary if you are going to create a PetaLinux project so that the finished image can appropriately setup the board when it is powered on.

Detailed instructions for installing PetaLinux Tool Chain can be found in the following link location: https://aerotenna.readme.io/docs/using-petalinux-os

Step 2: Creation of the Petalinux project

Using the HDF file generated in the previous phase, in this step we will establish a fundamental PetaLinux project. On a system running Linux OS, several procedures are carried out. You should use a virtual machine with Linux OS and PetaLinux 2018.3 installed if your primary computer is Windows. PetaLinux projects can be created from two sources: Board Support Packages (BSP) and template projects.

Finally, one can copy the bootable image file and root system files to a SD card. Then, if you have a custom application for your system, you can cross-compile it with your Linux OS and Petalinux OS. By doing so, the custom application can be bootable from the embedded OS.

3.1.2 Implementation of FFT system in FPGA

3.1.2.1 Hardware implementation of FFT system in FPGA

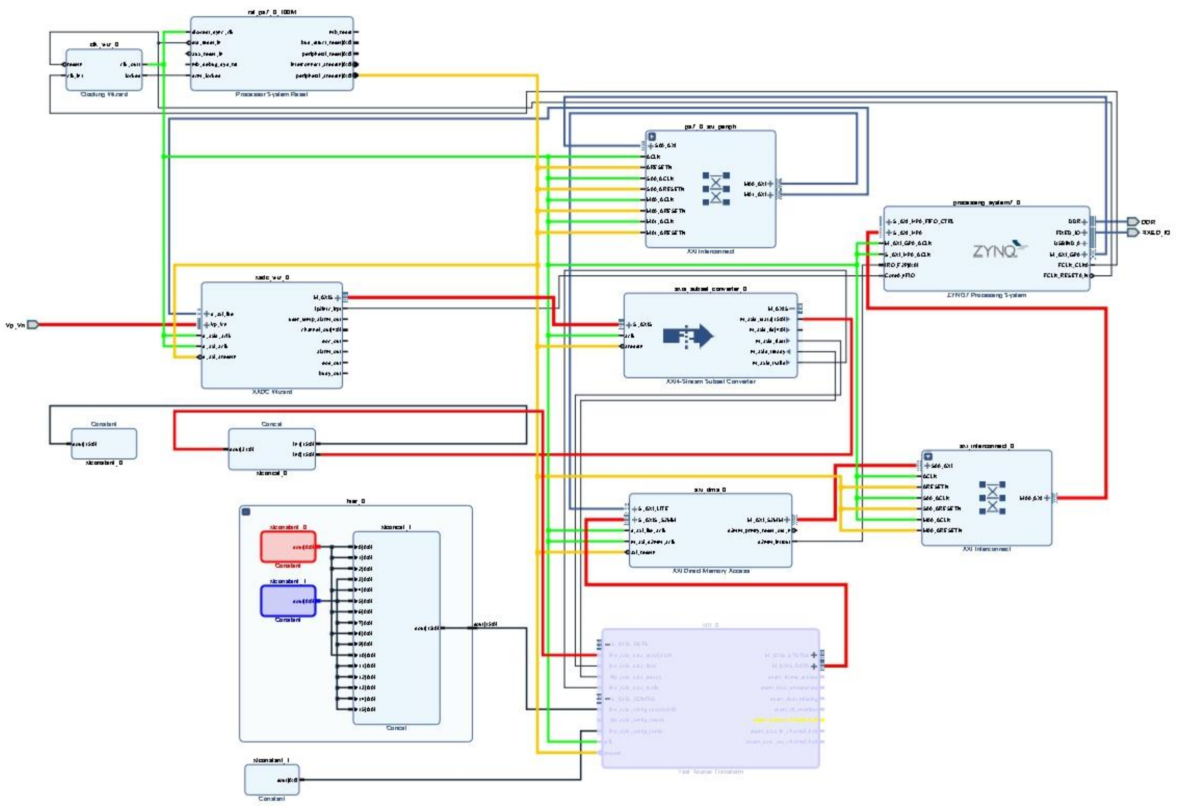
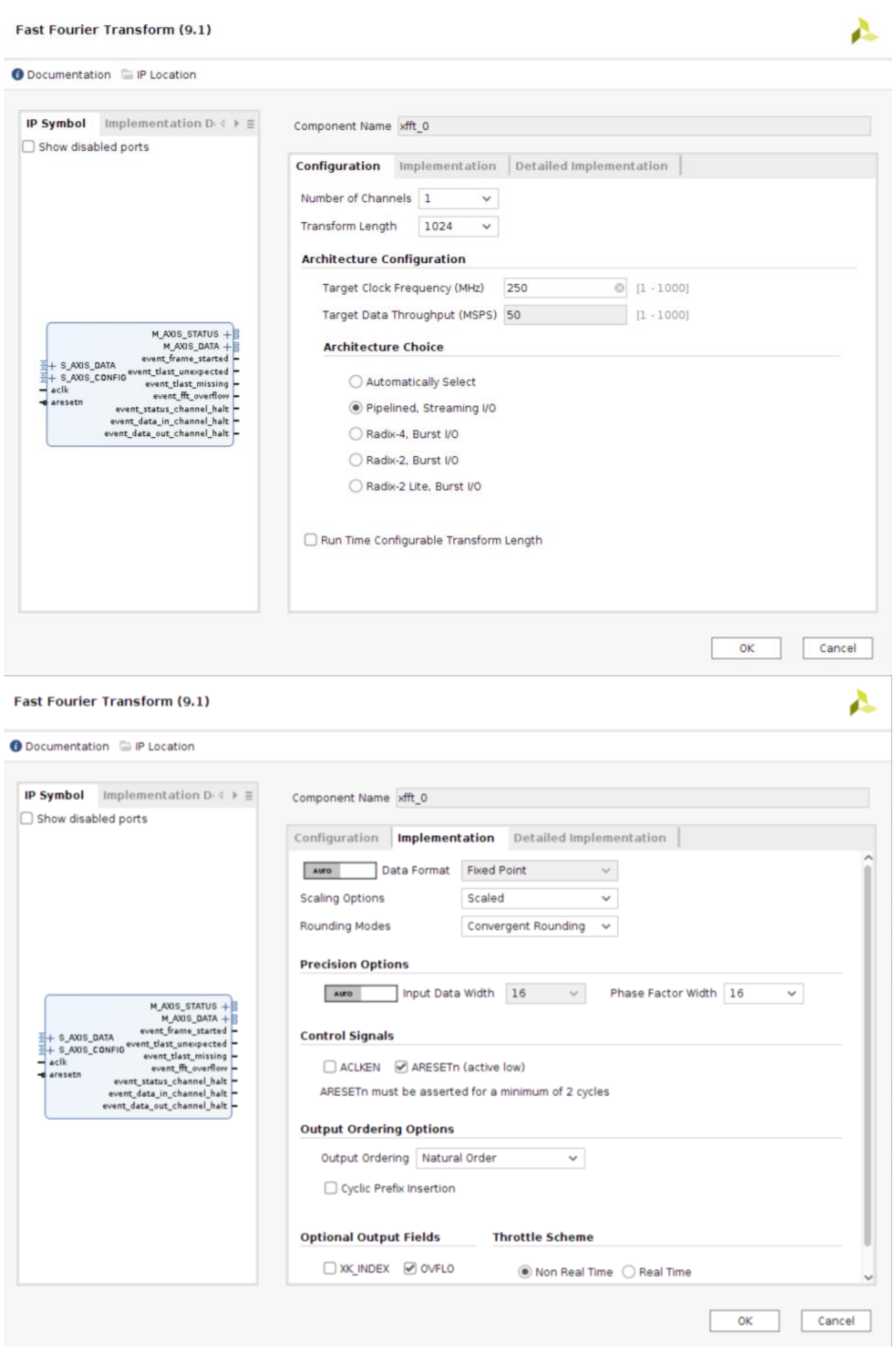


Figure 3.14 Overall hardware schematic of FFT system in FPGA

The complete hardware schematic of FFT system in FPGA is illustrated in Figure 3.14 (more visible version of the schematic is attached in Annex I). Once the functionality testing and configuration verification of the main peripherals, such as the FFT IP, the DMA IP, and the XADC IP, are succeed in the previous lab works mentioned in 3.1.1, the implementation of FFT system in FPGA is straight forward. Obviously, there must be some differences in the configuration of the FFT IP due to different input signal source. In other word, we use locally generated complex signal in 3.1.1.2 as input of the FFT IP while the real analog signal is the input in this final schematic (shown in Figure 3.12) of FFT system in FPGA.

As usual, the main trace of the signal flow is marked as red in the Figure 3.12. The real analog signal comes through XADC IP’s dedicated analog input Vp_Vn. Then this signal directly connects to the input of the XADC IP of which configuration is the same with that of the one shown in Figure 3.11. The XADC IP itself is a 12-bit ADC. The output of XADC is with width of 16 bits and its higher 4 bits are zeros. When this stream passes throughout the AXI4-sream Subset Converter, the TLAST signal is added to the signal. Now, it is the time to input the FFT IP with the sampled stream value of the real analog signal. According to the datasheet and configuration mentioned below, in case of real input signal, imaginary part of the input signal must be zeros. For this purpose, I used the Concat IP which merges several inputs into a single output. The 16 bits of the first input of this Concat IP is connected to the output of the AXI4-sream Subset Converter and considered as the real part of the input signal of the FFT IP. The 16 bits of the second input of this Concat IP is connected to the Constant IP whose constant value is 0x00 and considered as the imaginary part of the input signal of the FFT IP.

Finally, the 1024-bit FFT is computed by the FFT IP and its AXIStream output is connected to the DMA IP to be converted into the memory map data. Then this memory map data is written to the memory.


Fast Fourier Transform (9.1)

Documentation IP Location

IP Symbol | Implementation D

☐ Show disabled ports

Component Name xfft_0

Configuration | Implementation | Detailed Implementation

Number of Channels 1

Transform Length 1024

Architecture Configuration

Target Clock Frequency (MHz) 250 [1 - 1000]

Target Data Throughput (MSPS) 50 [1 - 1000]

Architecture Choice

○ Automatically Select

● Pipelined, Streaming I/O

○ Radix-4, Burst I/O

○ Radix-2, Burst I/O

○ Radix-2 Lite, Burst I/O

☐ Run Time Configurable Transform Length

OK Cancel

Fast Fourier Transform (9.1)

Documentation IP Location

IP Symbol | Implementation D

☐ Show disabled ports

Component Name xfft_0

Configuration | Implementation | Detailed Implementation

AUTO Data Format Fixed Point

Scaling Options Scaled

Rounding Modes Convergent Rounding

Precision Options

AUTO Input Data Width 16 Phase Factor Width 16

Control Signals

☐ ACLKEN ☑ ARESETn (active low)

ARESETn must be asserted for a minimum of 2 cycles

Output Ordering Options

Output Ordering Natural Order

☐ Cyclic Prefix Insertion

Optional Output Fields

☐ XK_INDEX ☑ OVFLO

Throttle Scheme

● Non Real Time ○ Real Time

OK Cancel

Figure 3.15 Hardware configuration of FFT IP with the real analog signal

As shown in Figure 3.15, configuration of the FFT IP is: Number of channel 1, transform length is 1024, architecture choice is Pipelined (continuous stream data processing), Streaming I/O, Data format is Fixed point (sampled real signal), scaling option is scaled, Rounding Mode is Convergent Rounding, Input data width is 16 bits (16 bits for real part and 16 bits for imaginary (16 bits of zeros) part of the input signal), and Output ordering is Natural order.

Three bits from input can be added to output in case of Radix-4 butterfly algorithm, that is why the bit growth needs to be considered to prevent the data overflow. The same effect happens in Pipelined Streaming I/O designs of FFT.

The highest amount of bit expansion in the stage determines how many output data bits must be shifted to the right after it is complete. Data is only shifted in conditional block scaling if bit growth happens. The entire block of data is moved to the right and the block exponent is changed if one or more outputs increase.

That is why SCALE_SCH register of the FFT IP must be configured. In the Pipelined Streaming I/O architecture, overflows are totally avoided by the conservative schedule SCALE_SCH = [10 10 10 10 11]. The last group only has one stage when the point size is not a power of four, and the higher bit growth for the last group is one bit. Consequently, the scaling schedule's two MSBs can only be 00 or 01 [16].

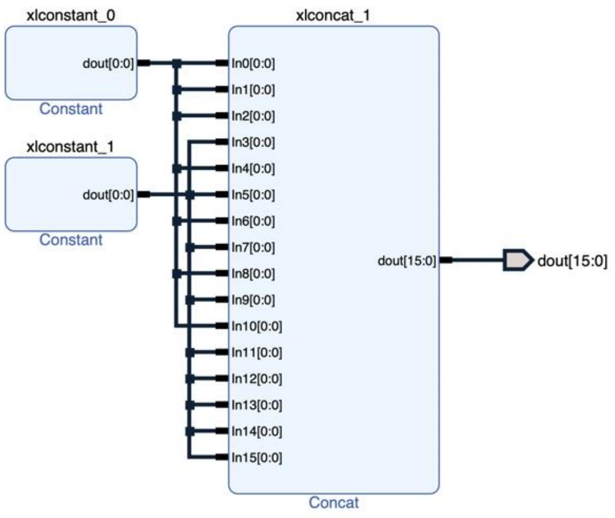


Figure 3.16 Scaling schedule of 1024-bit FFT with Pipelined Streaming I/O architecture

The scaling schedule mentioned above can be done by writing conservative schedule to the SCALE_SCH register. To do so, we must write SCALE_SCH = [10 10 10 10 11] by writing constant values to the s_axis_config_tdata port of the FFT IP as shown in Figure 3.16.

3.1.2.2 Software implementation of FFT system in FPGA

In the software part, configuring the XADC IP and FFT IP by using their AXILite interfaces. Once these configurations are done, all we have to do is wait for the receiving data which is the output of the FFT IP in the single precision data type.

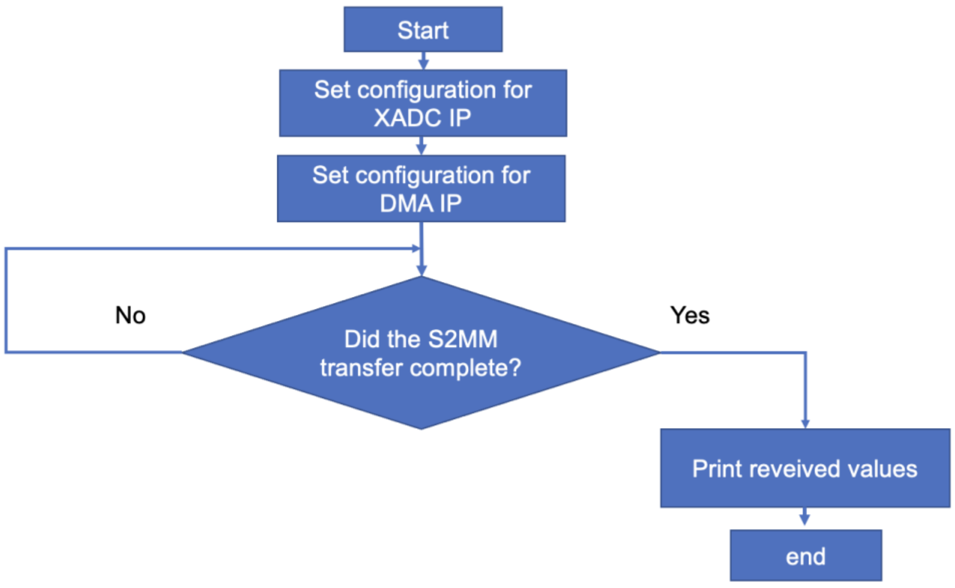

Start
Set configuration for XADC IP
Set configuration for DMA IP
No
Yes
Did the S2MM transfer complete?
Print reveived values
end

Figure 3.17 Algorithm of the software of FFT system in FPGA

The algorithm of the software is shown in Figure 3.17. Since most of the configuration of IPs are completed in the hardware part, some minor configurations of the IPs, such as a transfer word length for the DMA IP and resetting the IPs, are done in the bare-metal application. Then once the IPs are configured successfully in both hardware and software, the only thing to do in the software is to wait for the 1024-point FFT output of FFT IP. Finally, one can use the FFT output for filtering a signal by using its frequency components. For example, we can get rid of an interference by omitting the frequency component corresponds to the interference signal from the FFT output and input that remaining bunch of data to Inverse Fourier Transform (IFFT). In fact, the FFT IP itself has a mode to compute IFFT.

3.2 Overall design flow of “Thermal control” implementation

Implementation of the thermal control is straight forward as of any other embedded systems. Deciding a solution, completing scheme design, drawing layout of PCB, soldering and testing are the all the steps which I walked through its implementation.

3.2.1 Implementation of “Thermal control”

3.2.1.1 Hardware implementation of “Thermal control”

From choosing electronic components to designing a schematic, all the steps are explained in this section. Choice of the components, these days, are highly depending on its availability in stocks of the vendors rather than inventors’ decisions due to a shortage of such component in the market under COVID situation. As most of the engineers are experiencing, some components I had chosen for the thermal control system had not been available in the market. Therefore, during the first phase of this project, it costed me spending several days to choose the components, check the technical characteristic and availabilities because once one of the components is changed, the whole schematics may be changed. Thanks to the variety of the vendors in the market, satisfied elements of the schematic has been chosen.

After decided every single component of the thermal control, the schema designing task is carried out. In this step, the software used was KiCAD which is free and can be used for professional purposes. Designing the schematic for thermal control is straight forward as the purpose of the system is to control a Peltier cell, which must be fixed in the box to balances the temperature inside, based on surrounding temperature. As seen in the Figure 3.18, a microcontroller takes control of whole system, for example it gets digitized values of temperature from the ADC through its SPI interface and controls Peltier cell through the H-bridge by using a software PID which controls values of PWM. An analogy digital converter (ADC) converts an analog signal to digital signal. Alongside those electronic components, what we need for overall system are thermal sensors to sense the temperature and the Perltier sell to cool or heat inside of the box.

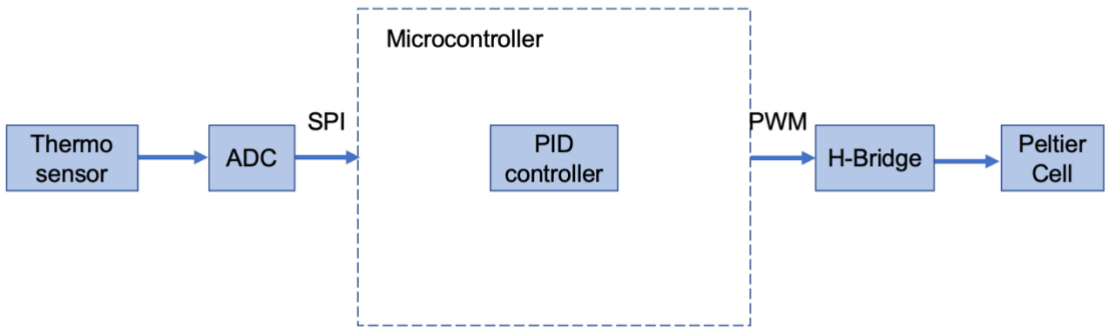


Figure 3.18 Block diagram of the hardware implementation of "Thermal control"

Overall hardware schematic of the Thermal control is attached in Annex II. Designing a layout of PCB for the electronic system is what every inventor does after completing schematic design. This step plays an important role in the electronic system design because this is when one can minimize the noises by introducing interesting solutions, like placing the coupling capacitors correctly and managing the planes (layers) or traces efficiently, for example, the coupling capacitors must be place as near as possible to the dedicated pins, and ground planes of analog and digital sources could minimize some noises as separately generated in the other sides of the PCB. Another important aspect during phase is to create a new footprint for certain, or for your custom elements such as a heatsink.

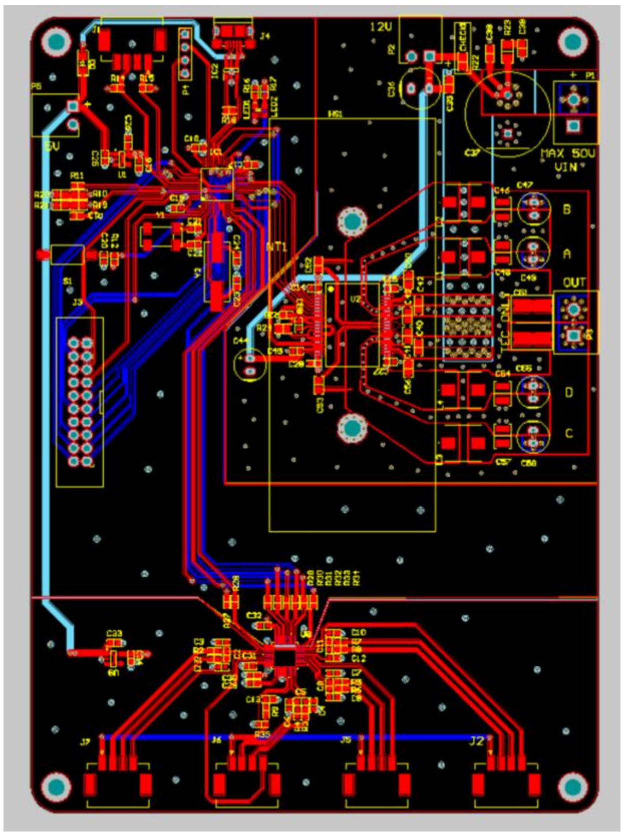


Figure 3.19 Layout design of PCB

For this project, the PCB has been done with four layers, two for tracing, one for the ground plane and one for the power plane. Some empty pins of STM32 have been designed for possible development of the system.

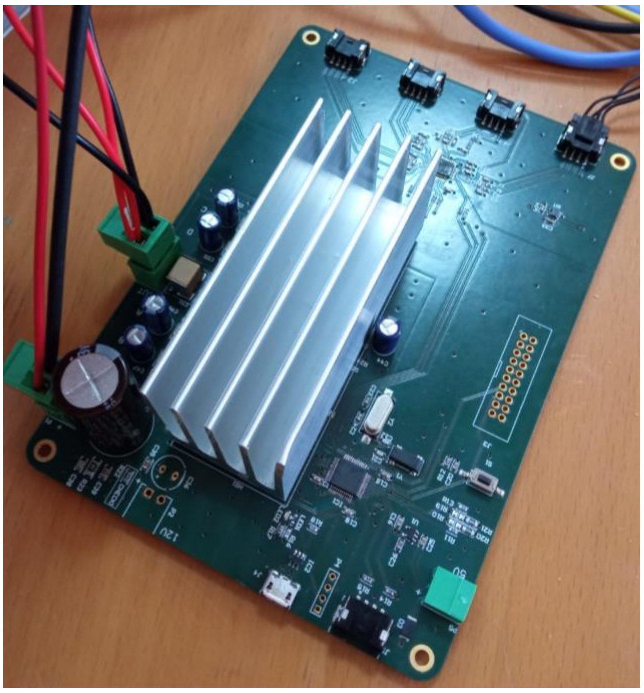


Figure 3.20 The final electronic board of the "Thermal control"

Once the soldering is completed successfully, I moved to the software part of the project before the last step which is testing the whole system.

3.2.1.2 Software implementation of "Thermal control"

In the software part, configuration of the other devices such as the ADC and H-bridge is done in STM32 by using their SPI interfaces.

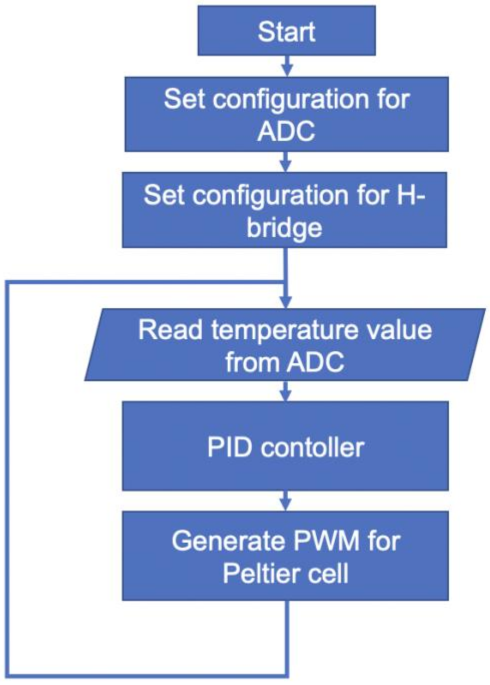

Start
Set configuration for ADC
Set configuration for H-bridge
Read temperature value from ADC
PID contoller
Generate PWM for Peltier cell

Figure 3.21 Algorithm of the software of "Thermal control"

The algorithm of the software is shown in Figure 3.21. What it does after configuring the control registers of the other devices is to read output of ADC which is the digitized value in 16 bits of the thermal resistor. Then the PID controller designed in STM32 controls the value of PWM for Peltier cell.

The use of PID controller here is to keep the temperature of the other system around the chosen reference temperature while minimizing the energy utilization of the Peltier cells. Thanks to the mathematical model of PID controller and its recursive operation, it reduces the fluctuation of the temperature to its minimum level.

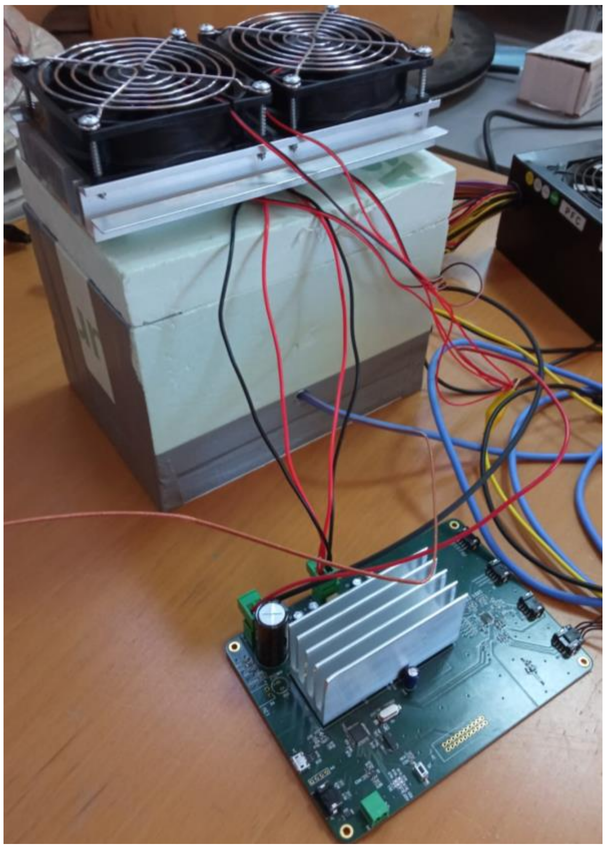


Figure 3.22 Testing of the “Thermal control”

In the figure 3.22, the 2 Peltier cells with fans are connected to Thermal control board and attached to the box made of an isolation material. The thermal resistor or thermistor (1 out of 4 possible headers dedicated to thermistor is used) is put inside the box through a small hole.

As for the testing of the board, I completed an experiment where I wanted to know how much time a PID controller which is designed in STM32 microcontroller took to stabilize the temperature inside the box made of a thermal isolation material. I put a cup with 0.2 litres water ($25°C$) inside the box and recorded temperature (through UART port of PC) value using the thermometer every seconds. In this experiment, I wanted to examine if the tunning of PID controller is correct or not.

CHAPTER 4 RESULTS

4.1 Results of FFT system in FPGA

After the hardware and software implementation, it is time to show the results of the FFT system in FPGA and to verify the result with mathematical computation. The real analog signals are generated by a signal generator named FY6800 of KMOON. Then these analog signals are directly connected to the dedicated analog input port, V_p & V_n, of Arty Z7 board. To confirm the result, two separated experiments are done with two different signals which are a sine wave and a square wave.

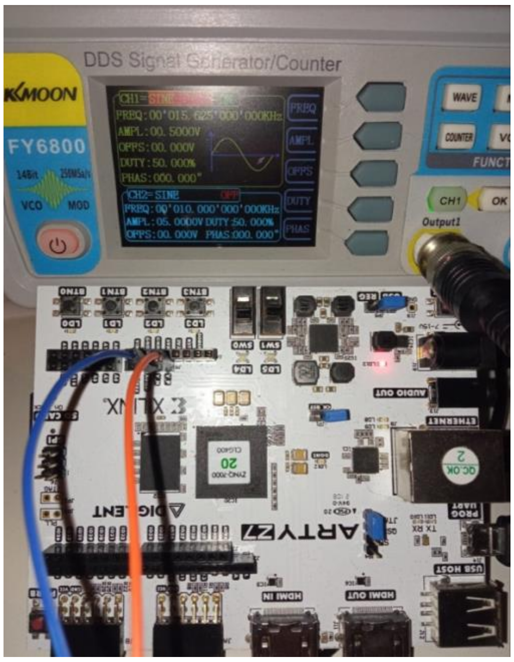

DDS Signal Generator/Counter
KMOON
FY6800
CH1=SINE
FREQ:00'015.625'000'000KHz
AMPL:00.5000V
OFFS:00.000V
DUTY:50.000%
PHAS:000.000°
CH2=SINE
FREQ:00'010.000'000'000KHz
AMPL:05.0000V DUTY:50.000%
OFFS:00.000V PHAS:000.000°
ARTY Z7
DIGILENT
XILINX

Figure 4.1 Connection of the analog signal (sine wave)

In the Figure 4.1, physical connection of the analog signal and its configuration. As displayed in the signal generator's display, the signal type is the sine wave, frequency is 15.625 KHz due to the bin size (976.5625 Hz). The amplitude of the signal is 0.5V because in the hardware part we use the XADC IP in the bipolar mode meaning input is between -0.5 and +0.5.

In the overall system, we have following value to compute FFT.

- Sampling rate: $f_s = 1MSPS$ (according to the configuration of the XADC IP in 3.1.1.3)
- FFT size: $N_{fft} = 1024$ (according to the configuration of the FFT IP in 3.1.2)
- Input signal: $f_{in} = 15.625\ KHz$

From the known values above, the frequency resolution can be calculated as follows:

$$\Delta f = \frac{f_s}{N_{fft}} = \frac{1MSPS}{1024} = 976.5625\ \mathrm{Hz} \qquad (4.1)$$

Therefore, the frequency bin size for each bin of FFT IP is 976.5625 Hz. By using this bin size, we can calculate the bin location which corresponds to the input signal.

$$k_f = \frac{f_{in}}{\Delta f} = \frac{15.625\ KHz}{976.5625\ \mathrm{Hz}} = 16 \qquad (4.2)$$

The meaning of the value above is where we must expect a peak in output of the FFT IP.

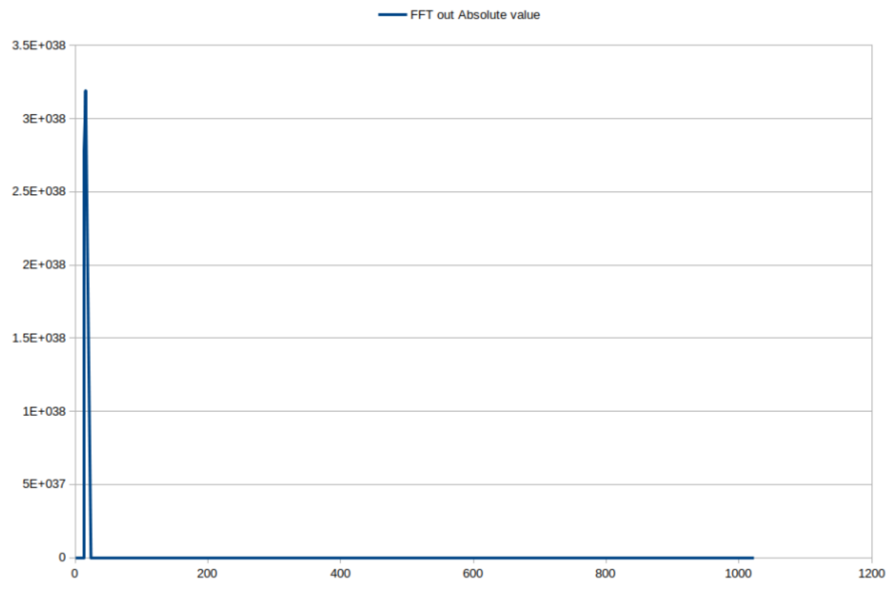


Figure 4.2 Output of 1024 point FFT of the FFT IP (sine wave)

Figure 4.2 shows the output of 1024 point FFT computed by FFT IP in PL in case of sine wave. In this experiment, I am not expecting several peaks in the graph because the input signal is generated without any noises and sine wave signal type has only one frequency component. The bin location of the highest peak is 16th bin of the FFT output as expected. The bin location of the frequency component in the output of the FFT IP and mathematically computed bin location are exactly the same. Therefore, the output of the FFT IP is satisfied in case of the sine wave.

In the second experiment right below, I used a square wave as the input signal.

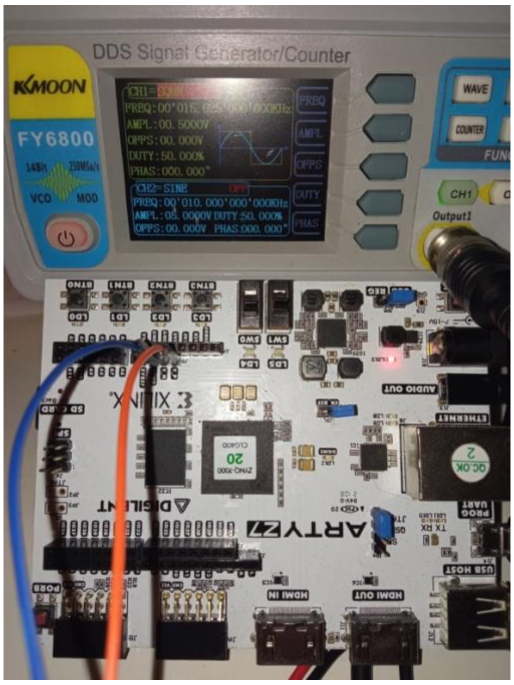


Figure 4.3 Connection of the analog signal (square wave)

In the Figure 4.1, physical connection of the analog signal and its configuration. As displayed in the signal generator’s display, the signal type is the square wave, Frequency is 15.625 KHz due to the bin size (976.5625 Hz). The amplitude of the signal is 0.5V because in the hardware part we use the XADC IP in the bipolar mode meaning that input is between -0.5 and +0.5.

The computation of bin location is the same as ones mentioned the first experiment because the input signal (except the type of signal) and the hardware configuration are the same.

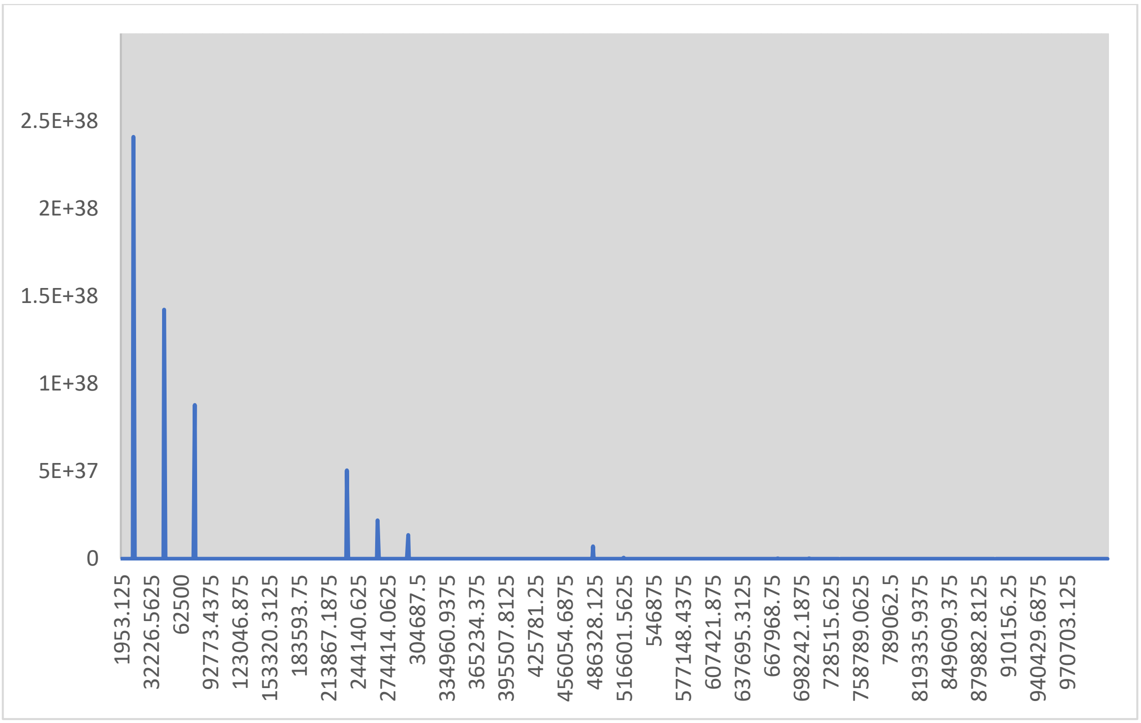


Figure 4.4 Output of 1024 point FFT of the FFT IP (square wave)

In Figure 4.4, X axis is frequency in Hz, Y axis is absolute value of FFT output. This graph shows the output of 1024 point FFT computed by FFT IP in PL, in case of square wave. In this experiment, I am expecting several peaks in the graph because the square wave signal type has several frequency components due to its shape. The bin location of the highest peak is 16th as expected. There are several smaller peaks (at bin 48 (45898.4375 Hz), at bin 80 (77148.4375 Hz), at bin 240 (233398.438 Hz), at bin 272 (264648.438 Hz), at bin 304(295898.438 Hz) and at bin 498 (485351.563) … so on.) in the graph as expected as well. The bin location of the main frequency component in the output of the FFT IP and mathematically computed bin location which is 16th bin are exactly the same. Therefore, the output of the FFT IP is satisfied in case of the square wave.

4.2 Results of Thermal control

As the result of the thermal control, I can present the temperature values which is stabilized by the PID controller designed in STM32. This PID settings of PID controller is important for thermal control because it controls PWM to the Peltier Cell. By doing so, less energy is consumed while maintaining the reference temperature.

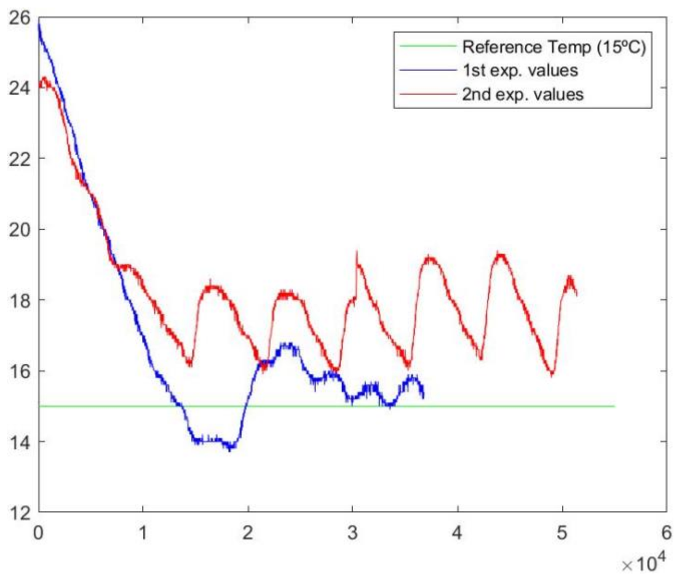


Figure 4.5 The records of measurements during the experiment

In the Figure 4.5, the measured temperature inside the box has been recorded in two experiments (red and blue). Both experiments are started from around $25°C$ and the PID controller must stabilize the temperature at $15°C$. In the Figure 3.19: Y axis is temperature in $°C$, X axis is number of samples (sampling time is 0.2 second). The curve in blue is the temperature values when the PID has tunning of $K_p = 50, K_i = 1, K_d = 0.1$. The curve in red is the temperature values when the PID has tunning of $K_p = 22.6994, K_i = 766.2856, K_d = 191.5714$.

As seen in the graph above, the first experiment in blue has better values than the other one. This is because there is an over-shoot in the graph which reaches 13.7 $°C$. After that the fluctuation of the blue graph is getting closer to the reference temperature over time while the other has not any over-shoot and keeps fluctuating in almost same size. The whole system uses less energy to control the Peltier Cell when the fluctuation gets smaller and the temperate gets more stable. Thanks to the estimation by the PID controller, value of PWM is set properly and the system uses less current.

CONCLUSIONS

Thanks to FPGAs' parallel operation, a FPGA based signal processing is the efficient way in terms of processing speed. In other word, FPGAs can process signals from number of channels once. For example, the FFT IP of XIILINX's Zynq 7000 has 12 channels, and these channels can be used parallelly. One can use this IP as much as he or she wants. The number of inputs or outputs can be limited by the number of physical pins. The number of IPs can be limited by the number of logic cells inside the PL (in case of Zynq 7000, there are 6.6M logic cells). Then one can design a new system with unlimited number of IPs but with limited number of logical cells.

The combination of a theoretical background and a hardware configuration of certain IP in Vivado HLS can lead us the correct way to implement any system in XILINX's FPGA with a support of a software implementation in XILINX SDK. The theoretical aspect mentioned in the Chapter 1 & 2, the configurations of IPs mentioned in Chapter 3 are finally meet in the last chapter where we get the results of FFT system in FPGA. With correct configurations of every IPs in the system, the "FFT system in FPGA" performs correct FFT outs which are validated with mathematical computations. To use the "FFT system in FPGA" as a filter, an interference of the given signal can be eliminated by omitting the frequency component corresponding to the interference signal from the FFT output and input that remaining bunch of data to Inverse Fourier Transform (IFFT). In fact, the FFT IP itself has a mode to compute IFFT.

For the embedded systems, like the "Thermal control", the choice of electronic components is important because they can play critical roles in the performance of the system. For example, the sensitivity level of a sensor depends on the ADC and the H-bridge driver's maximum current may limit the consumption of the Peltier cell in the "Thermal control". To keep the power consumption of the Peltier cell in its minimum level while it is cooling down or heating up the certain object, such as the FPGA based signal processing unit, the PID controller which is developed in STM32 is utilized. Thanks to the recursive operation of the PID controller, it is possible to stabilize the temperature of the object around the reference temperature and to keep the current which the Peltier cell uses in the minimum level while cooling or heating.

The future development of this project could start from the studying the FFT IP with the rounding mode of "Truncation". According to the datasheet of the FFT IP, it allows the IP to truncate the smaller peaks of FFT output. This mode has to be studied carefully with real analog signals because it may reduce processing time. In other word, If we could truncate the frequency components of the interferences, we would introduce the output of the FFT to the input of IFFP. It may open the door to implement "FFT system in FPGA" without interaction of PS (software),

ACRONYMS

ADC	Analog to digital converter
AMBA	Advanced Microcontroller Bus Architecture
APU	Application Processing Unit
AXI	Advanced eXtensible Interface
CLB	Configurable logic blocks
CPU	Central Processing Unit
DAC	Analog converter
DC	Direct Current
DFT	Discrete Fourier Transformation
DMA	Direct Memory Access
DRAM	Direct Random Access Memory
DSP	Digital signal processing
EMIO	Extended-MIO
FF	Flip-Flops
FFT	Fast Fourier Transform
FPGA	Programmable Gate of Array
FSBL	First Stage Bootloader
HDF	Hardware Description File
HLS	High-Level Synthesis
IP	Intellectual Property
I/O	Input/Output
LUT	Look-up Table
M2M	Memory-to-Memory
MM2S	Memory-to-Stream
MSPS	Mega sample per second
OS	Operation system
PCB	Printed Circuit Board
PL	Programmable logic
PMOD	Peripheral Module Interface
PS	Processing system
PWM	Phase Wide Modulation
RAM	Random Access Memory
ROM	Read-only Memory
S2MM	Stream-to-Memory
SDK	System Development Kit
SNR	Signal-to-Noise Ratio
SoC	System on Chip
SPI	Serial Peripheral Interface
SRST	System Reset
UART	Universal Asynchronous Receiver/Transmitter
UAV	Unmanned Air Vehicle

REFERENCES

[1] M. Jou Barrancos, “Development and testing of processing software for an airborne soil moisture mapper,” Barcelona, Spain, 2018.

[2] A. Jamal Rashed Daoud, “Implementation of interference mitigation techniques for L-band radiometers on-board Unmanned Aerial Systems,” Barcelona, 2020.

[3] J. G. M. D. G. Proakis, Digital Signal Processing: Principles, Algorithms and Applications. (Fourth Edition), New Jersey, 2007.

[4] D. D. f. O. R. H. P. D. U. V.-4. FPGAs, 2005. [Online]. Available: https://www.xilinx.com/publications/archives/books/dsp.pdf.

[5] S. M. L. B. H. T. W. Kuo, Real Time Digital Signal Processing (Second Edition), West Sussex, 2006.

[6] R. M. J. L. G. Y. Y. Woods, FPGA-based Implementation of Signal Processing Systems (Second Edition), United Kingdom, 2017.

[7] “www.nti-audio.com,” [Online]. Available: https://www.nti-audio.com/es/servicio/conocimientos/transformacion-rapida-de-fourier-fft. [Accessed 01 07 2022].

[8] realpars, “www.realpars.com,” [Online]. Available: https://realpars.com/pid-tuning/#:~:text=In%20the%20most%20simplistic%20terms,direct%20acting%2C%20as%20discussed%20previously..

[9] XILINX, “Vivado Design Suite User Guide,” 202.

[10] K. D. Team, “www.kicad.org,” 2021. [Online]. Available: https://www.kicad.org/about/kicad/.

[11] XILINX, “Zynq-7000 SoC Data Sheet,” XILINX, 2018.

[12] Digilent, “Digilent,” Digilent, [Online]. Available: https://digilent.com/reference/programmable-logic/arty-z7/reference-manual.

[13] XILINX, LogiCORE IP AXI XADC v1.00a Product Guide, 2012.

[14] XILINX, “AXI Interconnect (v1.06.a),” XILINX, 2012.

[15] ST, “STM32F103x8 datasheet,” ST, 2022.

[16] R. Sole, “hardwaresfera.com,” 11 July 2020. [Online]. Available: https://hardwaresfera.com/en/articulos/que-es-celula-peltier/.

[17] XILINX, “Fast Fourier Transform v9.1 LogiCORE IP Product Guide,” 4 May 2022. [Online]. Available: https://docs.xilinx.com/viewer/book-attachment/jKn_d6EeSeSm4b25FBbCOA/KtJ2q9rEZvP4hqj3BbbU_Q.

ANNEX I. OVERALL HARDWARE SCHEMATIC OF FFT SYSTEM IN FPGA

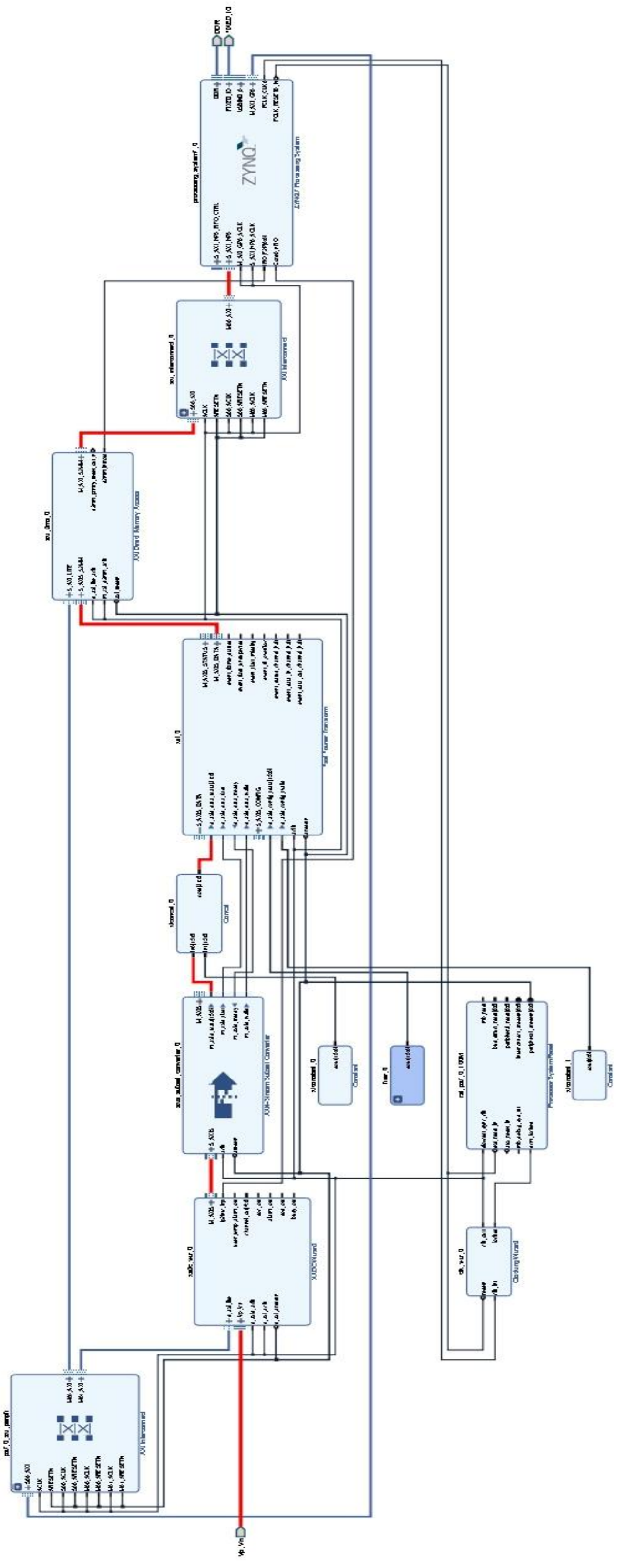


Figure I.1 Overall hardware schematic of FFT system in FPGA (More Visible)

ANNEX II. OVERALL HARDWARE SCHEMATIC OF THERMAL CONTROL

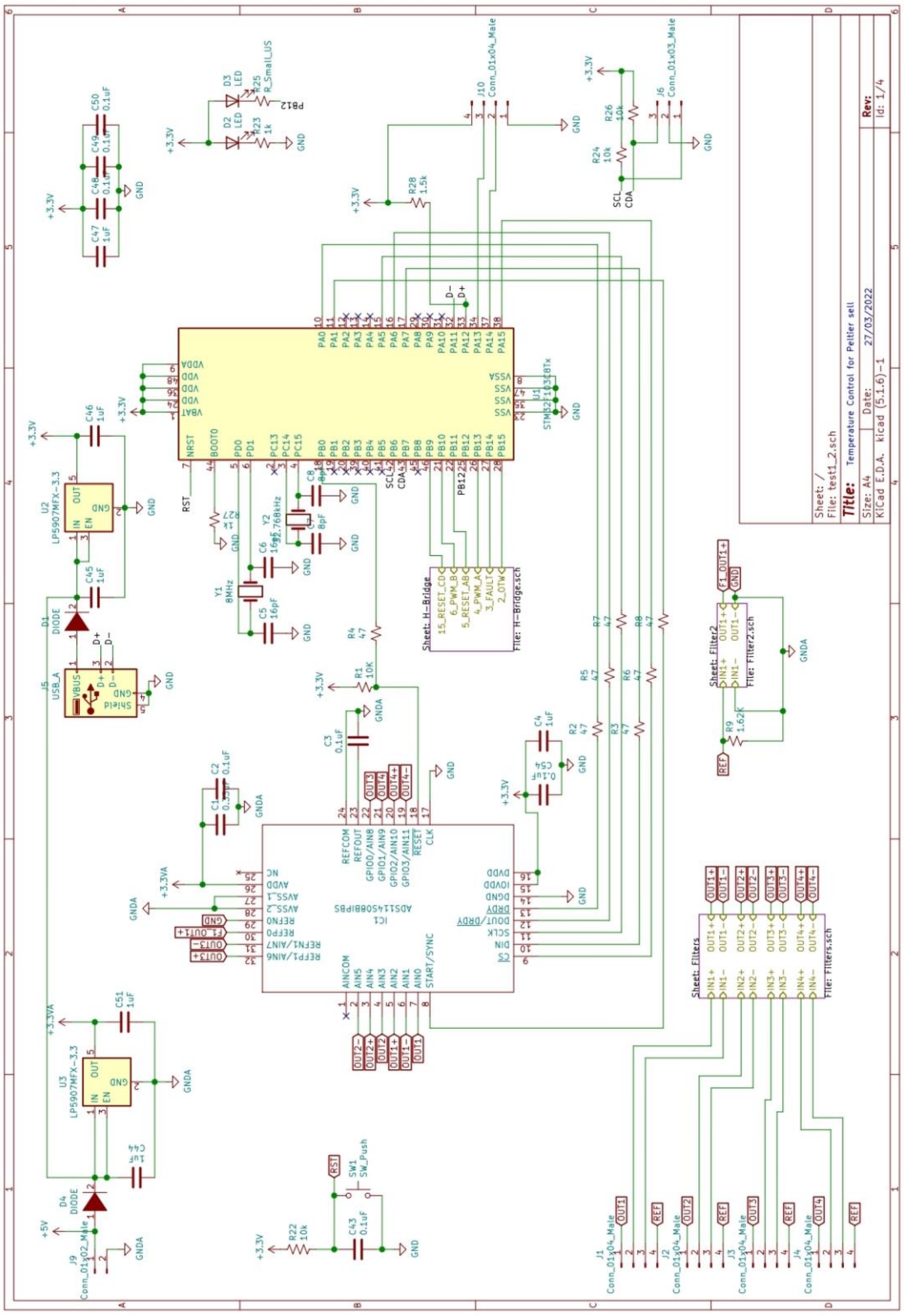


Figure I.2 Overall hardware schematic of Thermal control (Part 1

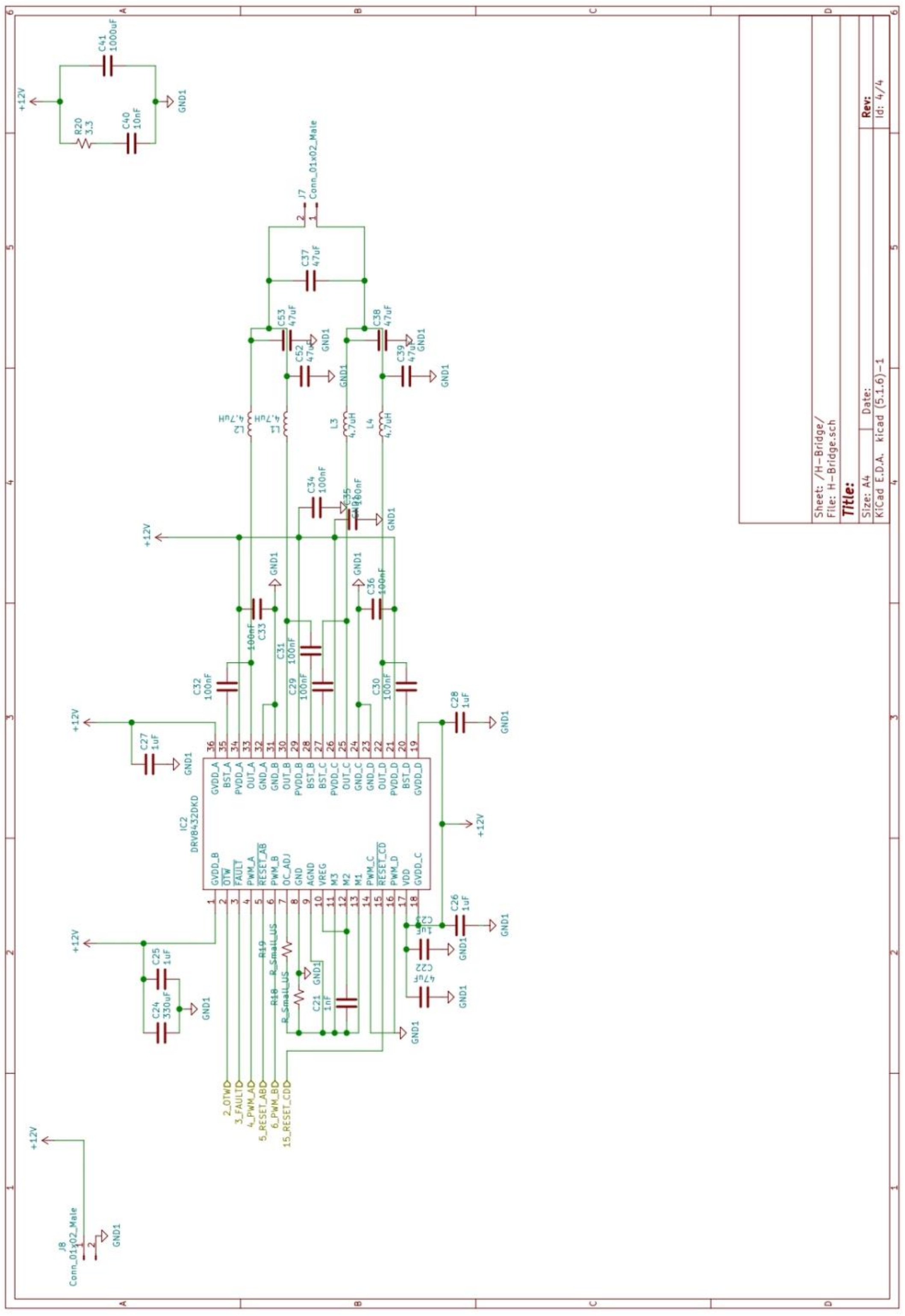


Figure I.3 Overall hardware schematic of Thermal control (Part 2)

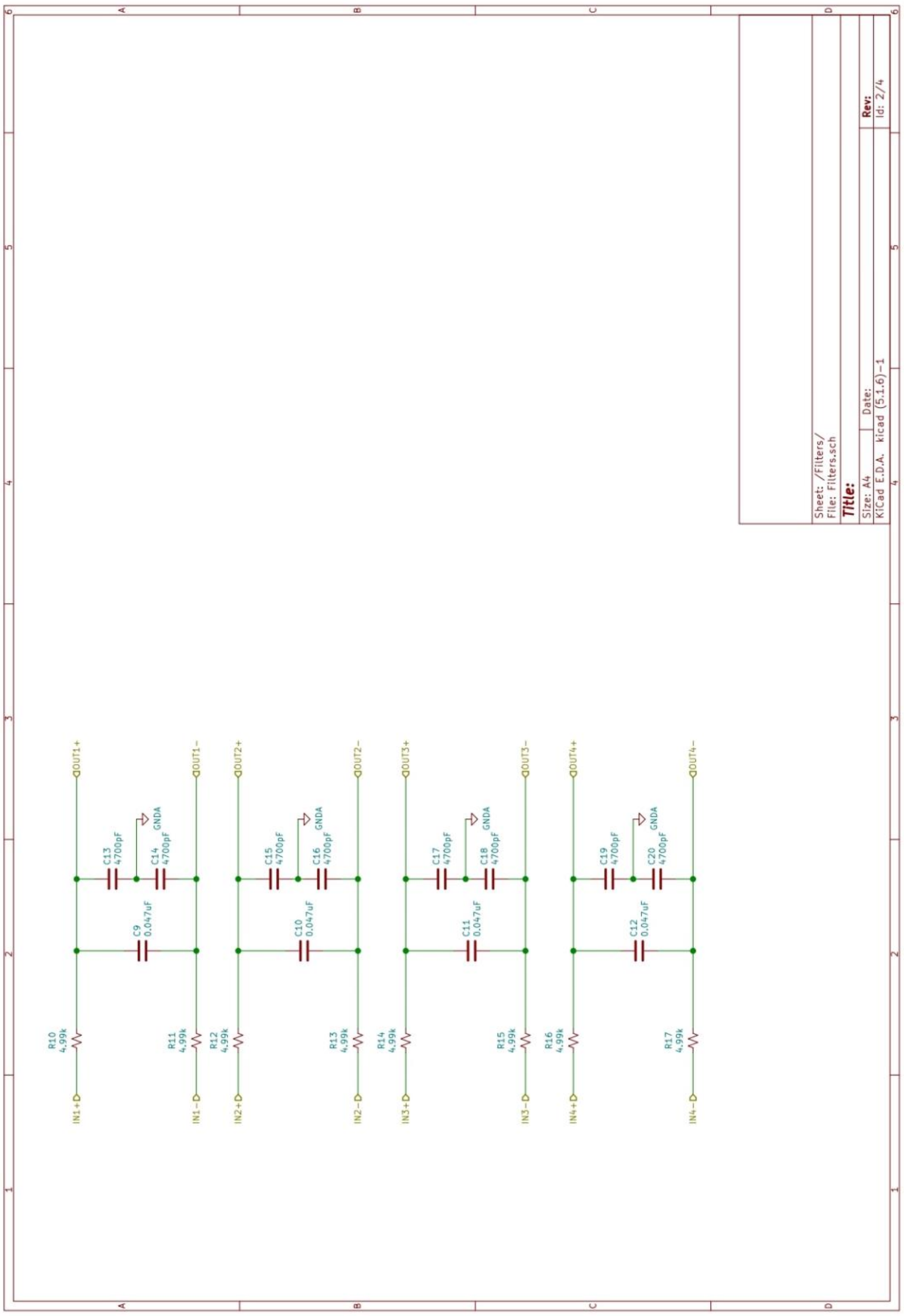


Figure I.4 Overall hardware schematic of Thermal control (Part 3)