People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdes

Institute of Electrical and Electronic Engineering
Department of Electronics

Final Year Project Report Presented in Partial Fulfilment of
the Requirements for the Degree of

MASTER

In **Electronics**

Option: **Computer Engineering**

Title:

FPGA-Based Phasor Measurement Unit Prototype

Presented by:

- **TALAMALI Youcef**
- **AMROUCHE Yacine**

Supervisor:

Dr. MAACHE A

Registration Number:………./2019

Abstract

Phase measurement is required in electronic applications where a synchronous relationship between the signals needs to be preserved. As the world continues to move towards a Smarter Grid day by day, it has become the necessity to incorporate real-time monitoring of the grid wherein the instantaneous snapshot of the health of the grid can be made available.

Traditional electronic system which are used for time measurement are designed using a classical mixed-signal approach. With the advent of reconfigurable hardware such as field-programmable gate arrays (FPGAs), it is more advantageous for designers to opt for all-digital architecture.

This project is about the design and implementation of a part of Phasor Measurement Unit (PMU) Prototype based on FPGA. It discusses how an FPGA can be used to estimate the phasors of a one-phase system. An example sinusoidal input signal is generated by a Function Generator and then sampled using the Analog to Digital Converters (ADC) on the FPGA board. The design then stores digital data into a local FIFO, which is passed to a 1024- Point FFT hardware core to get the spectrum of the signal and hence calculate the frequency and the phase difference.

The system uses the Intel DE10 FPGA board (donated by the Intel University Program) and the Quartus Prime suite to design and implement the system. One of the aims of this project is to evaluate the potentials of the newly acquired DE10 FPGA board. The final output of the FFT core are transmitted back to local host through Quartus SignalTap II logic Analyzer Tool.

Keywords: PMU, FPGA, VHDL, ADC, FFT, Quartus.

Dedication

This project is dedicated to my dear parents whose faith in me never once wavered for their love, endless support and encouragement ,also to my lovely sisters , brothers, nephews and nieces . I would like to thank them for their support and unconditional love during all my studies. I further extend my gratitude to all members of TALAMALI's family.

I also dedicate this project to all my good friends, especially the squad for all the moments and memories shared together, mostly the pep talks that brought humor along with encouragement through the years spent in university, without forgetting my childhood friends for always having my back no matter the circumstances.

Last and not least, I dedicate this humble work to everyone who has taught, encouraged, and advised me during all my studies.

.***Talamali youcef***

This work is first dedicated to my parents and my brothers for their never fading love, support, belief and patience and most importantly for their excellent guidance that made me become the person I am today, to my brothers and sisters and m and nieces for their unconditional love and care and all my family for their encouragement.

I also dedicate this project to all my good friends, especially the squad for all the moments and memories shared together, without forgetting my friends of the mosk for always having my back no matter the circumstances.

.Amrouche Yacine

Acknowledgment

All praise and thanks giving to Allah the most powerful and most merciful who give us the ability and patience to accomplish this humble work.

We would like to express our gratitude to our supervisor Dr. A. MAACHE for his guidance and support during our work. We are grateful to our teachers, academic staff and workers at the Institute of Electrical and Electronic Engineering who prepare for us the environment to work and offer to us their valuable help.

We would also want to thank all those people who supported us through the process of writing this report.

We would like to acknowledge the donation of the DE10 Standard board done by the Intel University Program.

Table of Contents

List of Figures

List of Abbreviations

AC	Analog Current
ADC	Analog to Digital Convertor
ASIC	Application Specific Integrated Circuit
ARM	Acorn RISC Machine
CPU	Central Processing Unit
DC	Direct Current
DFT	Discrete Fourier Transform
DE10 board	Development and Education board Cyclone V
CT	Current Transformer
EMS	Energy Management System
EPROM	Erasable Programmable Read Only Memory
FFT	Fast Fourier Transform
FIFO	First-In, First-Out
FPGA	Field programmable gate array
FSM	Finite State Machine
GPS	Global Positioning System
GUI	Graphical User Interface
HDL	Hardware Description Language
HPS	Hard Processor System
IC	Integrated Circuit
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronic Engineers
PC	Personnel Computer
PDC	Phasor Data Concentrator
PLD	Programmable Logic Device
PMU	Phasor Measurement Unit
PPS	Pulse Per Second
PROM	Programmable Read Only Memory

PT	Potential Transformer
RAM	Random Access Memory
ROM	Read Only Memory
RTL	Register Transfer Level
SCADA	Supervisory Control And Data Acquisition System
SDRAM	Synchronous Dynamic Random Access Memory
SoC	System on Chip
SoPC	System on Programmable Chip
SPI	Serial Peripheral Interface
SRAM	Static Random Access Memory
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
UTC	Universal Coordinated Time
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

Introduction

1.1 Overview

The load dispatch centers in a large power system supervise and control over the transmission network and it takes preventive actions to avoid any sort of system failure which can hamper electricity distribution. With ever increasing size and complexity of the power system, the ability to detect any faults in the power system is heavily dependent on the real time information available to the operator. Traditionally, analog and digital information (status of circuit breaker, power flow and frequency) is measured at the substation level and transmitted to load dispatch center using supervisory control and data acquisition system (SCADA) or energy management system (EMS). The major limitation of SCADA or EMS is the inability to accurately calculate the phase angle between a pair of substations. In SCADA or EMS, phase angle is either estimated from available data or is calculated offline. Phasor Measurement Units (PMU) overcome the limitations of SCADA and EMS by accurately calculating the phase angle between a pair of grids.

Synchronized phasor measurement units were introduced in the mid-1980s as a solution for the need of more efficient and safer monitoring devices for Electric Power Systems (EPS). Since then, measuring Electric Power System (EPS) parameters of voltage and current in relatively distant buses has received great attention from researchers. Such measurements are performed by phasor measurement units (PMUs), synchronized by Global Positioning System (GPS) satellites.

A commercial PMU measures the voltage and angle of a particular grid   at 25 samples per second. The phase information is synchronized with Global Positioning Systems (GPS) satellite and is transmitted to Phasor Data Concentrator (PDC) through a high-speed communication network. The time stamped phase information is called synchrophasor. There are several benefits of PMU such as monitoring of EPS and network protection. The measurement of voltage and current in remote bus allows the operator to make a concrete decision about the maintenance and security of the system in the face of various uncertainties [1].

Figure 1.1 shows an example of a standalone PMU device

Figure 1.1 Standalone PMU device from Arbiter, model 1133A power sentinel [2]

1.2 Literature Review

The measurement of voltage phase angles using synchronized clocks for power system applications dates back to the early 1980s when measurements of voltage phase angles were carried out between Montreal and SEPT-ILES [3], [4], and parallel efforts by Bonanomi in 1981 [5].

However, the synchrophasor technology available today emerged from the early efforts by Phadke et al. at Virginia Tech as described in [6], [7]. Phadke demonstrated the first synchronized PMU in 1988, and in 1991 Macrodyne Inc. launched the first commercial PMU product [8]. Due to the cost of early PMU devices, PMU technology has historically been limited to transmission system applications where the business case justified expensive phasor analysis equipment. One of the early applications that is important to mention is the implementation of the wide-area protection system Syclopes in France in the early 1990s, which was the first functional application of early forms of PMUs [9].

The cost of the components from which PMUs are assembled (such as GPS receivers, microprocessors, and storage devices) have dropped significantly due to recent developments across the electronics sector. As a consequence, PMUs have reached price points that have made them an attractive tool for the distribution systems and embedded generation. Many PMUs are sold as dedicated devices which offer event recorder type functionality. Costs for such units vary between US $6000 and US $15000 depending on the specification. Many equipment vendors have begun to offer PMU functionality as a supplementary feature on other products in their range, such as protection relays [10].

The standard for PMU devices is maintained by the IEEE C37.118 Working Group. IEEE Std. C37.118 [1] was released in 2005 and subsequently updated in 2011. The latest release comes in two parts; IEEE C37.118.1-2011 [1] describes how synchrophasors should be estimated and gives certification requirements while IEEE C37.118.2-2011 [2] describes data representation and data transfer. Concerns have been raised regarding the transient performance of PMUs under the 2005 standard [1], [11], [12]. These concerns are addressed in the 2011 release of the standard. IEEE C37.118.1-2011 states that it defines synchrophasors, frequency, and rate-of-change-of frequency measurement under all operating conditions [1].

1.3 Motivation

Many researchers designed PMU based on microcontrollers, but microcontroller is sequential in nature thereby degrading the efficiency of the system. By using FPGA, we are capable of measuring currents and voltages with parallel measurement, in other words the data for current and voltage will be read at the same time and at the same clock. It’s different from microcontroller that using a sequential programming language. In every task, it needs a couple of execution time, from first to the last task must be done by sequentially. Since there exist some gap of measurement, it will give us an uncertainty of the accurate time between compared value. It also takes longer delay than FPGA has.

The advantages of using FPGA in phasor estimation is that the 1024-point FFT hardware core can be pipelined. In other words, it can accept input data every clock cycle and generate output data every clock cycle, after a certain time delay. These huge computations can be handled well with a parallel processor such as FPGA. [13]

1.4 Project Objectives

The main goal of this project is to design and implement a Phasor Measurement Unit (PMU) prototype using the DE10 Standard FPGA.

1.5 Organization of the Report

This report is organized into four chapters. Chapter two outlines the theoretical background of the project and lists the components used, their description and principal of operation. Chapter three presents the design and implementation of the project's hardware, showing the interface between the different components of the system. Chapter four presents the finals results including the simulations done and the data transferred to PC. The conclusion summarizes the work presented in this report and provides suggestions for further work.

Chapter 2

Theoretical Background

This chapter introduces the theoretical background related to the design and implementation of the PMU prototype starting from the adopted signal model, the theoretical definition of phasors. This chapter also describes the FPGA used in our project.

2.1. Signal Model

Electrical power is traditionally delivered from the generators to the end-users through an infrastructure that is mainly composed by AC power systems. As a consequence, during normal operating conditions of the power system, voltage and current waveforms are usually modeled as signals characterized by a single sinusoidal component with constant parameters:

$$y(t) = Y_m cos(\omega t + \phi) \tag{2.1}$$

2.2. Phasor:

For a detailed analysis of an AC circuit, it is useful to know the magnitude, frequency and phase angle of the time-varying quantities during a specific time interval. The mathematical tool used to accomplish this task is called Phasor.

Let us consider equation (2.1), where Y_m represents the maximum value or peak amplitude; ω = 2πfo is the angular frequency of the signal in radians per second (f_0 is the fundamental frequency); and φ is the phase angle in radians. Keeping in mind the Euler's identity ($e^{jx} = cos\ x + j\ sin\ x$), one can observe that Eq. (2.1) can also be rewritten as

$$y(t) = Re\{Y_m e^{j(\omega t+\phi)}\} = Re[\{e^{j2\pi f_o t}\} Y_m e^{j\phi}] \tag{2.2}$$

once the system frequency is known, the term $e^{j2\pi f_o t}$ can be neglected. Therefore Eq. (2.2) may be represented by a complex number V given by

$$y(t) \longleftrightarrow V = Y_m e^{j\phi} = Y_m[cos\ \phi + j\ sin\ \phi]. \tag{2.3}$$

Assuming that both voltage and current signals are given by Eq. (2.3), one can observe that this representation is at odds with the calculattion of average power, therefore the RMS quantities must be taken into account for the correct phasor representation of sinusoidal signals, as illustrated by the complex number Y that follows.

$$y(t) \longleftrightarrow Y = \left(\frac{Y_m}{\sqrt{2}}\right) e^{j\phi} = \left(\frac{Y_m}{\sqrt{2}}\right) [cos\ \phi + j\ sin\ \phi]. \qquad (2.4)$$

The phase angle of a phasor brings the information about the fraction of the sinusoid's period in which the time, or the angular displacement ωt, is advanced or delayed to an arbitrary reference. It is very important to correlate different alternating-waves between them, thus, the phasors represent an equilibrium point or the steady-state condition of the AC circuit, that is, one can assume that the phasors are time-invariant, as illustrated in **Figure.2.1.**

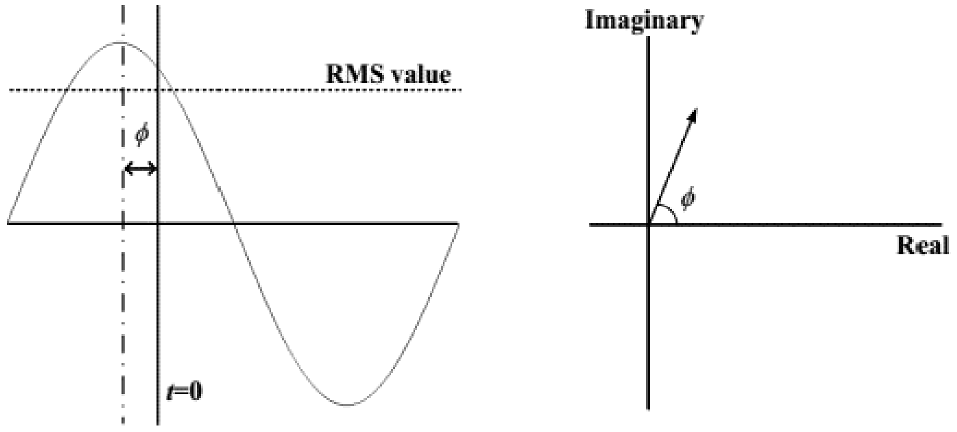


Figure 2.1: Phasor representation of a sinusoid.

However, in practical cases, a time interval must be considered to perform the phasor calculation. This time interval is also known as "data window" or "observation interval", being fundamental in phasor estimation of practical waveforms. In essence, the phasor representation is related to a pure sinusoidal signal, but the existing signals in electric power systems may be distorted by harmonics. In this way, it is advised to extract the envisaged frequency component(s) of the signal to also be represented by phasor notation. These tasks have been properly performed by the classical Fourier's theory. Due to the fact, the main key points about phasor representation using the aforementioned theory is presented and discussed in greater detail in the later sections.

2.3.Fast Fourier Transform

We should mention that the Fourier transform is a very important part of many engineering applications. FFTs are an important part of any digital spectrum analyzer.

FFTs can also be used when implementing a spectrogram, such as the one shown in **Figure 2.2** below. Such spectrograms make it easier to understand artifacts of speech and other sounds, or even radio frequency waveforms, by visual inspection [14].

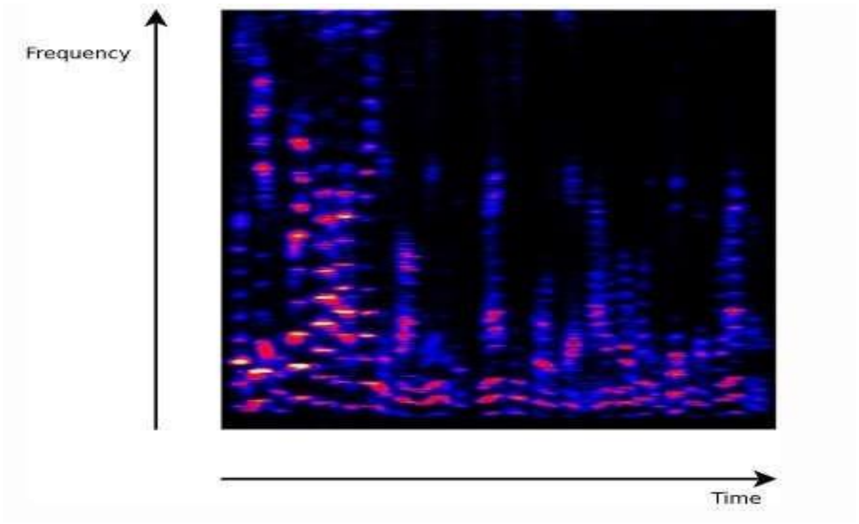


Figure 2.2 Spectral representation of speech

Convolutions and/or correlations can often be implemented much faster and cheaper using an FFT implementation of the Fourier transform. This means that digital filters can be implemented with Fourier transform enabled convolutions faster/better/cheaper. Fourier transform are used to understand and analyze control systems. Fifth, Fourier transforms are used not only in filter implementations, but they are also used in the filter design process. And finally, like we used it in our design Fourier transform can be used to evaluate phasor measurements.

The Fast Fourier Transform (FFT) is a specific implementation of the Fourier transform, that drastically reduces the cost of implementing the Fourier transform Prior to the invention of the FFT, a Discrete Fourier transform could only be calculated the hard way with N^2 multiplication operations per transform of N points. Since Cooley and Tukey published their algorithmic implementation of the Discrete Fourier transform, they can now be calculated with $O(N \log_2(N))$ multiplies. Needless to say, the invention of the FFT immediately started to transform signal processing. But before talking about the FFT we should understand a little more about what a Fourier transform is first [14].

A Fourier transform is a linear operator that decomposes a signal from a representation in time, to a time-less representation in frequency.

$$X\left(e^{j2\pi f}\right) = \sum_{n=-\infty}^{\infty} x[n]\, e^{-j2\pi f n} \tag{2.5}$$

This is the definition we will first come across when studying Fourier transform. This form above is easy to work with mathematically with just pen and paper. There are two problems with this nice mathematical definition when it comes to working with an engineering reality. The first problem is that digital algorithms do not operate upon continuous signals very well. Computers and other digital signal processors do a much better job with sampled signals. Hence, we'll switch from discussing the pure Fourier transform shown above and examine the Discrete-time Fourier transform instead. For this, we will switch from a continuous incoming signal, x(t) to its sampled representation, x[n]. The Discrete-time Fourier transform can then be applied to our signal.

$$X\left[\frac{k}{N}\right] = \sum_{n=0}^{N-1} x[n]\, e^{-j2\pi \frac{k}{N} n} \tag{2.6}$$

While this discrete-time transform works very nicely for representing the response of certain digital filters, it's still not all that practical. This brings us to the second problem: Computers can't handle an infinite number of samples, nor can they handle an infinite number potential frequencies. Both of these need to be sampled and finite. Fixing this second problem brings us to the Discrete Fourier transform.

$$X(f) = \int^{\infty} x(t) e^{-j2\pi f t} dt \tag{2.7}$$

Now, not only is the x[n] used in this transform discrete, but the frequency index, k/N, is as well. All three of these representations are very tightly related. Mathematically, there are major and significant differences between these transforms. Practically, however, only this last transform can ever be computed digitally. Therefore, the first two expressions of the Fourier transform and then the discrete time Fourier transform can only ever be digitally approximated by the Discrete Fourier transform [14].

It is this third representation of the Fourier transform, known as the Discrete Fourier transform, that we will be discussing the implementation. We are also going to argue that this is the only representation of the Fourier transform that can be numerically computed for any sampled finite sequence. If we look at equation (2.7) , we can see it takes as input N data samples, x[n], and calculates one summation across those inputs for every value of k to produce N samples out, X[k/N]. Given that there's a complex multiplication required for every term in that summation of N numbers, and that there are N relevant outputs, this will cost N^2 painful multiplications to calculate. If we just left things there, this transform would be so hard to calculate that no one would ever use it. Cooley and Tukey, however, described a way that the Discrete Fourier transform can be decomposed into two transforms, each of them being half the size of the original, for the cost of only N multiplies. If you then repeat this $\log_2(N)$ times, you'll get to a one-point transform, for a total cost of N $\log_2(N)$multiplies. At this cost point, the Discrete Fourier transform becomes relevant. Indeed, it becomes a significant and fundamental DSP operation.

An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors [2]. As a result, it manages to reduce the complexity of computing the DFT from $O(n^2)$, which arises if one simply applies the definition of DFT, to O(nlog n), where n is the data size. The difference in speed can be enormous, especially for long data sets where N may be in the thousands or millions. In the presence of round-off error, many FFT algorithms are much more accurate than evaluating the DFT definition directly. There are many different FFT algorithms based on a wide range of published theories, from simple complex-number arithmetic to group theory and number theory [14].

2.4.Phasor Calculation for 3-phase system

Consider a balanced 3-phase power system operating at a nominal frequency of f0, then the voltage waveform can be represented as

$$x1(t) = Xm\cos(2\pi f0t + \varphi 1)$$
$$x2(t) = Xm\cos(2\pi f0t + \varphi 2)$$
$$x3(t) = Xm\cos(2\pi f0t + \varphi 3) \qquad (2.7)$$

Here **Xm** represents the maximum amplitude of the signal and **Φ** represents the phase angle. The phase angles are 120 degree or $\frac{2\pi}{3}$ radian apart. The time domain sample of the power system can be represented as

$$X_{n1} = X_m cos(\frac{2\pi n}{N} + \varphi_1)$$
$$X_{n2} = X_m cos(\frac{2\pi n}{N} + \varphi_2) \qquad (2.8)$$
$$X_{n3} = X_m cos(\frac{2\pi n}{N} + \varphi_3)$$

Here N is the number of samples, which is an integer multiple of fundamental frequency. f_0 and n represents the sample index in the array which ranges from 0 to N−1.

The generalized expression for N-point can be represented as

$$X = \frac{1}{N}\sum_{n=0}^{N-1} x_n(cos\frac{2\pi n}{N} - jsin\frac{2\pi n}{N}) \qquad (2.9)$$

N-point DFT of the signal can be found out using

$$X_{real} = \frac{\sqrt{2}}{N}\sum_{n=0}^{N-1} x_n(cos\frac{2\pi n}{N}) \qquad (2.10)$$

$$X_{img} = \frac{\sqrt{2}}{N}\sum_{n=0}^{N-1} x_n(cos\frac{2\pi n}{N}) \qquad (2.11)$$

The real and imaginary part of the above expression can be rewritten as

$$X_k = \frac{\sqrt{2}}{N}\sum_{n=0}^{N-1} x_n(cos\frac{2\pi n}{N} - jsin\frac{2\pi n}{N}) \qquad (2.12)$$

$$X_{nominal} = \frac{\sqrt{2}}{N}\sum_{n=0}^{N-1} x_n(cos\frac{2\pi n}{N} - jsin\frac{2\pi n}{N}) \qquad (2.13)$$

The phasor estimate at nominal frequency is represented by this complex quantity $X_{nominal}$, whose $|X_{nominal}| = \sqrt{X_{real}^2 + X_{img}^2}$ magnitude gives the RMS magnitude of the signal. The phase angle can be computed using the trigonometric property, $\phi_{nominal} = atan(\frac{X_{img}}{X_{real}})$.

2.5. Field Programmable Gate Array

2.5.1. Overview

A field-programmable gate array (FPGA) is a logic device that contains a two-dimensional array of generic logic cells and programmable switches. The conceptual structure of an FPGA device is shown in **Figure 2.3**. A logic cell can be configured (i.e., programmed) to perform a simple function, and a programmable switch can be customized to provide interconnections among the logic cells. Once the design and synthesis are completed, a simple adaptor cable has to be used to download the desired logic cell and switch configuration to the FPGA device and obtain the custom circuit [15].

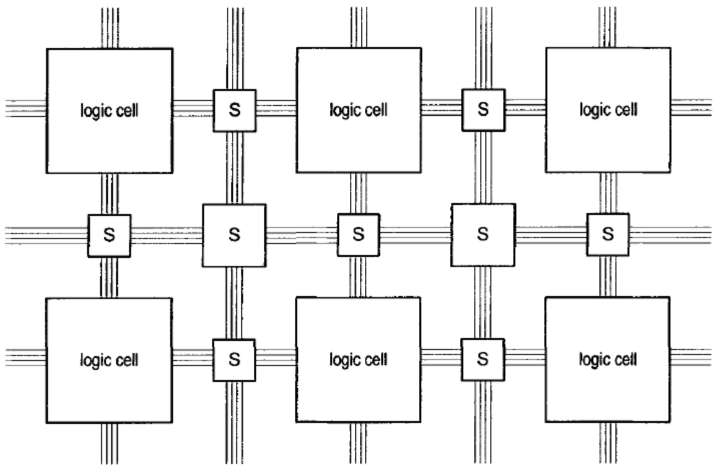


Figure 2.3 Conceptual Structure of an FPGA Device

An FPGA can be used to solve any problem which is computable. This is trivially proven by the fact FPGA can be used to implement a soft microprocessor, such as the Xilinx MicroBlaze or Altera Nios II. Their advantage lies in that they are sometimes significantly faster for some applications because of their parallel nature and optimality in terms of the number of gates used for a certain process [16].

2.5.2. Applications of FPGAs

Specific applications of FPGAs include digital signal processing, software-defined radio, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection and a growing range of other areas [16].

2.5.3. DE10-Board :

The DE10-Standard Development Kit presents a robust hardware design platform built around the Intel System-on-Chip (SoC) FPGA, which combines the latest dual-core Cortex-A9 embedded cores with industry-leading programmable logic for ultimate design flexibility. Altera’s SoC integrates an ARM-based hard processor system (HPS) consisting of processor, peripherals and memory interfaces tied with the FPGA fabric using a high-bandwidth interconnect backbone. The DE10-Standard development as shown in **Figure 2.4** board includes hardware such as high-speed DDR3 memory, video and audio capabilities, Ethernet networking, and much more. [17]

The following hardware is provided on the board:

•Intel Cyclone V SE 5CSXFC6D6F31C6N device

• Serial configuration device –EPCS128

• USB-Blaster II onboard for programming; JTAG Mode

• 64 MBSDRAM (16-bit data bus)

• 4 push-buttons

• 10slide switches

• 10red user LEDs

• Six 7-segment displays

• Four 50MHz clock sources from the clock generator

• VGA DAC (8-bit high-speed triple DACs) with VGA-out connector

• PS/2 mouse/keyboard connector

• IR receiver and IR emitter

• One HSMC with Configurable I/O standard 1.5/1.8/2.5/3.3

• A/D converter, 4-pin SPI interface with FPGA

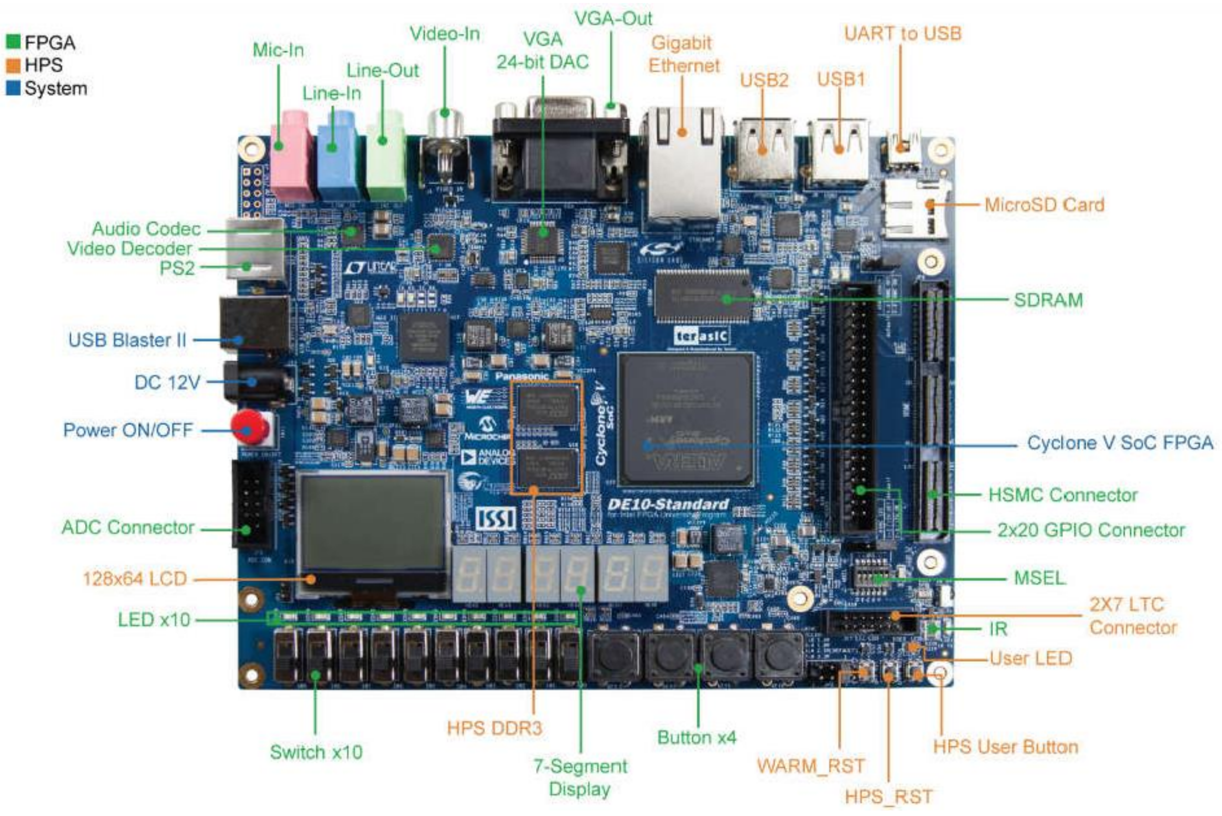


Figure 2.4 DE10-Standard development board (top view) [17]

2.5.4. Quartus PRIME Software

Quartus prime is a software development suit tool developed by **Intel FPGA** (former Altera) Company. It provides a graphic interface for users to access tools and display relevant files. Some differences may exist between different versions. The default Quartus prime GUI window is shown in the below **Figure 2.5**

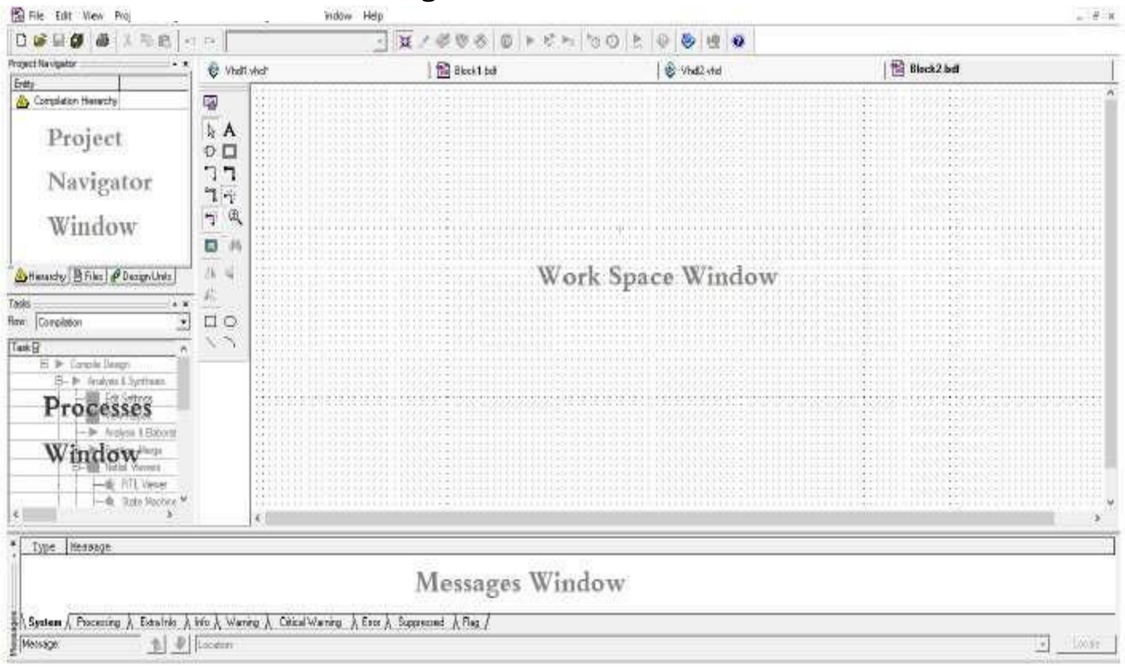
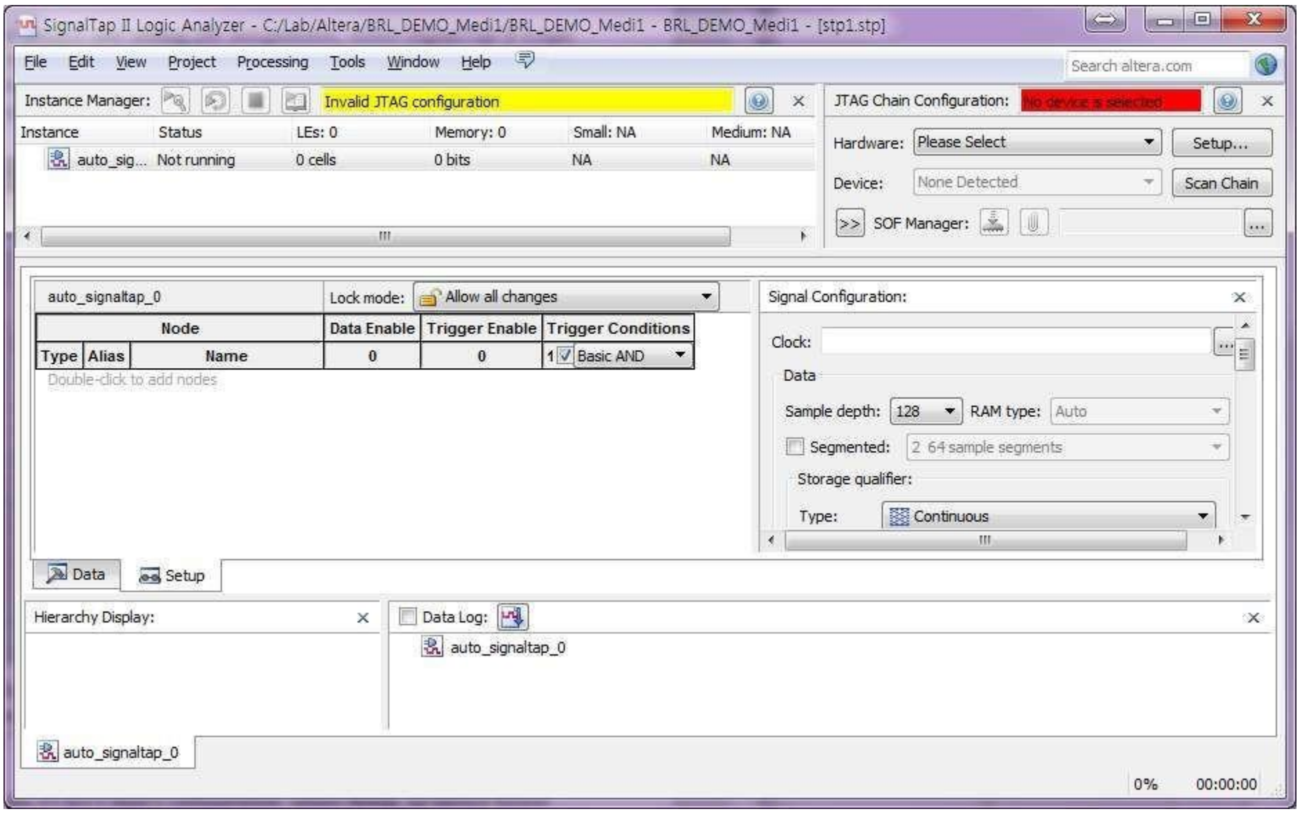


Figure 2.5 Typical Quartus PRIME GUI Window

2.5.4.1. Signal-tap Logic Analyzer

The Signal-Tap II Embedded Logic Analyzer is a system-level debugging tool provided by Quartus Prime, that captures and displays signals in circuits designed for implementation in Intel/Altera's FPGAs. Signal-Tap runs on the chip, with your design, in real hardware (not simulation) to provide waveforms of logic signals within the design. Signal-Tap uses significant hardware resources on the FPGA to allow flexible triggering and to record waveforms of your logic signals for later viewing on the PC running Quartus Prime. All communication is done through the JTAG programming cable that you use to program the FPGA. No extra hardware external to the FPGA is necessary. **Figure 2.6** depicts the overall window of this tool [18].


SignalTap II Logic Analyzer - C:/Lab/Altera/BRL_DEMO_Medi1/BRL_DEMO_Medi1 - BRL_DEMO_Medi1 - [stp1.stp]
File Edit View Project Processing Tools Window Help
Search altera.com
Setup...
Scan Chain
Data
Setup
Double-click to add nodes

Instance Manager: Invalid JTAG configuration

JTAG Chain Configuration: No device is selected

Instance	Status	LEs: 0	Memory: 0	Small: NA	Medium: NA
auto_sig...	Not running	0 cells	0 bits	NA	NA

Hardware: Please Select
Device: None Detected
SOF Manager:

auto_signaltap_0 Lock mode: Allow all changes

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	0	0	1 ✓ Basic AND

Signal Configuration:
Clock:
Data
Sample depth: 128 RAM type: Auto
Segmented: 2 64 sample segments
Storage qualifier:
Type: Continuous

Hierarchy Display:
Data Log: auto_signaltap_0
auto_signaltap_0
0% 00:00:00

Figure 2.6 Overall windows of the SignalTap II Logic Analyzer

2.6. Analog to Digital Converter

2.6.1. Overview

Analog to digital converter converts continuous analog signal to discrete digital numbers. ADC's differ from each other by two main parameters, the resolution which indicates the number of discrete values it can produce over the range of analog values and the step size (quantization value) which is based on the reference voltages of the ADC and it can be found as:

$$\Delta V = \frac{V_{ref+} - V_{ref-}}{2^{n\_bits} - 1} \tag{2.8}$$

2.6.2. The LTC2308

The DE-10 Board FPGA has a 12 Bit ADC, The LTC2308 is a low noise, 500ksps,8-channel,12-bit successive approximation register (SAR) A/D converter. The LTC2308 includes a precision internal reference, a configurable 8-channel analog input multiplexer (MUX) and an SPI-compatible serial port for easy data transfers. The ADC may be configured to accept single-ended or differential signals and can operate in either unipolar or bipolar mode. A sleep mode option is also provided to save power during inactive periods. Conversions are initiated by a rising edge on the CONVST input. Once a conversion cycle has begun, it cannot be restarted until the current conversion is complete. The time taken by each conversion for each channel is 1.3 µs [19].

Figure 2.7 represents SPI timing specifications for this ADC

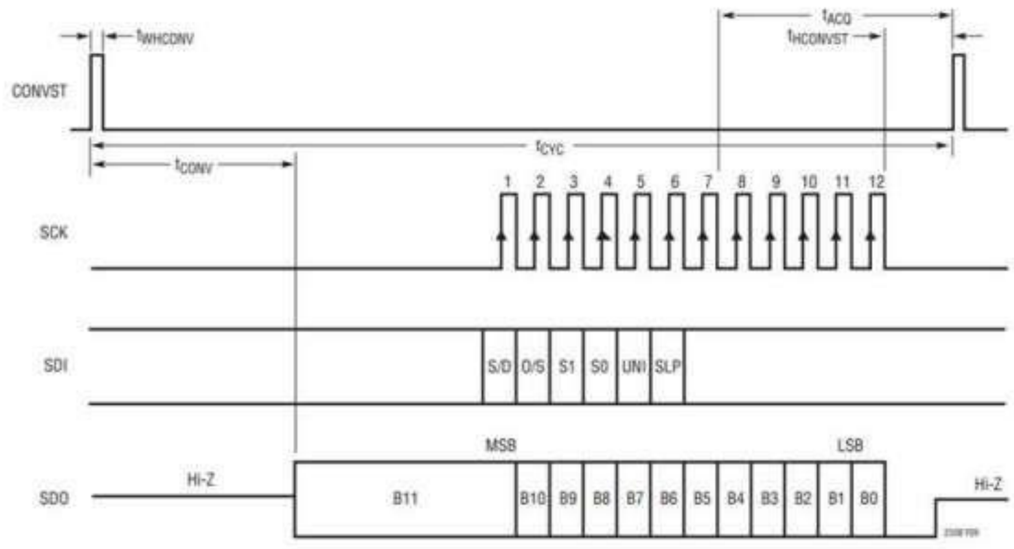


Figure 2.7 LTC2308 Timing with a Short CONVST Pulse [19]

Chapter 3

Hardware Design

This chapter describes the design and implementation of a part of a Phasor Measurement Unit using FPGA. The block diagram of such a unit is shown in **Figure 3.1.**

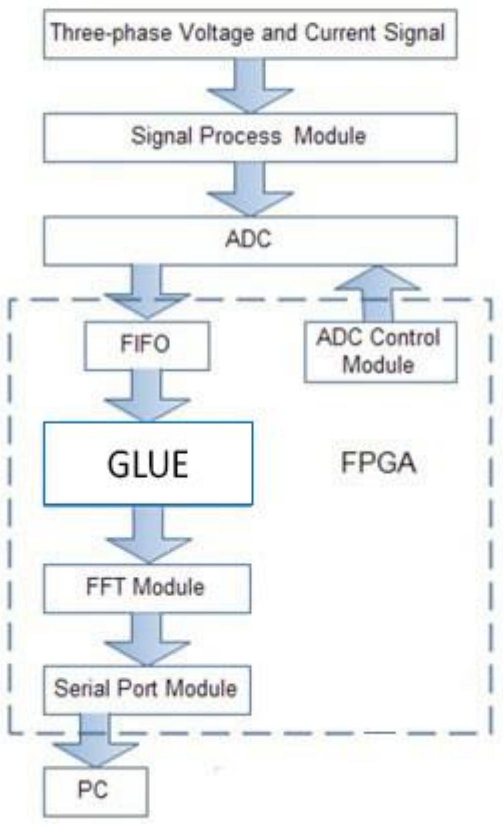


Figure 3.1: Block diagram of our prototype PMU

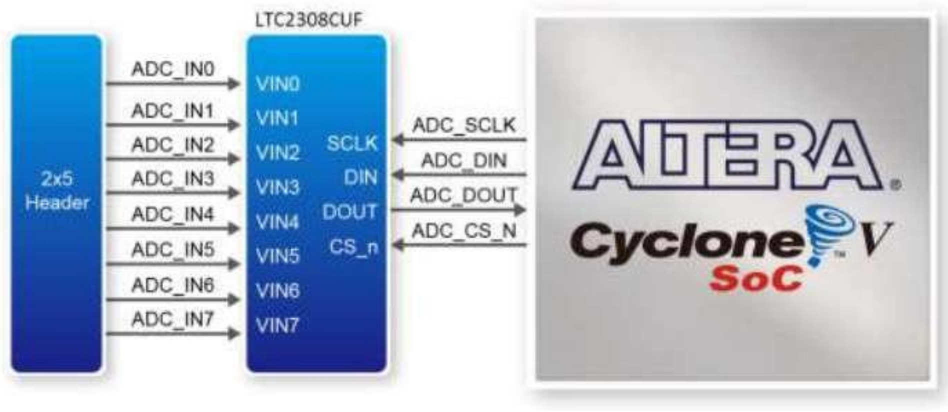
3.1. Signal Acquisition and Sampling

For the calculation of a phasor, the data (i.e. the sampled voltage signal)

must be acquired. When the PMU is tested in real-world scenarios a means of getting the signals from the transmission lines is necessary, which is accomplished using a Potential Transformer (PT) and a Current Transformer (CT) in the substations. This signal is further stepped down using the Hall Effect voltage sensors. However, in our laboratory setup, we used a Function Generator to generate sinusoidal input signal to mimic the signals read from voltage and current sensors.

3.1.1. Sampling of the signals using ADC (Analog to Digital Converter)

As explained in the last chapter, the on-board ADC is connected to the FPGA through an SPI-interface as seen in Figure 3.2. A state machine is designed and implemented in VHDL to control this ADC



. **Figure 3.2**: Connections between the FPGA, 2x5 header, and the A/D converter [19]

3.2 FFT Calculation

To implement the phasor calculation unit, the DE10-board FPGA has been used as the computational unit. For a 3-phase system, the voltage samples are stored in FIFOs on the FPGA which is updated every time a new sample comes in. A counter in the FIFO keeps track of the number of accumulated samples. As long as the FIFO store data, the phasor calculation task is initiated, and the FFT unit is used to calculate the spectrum of the input signal.

3.3 PMU SoC System Design

The top-level file containing the SoC overall project is illustrated in **Figure 3.3**.

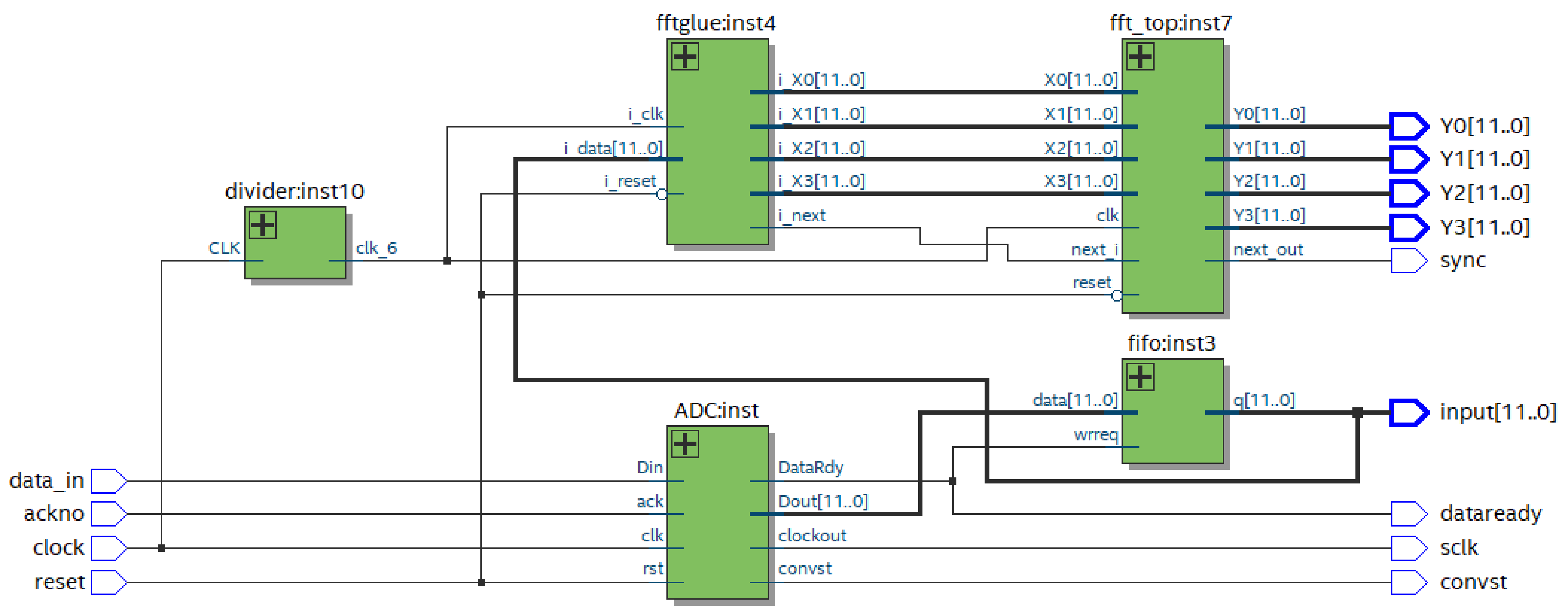


Figure 3.3 Overall Block Diagram of the On-Chip System

This design consists of two parts:

3.3.1. The Off-Chip Hardware Unit

The off-chip hardware unit is composed of:

3.3.1.1. Function Generator:

It generates the analog signals to be processed; in our case, it replaces the 3-phase signals after signal conditioning.

3.3.1.2. Data Acquisition Unit:

It samples analog signals coming from the function generator and converts them into digital data.

3.3.1.3. The PC Unit

The PC runs the Quartus Prime development suite tools to develop, debug and download the overall system onto the FPGA chip of the DE10 board.

3.3.2. The On-Chip Hardware Unit

The on-chip hardware is implemented on the FPGA; it controls the Off-Chip hardware unit and communicates with the PC via the USB Blaster cable.

The SoC system main entity has the following inputs and outputs:

- Data-in: 1-bit input port connected to the ADC block, which represent the serial digital data coming-in from the LTC2308 ADC to the FPGA.
- Data-out: 12-bit output port coming out of the ADC block, which represent the sampled parallel data.
- Data ready: 1-bit output ports from the ADC to tell if the Data in the output register is valid or not. It is also used for handshaking with the FIFO block.
- Convst: 1-bit output used to send the start of conversion signal to the ADC.
- Sclk: 1-bit output used to provide slower serial data clock for the ADC
- Reset : 1-bit input asynchronous reset connected to the ADC and FFT block., to reset the hardware.
- clk: 1-bit input port which represents the 50MHz clock input used as

the overall design clock source.

- result: 24-bit output port coming out from the FFT block used to calculate the spectrum of the input signal sequence.

- sync: 1-bit output port from the FFT block to indicate that we have got the calculated results from the FFT core.

The overall block symbol of the SoC project is shown in **Figure 3.4**

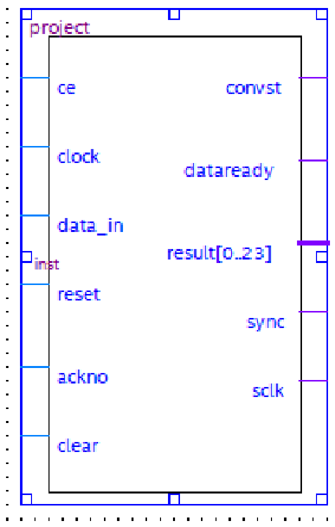


Figure 3.4 PMU SoC block symbol

3.4. PMU SoC project blocks

The SoC project is made of several custom blocks developed in VHDL involving the data acquisition and sampling of the analog input unit i.e. the ADC unit, frequency divider unit, data buffering unit, and the FFT unit. Each block will be detailed in the following sections

3.4.1. Clock Divider block

This block is used to synchronize data communication between the different blocks by generating multiple ranges of clock frequencies, as we can see in

Figure 3.5. The main frequencies generated by this block were : 25Mhz which was supplied to the ADC block and the FIFO block, the 12.5Mhz sampling frequency which was supplied to the slow clock of the ADC and the 6Mhz clock which was used with The FFT block.

The reason why we chose these frequencies was because of the following:

- The time for the whole power-up and conversion process before the data is ready to be output in the ADC is 5.5us (max), and we know that in order to sample one pulse we require 1.6us, that's why we used a clock of 80ns i.e. 12.5Mhz, hence the sampling period will be 6.4us, to make sure the output is ready and is correct.
- Since the FFT block is very fast, in order to synchronize the FFT with the arriving sampled signal from the FIFO block it was necessary to use a slower clock, that is why we used the 6Mhz clock.

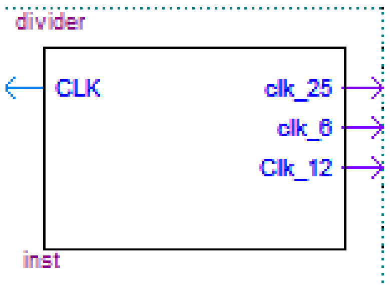


Figure 3.5: block diagram of the clock divider unit

3.4.2. The ADC controller block:

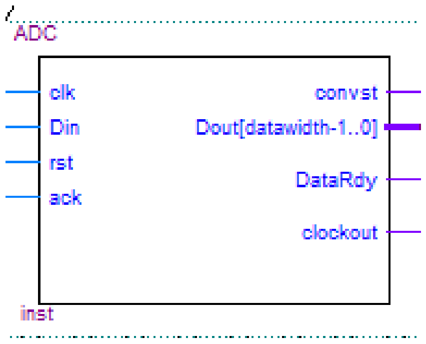


Figure 3.6: block diagram of the ADC controller unit

As we can see from Figure 3.6, the ADC controller read the digital data from the on-board ADC using the Din serial input. In typical PMUs, the ADC conversion is started on reception of 1 pulse per second (PPS) from the GPS module. However, in our case we did not have a GPS module. So, a locally generated trigger PPS signal is used. The sampling process then begins as the state machine in **Figure 3.7** depicts.

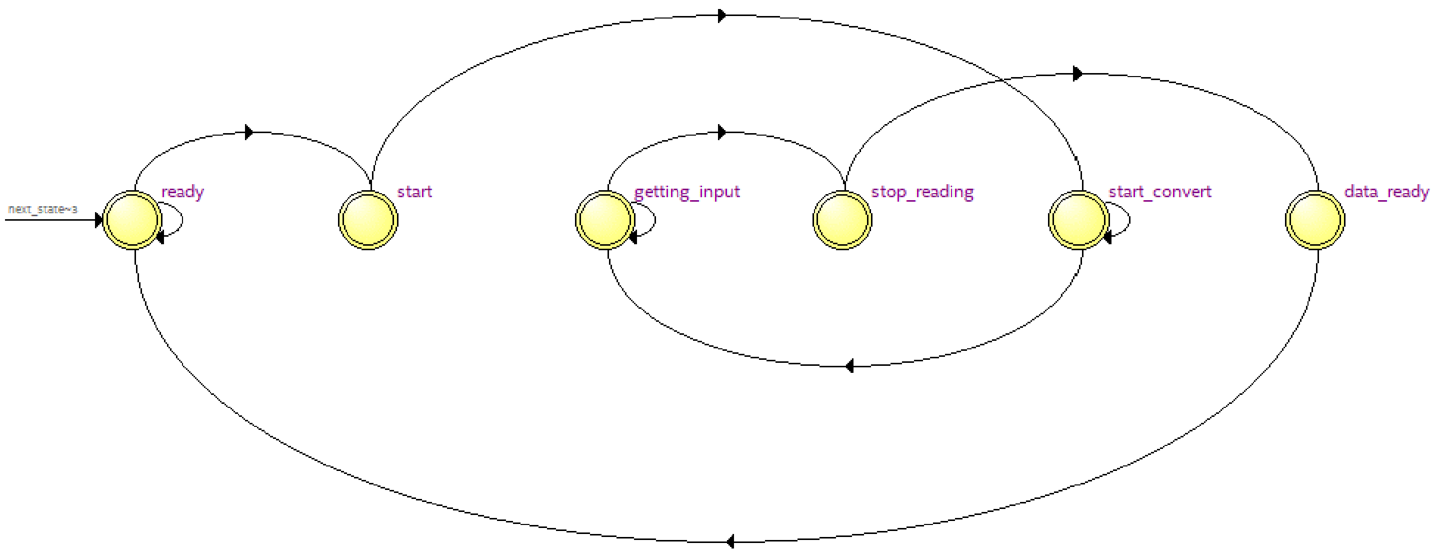


Figure 3.7: State machine representing data acquisition from the ADC

Since the ADC in the DE-10 board FPGA is a 12-bit ADC, it gives a data reading of 0 to 2^{12}-1 for an input voltage range of 0 volt to 4.096 volt. Hence, it is necessary to map the digital data readings with the actual measured values using the linear relationship between them.

3.4.3. FIFO block (storage block):

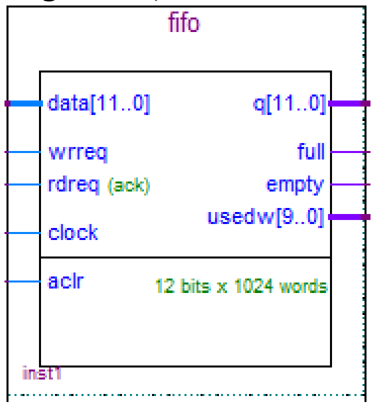
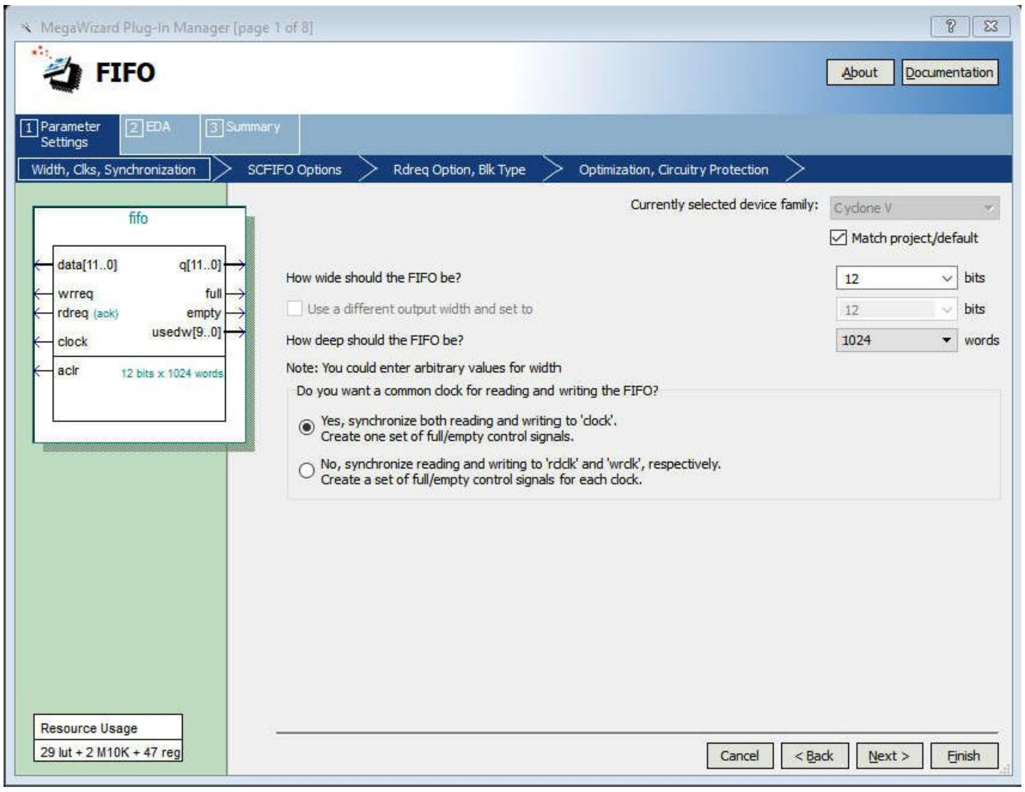


Figure 3.8: block diagram of the storage unit

Once the data is sampled, it needs to be stored temporally in order to be processed by the FFT block as shown in **Figure 3.8**. The storage process begins once the ADC sends to the FIFO a “dataready” signal. The FIFO block used was initially generate from the IP core library in Quartus Prime. It was configured to accept a 12-bit width and 1024 words depth as shown in **Figure 3.9.** This block is also provided with full and empty signal which are used to enable the FFT block once the data is available. This IP block is clocked at the same clock as the ADC.



Currently selected device family: Cyclone V

[✓] Match project/default

How wide should the FIFO be? 12 bits

[ ] Use a different output width and set to 12 bits

How deep should the FIFO be? 1024 words

Note: You could enter arbitrary values for width

Do you want a common clock for reading and writing the FIFO?

[✓ Yes, synchronize both reading and writing to 'clock'. Create one set of full/empty control signals.]

No, synchronize reading and writing to 'rdclk' and 'wrclk', respectively. Create a set of full/empty control signals for each clock.

Resource Usage

29 lut + 2 M10K + 47 reg

Figure 3.9: Configuration of the FIFO block in the mega function wizard

3.4.4. The Interfacing Logic:

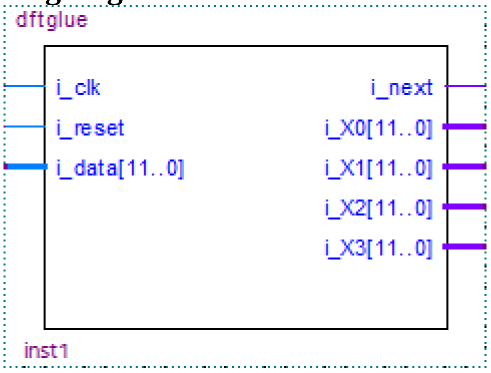


Figure 3.10: block diagram of the Interfacing logic unit

This “Glue logic” as named in **Figure 3.10** was used to interrface the FIFO block with the FFT block, since The FFT block needs to meet some conditions to be enabled. This block’s main task was to meet these conditions such as triggering the “ce” signal of the FFT and make the data length wider in order to be accepted by the FFT.

3.4.5. The FFT block

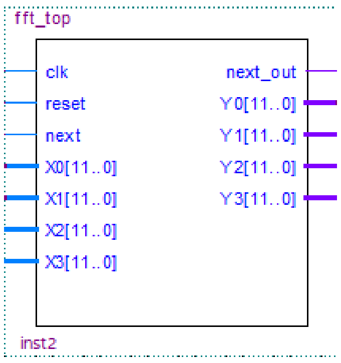


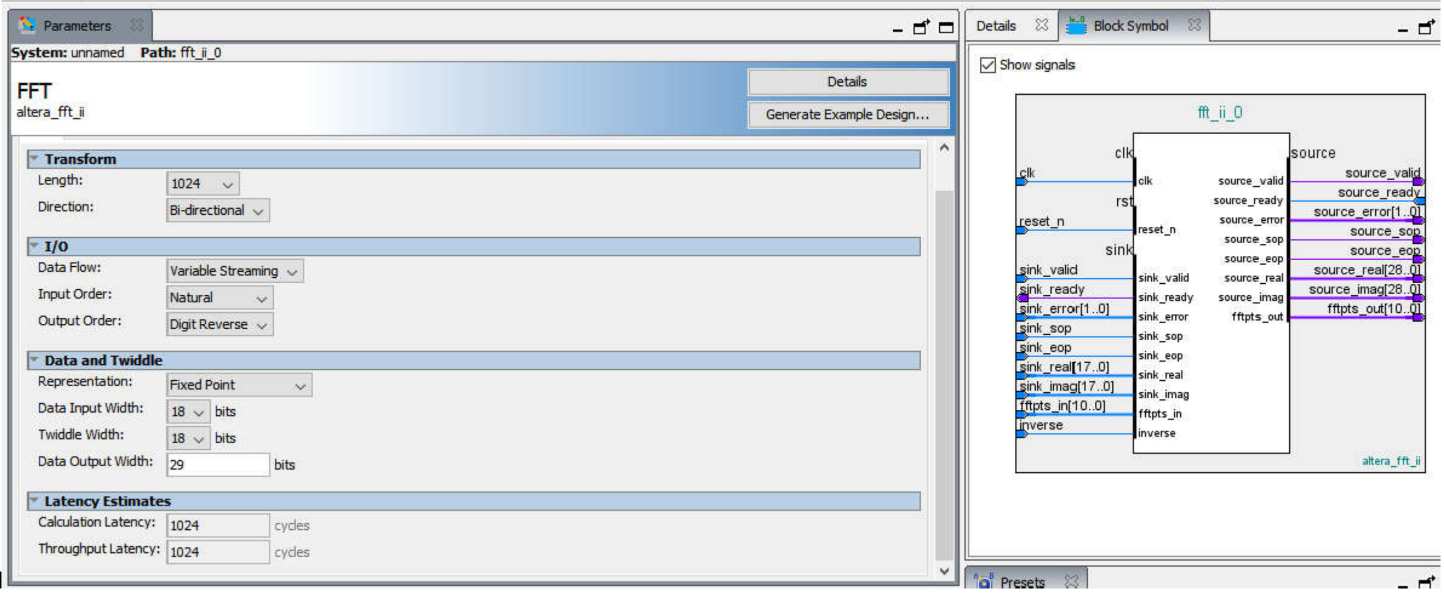
Figure 3.11: block diagram of the FFT unit

This is the main block of our implemented design as it processes the sampled data in order to finally calculate its spectrum. Given that implementing is an FFT block is a huge and a complex task, we evaluated three different (already existing) FFT hardware cores to test our system:

3.7.5.1. The Intel FFT core integrated in the IP catalog in Quartus Prime:

This core represents the best solution in terms of efficiency and reliability. However, we did not use this core, since it has two major issues:

- This core requires sending data via bus which should be controlled by a microprocessor. Hence, it is not suited for our system design which is based on direct execution on pure hardware blocks.
- As shown in **Figure 3.9** this core has many handshaking signals which makes it very complex to interface with our glue logic.



Parameters

System: unnamed **Path:** fft_ii_0

FFT
altera_fft_ii

Transform
Length: 1024
Direction: Bi-directional

I/O
Data Flow: Variable Streaming
Input Order: Natural
Output Order: Digit Reverse

Data and Twiddle
Representation: Fixed Point
Data Input Width: 18 bits
Twiddle Width: 18 bits
Data Output Width: 29 bits

Latency Estimates
Calculation Latency: 1024 cycles
Throughput Latency: 1024 cycles

Figure 3.12: FFT core integrated inside Quartus's mega wizard

3.7.5.2. Open Source FFT core by Gisselquist Technology, LLC

This is an open source FFT core, which is implemented by **Gisselquist Technology, LLC**. In order to be able to generate the Verilog code for this core we followed a number of steps [13]:

- Running the FFT Core Generator software:

  Inside Linux shell we run the following commands:

```
$ git clone https://github.com/ZipCPU/dblclockfft
$ make
```

  Once the "make" command completes, we got an 'fftgen' exectable program in the sw/ subdirectory which is used to generate customizable FFT cores depending on user requirements.

- Generating the FFT core

  This core offers several features, but since we needed just a 1024 point FFT with 12-bit input/output width, the following commands is used to generate the core:

```
$ ./fftgen  -f 1024  -n 12  -m 12
```

The block diagram of the generated core is represented in **Figure 3.13**

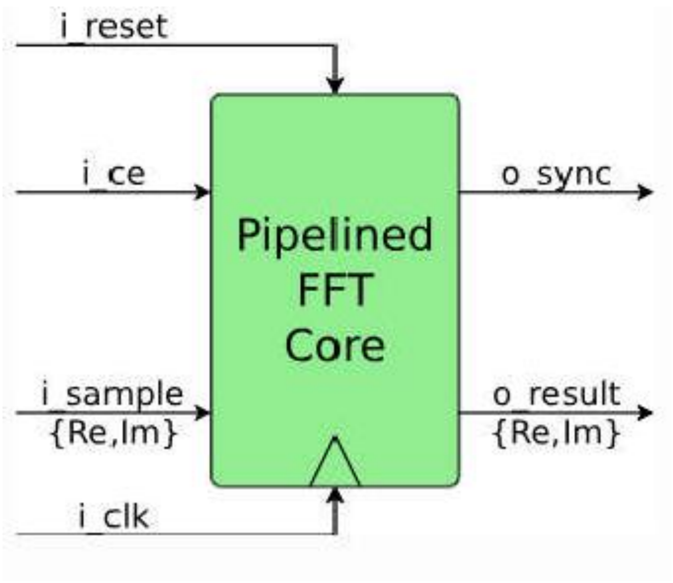


Figure 3.13: Block diagram of the generated core

The input and output ports of this core are:

i_clk: synchronous to the 6Mhz clock divider output.

i_reset: is a positive edge asynchronous reset signal.

i_ce: is a global CE signal. It is set to 1 on every clock where a valid new sample is available on the input.

i_sample: is actually a pair of values, both real and imaginary, stuffed into one signal bus. The real portion is placed in the upper bits, and the imaginary portion is placed in the lower or least significant bits.

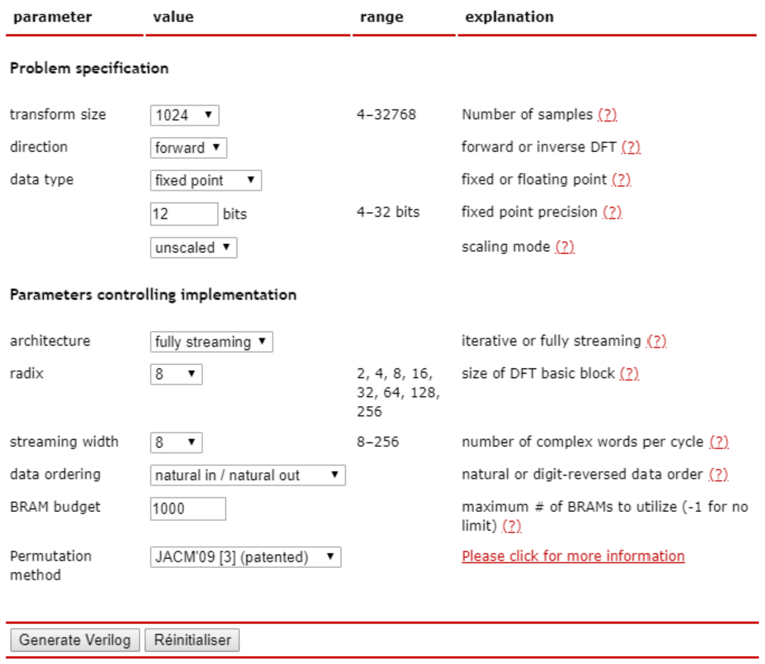
o_result: is the output of one FFT bin from the FFT. It is in the exact same format as i_sample.

o_sync: is the last output in the port list. This signal will be true when o_result contains the first output bin coming out of the FFT.

Unfortunately, despite the core being simple to interface with, it did not work properly [13].

3.7.5.3. Spiral core:

The Spiral DFT/FFT is a web-based IP Generator which automatically generates customized Fast Fourier Transform (FFT) IP cores in a synthesizable RTL Verilog. The user has control over variety of parameters that control the functionality and cost/performance tradeoffs such as area and throughput [20]. We selected the parameter that suits our design and generated it as shown in **Figure 3.14**,



parameter	value	range	explanation
Problem specification			
transform size	1024	4–32768	Number of samples (?)
direction	forward		forward or inverse DFT (?)
data type	fixed point		fixed or floating point (?)
	12 bits	4–32 bits	fixed point precision (?)
	unscaled		scaling mode (?)
Parameters controlling implementation			
architecture	fully streaming		iterative or fully streaming (?)
radix	8	2, 4, 8, 16, 32, 64, 128, 256	size of DFT basic block (?)
streaming width	8	8–256	number of complex words per cycle (?)
data ordering	natural in / natural out		natural or digit-reversed data order (?)
BRAM budget	1000		maximum # of BRAMs to utilize (-1 for no limit) (?)
Permutation method	JACM'09 [3] (patented)		Please click for more information

Figure 3.14 The web interface FFT core generator [20]

The block of the generated core is shown in **Figure 3.15**

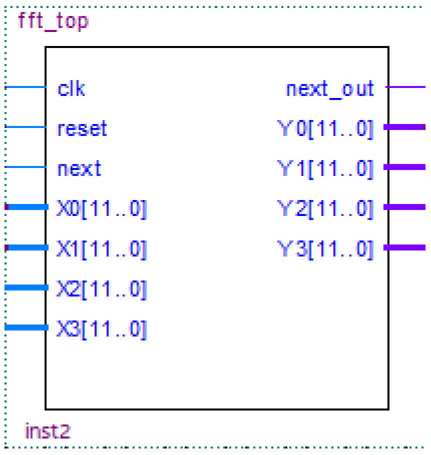


Figure 3.15: Block diagram of the spiral FFT core

The input/output ports of this core are:

clk: synchronous to the clock input.

reset: is a positive edge asynchronous reset signal.

next: this input (asserted high), is used to instruct the system that the input stream will begin on the following cycle.

$X_{0,1,2,3}$: is actually a pair of values of 12 bits, both real and imaginary, used to make the core work in parallel. The real portion is placed in the upper bits, and the imaginary portion is placed in the lower for example X_0 is the real part and X_1 is the imaginary part and so on.

$Y_{0,1,2,3}$: is the outputs from the FFT. It is in the exact same format as **$X_{0,1,2,3}$**.

Next_out: The output signal 'next_out' (also asserted high) indicates that the output vector will begin streaming out of the system on the following cycle.

The design uses a system of flag signals to indicate the beginning of the input and output data streams. The 'next' input (asserted high), is used to instruct the system that the input stream will begin on the following cycle. This system has a 'gap' of 512 cycles. This means that 512 cycles must elapse between the beginning of the input vectors.

The output signal 'next_out' (also asserted high) indicates that the output vector will begin streaming out of the system on the following cycle.

The system has a latency of 1373 cycles. This means that the 'next_out' will be asserted 1373 cycles after the user asserts 'next' [20].

Chapter 4

Implementation and Results

This chapter shows the implementation of our prototype Phasor measurement unit using FPGA. This chapter also describes and analyses the results captured using SignalTap II Logic Analyzer Tool. **Figure 4.1** depicts the system implemented**.** The Function Generator generates a sinusoidal signal with a 50Hz frequency. This signal is fed to one of the analog inputs of the on-board ADC.

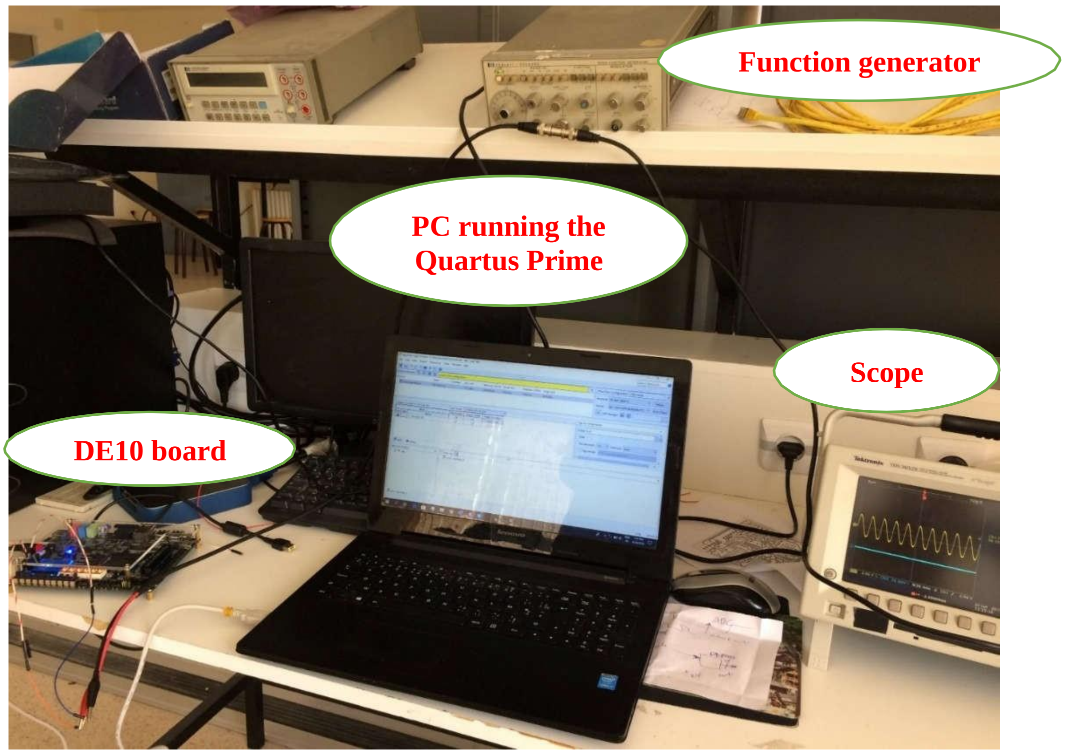
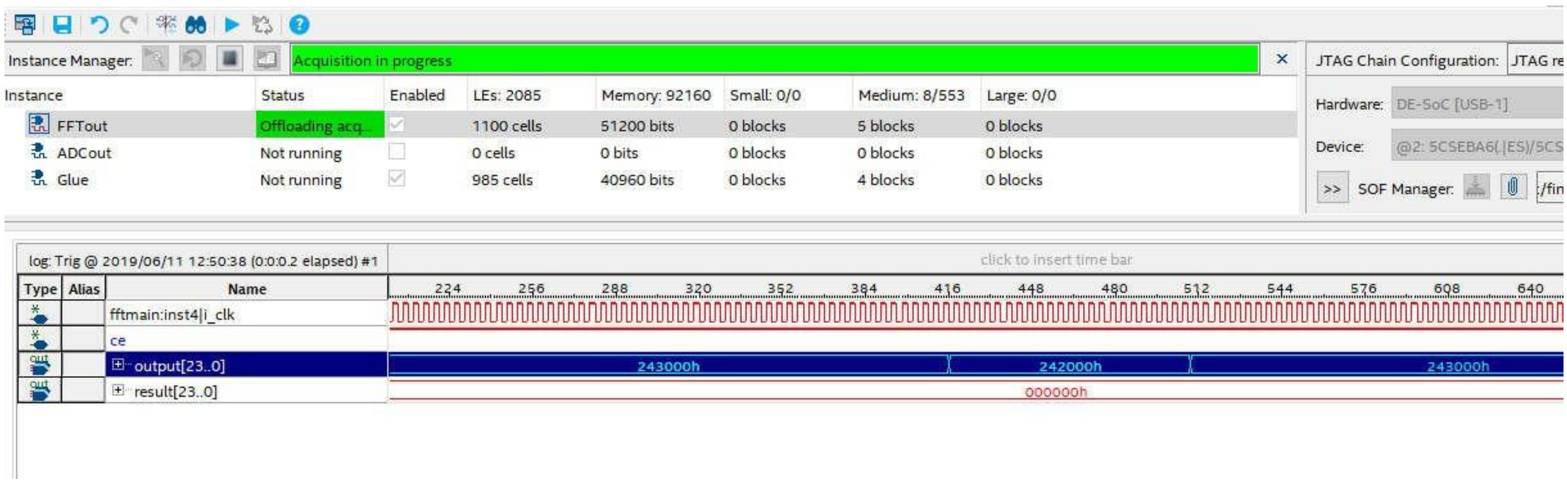


Figure 4.1: The experimental setup to test the PMU system

4.1. Testing the FFT hardware core

4.1.1. Open Source FFT core by Gisselquist Technology, LLC

Figure 4.2 shows a real-time testing of the first FFT core using SignalTap logic analyzer. It can be seen that the ADC is transmitting data through the signal ‘output’ to the core and the core’s chip enable (‘ce’) is turned on, but no data is coming out from the core i.e. ‘results’. This indicates that the core is not working since all interfacing conditions are satisfied and no output data is coming out.



Instance Manager: Acquisition in progress

JTAG Chain Configuration: JTAG re

Hardware: DE-SoC [USB-1]

Device: @2: 5CSEBA6(.|ES)/5CS

SOF Manager: :/fin

Instance	Status	Enabled	LEs: 2085	Memory: 92160	Small: 0/0	Medium: 8/553	Large: 0/0
FFTout	Offloading acq...	✓	1100 cells	51200 bits	0 blocks	5 blocks	0 blocks
ADCout	Not running		0 cells	0 bits	0 blocks	0 blocks	0 blocks
Glue	Not running	✓	985 cells	40960 bits	0 blocks	4 blocks	0 blocks

log: Trig @ 2019/06/11 12:50:38 (0:0:0.2 elapsed) #1

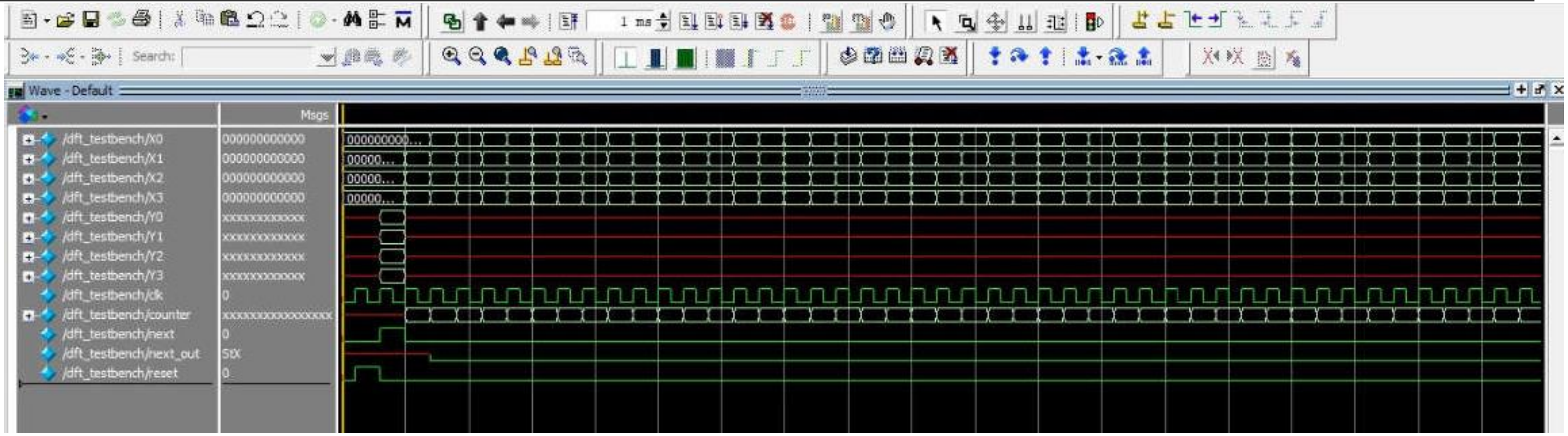
click to insert time bar

Type	Alias	Name	224 … 640
		fftmain:inst4|i_clk	
		ce	
		output[23..0]	243000h · 242000h · 243000h
		result[23..0]	000000h

Figure 4.2 Actual real-time testing of the open source FFT core

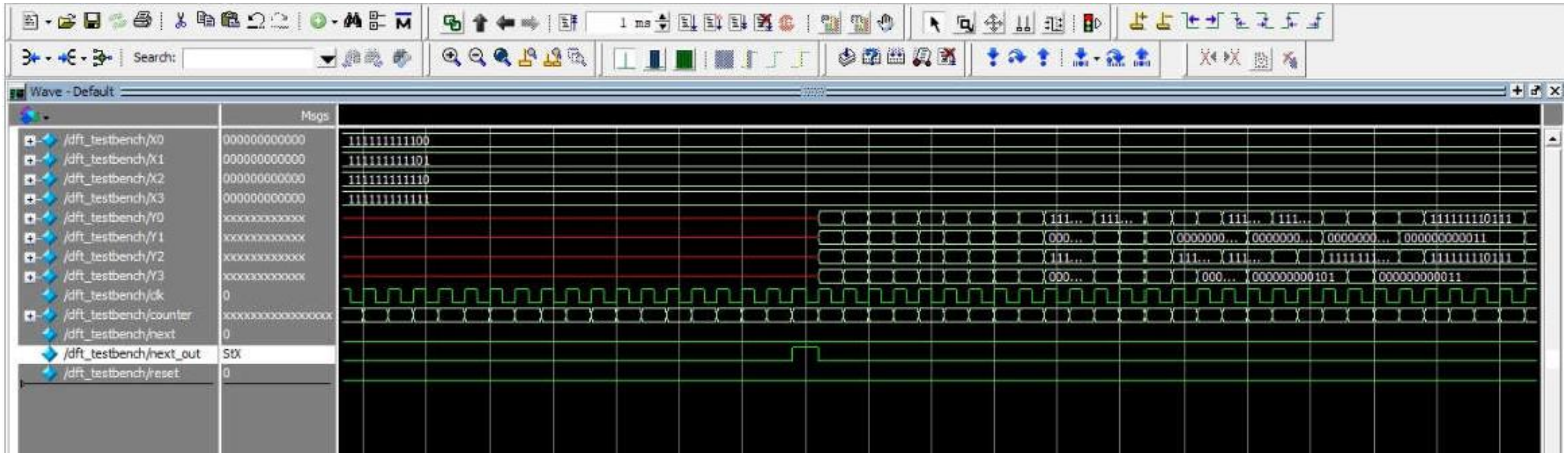
4.1.2. Spiral FFT Core Simluation

As shown in Figure 4.3, ModelSim was used to test the core through simulation. Figure 4.3 (a) shows the beginning of simulation where the signal ‘next’ is given a pulse to start the FFT computations. Figure 4.3 (b) shows the end of simulation where the signal ‘next_out’ is asserted high by the FFT core indicating the end of FFT computations. Since the simulation takes a large number of cycles, we cannot show the full length of the waveforms. As it can be noticed, the core is functional since there are results generated by the core for given input signals. Also it can noticed that this core is triggered once it receives a “next” signal which means that it begins processing the data when the “next” signal is one, when the data are processed the core set the “next_out” signal to one and the data is available on the “Y” signals.



```
Wave - Default
                              Msgs
/dft_testbench/X0             000000000000
/dft_testbench/X1             000000000000
/dft_testbench/X2             000000000000
/dft_testbench/X3             000000000000
/dft_testbench/Y0             xxxxxxxxxxxx
/dft_testbench/Y1             xxxxxxxxxxxx
/dft_testbench/Y2             xxxxxxxxxxxx
/dft_testbench/Y3             xxxxxxxxxxxx
/dft_testbench/clk            0
/dft_testbench/counter        xxxxxxxxxxxxxxxx
/dft_testbench/next           0
/dft_testbench/next_out       StX
/dft_testbench/reset          0
```

(a)



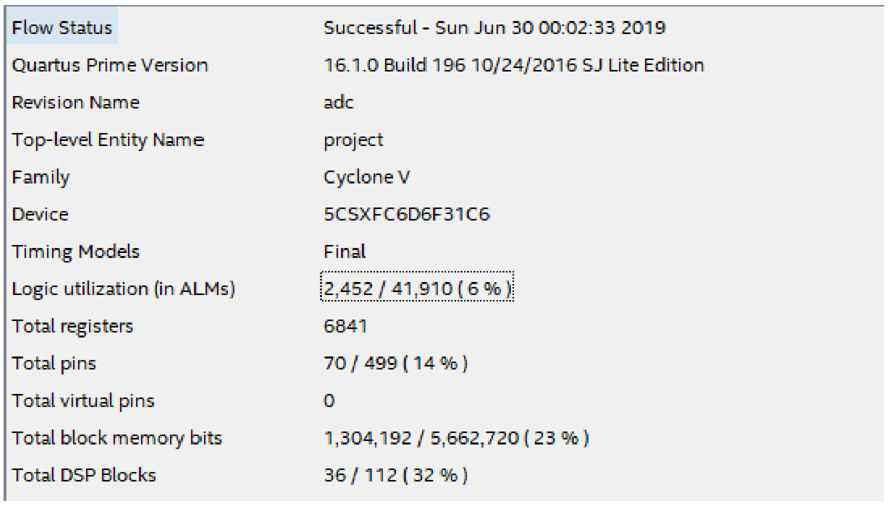
```
Wave - Default
                              Msgs
/dft_testbench/X0             000000000000    111111111100
/dft_testbench/X1             000000000000    111111111101
/dft_testbench/X2             000000000000    111111111110
/dft_testbench/X3             000000000000    111111111111
/dft_testbench/Y0             xxxxxxxxxxxx
/dft_testbench/Y1             xxxxxxxxxxxx
/dft_testbench/Y2             xxxxxxxxxxxx
/dft_testbench/Y3             xxxxxxxxxxxx
/dft_testbench/clk            0
/dft_testbench/counter        xxxxxxxxxxxxxxxx
/dft_testbench/next           0
/dft_testbench/next_out       StX
/dft_testbench/reset          0
```

(b)

Figure 4.3 Simulation of the Spiral FFT core: (a) the beginning of simulation (b) the end of simulation

4.2.Project Compilation Report

The compilation of the whole system was successful. The summary of the compilation is shown in **Figure 4.4**. The entire system uses 6% of the total logic elements, 6841 registers, 14% of the total pins and 23% of the total memory bits, also as we can see the designed system uses 32% of DSP blocks, which can reflect that the FFT core is synthesized correctly.

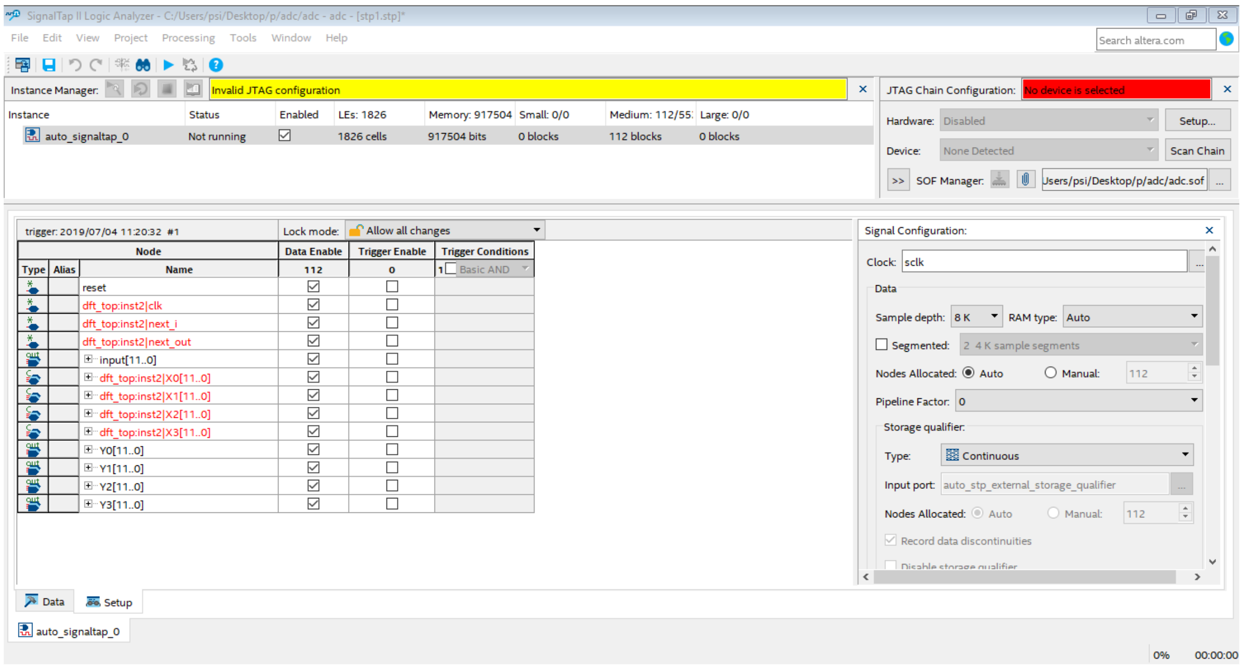


Flow Status	Successful - Sun Jun 30 00:02:33 2019
Quartus Prime Version	16.1.0 Build 196 10/24/2016 SJ Lite Edition
Revision Name	adc
Top-level Entity Name	project
Family	Cyclone V
Device	5CSXFC6D6F31C6
Timing Models	Final
Logic utilization (in ALMs)	2,452 / 41,910 (6 %)
Total registers	6841
Total pins	70 / 499 (14 %)
Total virtual pins	0
Total block memory bits	1,304,192 / 5,662,720 (23 %)
Total DSP Blocks	36 / 112 (32 %)

Figure 4.4 Compilation Summary

4.3.Results of the implemented system

After the design has been compiled, the SignalTap II Logic Analyzer Tool is launched and required nodes and clocks are added to test the design in real time as shown in the bellow **Figure 4.5.**



Instance Manager: Invalid JTAG configuration

Instance	Status	Enabled	LEs: 1826	Memory: 917504	Small: 0/0	Medium: 112/55:	Large: 0/0
auto_signaltap_0	Not running	☑	1826 cells	917504 bits	0 blocks	112 blocks	0 blocks

JTAG Chain Configuration: No device is selected

Hardware: Disabled

Device: None Detected

SOF Manager: Users/psi/Desktop/p/adc/adc.sof

trigger: 2019/07/04 11:20:32 #1 Lock mode: Allow all changes

Node			Data Enable	Trigger Enable	Trigger Conditions
Type	Alias	Name	112	0	1 ☐ Basic AND
		reset	☑	☐	
		dft_top:inst2|clk	☑	☐	
		dft_top:inst2|next_i	☑	☐	
		dft_top:inst2|next_out	☑	☐	
		input[11..0]	☑	☐	
		dft_top:inst2|X0[11..0]	☑	☐	
		dft_top:inst2|X1[11..0]	☑	☐	
		dft_top:inst2|X2[11..0]	☑	☐	
		dft_top:inst2|X3[11..0]	☑	☐	
		Y0[11..0]	☑	☐	
		Y1[11..0]	☑	☐	
		Y2[11..0]	☑	☐	
		Y3[11..0]	☑	☐	

Signal Configuration:

Clock: sclk

Data

Sample depth: 8 K RAM type: Auto

☐ Segmented: 2 4 K sample segments

Nodes Allocated: [✓ Auto] Manual: 112

Pipeline Factor: 0

Storage qualifier:

Type: Continuous

Input port: auto_stp_external_storage_qualifier

Nodes Allocated: Auto Manual: 112

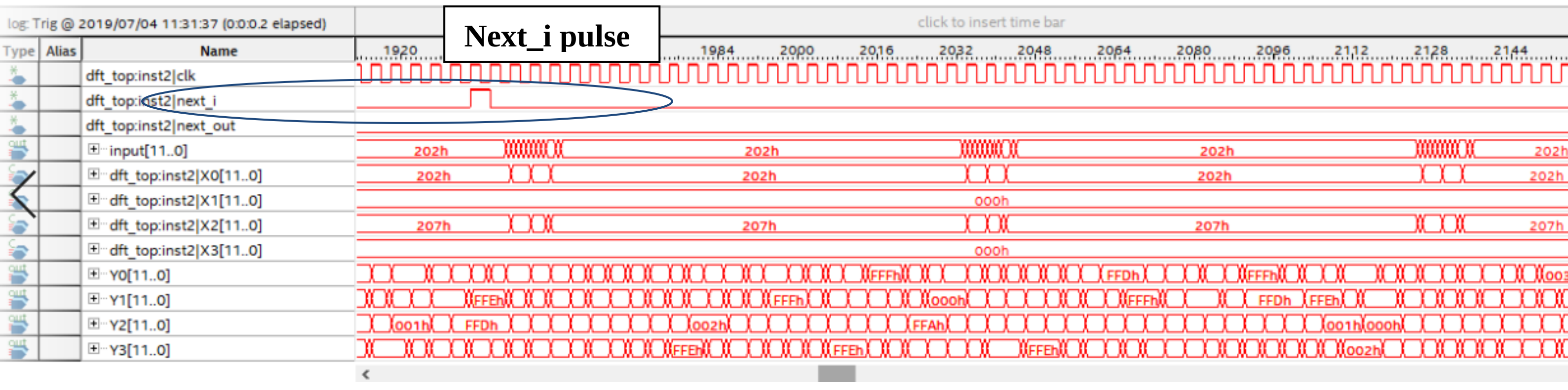
☑ Record data discontinuities

☐ Disable storage qualifier

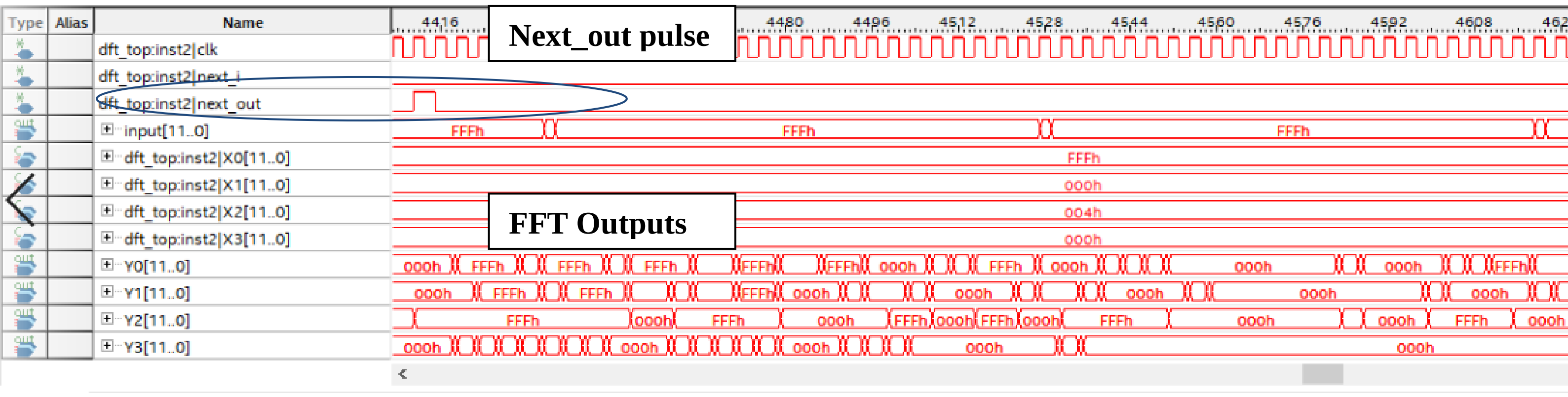
Figure 4.5: Adding design signals to Signal Tap II Logic Analyzer Tool

After the programmer and the device have been verified, the design is recompiled, it can be remarked that the design took longer to recompile since the tool took enormous resources for testing.

The generated waveforms of the tested prototype is shown in **Figure 4.6**



(a)



(b)

Figure 4.6: Waveforms of the tested design
(a)the beginning of simulation (b) the end of simulation

As seen in the waveform generated in **Figure 4.6** We are getting some data inputs from the interfaced logic which is connected to the storage block which is also connected to the data acquisition unit (ADC), hence the data from the ADC is well received.

Also, as it can be seen we are getting a pulse to the “next” signal in order to trigger the core. Hence the conditions to turn the FFT core on are satisfied.

At last it is noticed that after a certain time after a “next” pulse is generated there is a “next_out” signal associated with Y’s data i.e. the FFT core is working.

Conclusion

As the PMU technology has to provide estimations with high degree of accuracy, the hardware platform that will be used for this phasor estimation algorithm should have a high degree of determinism. In this respect, the use of FPGA to implement the PMU algorithm is highly justified.

The FPGA-based PMU was based on the Cyclone V Assembly, VHDL, Verilog HDL. These platforms are open source software; hence, the overall implementation is low cost and can be easily configured.

It can be concluded that SoC-based PMU was designed and implemented and the objectives of this work have been successfully met and they are as follows:

- Implementation of a Phasor Measurement Unit System based on FPGA
- Apply every single knowledge of Hardware design and computer engineering that were acquired during the past five years.

Effort and time were spent to debug software bugs and hardware problems to improve the system and make it operational. The implementation of the SoC-based PMU System was carried and the final prototype was fully functional. Like any other project, this work can be enhanced and improved by adding some features, mainly:

- The project was designed using a small single-phase function generator system for study purpose, for wide area monitoring purpose, the prototype can be implemented with some modifications, for example, by testing the system with 3 phases, increasing the sampling rate for better accuracy, attaching a GPS module in the system for time stamping etc.
- In the future scope of research, the prototype is to be used in studying the effects of faults on the Phasor estimates, power monitoring system, and other benefits that could be gained from phasor measurement unit.

References

[1] IEEE Standard for Synchrophasor Measurements for Power Systems, IEEE Std. C37.118.1-2011. [Online]. Available: http://standards.ieee.org/findstds/standard/C37.118.1-2011.html.

[2] Paolo Castello, Algorithms for the synchrophasor measurement in steady-state and dynamic conditions, University of Cagliari, March 2014

[3] IEEE Standard for Synchrophasor Data Transfer for Power Systems,IEEE Std. C37.118.2- 2011. [Online]. Available: http://standards.ieee.org/findstds/standard/C37.118.2- 2011.html

[4] .G. Missout and P. Girard, Measurement of bus voltage angle between montreal and SEPT-ILES, IEEE Trans. Power App. Syst., vol. PAS-99, no. 2, pp. 536-539, Mar. 1980

[5] G. Missout, J. Beland, G. Bedard, and Y. Lafleur, Dynamic measurement of the absolute voltage angle on long transmission lines, IEEE Trans. Power App. Syst., vol. PAS-100, no. 11, pp. 4428-4434, Nov. 1981.

[6] P. Bonanomi, Phase angle measurements with synchronized clocks principle and applications, IEEE Trans. Power App. Syst., vol. PAS-100, no. 12, pp. 5036-5043, Dec. 1981.

[7] A. G. Phadke and J. S. Thorp, History and applications of phasor measurements, in Proc. IEEE PES PSCE, 2006, pp. 331-335.

[8] A. G. Phadke, Synchronized phasor measurements A historical overview, in Proc. IEEE/PES Transmiss. Distrib. Conf. Exhib. Asia Pacific, Oct. 6-10, 2002, vol. 1, pp. 476-479.

[9] A. G. Phadke and J. S. Thorp, Synchronized Phasor Measurements and Their Applications. New York, NY, USA: Springer-Verlag, 2008

[10] P. Denys, C. Counan, L. Hossenlopp, and C. Holweck, Measurement of voltage phase for the French future defence plan against losses of synchronism, IEEE Trans. Power Del., vol. 7, no. 1, pp. 62-69, Jan. 1992..

[11] A.Agarwal, N.Verma, H. Tiwari, J. Singh, Varun Maheshwari Design and Development of Phasor Measurement Unit on FPGA,

[12] B. Kasztenny and M. Adamiak, Implementation and performance of synchrophasor function within microprocessor based relays, in Proc. 61st Annu. Georgia Tech. Protect. Relaying Conf., Atlanta, GA, USA, May 2-4, 2007, pp. 1-43.

[13] D. M. Laverty, D. J. Morrow, R. Best, and P. A. Crossley, Performance of phasor measurement units for wide area real-time control, in Proc. IEEE PES Gen. Meeting, Jul. 2630, 2009, pp. 1-5.

[14] Open source FFT core, [Online]: https://zipcpu.com/dsp/2018/10/02/fft.html. Accessed on 30-06-2019

[15] R. Wisniewski, in *Synthesis of compositional microprogram control units for programmable devices*, Zielona Góra: University of Zielona Góra, 2009, p. 153.

[16] Intel FPGA, DE10 Standard User Manual, 2017

[17] Mike Pridgen, "Tutorial for Quartus" SignalTap II Logic Analyzer",

http://www.mil.ufl.edu/4712/docs/SignalTa p_Tutorial.pdf.

[18] Linear Technology, LTC2308 ADC datasheet, 2007.

[19] Spiral DFT core generator, [Online]: https://www.spiral.net/hardware/dftgen.html. Accessed on 30-06-2019

Appendix

Using Signal Tap II Logic Analyzer

1- First we need to have a complete Compiled project, so we set pin assignments and we should have no error.

2- Select SignalTap II Logic Analyzer: After compilation we have to ensure the JTAG programmer (USB-Blaster) is connected between the board and the PC. We open SignalTap II Logic Analyzer by selecting “Tools | SignalTap II Logic Analyzer” or we can open pre-existing SignalTap II Logic Analyzer file (*.stp) from “File | Open”.
After that we Select Hardware, If not appear USB-Blaster, we click Setup to select the programmer. Shown in **Figure I**

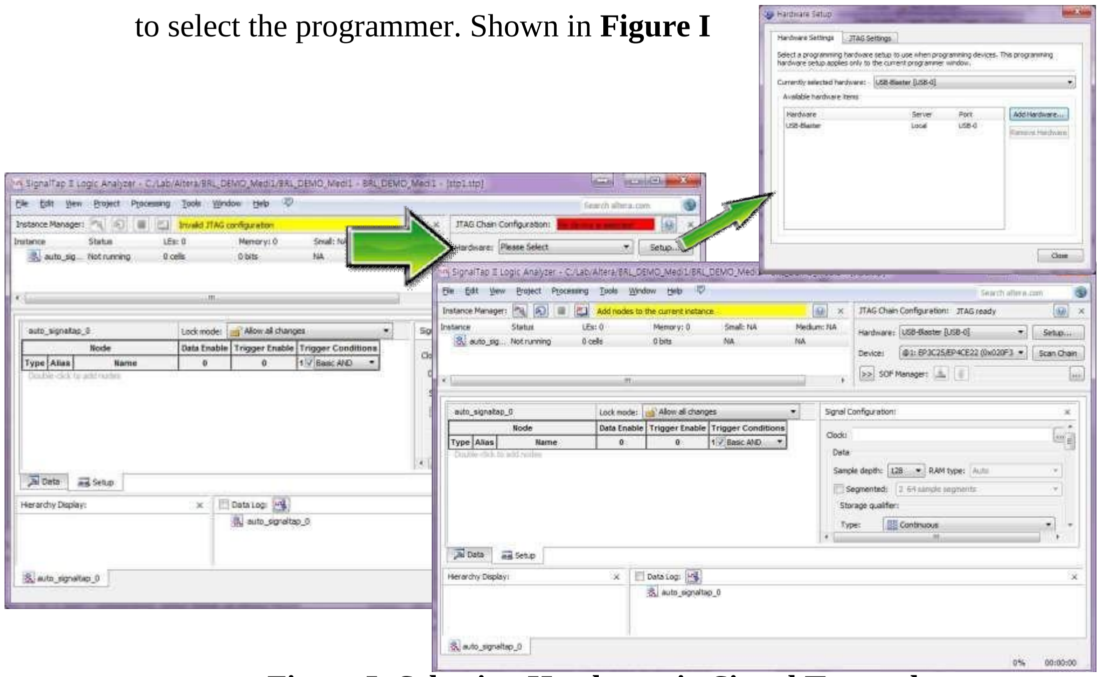


Figure I: Selecting Hardware in Signal Tap tool

3- Add nodes to be analyzed: we double click to add necessary nodes, click List to view nodes, then Add nodes to be analyzed Select nodes. As shown in **Figure II**

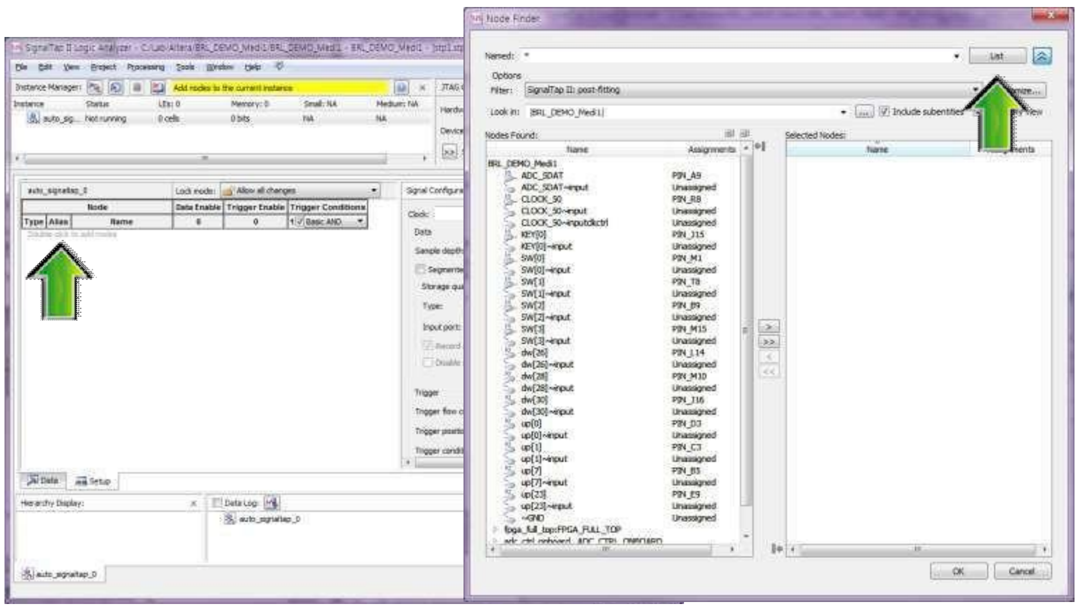
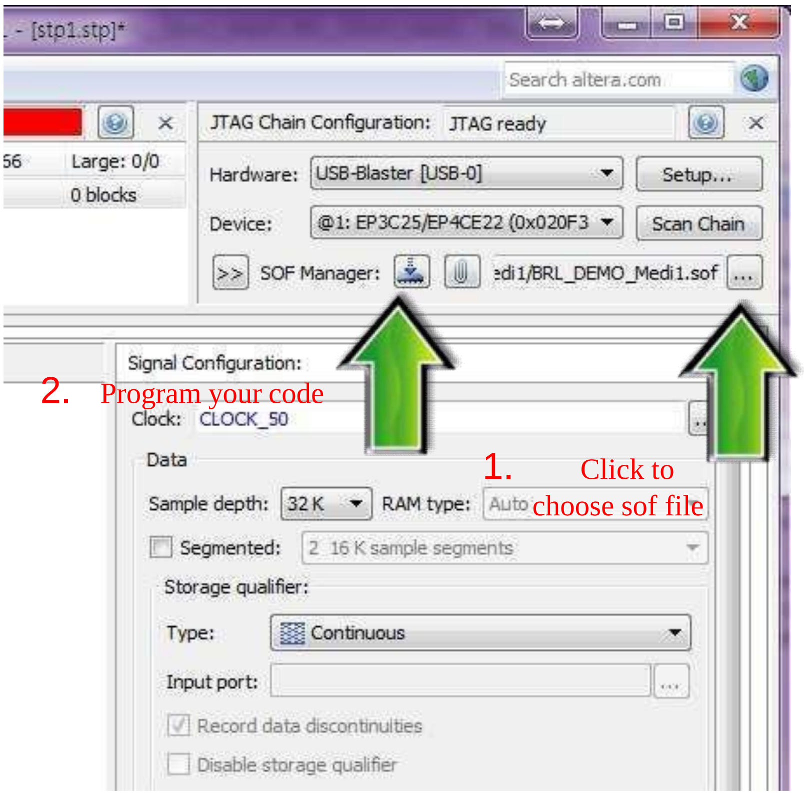

Node Finder
Named: *
List
Filter: SignalTap II: post-fitting
Look in: |BRL_DEMO_Medi1|
Include subentities
Nodes Found:
Selected Nodes:
OK
Cancel
SignalTap II Logic Analyzer
auto_signaltap_0
Double-click to add nodes

Figure II: Adding your desired nodes to test

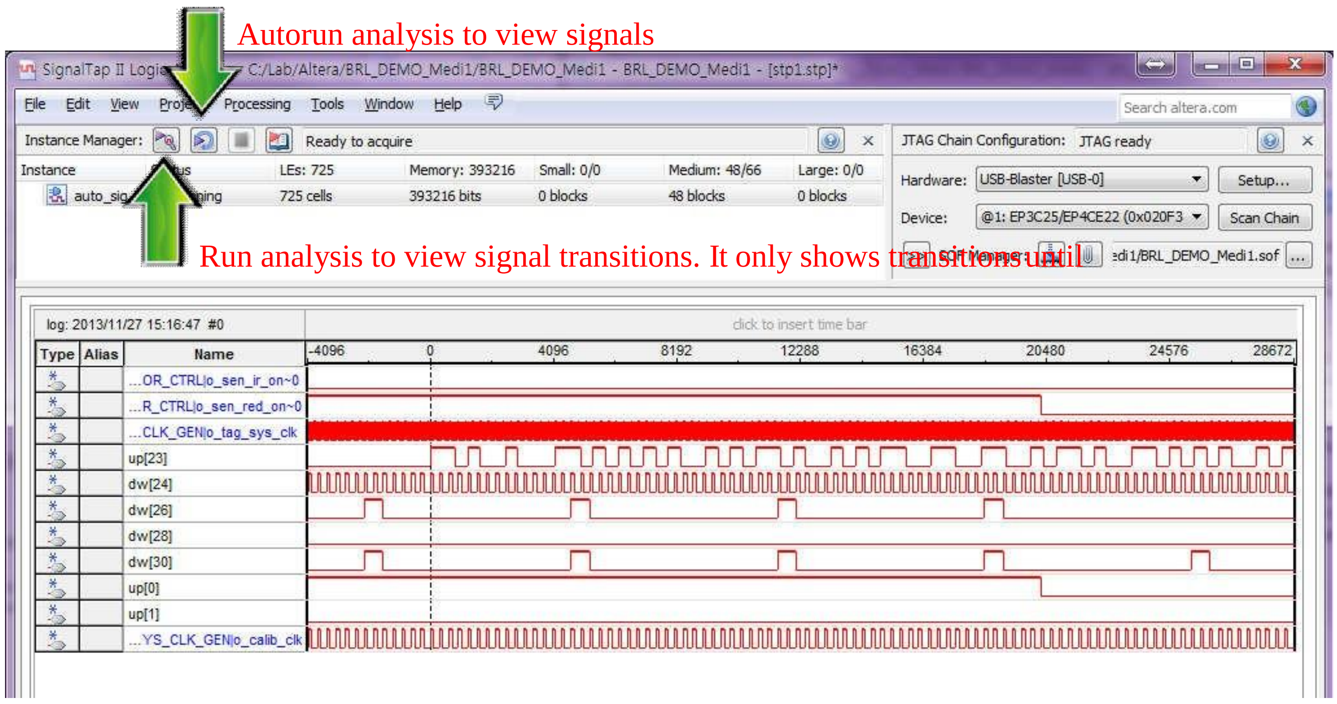
4- Select proper clock Basically, the clock need to be set to FPGA clock.
5- Choose Sample depth depending on RAM attached to FPGA
6- After setting for SignalTap II Logic Analyzer, you need to compile your project again.Select „sof‟ file and program your code Select „sof‟ file to be downloaded first.Program your project on the board. As shown in **Figure III**


- [stp1.stp]*
Search altera.com
2. Program your code
1. Click to choose sof file
Setup...
Scan Chain
Record data discontinuities
Disable storage qualifier

JTAG Chain Configuration: JTAG ready
Large: 0/0
0 blocks
Hardware: USB-Blaster [USB-0]
Device: @1: EP3C25/EP4CE22 (0x020F3
SOF Manager: edi1/BRL_DEMO_Medi1.sof
Signal Configuration:
Clock: CLOCK_50
Data
Sample depth: 32 K RAM type: Auto
Segmented: 2 16 K sample segments
Storage qualifier:
Type: Continuous
Input port:

Figure III Selecting the clock and programming the tool

7- Run analysis to view signals as shown in **Figure IV**


Autorun analysis to view signals
Run analysis to view signal transitions. It only shows transitions until
File Edit View Project Processing Tools Window Help
Search altera.com
Setup...
Scan Chain
click to insert time bar

SignalTap II Logic Analyzer - C:/Lab/Altera/BRL_DEMO_Medi1/BRL_DEMO_Medi1 - BRL_DEMO_Medi1 - [stp1.stp]*
Instance Manager: Ready to acquire
JTAG Chain Configuration: JTAG ready

Instance	Status	LEs: 725	Memory: 393216	Small: 0/0	Medium: 48/66	Large: 0/0
auto_signaltap_0	Not running	725 cells	393216 bits	0 blocks	48 blocks	0 blocks

Hardware: USB-Blaster [USB-0]
Device: @1: EP3C25/EP4CE22 (0x020F3
SOF Manager: edi1/BRL_DEMO_Medi1.sof

log: 2013/11/27 15:16:47 #0

Type	Alias	Name
		...OR_CTRL|o_sen_ir_on~0
		...R_CTRL|o_sen_red_on~0
		...CLK_GEN|o_tag_sys_clk
		up[23]
		dw[24]
		dw[26]
		dw[28]
		dw[30]
		up[0]
		up[1]
		...YS_CLK_GEN|o_calib_clk

Time axis: -4096, 0, 4096, 8192, 12288, 16384, 20480, 24576, 28672

Figure IV Running the Analysis on Signal