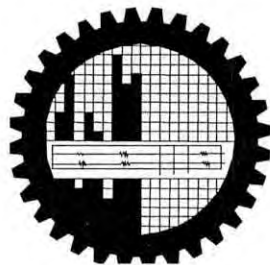


Design of an FPGA Based High Precision Digital Energy and Power Quality Meter

A thesis submitted to the Department of Electrical and Electronic Engineering (EEE)
of
Bangladesh University of Engineering and Technology (BUET)
in partial fulfillment of the requirement for the degree of
**MASTER OF SCIENCE IN ELECTRICAL AND ELECTRONIC
ENGINEERING**

by
Mohammad Ashraful Anam



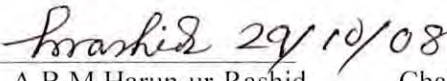


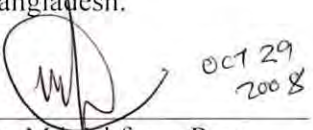
**DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
(EEE)
BANGLADESH UNIVERSITY OF ENGINEERING AND TECHNOLOGY
(BUET)
2008**



#107250#

The thesis titled "Design of an FPGA Based High Precision Digital Energy and Power Quality Meter" submitted by Mohammad Ashraful Anam, Roll No.: 040306115P, Session: April 2003 has been accepted as satisfactory in partial fulfillment of the requirement for the degree of MASTER OF SCIENCE IN ELECTRICAL AND ELECTRONIC ENGINEERING on October 29, 2008.

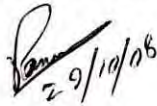
BOARD OF EXAMINERS

1. 
 Dr. A.B.M Harun-ur-Rashid Chairman
 Professor (Supervisor)
 Department of Electrical and
 Electronic Engineering
 BUET, Dhaka—1000,
 Bangladesh.
2. 
 Dr. Aminul Hoque Member (Ex-Officio)
 Professor and Head
 Department of Electrical and
 Electronic Engineering
 BUET, Dhaka—1000,
 Bangladesh.
3. 
 Dr. Kazi Mujibur Rahman Member
 Professor
 Department of Electrical and
 Electronic Engineering
 BUET, Dhaka—1000,
 Bangladesh.
4.  OCT 29
2008
 Dr. Md. Yshfaqur Raza Member
 Associate Professor (External)
 Department of Electrical and
 Electronic Engineering
 East West University, Dhaka
 Bangladesh

Declaration

I hereby declare that this thesis or any part of it has not been submitted elsewhere for the award of any degree or diploma.

Signature of candidate



(Mohammad Ashraful Anam)
(Roll: 040306115P)

Dedication

To My Parents

Table of Contents

Declaration	ii
Dedication.....	iii
Table of Contents	iv
List of Figures.....	vii
List of Abbreviations.....	x
Acknowledgement.....	xi
Abstract.....	xii
Chapter 1 Introduction	1
1.1 Background and Present State of the Problem	1
1.2 Literature Review	2
1.3 Thesis objective	4
1.4 Organization of the Thesis.....	4
Chapter 2 Energy Meters.....	6
2.1 Basic Idea about energy Meter	6
2.2 Electromechanical meters.....	6
2.3 Electronic Meters.....	7
2.4 Standard Construction of an energy meter	8
Chapter 3 The Proposed Meter.....	10
3.1 The Architecture of the Proposed Meter	10
3.2 Operating Principle.....	20
3.3 Design issues	21
Chapter 4 Implementation	23

4.1 Introduction	23
4.2 FPGA Structure	23
4.2.1 Logic Blocks.....	24
4.2.2 Interconnection wires and switches.....	25
4.3 Design Procedure.....	26
4.4 Device Selection.....	28
4.5 Design Architecture.....	30
4.5.1 Analog to Digital Converter	31
4.6 Data Acquisition FSM Block	33
4.6.1 Interfacing block.....	33
4.6.2 Storage Block	39
4.7 Soft Processor.....	41
4.8 Calculation Block:.....	42
4.9 DFT/FFT Block.....	45
4.9.1 Fourier Transform	45
4.9.2 Discrete Fourier Transform	45
4.9.3 Fast Fourier Transform.....	47
4.10 Acceleration Unit.....	49
4.10.1 SDRAM.....	49
4.10.2 Dual Port on-chip RAM	50
4.10.3 Hardware Multiplier	51
4.10.4 RAM Arbitrator.....	52
4.10.5 Other blocks	53
4.11 Storage Block	54

4.12 Meter Specification	54
Chapter 5 Simulation and Results.....	59
5.1 Necessary Files and Tools	59
Chapter 6 Measured Data	65
6.1 Calibration	65
6.2 Measured Forms	66
6.3 Measured Data.....	67
6.3.1 Voltage Variation	68
6.3.2 Power Factor response.....	68
6.3.3 Harmonic Distortion.....	69
6.4 Comparison.....	70
Chapter 7 Conclusion and Future Work	71
7.1 Conclusion.....	71
7.2 Future Work.....	71
References	73
Appendix A	77
Appendix B.....	79
Appendix C.....	82
Appendix D	89
Appendix E.....	92
Appendix F	94
Appendix G	96

List of Figures

Figure 2.1 Block diagram of a typical 3 phase energy meter	8
Figure 3.1 Typical voltage divider network	10
Figure 3.2 Typical AC current step down network	11
Figure 3.3 Sequential or multiplexed sampling of current and voltage.	11
Figure 3.4 Simultaneous or aligned sampling of voltage and current.	12
Figure 3.5 Nios processor structure implemented in the FPGA	13
Figure 3.6 Simpler implementation using direct interface to ADC	13
Figure 3.7 Modified implementation with interfacing block	14
Figure 3.8 Final implementation with RAM and hardware multiplier	15
Figure 3.9 Cascadable Dedicated multiplier Register Transfer Level (RTL) Circuit	16
Figure 3.10 Block diagram of proposed meter	16
Figure 3.11 Block diagram of FFT processor of the proposed meter	18
Figure 3.12 Flow diagram of energy calculation of the proposed meter	19
Figure 4.1 General structure of an FGPA	24
Figure 4.2 Typical logic block of an FPGA	25
Figure 4.3 Logic Block Pin Locations for a 4 input LUT	25
Figure 4.4 Switch box interconnect topology in an FPGA	26
Figure 4.5 Typical CAD flow in FPGA design process	27
Figure 4.6 Cyclone II FPGA chip in DE2 board	29
Figure 4.7 Block diagram of the DE2 board used in the proposed meter implementation	30
Figure 4.8 Hardware architecture of the proposed meter consisting of modular components	30
Figure 4.9 Generation of aliasing signal at lower sampling frequency	31

Figure 4.10 AD73360 in the proposed implementation	32
Figure 4.11 State Diagram of the Finite State Machine (FSM) interfacing module	34
Figure 4.12 16 Bit structure of ADC control word	35
Figure 4.13 Signaling gates for write complete and read complete operation	38
Figure 4.14 Partial RTL of the interfacing block connected to GPIOs	39
Figure 4.15 Storage block of the interfacing module	40
Figure 4.16 Flow diagram of storage block of the interfacing module	41
Figure 4.17 AC Signal (Voltage and Current)	44
Figure 4.18 Sampled Voltage and Current Signal	44
Figure 4.19 Block diagram of hardware accelerated FFT calculation	48
Figure 4.20 FFT block synthesized using Verilog	48
Figure 4.21 SDRAM IO Cell	50
Figure 4.22 PLL required for SDRAM	50
Figure 4.23 Synthesized RAM Block	51
Figure 4.24 Partial section of the on-chip memory	51
Figure 4.25 Hardware Multiplier	52
Figure 4.26 RAM Arbitrator	52
Figure 4.27 Accelerator Subroutine State Machine	53
Figure 4.28 CPU Accelerator Interface Instance	53
Figure 4.29 Block Diagram of Nios and FSM interconnect	57
Figure 4.30 Physical implementation of the entire design	58
Figure 5.1 State 0. ADC reset.	60
Figure 5.2 State 1. Turn on channel 1 and 2.	60
Figure 5.3 State 2. Turn on channel 3 and 4.	61
Figure 5.4 State 3. Channel 5 and 6 turned on.	61

Figure 5.5 State 4. Setting sampling rate 1kHz.	62
Figure 5.6 State 5. Setting non inverted mode of ADC.	62
Figure 5.7 State 6. Start data mode.	63
Figure 5.8 State 7. Readdata state	63
Figure 6.1 Input Voltage Data	66
Figure 6.2 Signal at Channel 1 (voltage channel) of the ADC	66
Figure 6.3 Sampled voltage data received from ADC	67
Figure 6.4 Half cycle discrete voltage data as received from ADC	67
Figure 6.5 Meter Accuracy, voltage variation testing	68
Figure 6.6 Meter Accuracy, power factor testing	69
Figure 6.7 THD output from the meter	70
Figure 6.8 Performance comparison of proposed meter and an induction meter	70

List of Abbreviations

ASIC	Application Specific Integrated Circuit
CAD	Computer Aided Design
CT	Current Transformer
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
EPROM	Erasable Programmable Read Only Memory
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
HD	Harmonic Distortion
HDL	Hardware Description Language
LCD	Liquid Crystal Display
LE	Logic Elements
LUT	Look Up Table
GCC	GNU C Compiler
GNU	GNU's Not Unix
IEC	International Electrotechnical Commission
PT	Potential Transformer
RAM	Random Access Memory
RMS	Root Mean Square
RTC	Real Time Clock
SoC	System on a Chip
THD	Total Harmonic Distortion
VLSI	Very Large Scale Integration

Acknowledgement

I would like to express my profound and sincere gratitude to my supervisor Dr. A.B.M Harun-ur-Rashid, Professor, Department of Electrical and Electronic Engineering (EEE), Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh, whose patient guidance and encouraging attitude have motivated me much to have this thesis materialized.

I would also like to thank Professor Dr. Aminul Hoque, Head of the Department of Electrical and Electronic Engineering (EEE), BUET, for providing me the lab facilities.

I am also grateful to all the members of my family especially to my father and mother for their cooperation.

Finally, I am grateful to the Almighty for giving me the strength and courage to complete this thesis.

Abstract

Traditional analog energy meters are unable to cope with the rapidly changing load and cannot measure energy consumed by high frequency harmonic contents of the power. Single phase digital energy meters have been developed for this purpose but these use separate energy calculation chip and other peripheral chips. These make the meter large in size and costly. Besides, these meters are unable to measure the quality of power. This thesis presents a complete digital design of a three phase energy meter in a single Field Programmable Gate Array (FPGA) chip with additional capability to measure the quality of the power. The proposed energy meter measures all the harmonic contents of the power including the fundamental component. As a result the accuracy of the meter is very high even in the presence of harmonics in the power grid. Having a single chip FPGA design makes it cost effective and at the same time achieves additional advantages like less power consumption, less space and components requirements. This also makes the entire system programmable, reconfigurable and upgradeable based on changing requirements at any point in future even after the meters have been deployed at user premises. The energy meter is further optimized with fast on-chip memory and parallel path processing so that it will be able to perform all calculations including voltage measurement, current measurement, power measurement, phase difference measurement in real-time. An on-chip Fast Fourier Transform (FFT) processor is also designed to calculate and display the third harmonic distortion. Design synthesis of the meter is done in Verilog Hardware Descriptive Language (HDL) and the design tool used is Quartus II. The meter is implemented on Altera DE2 board containing Cyclone II FPGA chip. A 16 bit 6 channel simultaneous sampling A/D converter AD73360 is used as the only external component in the meter. The energy meter operates at 3.3 V and draws its power from the power line. The entire design required 8317 logic elements, 3493 dedicated logic register, 32 embedded multiplier, 4 digital PLL, 388 K memory units and 1 MB SDRAM memory. Finally the performance of the meter is compared with a traditional analog meter. While extreme precision was not achieved due to lack of external component precision, the meter was able to achieve 0.2 class.



Chapter 1

Introduction

1.1 Background and Present State of the Problem

Electromechanical induction meters whose operation is based on counting the rotation of a disk, has been the predominant energy measurement method for decades. These analog meters are unable to perform properly with the changing loads of today which comprise of non sinusoidal inputs. However recent advancement on measurement technology and very large scale integration (VLSI) technologies has lead smaller smart meters that are able to perform better than these conventional analog meters. These digital energy meters use a microcontroller for control purpose, a separate energy calculation chip for energy measurements and other peripheral chips. These make the meter large in size and costly. Again, the calculations of RMS voltage, current, power and energy require a number of multiplication, addition and division operations. Current generation of microcontroller based design is able to perform only a limited number of these mathematical operations per second. As a result these energy meters are used mostly for single phase energy measurement. Again, additional features like power quality analysis require a lot of computational power that cannot be performed with these types of designs. To implement signal analysis features as well as energy measurement features within a single hardware implementation, it is required to reduce computational time. The computation time can be minimized by using hardware multipliers that requires less clock cycles than a microcontroller implementation. For signal analysis all sine and cosine calculations can be pre-calculated which reduces calculations significantly. Moreover the whole system has to be reduced to a single chip in order to make the system cost effective.

1.2 Literature Review

The earliest energy measurement technique applied to meters is the magnetic flux based single phase induction meters where a disk rotates based on the intensity of the flux produced by voltage and current [1,2]. This technique is capable of providing energy measurement of considerable accuracy but only for sinusoidal voltage and currents. A comparative analysis of the performances of both a traditional induction meter and various types of other commercial meters in the presence of harmonic distortion reveals this scenario [3]. In accordance with current standards, these meters are designed to operate in sinusoidal conditions and their performance is not tested generally in the presence of harmonic distortion [4-7]. However, with the increase of pollution levels in power systems, the same meters may be used even in the presence of distorted voltages and/or currents; in such cases, their accuracy is very different from the nominal conditions, and the various meters may lead to different measurements of energy for the same load conditions. Electric energy meters have been always designed to account energy under sinusoidal conditions and therefore no uncertainty specification was provided outside these conditions. This was considered acceptable when voltage and current distortion levels were low and the old, well known induction meters were used. In fact, under these conditions, the measurement errors caused by distortion did not, generally, penalize the customers [8]. The present situation is, however, quite different: distortion is not negligible any longer, especially in low voltage systems, and the modern electronic energy meters can be much more sensitive to distortion [9-16] than the induction ones, depending on the implemented measurement algorithm.

To remedy some of the drawbacks inherent in analog meters, digital meters were introduced. These meters operate by continuously measuring the instantaneous voltage and current. The voltage and current is constantly sampled at a certain rate [17,18]. The voltage is dropped down to an acceptable level through a potential transformer and current usually through a current transformer.

Multiplying the two instantaneous values give instantaneous electrical power which is then integrated against time to give energy in kilowatts.

The digital meter usually comprises of integrated single board consisting of hardware for automated measurement and billing system for public utilities [19-21]. Newer systems might have a combination of hardware and software structures. The hardware structure consists of a digital energy meter module which is interfaced to a single microcontroller chip. The software structure runs the entire process via the microcontroller input/output ports. This type of meter does not contain any rotating parts, and the energy consumption is usually shown in a four-digit display. Besides that, energy consumption is stored in the microcontroller's EPROM or Flash memory. Early implementation of these kinds of meter has about 98% accuracy. Recent implementation has improved its accuracy to near 100% for sinusoidal loads.

Nowadays, with the increasing diffusion of power electronics, the typical characteristics of supply networks are far from being sinusoidal. Driesen, J., Van Craenenbroeck, T., Van Dommelen, D. has shown that there is considerable measurement error of energy meters operating under harmonic distortion [23]. Their generated curves show that the third harmonic component has the highest magnitude but the total summation of the other higher harmonics also plays a considerable role. Other follow up research confirm this situation [24-25].

Luo, Xu and Zheng proposed a new calculation algorithm based on harmonic load that is able to reduce error due to non sinusoidal voltage and current. Their implementation requires a DSP and discrete Fourier transform (DFT) [26]. This is further improved by Ovidiu and Gabriel [27] whose solution using microcontrollers and DSP facilitates the transition from widely-used mechanical meters that yield limited data, to electronic meters that provide extensive information on customer energy usage. Simultaneous sampling of both voltage channel and current are performed and energy computations consist mainly of multiplication and addition operations, easily be handled by this DSP. To increase accuracy, high sampling rate is used [28-29].

A built in switching regulator with microcontroller can be modified as an energy meter [30]. For many systems that have a built-in switching regulator, adding a single wire between the regulator and the microcontroller enables real-time energy metering by counting the switching cycles of the regulator. Furthermore these regulators can be built within an FPGA [31-33]. But even though the relationship between load current and switching frequency is quite linear and this simple design can be applied to a variety of regulators, this can achieve a maximum accuracy of class 1.

1.3 Thesis objective

The objectives of this thesis are to develop a single chip three phase digital energy meter which is capable of measuring the true energy consumed by the load including the harmonic contents of the power. The meter should also measure the quality of the power by calculating and displaying the harmonic distortion of the power. The entire system should be programmable, reconfigurable and upgradeable based on changing requirements at any point in future even after the meter have been deployed at user premises. In order to implement the whole system in a single chip, algorithm and designs are devised that reduce the calculation time required for energy calculation by off loading real time multiplications to dedicated multiplier blocks and by storing the pre-calculated sine and cosine values in a ROM table. Enhancements to increase accuracy with all types of sinusoidal and non sinusoidal inputs are also devised.

1.4 Organization of the Thesis

Chapter two describes the basic electro mechanical energy meter. It also introduces the newer digital energy meters and its components. Chapter three analyzes the hardware architecture based on the target goal. Chapter four provides the implementation of the design over a DE2 FPGA board with Cyclone II processor. Chapter five provides a FPGA simulation of the proposed meter

scheme. Timing analysis is also shown in this chapter. Chapter six compares the implementation with a standard induction meter. Finally, chapter seven concludes the thesis with some recommendations for future research.

Chapter 2

Energy Meters

2.1 Basic Idea about energy Meter

An electric energy meter is a device that measures the amount of electrical energy supplied to a residence or business. The most common type is properly known as a kilowatt-hour meter or a joule meter. Utility companies record the power consumption values measured by these meters and charge the users accordingly. Modern electricity meters operate by continuously measuring the instantaneous voltage (volts) and current (amperes) and finding the product of these to calculate instantaneous electrical power (watts) which is then integrated over time to give energy used (joules, kilowatt-hours etc). Energy meters fall into two basic categories, electromechanical and electronic.

2.2 Electromechanical meters

The most common type of electricity meter is the electromechanical induction meter. This technology has been used for decades to calculate energy consumption by the utility companies. The electromechanical induction meter operates by counting the revolutions of an aluminium disc which is made to rotate at a speed proportional to power. The number of revolutions is thus proportional to the energy usage. The meter itself also consumes a small amount of power, typically around two watts.

The metallic disc is acted upon by two coils. One coil produces a magnetic flux in proportion to the voltage and the other produces a magnetic flux in proportion to the current. This produces a circulating flow of electrons, or a current within the disk known as eddy current. The effect of this is a force exerted on the disc in proportion to the product of the instantaneous current and voltage. This

causes the disc to rotate at a speed proportional to the power being used. A permanent magnet exerts an opposing force proportional to the speed of rotation of the disc and this force acts as a brake which causes the disc to stop spinning when power stops being drawn rather than allowing it to spin faster and faster.

The aluminium disc is supported by a spindle which has a worm gear driving a counter register. The register is a series of dials which records the amount of energy used. The dials may be of the cyclometer type, an odometer-like display that is easy to read where for each dial a single digit is shown through a window in the face of the meter, or of the pointer type where a pointer indicates each digit. The amount of energy represented by one revolution of the disc is denoted by watt-hours per revolution.

2.3 Electronic Meters

Electronic meters can be analog or digital but most are a combination of the two and use mostly solid state devices. These meters operate by continuously measuring the instantaneous voltage and current usually through a potential transformer (PT) and current transformer (CT) respectively. Calculating the product of these two give instantaneous electrical power which is then integrated against time to give energy in kilowatt-hour. Most electronic meters use a current transformer as transducer to measure the current. This feature enables the placement of the CT even outside the meter. The meter can be located remotely from the main current-carrying conductors, which is a particular advantage in large-power installations. It is also possible to use remote current transformers with electromechanical meters though this is less common. In addition to measuring electricity used, solid state meters can also record other parameters of the load and supply such as maximum demand, power factor and reactive power used etc. They can also include electronic clock mechanisms to compute a value, rather than an amount, of electricity consumed, with the pricing varying by the time of day, day of week, and season.

2.4 Standard Construction of an energy meter

Modern meters typically consist of power supply, transformers, a metering engine, a processing and communication engine i.e a microcontroller, other add-on modules such as RTC, LCD display and communication ports/modules. A block diagram of a typical meter [34] is shown in Figure 2.1.

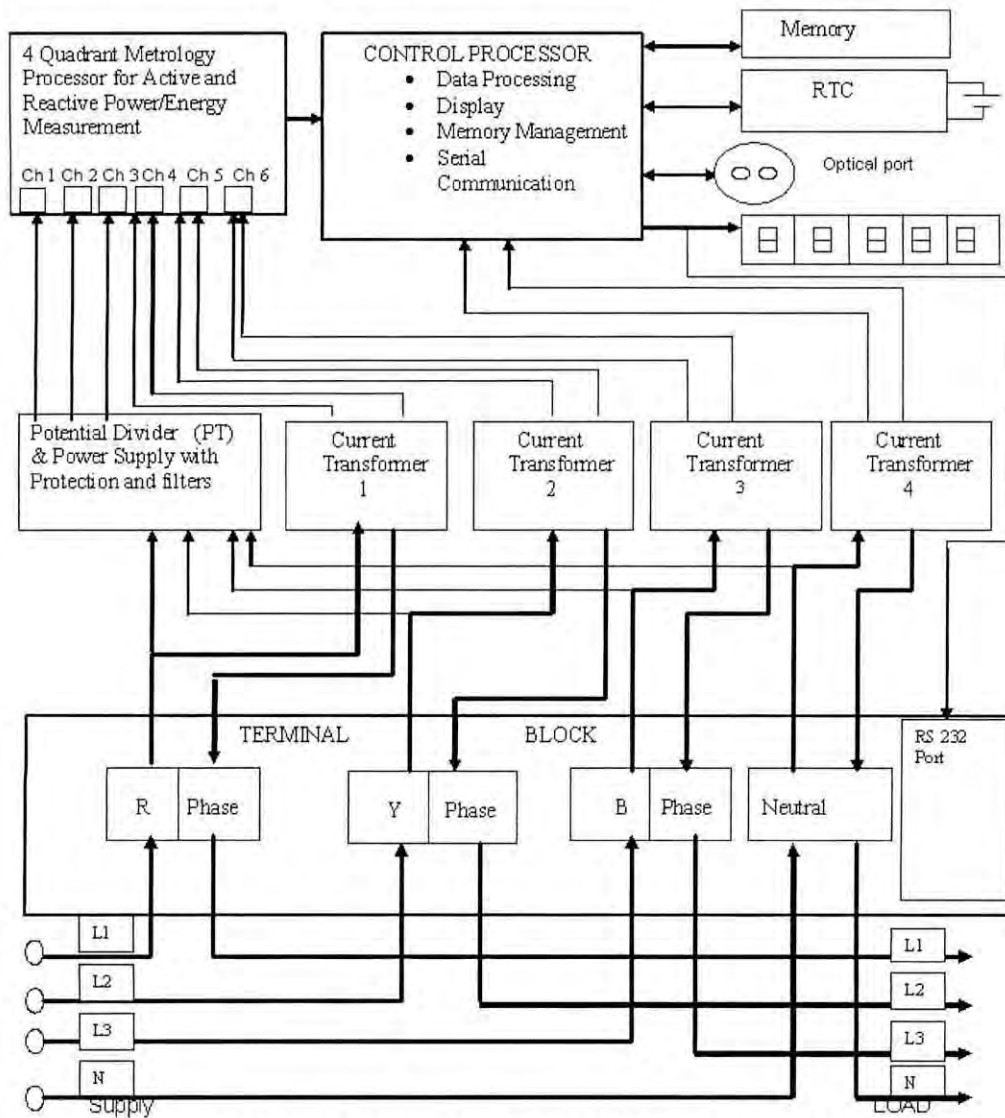


Figure 2.1 Block diagram of a typical 3 phase energy meter

A short description of the different parts of an energy meter is given below:

Power supply: This part is responsible for powering the meter itself. Usually it is derived from the supply line. Some meters have backup battery so that the meter can operate even when there is no power in the supply lines.

Transformers: Usually these are step down transformers that produce a lower current and voltage suitable for the electronic part to sense and measure. Potential transformers (PT) are required to step down the voltage and current transformer (CT) to produce a voltage proportional to the current through the transformer.

Metering Engine: The metering engine takes voltage and current as inputs and has a voltage reference, samplers and quantizers followed by an ADC section to yield the digitized equivalents of all the inputs. These inputs are then processed using a Digital Signal Processor (DSP) to calculate the various metering parameters such as power, energy etc.

Processing and communication section: This section calculates various derived quantities from the digital values generated by the metering engine. It also manages communication using various protocols and interfaces with other add-on modules connected as slaves to it like PC interface module or bluetooth module to transmit data.

RTC and other add-on modules: These are attached as slaves to the processing and communication section for various input/output functions. On a modern meter most, if not all of this will be implemented inside the microprocessor, such as the Real Time Clock (RTC), LCD controller and memory.

Chapter 3

The Proposed Meter

3.1 The Architecture of the Proposed Meter

Digital implementation of any system that operates on analog signal requires that the analog signals involved be sampled and digitized. In case of a three phase energy meter the signals involved are the three phases of voltage and three phases of current. These voltage and current signals need to be converted into their digital equivalent before any operation can be performed on them. But the voltage and current signals are too high to be measured directly, so they need to be stepped down to a lower values that an analog to digital converter (ADC) is able to measure. To step down voltage, a simple voltage divider network (Figure 3.1) is used.

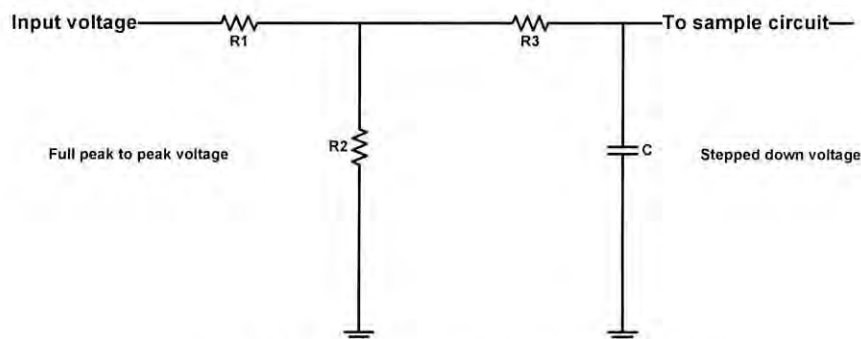


Figure 3.1 Typical voltage divider network

Using simple divider network instead of any special transducer has the advantage that these resistors do not exhibit any non linear characteristics and hence do not introduce any error in the stepped down signal which is inherent in any PT. The signal is also passed through a low pass filter.

Current inputs also require stepping down to a level that can be sampled by the analog to digital converter. The current signal also needs to be converted to voltage signal since it is easier to measure voltage values. A high accuracy current transformer with typical turn ration of 1000:1 is used to reduce the current and then

this current is passed through a resistor of low resistance. This produces a voltage across the resistor which is proportional to the input current. This voltage is sampled by the ADC to measure the input current (Figure 3.2).

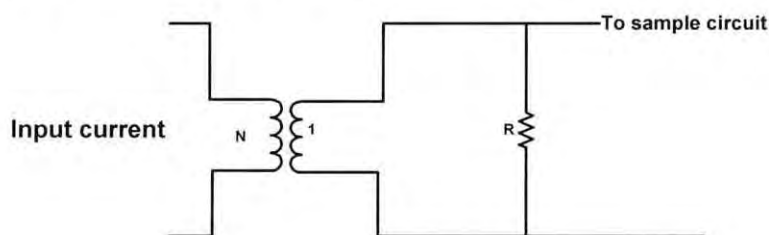


Figure 3.2 Typical AC current step down network

After both voltage and current is stepped down and converted to a voltage that can be directly sampled by the analog to digital converter (ADC), these inputs are fed into a high resolution 16 bit six channel ADC. The ADC used in this design is AD73360 which is a six-input channel analog front-end processor featuring six 16-bit A/D conversion channels each of which provides 77 dB SNR over a dc to 4 kHz signal bandwidth. Each channel also features a programmable input gain amplifier (PGA) with gain settings in eight stages between 0 dB to 38 dB. The AD73360 is particularly suitable for our purpose as each channel samples synchronously, ensuring that there is no (phase) delay between the conversions.

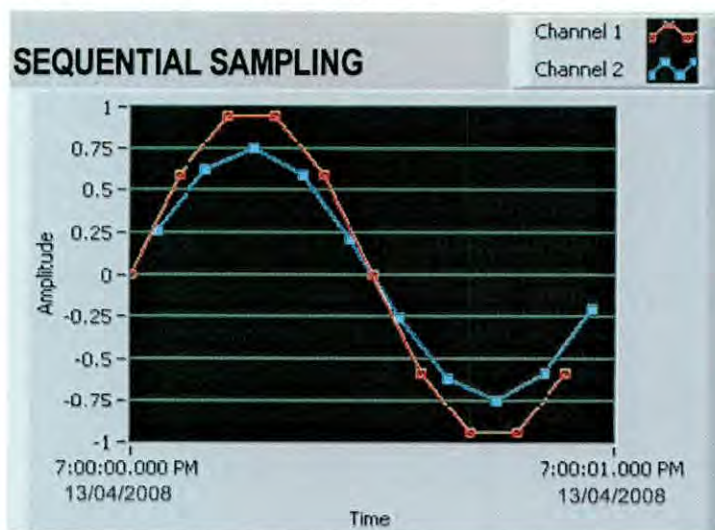


Figure 3.3 Sequential or multiplexed sampling of current and voltage.

Figure 3.3 shows a typical sampled signal that is sequential and not simultaneously sampled. If we consider channel1 as voltage signal and channel2 as current signal, then we can see that there is a phase difference between the sampled current and voltage channels. If we use this value to calculate instantaneous power by multiplying the sampled current and voltage value, this will give the wrong power calculation as the voltage and current are not of the exact same instant.

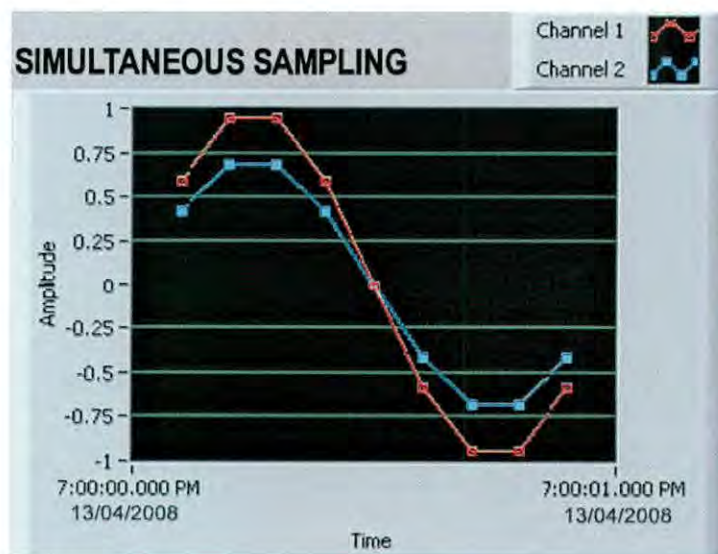


Figure 3.4 Simultaneous or aligned sampling of voltage and current.

This problem is solved by simultaneous or aligned sampling (Figure 3.4) of all 6 channels which gives the value of all 6 channels in the exact same moment. Other useful features of the ADC are an on-chip reference voltage, programmable sampling rates, a serial communication port and a signal conditioner [35].

The sampled data of the ADC needs to be collected and processed to calculate RMS voltage, current, power and energy. The ADC has a serial port (SPORT) with predefined signal structure to communicate from external sources. Using this signaling scheme, the ADC can be programmed to operate based on our requirement. Sampled data can also be collected from the ADC using this signaling scheme. To interface with the ADC and transfer data from it, a Nios II microprocessor is implemented in the FPGA chip (Figure 3.5) which also calculates power and energy from this data.

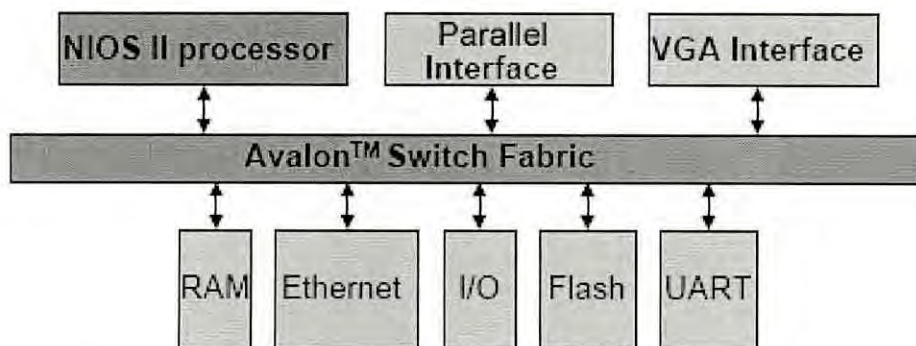


Figure 3.5 Nios processor structure implemented in the FPGA

The Nios II processor has parallel IO pins that can be used directly to interface with AD73360 ADC. ADC signaling scheme is coded in the microprocessor using assembly or ANSI C code. IO pins is evaluated like memory and interfaced with the ADC. In this method, the processor is responsible for constantly polling the IO lines and collect the data from the ADC. When each set of data is received, it is saved in the separate voltage and current buffer memory. When ten cycle of data is collected, the processor calculates the RMS values of signal as well as the energy. This scheme has the benefit that it is simpler to implement and requires very few components. Also any changes required, can be made virtually instantaneously as no additional hardware is synthesized within the FPGA except the Nios processor. The new firmware can be directly programmed into the Nios processor. The block diagram of this simplified interface is shown in Figure 3.6.

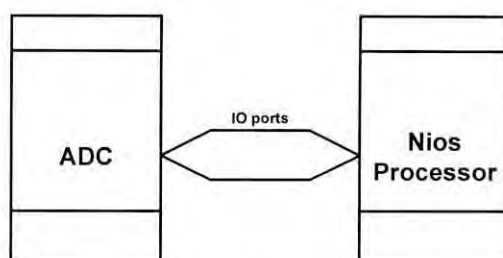


Figure 3.6 Simpler implementation using direct interface to ADC

However, a disadvantage of this simpler implementation is that when sampling rate is increased, the processor becomes too busy to perform all the required calculations within the short duration it gets. In this constant polling method, about 50% of the CPU cycle is used to interface with the ADC. This leaves less time for the processor to perform all the required calculations. This situation becomes worse when sampling frequency is increased or sample length is increased from ten to a twenty or more.

A revision of the structure of data collection from the ADC is necessary to solve this problem. A separate digital block is introduced in the system that is responsible for interfacing with the ADC. This block is designed digitally and synthesized within the FPGA block. The utility of this block is twofold:

- Interface with the ADC directly freeing up resource of the microprocessor
- Interrupt the processor only when one set of all six channel data has been collected.

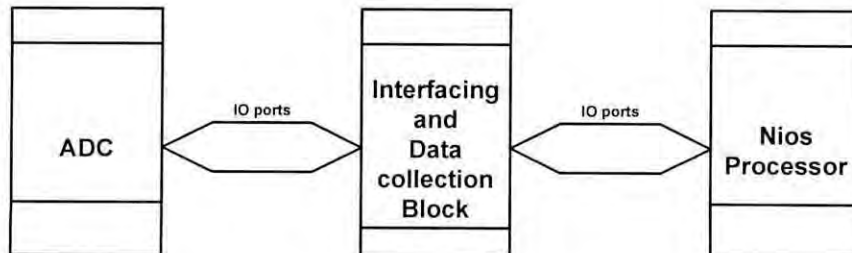


Figure 3.7 Modified implementation with interfacing block

Figure 3.7 shows the block diagram of the modified implementation to overcome the limitation of the initial design. Since the interfacing block is responsible for interfacing with the ADC, this frees up the processor for more intensive operations like calculation of rms current, rms voltage and energy. Using this scheme the processor overhead is reduced to about 12% for data collection from the interfacing block. This implementation has been tested to operator up to 1kHz sampling rate, 50 cycle frame data and for single phase energy measurement. This model also fails when all three phase (6 channel) data are processed and when

additional functions like signal processing is introduced. To remedy this situation the following performance bottlenecks are identified:

- Critical calculation like multiplying is performed by the microprocessor which takes most of the time
- Processor is unable to perform parallel multiplication operation
- The processor need not be interrupted at every set of data

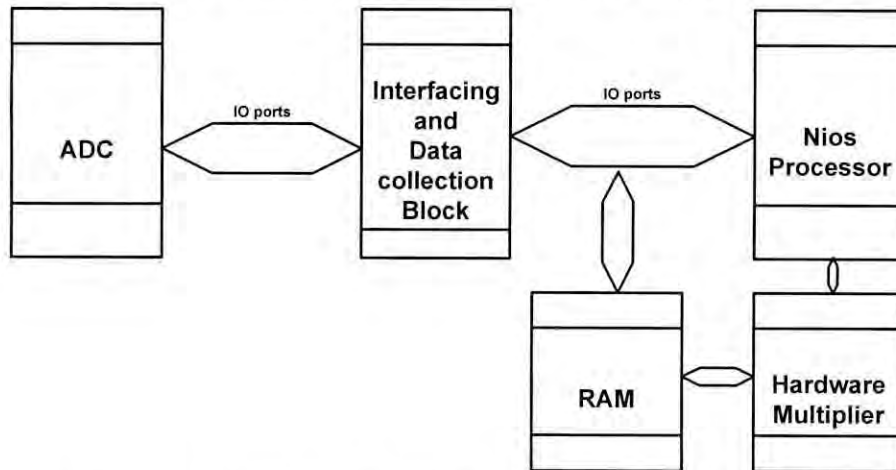


Figure 3.8 Final implementation with RAM and hardware multiplier

To remedy this situation, a Random Access Memory (RAM) block is added to the design. Instead of interrupting the processor whenever one set (6 samples) of data is available, the processor is only interrupted when a complete set of data for one second is available (Figure 3.8). Since the ADC is constantly supplying data, for a sampling frequency of 1kHz, there will be 6000 samples of data in one second for all six channels. This sampled data is saved in RAM instead of interrupting the processor 6000 times per second. The design consists of two RAM blocks each capable of storing all the sampled data of one second duration. When a complete set of data has been collected and saved in the RAM blocks, the interfacing block interrupts the processor signaling it that data is ready while continuing to save incoming data from the ADC to the other RAM block. This makes sure that none of the sampled data is lost and increases measurement accuracy. The Nios processor when interrupted, performs calculation over the sampled data in RAM but this time the processor has a entire second to perform all the calculations and

during this time it will not be interrupted for any other operation. Instead of processing the data sequentially as in traditional microprocessor, the proposed implementation uses hardware acceleration to speed up calculation. It achieves this by using the dedicated multipliers (Figure 3.9) in parallel and moving data directly from RAM instead of through the processor.

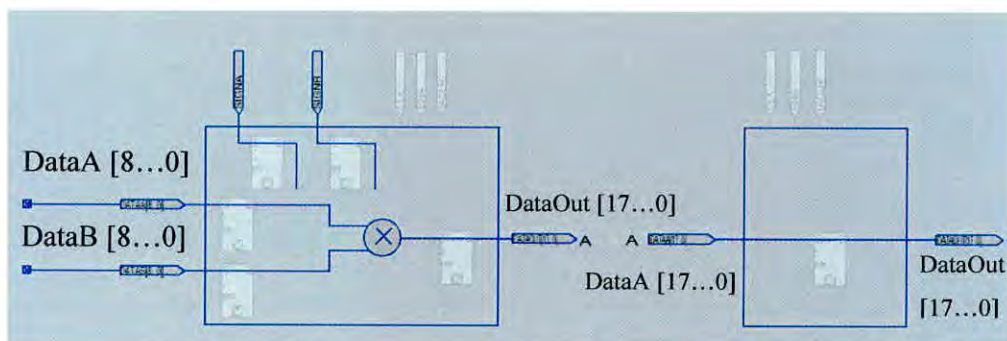


Figure 3.9 Cascadable Dedicated multiplier Register Transfer Level (RTL) Circuit

The block diagram of the overall system is shown in Figure 3.10 on the next page.

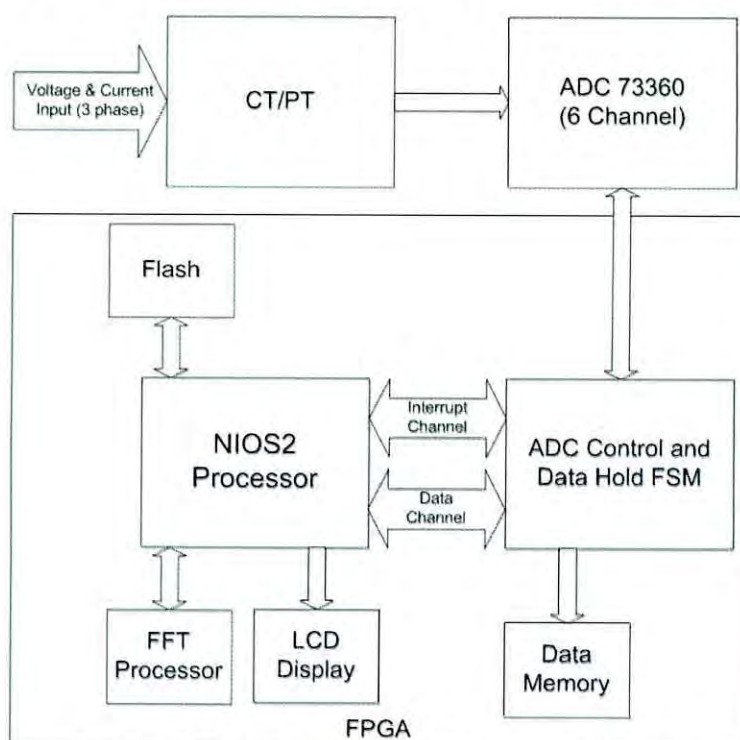


Figure 3.10 Block diagram of proposed meter

The design (Figure 3.10) consists of a current transformer (CT) and voltage divider block for each phase, low-pass filter, analog to digital converter (ADC), control unit, Fast Fourier Transform (FFT) unit and LCD display. The three-phase AC current and voltage input signal goes through the CTs and divider network respectively and is converted to 0V~1V signal. This is then passed through a low pass filter and fed into the 16 bit six channel ADC (AD73360).

A control block is implemented which controls the ADC and latches the data for the Nios processor. This block also controls A/D sampling frequency, signal start and stop and the memory interface. The Nios II is a soft processor from Altera that is implemented over an FPGA. It is a 32-bit RISC soft-core architecture and can be implemented entirely in the logic and memory blocks of an Altera FPGA. One of the major advantages of the Nios II processor is that a designer can specify and generate a custom Nios II core specially designed for his specific requirements and application. A system designer can even extend the Nios II processor's functionality by defining custom instructions, custom peripherals or memory management units. The soft Nios processor implemented in our FPGA processes the data received from the ADC and outputs useful data to the LCD block like voltage, current and energy. The Nios processor also communicates and supplies data to the FFT block. Block diagram of the FFT unit is show in Figure 3.11. The FFT block processes the digital data from the Nios and performs FFT transform operation. This final processed data is also relayed back to the Nios processor and displayed on the LCD unit. The overall process is shown in the flow diagram of figure 3.12.

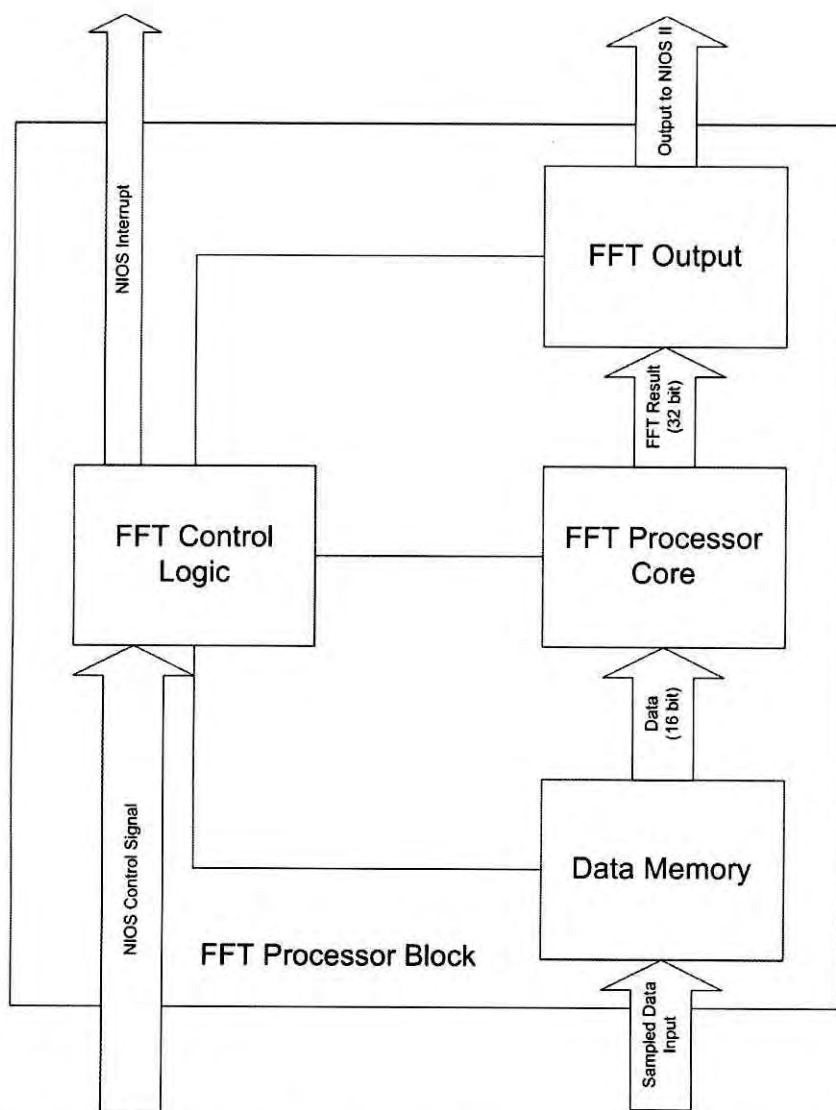


Figure 3.11 Block diagram of FFT processor of the proposed meter

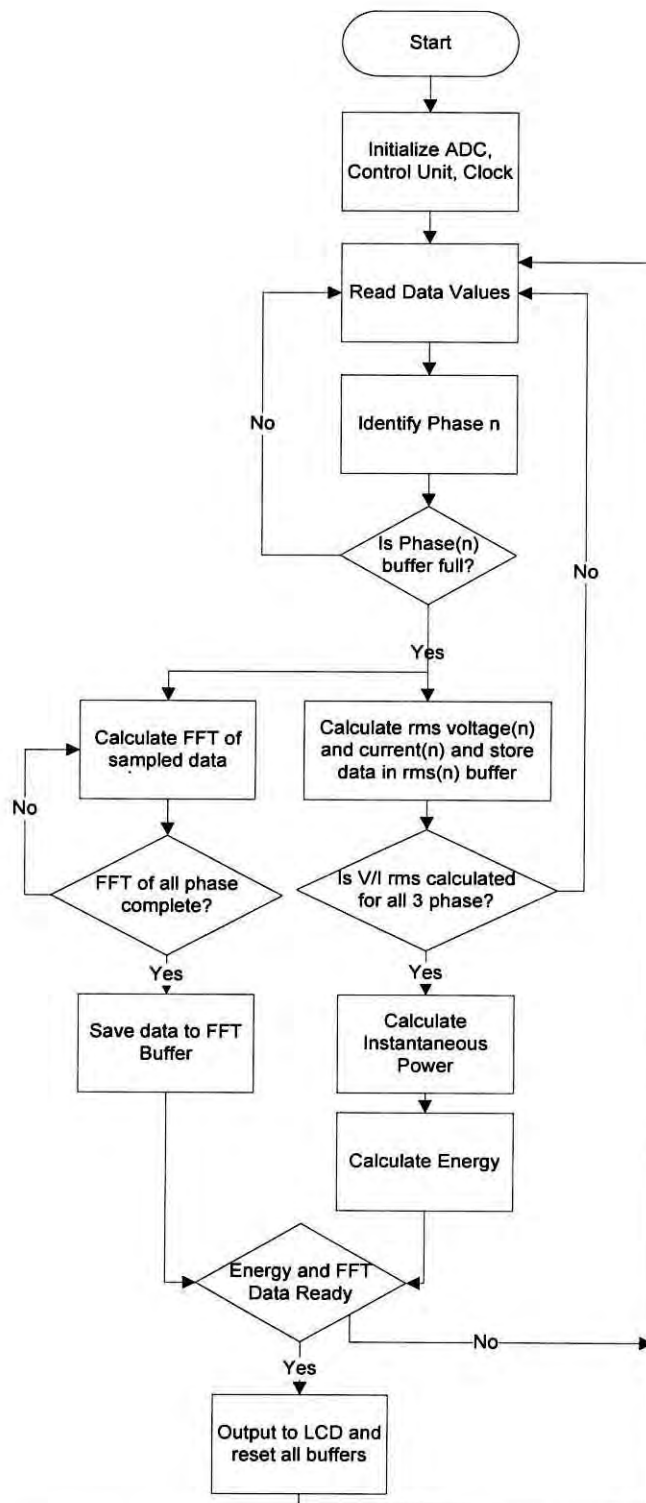


Figure 3.12 Flow diagram of energy calculation of the proposed meter

3.2 Operating Principle

The operating principle of the meter is the same as any other meter. Voltage and currents are simultaneously sampled at a certain interval. The sampled data is then digitized giving discrete values for each sample of voltage or current. The instantaneous power for discrete signal is defined as:

$$\text{Instantaneous Power} = V_i I_i \quad (3.1)$$

Average power is given by

$$\frac{1}{N} \sum_{n=1}^N V_i I_i \quad (3.2)$$

where N is the number of sample over one period of the wave and V_i and I_i are voltage and current respectively of a phase at instant i.

$$\text{Instantaneous energy} = V_i I_i \Delta t \quad (3.3)$$

Where Δt is the sampling interval. Total energy over a period is equal to summation of instantaneous energy over that period.

$$\text{Energy} = \sum_{n=1}^N V_i I_i \Delta t \quad (3.4)$$

The rms values of the voltage and current are derived from calculating the square root of the mean value of the square of input voltages and currents respectively over one period of the wave. The equation of rms value of voltage and current with respect of sampled discrete signals, are given by

$$V_{rms} = \sqrt{\sum_{n=1}^N \frac{1}{N} \times V_n^2} \quad (3.5)$$

$$I_{rms} = \sqrt{\sum_{n=1}^N \frac{1}{N} \times I_n^2} \quad (3.6)$$

After all the required values from each set of sampled voltage and current has been calculated, one cycle will be extracted from each of the sampled data and its Fourier transform (FFT) performed to determine the power quality.

To measure the wave shape distortion, we can use the quantity of the Total Harmonic Distortion, THD (Eqn. 3.7). THD is the ratio of the power of harmonic components to the power of fundamental frequency. To measure THD of current waveform, we can just find the sum of the rms of the harmonic components, I_n and the rms of the fundamental frequency, I_1 .

$$THD = \sqrt{\frac{\sum_{n=2}^N I_n^2}{I_1^2}} \times 100 \quad (3.7)$$

where I_n is the rms value of the nth harmonic current. Most of the harmonic problem is caused by the third component since the third harmonic is the second highest energy from the fundamental component. We can calculate third harmonic distortion (HD) of current waveform using the following equation:

$$HD = \frac{I_3}{I_1} \times 100 \quad (3.8)$$

3.3 Design issues

A complete digital implementation of such a large and complex system requires a lot of attention and planning. It is sometimes a lot more complex to design a digital filter than an analog filter. Since our entire system is implemented over a single FPGA, every component, system, filter, block, signal processor has to be implemented and realized digitally. Another major issue is timing requirements. The system has to be able to sample data in real time, calculate voltage and current rms values, instantaneous power and complex FFT operation on sampled data all within a very short time. Meeting this timing requirement was a major issue. Also the largest source of long-term errors in the meter is drift in the preamp, followed by the precision of the voltage reference. Both of these vary with temperature as

well, and vary wildly because most meters are outdoors. Characterizing and compensating for these is a major part of meter design.

Chapter 4

Implementation

4.1 Introduction

Design of VLSI (Very Large Scale Integration) circuits at register level is very time consuming and costly. This method is rarely implemented nowadays specially in designing prototypes. To speed up design process and add versatility and portability to designs, high levels languages are being introduced in the hardware design processes. Verilog is one such hardware descriptive languages (HDL) that enables the design of modular process independent VLSI circuits. Design process like platform independence and fitting can also be automated using this methodology. This chapter presents the realization of the meter in a FPGA chip using Verilog.

4.2 FPGA Structure

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together", somewhat like a one-chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory. An FPGA contains three main types of resource: logic blocks, I/O blocks for connecting the pins of the package and interconnection wires and switches. The general structure of an FPGA is shown in Figure 4.1. Field Programmable Gate Array (FPGA) is named so because its architecture resembles the gate array, but it can be programmed by the customer to perform a specific function. With an FPGA, the array consists of logic blocks that contain memories used to implement logic, multiplexers to select signal sources, and flip-flops. Because these are

programmed by the user, all the routing is placed on the chip. This is done by placing metal traces all over the chip but keeping them unconnected. At junctions between unconnected traces are transistors that can be turned on by setting a bit in the chip, thus creating a slow but definite connection between the traces.

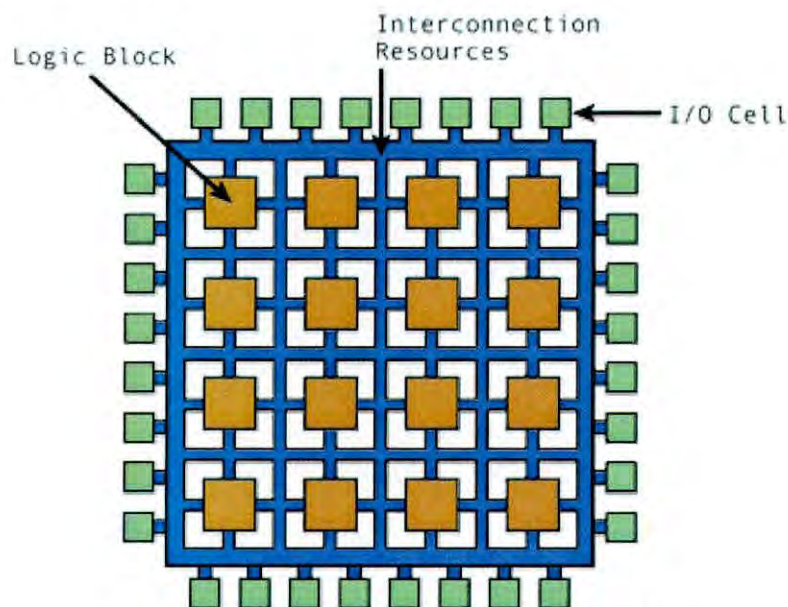


Figure 4.1 General structure of an FGPA

4.2.1 Logic Blocks

Each logic blocks in an FPGA has a limited number of inputs and one output. A very common type of block is the look up table (LUT) which contains storage cells. A classic FPGA logic block consists of a 4-input lookup table (LUT), and a flip-flop, as shown in Figure 4.2 [36]. Recently manufacturers have started moving to 6-input LUTs in their high performance parts. There is only one output, which can be either the registered or the unregistered LUT output. The logic block has four inputs for the LUT and a clock input. Since clock signals are normally routed via special-purpose dedicated routing networks in commercial FPGAs, they and other signals are separately managed.

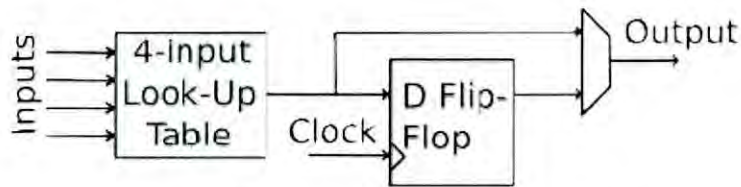


Figure 4.2 Typical logic block of an FPGA

The storage cells in LUTs are volatile and they lose their content as soon as power is removed. For this reason the FPGA has to be programmed every time it is powered on. To make the programming permanent, usually an erasable programmable read only memory (EPROM) or Flash memory is used with the FPGA. The storage cell values are loaded automatically at each power up into the FPGA. The logic cell fan out of pins of Figure 4.2 is shown in Figure 4.3.

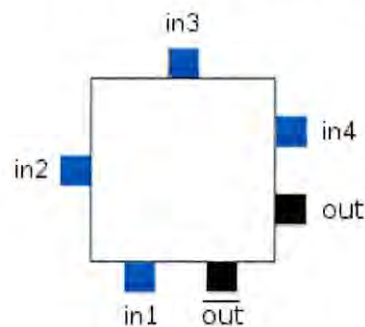


Figure 4.3 Logic Block Pin Locations for a 4 input LUT

4.2.2 Interconnection wires and switches

Each input in an FPGA is accessible from one side of the logic block, while the output pin can connect to routing wires in both the channel to the right and the channel below the logic block. Each logic block output pin can connect to any of the wiring segments in the channels adjacent to it. This interconnection between wires is done by programmable switches.

Generally, the FPGA routing is unsegmented so that each wiring segment spans only one logic block before it terminates in a switch box. By turning on some of the programmable switches within a switch box, longer paths can be

constructed. For higher speed interconnect, some FPGA architectures use longer routing lines that span multiple logic blocks.

Whenever a vertical and a horizontal channel intersect, there is a switch box. In this architecture, when a wire enters a switch box, there are three programmable switches that allow it to connect to three other wires in adjacent channel segments. The pattern, or topology, of switches used in this architecture is the planar or domain-based switch box topology. Figure 4.4 illustrates the connections in a switch box.

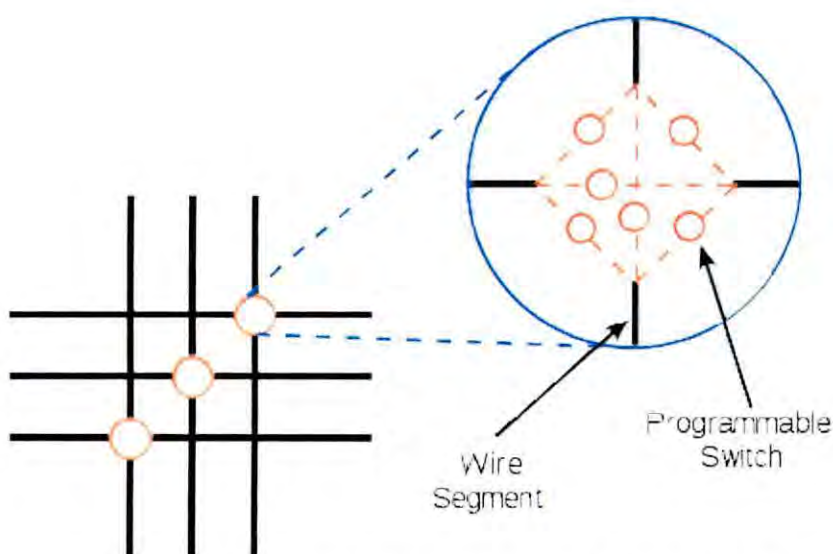


Figure 4.4 Switch box interconnect topology in an FPGA

4.3 Design Procedure

In any system designed by Verilog or any other HDL, the system is partitioned into smaller unique components that perform a specific task. These individual components are then designed one by one by Verilog code. After designing all the components, the interconnection between these components are defined. This effectively combines these components into a larger complex system or a single large circuit. Computer Aided Design (CAD) software makes it easy to implement a desired logic circuit by using a programmable logic device, such as a

field-programmable gate array (FPGA) chip. A typical FPGA CAD flow is shown in Figure 4.5.

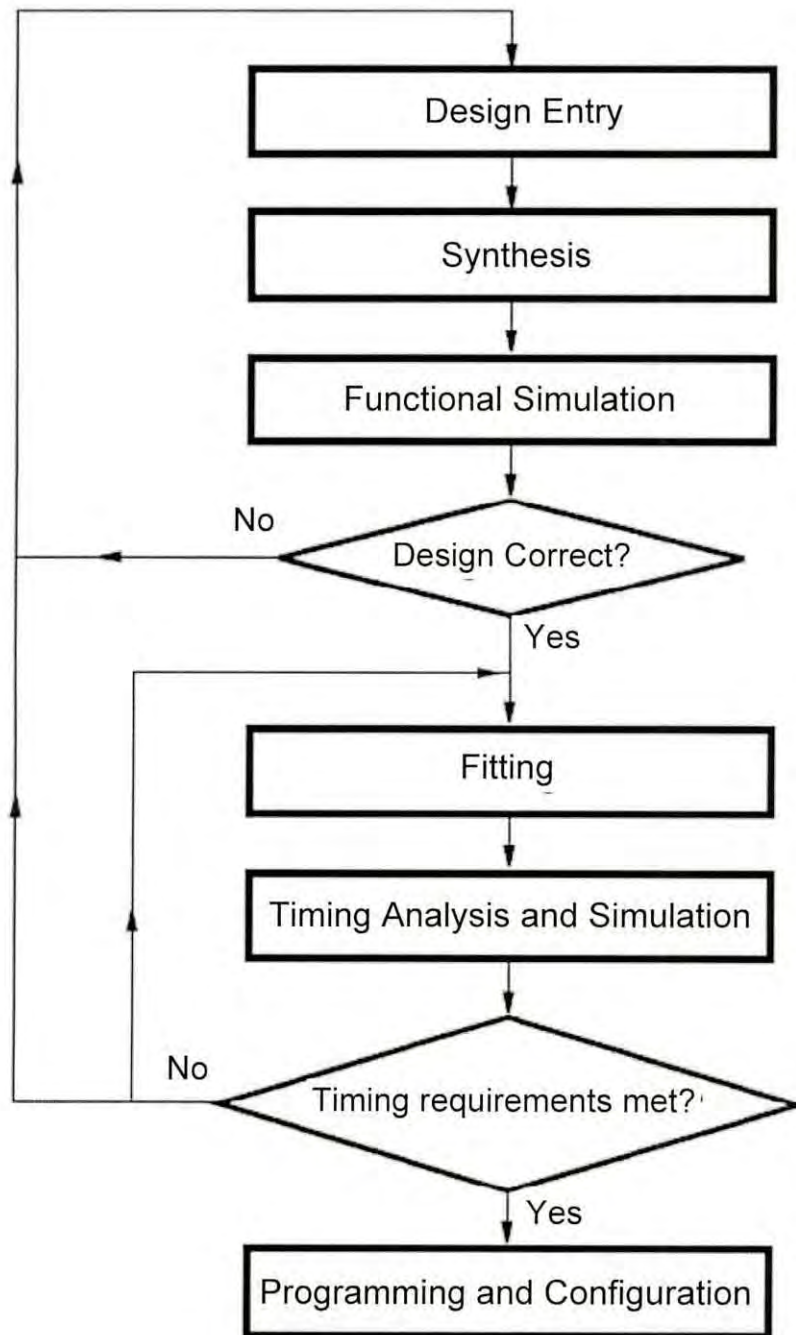


Figure 4.5 Typical CAD flow in FPGA design process

The computer aided design flow of Figure 4.5 involves the following steps:

- **Design Entry:** The desired circuit is specified either by means of a schematic diagram, or by using a hardware description language, such as Verilog or VHDL.
- **Synthesis:** The entered design is synthesized into a circuit that consists of the logic elements (LEs) provided in the FPGA chip.
- **Functional Simulation:** The synthesized circuit is tested to verify its functional correctness; this simulation does not take into account any timing issues.
- **Fitting:** The CAD Fitter tool determines the placement of the LEs defined in the netlist into the LEs in an actual FPGA chip; it also chooses routing wires in the chip to make the required connections between specific LEs.
- **Timing Analysis:** Propagation delays along the various paths in the fitted circuit are analyzed to provide an indication of the expected performance of the circuit.
- **Timing Simulation:** The fitted circuit is tested to verify both its functional correctness and timing.
- **Programming and Configuration:** Finally the designed circuit is implemented in a physical FPGA chip by programming the configuration switches that configure the LEs and establish the required wiring connections.

4.4 Device Selection

The DE2 development board which comes with Cyclone II EP2C35F672C6 FPGA is used in the development of the prototype. Verilog HDL was used for all synthesis using supplied Altera Quartus® II software. The Cyclone II chip contains:

- 33,216 LEs
- 105 M4K RAM blocks
- 483,840 total RAM bits
- 70s embedded multipliers

- 4 PLLs
- 475 user I/O pins
- FineLine BGA 672-pin package

The meter implementation required:

- 8,317 Logic Elements (LEs);
- 3,493 Dedicated logic registers
- 32 Embedded multiplier
- 365 I/O Pins;
- 4 Digital PLLs.
- 388,224 Memory Bits
- 1-MB SDRAM Memory

Figure 4.6 shows the Cyclone II FPGA chip located in the DE2 board used to implement this meter.



Figure 4.6 Cyclone II FPGA chip in DE2 board

The block diagram of the DE2 board is shown in Figure 4.7.

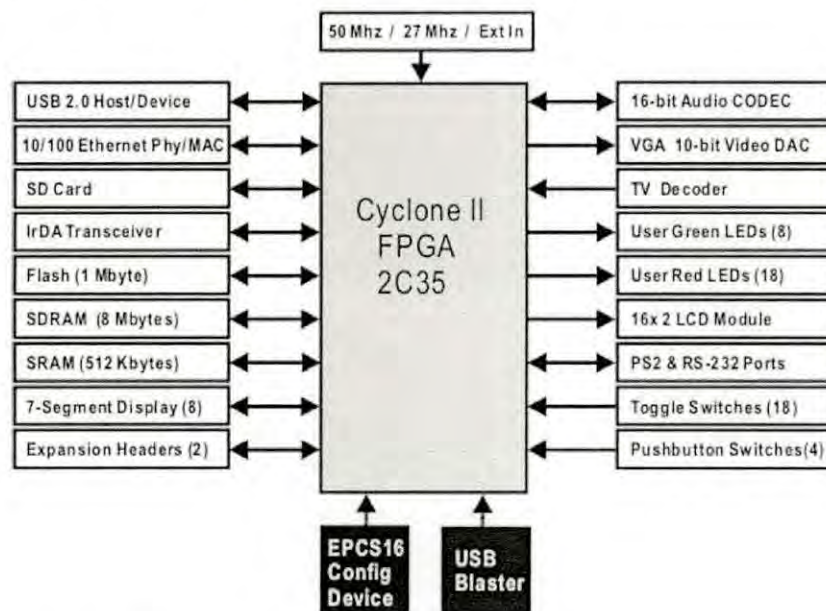


Figure 4.7 Block diagram of the DE2 board used in the proposed meter implementation

4.5 Design Architecture

The hardware architecture of the proposed meter is shown in Figure 4.8. Modular based design approach has been chosen to reduce complexity of the system. The main components are the ADC, the interfacing block (or FSM), the processor and the acceleration block consisting of RAM, hardware multiplier and the processor.

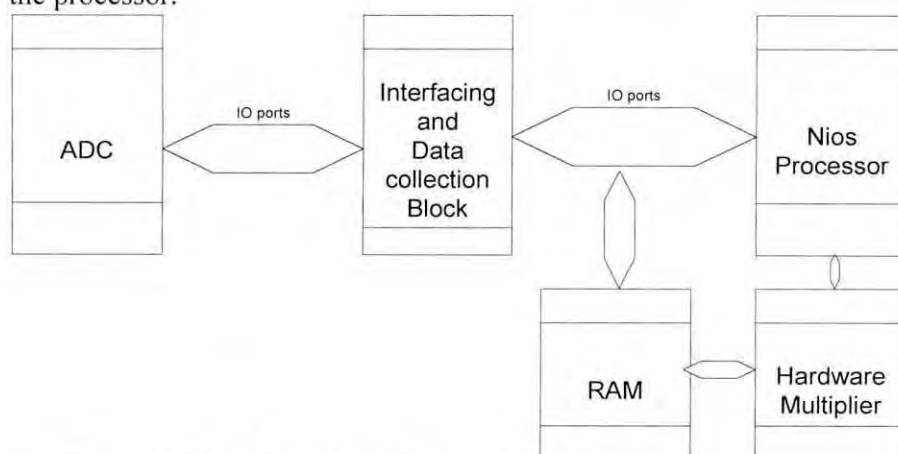


Figure 4.8 Hardware architecture of the proposed meter consisting of modular components

4.5.1 Analog to Digital Converter

A fast and efficient analog to digital converter (ADC) is required for a system like this. Choosing an ADC that operates with simultaneous sampling rather than sequential makes sure that there is no phase difference between each channel which is absolutely necessary for proper power calculation. The ADC is operated at sampling frequency of 1kHz which is twenty times the frequency of the input signal (50Hz). Such high sampling rate reduces aliasing errors. Figure 4.9 shows how aliased signal can be created with lower sampling rate than the required Nyquist rate. In Figure 4.9-A, the reconstructed waveform appears as an alias of DC, in Figure B, the sampled signal appears as a triangle waveform even though it is not. In Figure C, sampling rate is at $4f/3$ which creates alias waveform of incorrect frequency and shape. Our implementation avoids this situation by sampling ten times the minimum required rate.

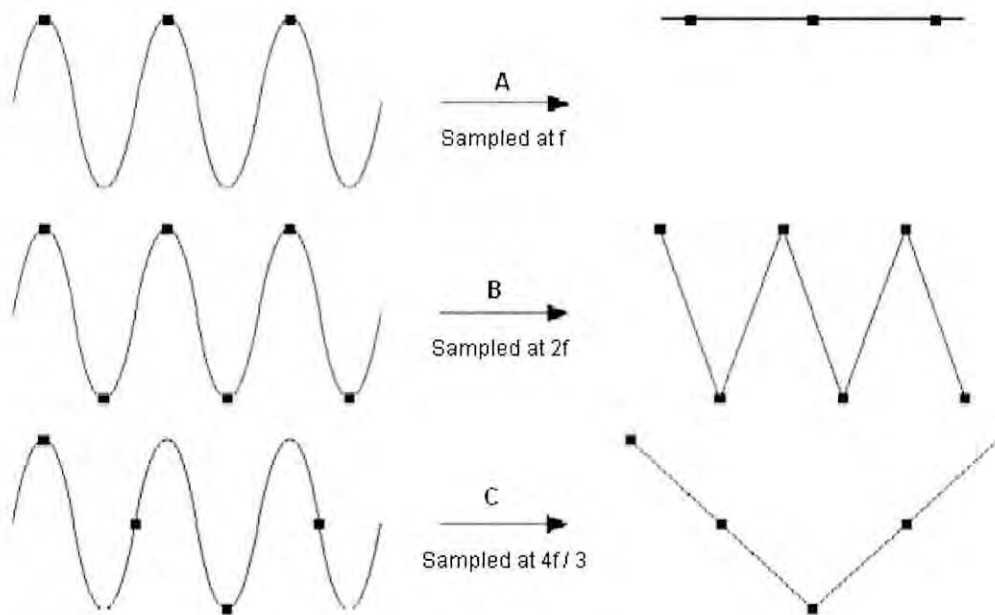


Figure 4.9 Generation of aliasing signal at lower sampling frequency

Considering these factors the ADC chosen is Analog Devices ADC73360 which has six 16-bit A/D conversion channels each of which provide 77 dB signal-to noise ratio over a dc to 4 kHz signal bandwidth. Each channel also features a

programmable input gain amplifier (PGA) with gain settings in eight stages between 0 dB to 38 dB. The AD73360 is particularly suitable for industrial power metering as each channel samples synchronously, ensuring that there is no (phase) delay between the conversions. The AD73360 also features low group delay conversions on all channels. In the prototype constructed the AD73360 was placed on a breadboard and connected to the DE2 FPGA board through I/O pins and wires as in Figure 4.10.

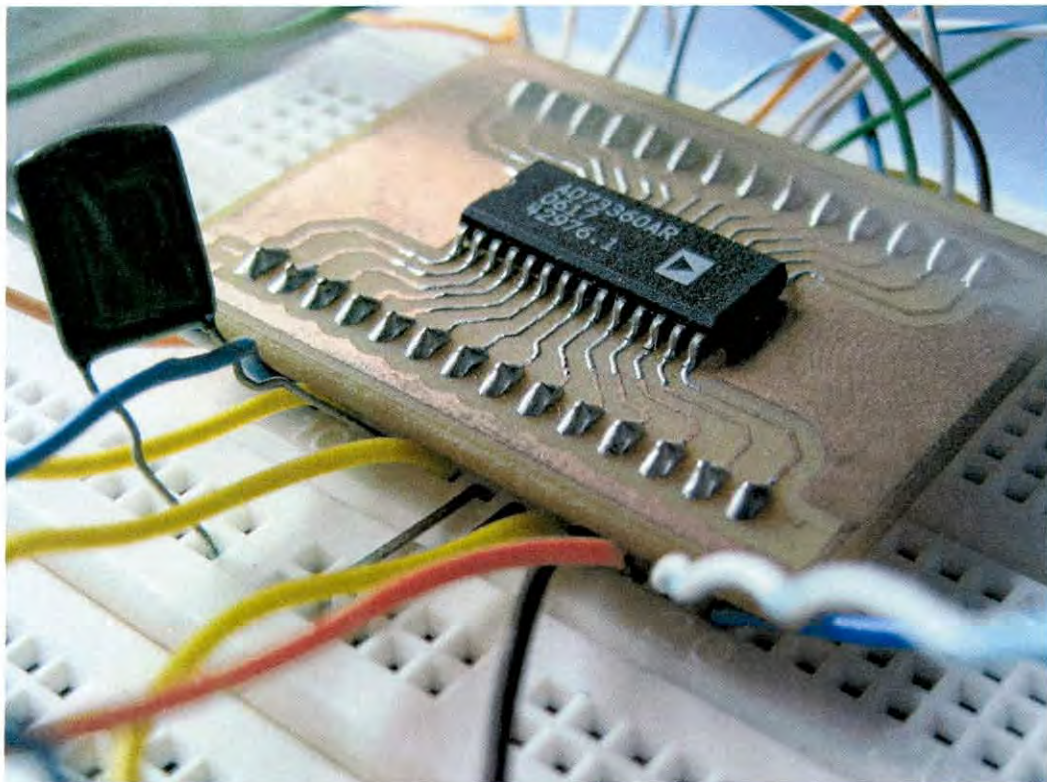


Figure 4.10 AD73360 in the proposed implementation

The sampling rate of the device is programmable with four separate settings offering 32 kHz, 16 kHz, 8 kHz and 4 kHz sampling rates (from a master clock of 16.384 MHz). A serial port (SPORT) allows easy interfacing of single or cascaded devices to industry standard DSP engines or in our case an FPGA. The functional pin description of the AD73360 is mentioned in a table in the appendix A.

4.6 Data Acquisition FSM Block

This is one of the primary blocks of the entire design and is necessary for interfacing the ADC to our soft processor and collecting data. This block is implemented through a finite state machine (FSM) which ensures definite state and synchronization with the ADC. Implementing this block through FSM also ensures that this block is practically implemented with the least number of logic elements. The function of this block is two folds; one is to interface with the ADC and provide synchronization mechanism and the second is reduce overhead of the processor by storing all digitized data in the RAM.

4.6.1 Interfacing block

The core of this block is a finite state machine that goes through several definite states each pertaining to a certain function. Three state machines work together to perform the complete operation. The main FSM is “fsm_logic” and there are two sub FSMs “readfsm” and “writefsm”. State 1 is for FSM initialization. State 2-6 is for ADC initialization and programming. State 7 is the read state in which the FSM loops for ever unless reset externally using the FSM reset pin. State 8 and 9 are used for transferring control to sub read and write FSMs. State 10 is for writing control words to the ADC and state 11 is the actual read FSM state that reads serial data values from the ADC line and transfers the data to state 7. The exact state functions are described below and shown in Figure 4.11.

State 0: This state initializes the FSM for operation. This state initializes the internal 1 bit register “reading” and “writing” and sets their value to zero. The SE and RESET output pin of the FSM is brought to low which disables the serial port of the ADC and also resets the ADC respectively. This state also sets the value of the wait register “waitreg” to the value “4” which means that the FSM will be in this state for 4 more clock pulse. This gives a little time for the ADC reset operation to complete.

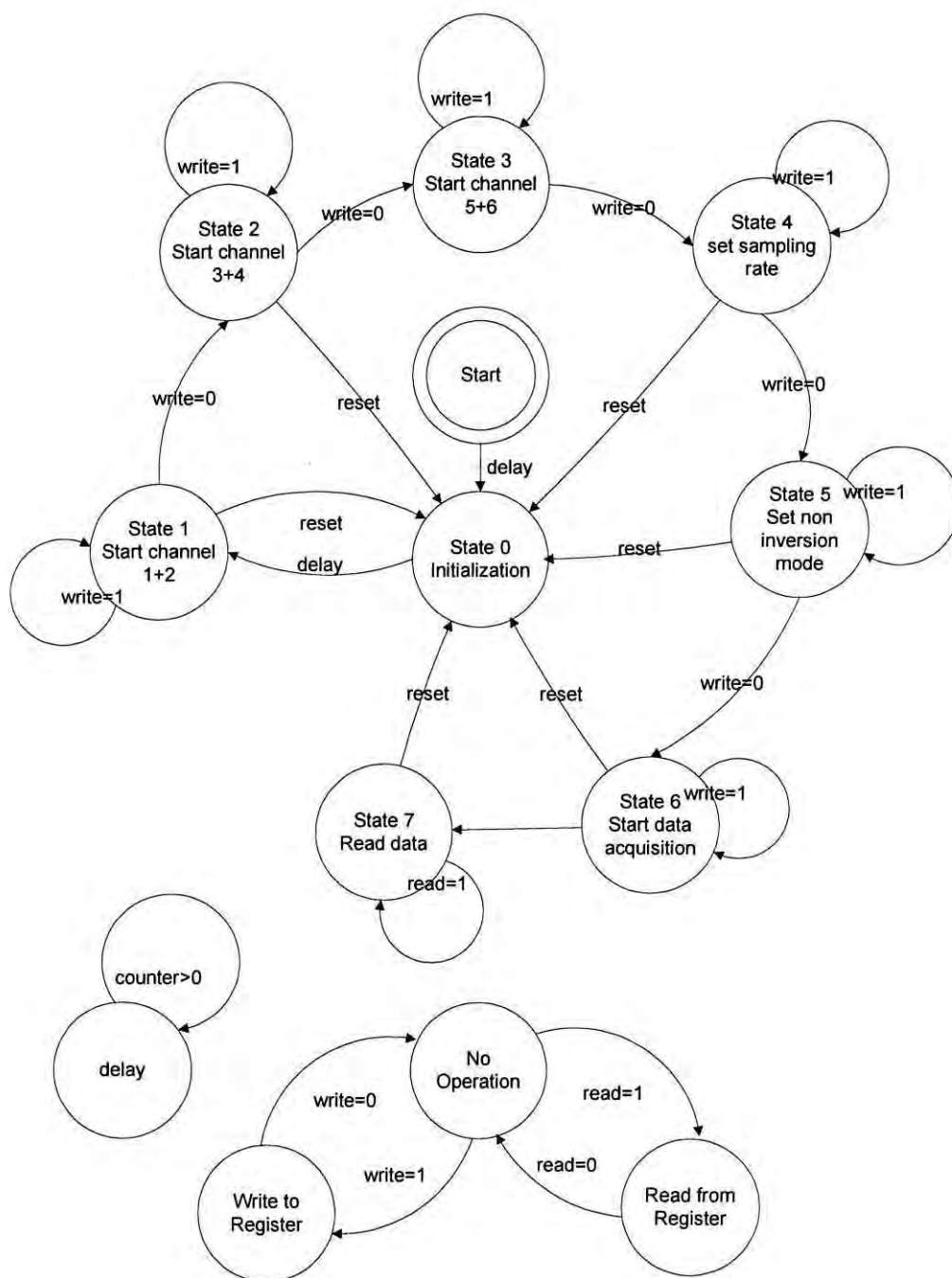


Figure 4.11 State Diagram of the Finite State Machine (FSM) interfacing module

State 1: This is the first state where the FSM starts communicating with the ADC. This operation starts by setting both the SE and RESET output pins of the FSM to high. This ensures that the ADC is in normal state and that its serial port is enabled for communication with the FSM. This state also sets the value of the “writing” register to 1 which disables the main FSM and transfers control to the minor FSM “writefsm”. This state sets the value of the “datatosend” register to binary “1000001110001000” which is written by the writefsm state machine. The write process is described in detail in the write state section below. The structure of the 16 bit value is shown in Figure 4.12.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
\overline{CD}	$R\overline{W}$	DEVICE ADDRESS			REGISTER ADDRESS			REGISTER DATA							

Figure 4.12 16 Bit structure of ADC control word

So the value “1000001110001000” means:

Bit 15: 1 – Specifies control word

Bit 14: 0 – Specifies write operation

Bit 13-11: 000 – Specifies device zero. In our case we have a single ADC, so this value is always 000.

Bit 10-8: 011 – Specifies the register to write to. Binary 011= decimal 3. This means register D needs to be written with the values in the next 8 bits

Bit 7-0: 10001000 – This 8 bit value is written to the register specified in the earlier 3 bits. In this case the first 1 stands for “power up channel 2 of the ADC”, the next 3 bits (000) specified 0 input gain (no additional gain). The next 4 bits instructs to power up channel 1 the same way.

State 2: This state is very similar to state 1. The only difference is that in this state the value “1000010010001000” is written to the register which instructs the ADC to power up channel 3 and 4 with zero input gain.

State 3: This state also performs the same operation as state 3. The only difference is that in this state the value “1000010110001000” is written to the register which instructs the ADC to power up channel 5 and 6 with zero input gain.

State 4: This state sets the sampling rate of the ADC. The ADC sampling rate is set by the register B. Bit 0 and bit 1 of the 8 bit registers sets the sampling rate based on table 4.1.

DR1	DR0	Sample Rate
0	0	1kHz
0	1	2kHz
1	0	4kHz
1	1	8kHz

Table 4.1 Decimation rate control of the ADC

By writing the value “1000011000111111” we choose the lowest sampling rate based on MCLK used. So if a 2.048MHz clock signal is applied to the ADC MCLK pin then the under default condition the sampling rate would be $2.048\text{MHz}/2048=1\text{kHz}$ which is the sampling rate in our case.

State 5: This state instructs the ADC to set non inverted mode for all operation of the ADC. The value written is “1000011100111111”.

State 6: State 6 write the value “1000000000000001” to the ADC which sets the ADC to data mode. In data mode all control data passed to the ADC are ignored and the ADC starts sampling data and providing it in its data channels.

State 7: The value of the register “reading” is set to 1 in this state enabling the minor FSM “readfsm” which reads from the ADC. The minor FSM (state 11) is described below. After the read operation is complete the system again comes back to this state. The system loops in this state until the FSM is reset.

State 8 (statewait): The system enters this state whenever the system must wait for a certain time. The system enters this state automatically irrespective of the present or next state whenever the register waitreg is greater than zero. At each clock pulse the system decreases the value of the waitreg register. When the waitreg register is down to zero, the system goes back to normal state and resumes operation exactly from where it got transferred to wait state. This state basically works as a delay and is used whenever the ADC requires a certain amount of time to complete its operation.

State 9 (donothing): This is the state of the FSM when the main FSM is in disabled state (reading or writing). The FSM does not perform any operation at all in this state.

writefsm: This is the minor FSM responsible for writing the 16 bit word to the ADC serially. This FSM takes the value in the 16 bit “datatosend” register and sends it serially through the SDI pin to the ADC. Each bit is send at the negative edge of the SDOFS input pin of the FSM. The ADC generates a pulse on its SDOFS pin every time it is ready to send or receive data. The system detects this pulse and after that two of these pulses have occurred, it waits for the serial communication clock SCLK positive egde to occur. When it detects the positive edge of the SCLK it puts each bit from the datatosend register serially through the SDI pin. This is repeated and clocked 16 times for each SCLK. After all 16 bits have been sent this minor FSM sets the writecomplete register to 1 signaling the major FSM that write operation has been complete and the main FSM can take control again and can resume normal operation.

readfsm: This FSM works similar to the writefsm and is responsible for reading from the ADC. It differs in one major way from the writefsm in that, it read six 16 bit values, one for each of the sampled channels rather than a single 16 bit word. The FSM does this by keeping track of which dataset it is currently reading in the “reading_6” register. Like the operation in writefsm, the readfsm disables the main

FSM and transfers control back to the main FSM when all 6 sets of data has been read. It signals the main FSM by setting the “readcomplete” register to 1 when read operation is complete.

A simple signaling mechanism consisting of single gate (Figure 4.13) is used to signal the main FSM whenever read or write operation is complete and the main FSM can resume its operation:

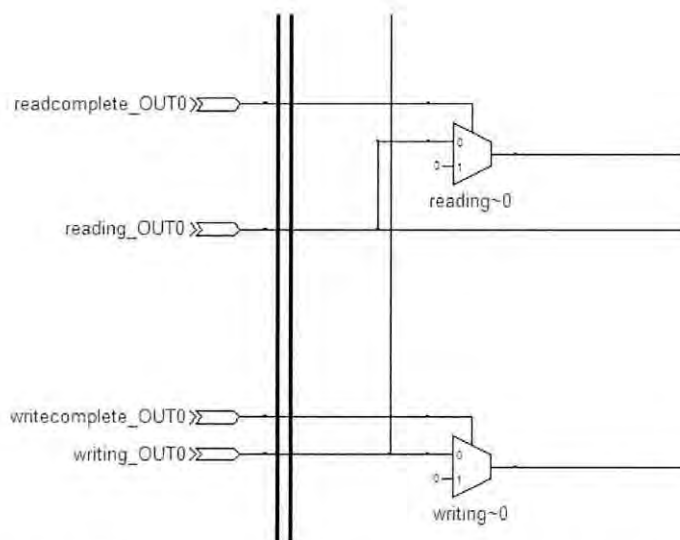


Figure 4.13 Signaling gates for write complete and read complete operation

After all the data for one second is acquired, the system interrupts the Nios processor making it read the data for processing. It was found that the system was sometimes unable to detect the edge of SDOFS pulse at higher frequencies resulting in loss of synchronization with the ADC.

The ADC is directly connected to the FSM block through the GPIO pins and controls all its operations. Figure 4.14 shows the partial interface of the FSM as it is connected to the GPIOs of the DE2 board. As can be seen from the figure, SDO, SCLK, SDOFS and other pins are directly connected to the FSM through the GPIO pins. Some of the RAM interface pins are also visible in the figure.

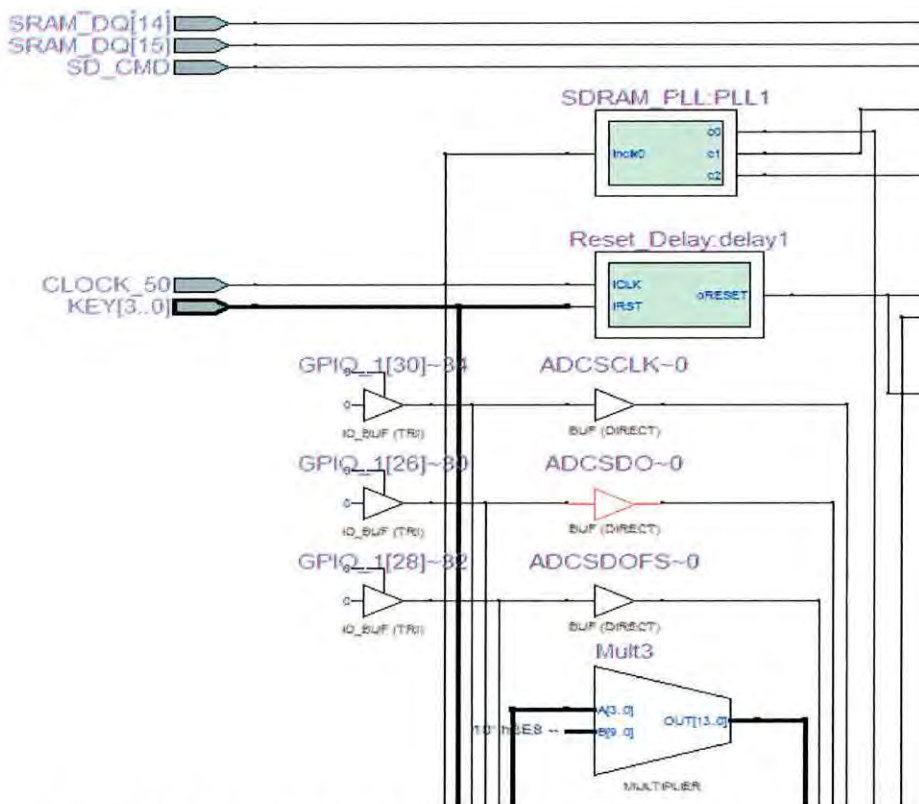


Figure 4.14 Partial RTL of the interfacing block connected to GPIOs

4.6.2 Storage Block

The storage block of the interfacing module stores the values captured from the ADC to the RAM. Instead of supplying the soft processor a constant flow of new data, this module stores the digitized and sampled value of the input AC signal in the random access memory (RAM). This module consists of twelve RAM modules each capable of storing one thousand 16 bit values. For a sampling frequency of 1kHz, there will be one thousand samples of each of the three voltages and three currents in one second. There are three sets of RAM for the three currents and three sets of RAM for the three voltage values. The storage block sequentially stores the voltage/current values received from the ADC to these RAMs until they are filled with one thousand set or one second of data. It does the same thing for the three voltage samples. When all six data set is complete and saved in RAM, this module interrupts the soft processor and starts saving the

new values in the other six sets of RAM. This method insures that no data is lost as the processor performs calculations over the previous 1000 values.

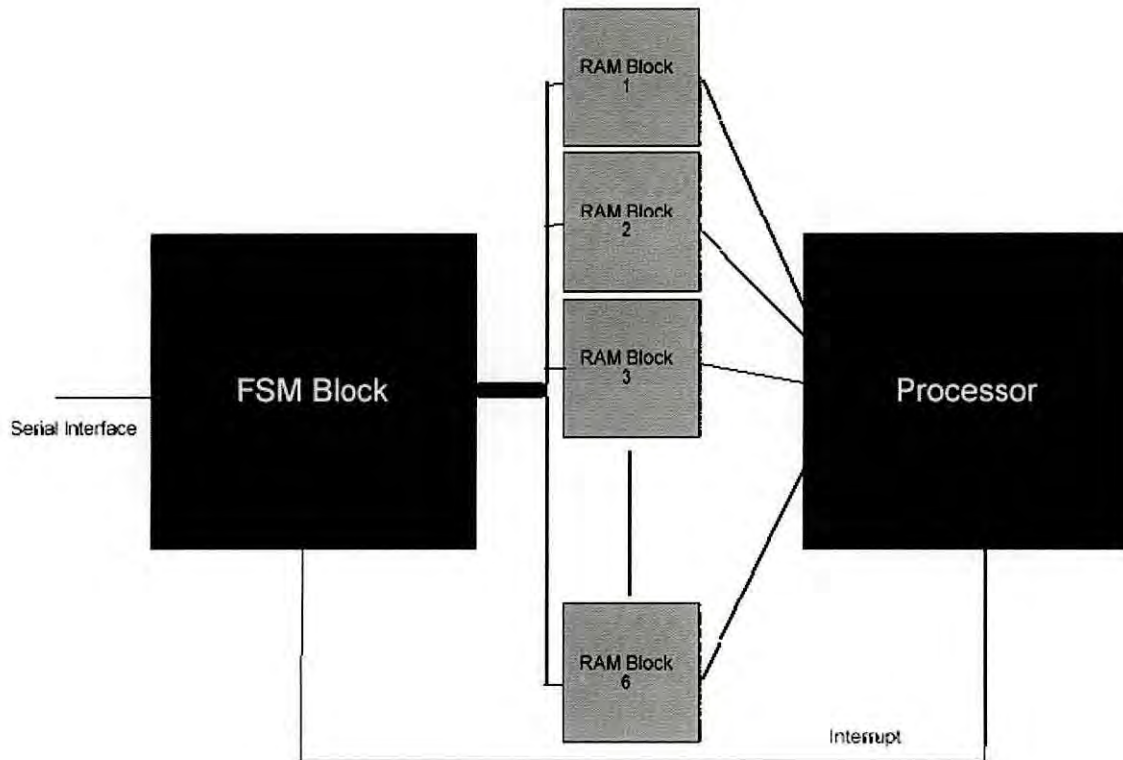


Figure 4.15 Storage block of the interfacing module

The complete flow diagram of the process is shown in Figure 4.16 on the next page.

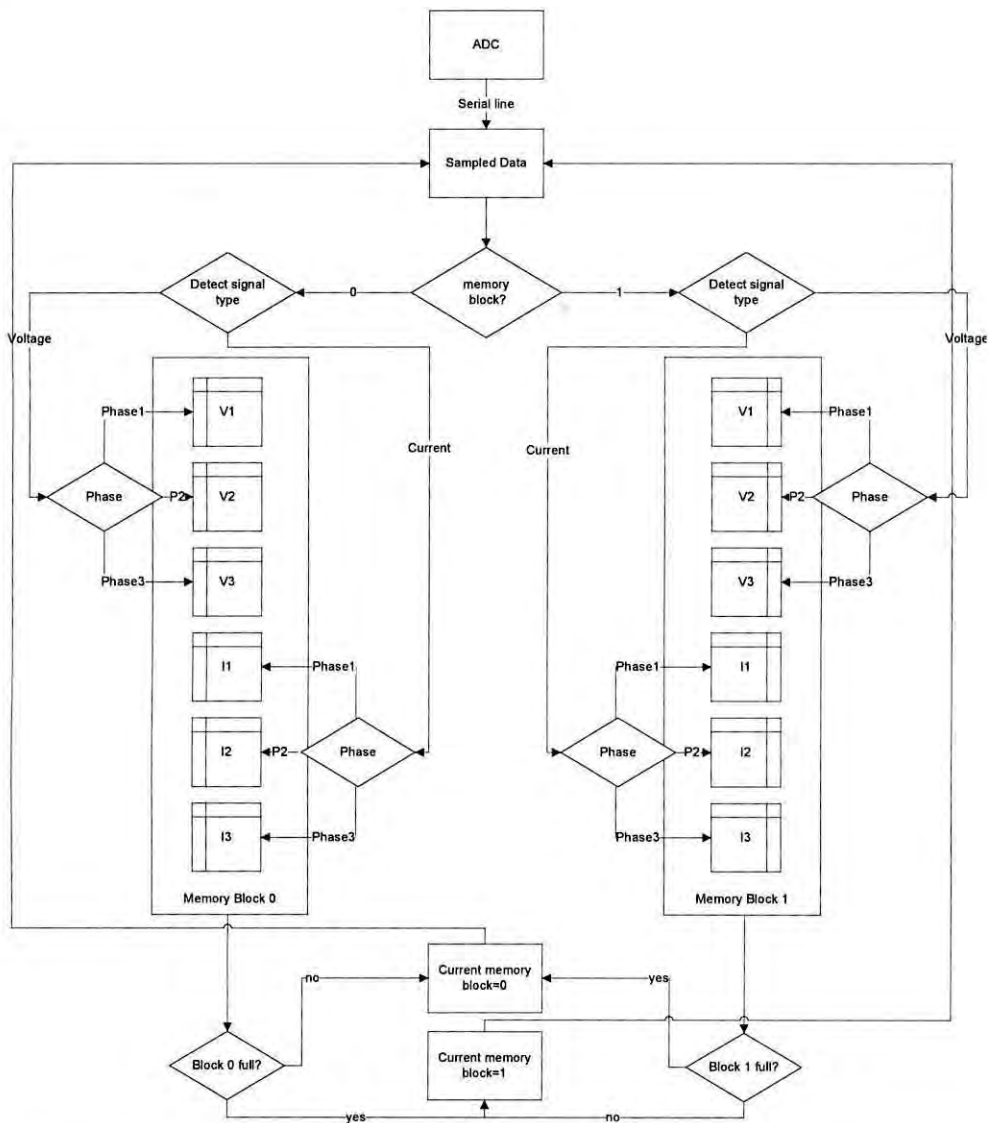


Figure 4.16 Flow diagram of storage block of the interfacing module

The I/O configuration and Verilog code of the complete block is mentioned in appendix B and C respectively.

4.7 Soft Processor

A 32 bit processor has been implemented over the FPGA to perform all the necessary calculations. The soft processor is the core of all the calculation. This processor is a 32 bit Nios II/f processor implemented over the FPGA. The

instruction set is RISC. This portion requires the most logic elements of the FPGA, as all the calculation instructions as well as data memory is implemented within this block. The soft processor does all the voltage, current and power calculation of the meter. The data from the ADC which is now stored in the RAM buffer are multiplied by a predetermined calibration factor for proper calculation before the data is fed to the summation circuit and also to the FFT unit. When all calculation has been performed, the data is sent to the LCD and shown in predetermined format. The processor starts performing its function when the phase buffers are full. There are six phase buffers and they contain voltage and current values of all three phases. Each data is squared and then added to the previous values, this repeats until all the buffers has been squared and added. The square root of the summations gives the rms value of the current and voltage. After all the rms values have been attained, the soft processor calculates the power. Another similar summation operation is used to calculate energy from instantaneous power.

The processor has been synthesized with the following additional component:

- 7 Segment display control
- VGA output module
- Three 32 bit parallel inputs (PIO) for reading voltage data from the FSM to the Nios processor.
- Three 32 bit parallel inputs (PIO) for reading current data from the FSM to the Nios processor.
- One 1 bit parallel input for interrupting the Nios processor whenever 6 sets of voltage and current data are ready from the FSM for the Nios processor.

4.8 Calculation Block:

We know that for rms and power claculation:

$$V_{rms} = \sqrt{\sum_{n=1}^N \frac{1}{N} \times V_n^2} \quad (4.1)$$

$$I_{rms} = \sqrt{\sum_{n=1}^N \frac{1}{N} \times I_n^2} \quad (4.2)$$

$$\text{Avg Active Power} = \sum_{n=1}^{\infty} V_n I_n \cos \theta_n \quad (4.3)$$

$$\text{Reactive Power} = \sum_{n=1}^{\infty} V_n I_n \sin \theta_n \quad (4.4)$$

Where V_n and I_n are the voltage and current at n^{th} instant of a particular phase and θ_n is the phase angle between them. This block uses equations (4.1) and (4.2) for rms calculations. But since we are calculating instantaneous power, we don't need to calculate the phase difference between the voltage and current value. Rather we use equation (4.5) to calculate instantaneous power at instant n directly.

$$\text{Instantaneous Power} = V_n I_n \quad (4.5)$$

Equation (4.6) denotes equation used for power calculation.

$$\text{Power} = \sum_{n=1}^{\infty} V_n I_n \quad (4.6)$$

The summation of the instantaneous power gives the entire power consumed at that time which when multiplied by the sampling interval gives energy.

$$\text{Instantaneous energy} = V_i I_i \Delta t \quad (4.7)$$

Where Δt is the sampling interval. Total energy over a period is equal to summation of instantaneous energy over that period.

$$\text{Energy} = \sum_{n=1}^N V_i I_i \Delta t \quad (4.8)$$

The calculation block uses the hardware adders to accelerate the addition process. The software part of this block is ANSI C code which is compiled by GNU C Compiler (GCC) and run on the 32 bit Nios processor.

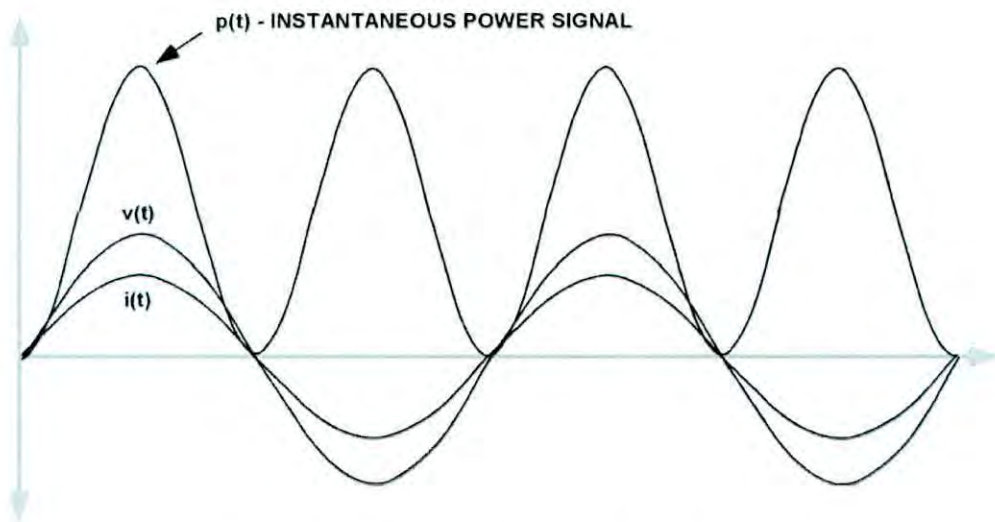


Figure 4.17 AC Signal (Voltage and Current)

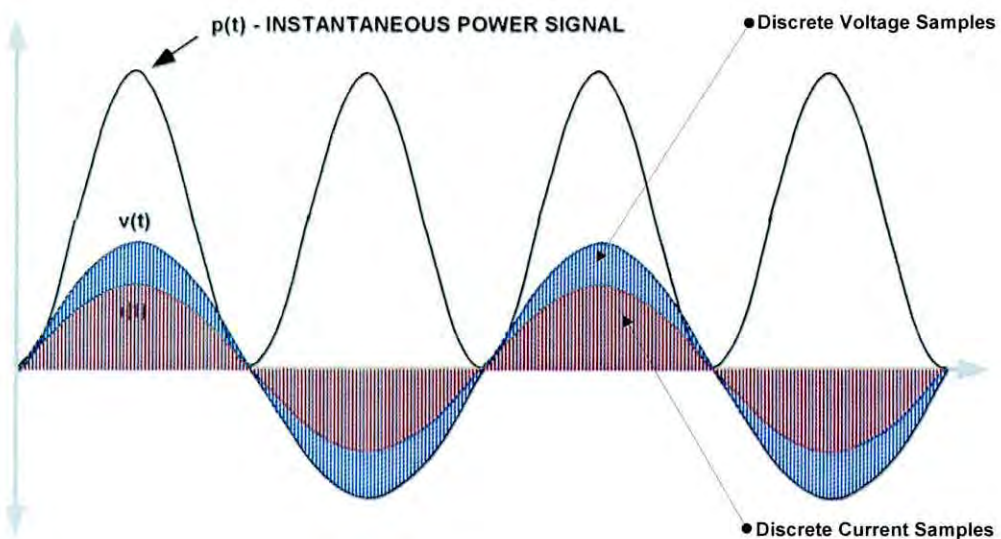


Figure 4.18 Sampled Voltage and Current Signal

The multiplication of instantaneous voltage and current gives instantaneous power. Accumulation of instantaneous power over time give the power consumed during that time. A typical AC signal is shown in Figure 4.17. This signal is continuously sampled. Figure 4.18 shows the original signal and the sampled discrete voltage and current signals. Since the sampled signals are exactly at the same instant, the direct multiplicand of those two gives the accurate instantaneous power.

The pseudo code that calculates the RMS values of voltage and current channels is listed in appendix D.

4.9 DFT/FFT Block

4.9.1 Fourier Transform

The Fourier transform maps a signal from the time domain into the frequency domain. The resulting frequency spectrum shows the frequency content of the original signal. The Fourier transform is based on decomposing a signal into an infinite set of orthogonal basis functions formed from the Fourier series.

4.9.2 Discrete Fourier Transform

The Fast Fourier Transform makes use of a discrete-time, finite-domain Fourier transform known as the Discrete Fourier Transform (DFT). The Discrete Fourier Transform is used for input signals that are both discrete in time and have a finite duration (finite domain) which the voltage and current signal in our case. The sampled data that we get from our ADC is both discrete and has a finite duration. The DFT transforms an input sequence of N complex numbers (x_0, x_1, \dots, x_{n-1}) into an output sequence of N complex numbers (X_0, X_1, \dots, X_{n-1}). The DFT is given by equation (4.9).

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}nk} \quad k = 0, \dots, N-1 \quad (4.9)$$

The exponential factor is often referred to as a twiddle factor and expressed with the notation in equation (4.10).

$$W_N^{nk} = e^{-j\frac{2\pi}{N}nk} \quad (4.10)$$

The DFT can be thought of as match filter that is convolving the input signal x_n with an orthogonal set of sinusoidal basis functions given by the twiddle factors $W(nk, N)$. Since the input sequence is real valued, the complex component

of the input is always zero. As a result, the complex output frequencies of the DFT are symmetric about the Nyquist frequency (half the sampling frequency). In our case, the input sequence can be thought of as the set of real valued samples from the voltage or current channel. More concretely, let N be 32 and the sampling frequency be 1.024 kHz. This means that the input is 32 samples of the current waveform, and the output is 32 equally spaced frequency bins of 32 Hz from 0 to 1.024 kHz.

We may decompose a periodic waveform $f(t)$ into the summation of a number of sinusoidal waveforms easily using DFT as in equation (4.11).

$$f(t) = A_0 + \sum_{\gamma=1}^{\infty} (B_{\gamma} \sin \vartheta \omega t + C_{\gamma} \cos \vartheta \omega t) \quad (4.11)$$

Where A_0 is the amplitude of the DC components and B_{γ} and C_{γ} are coefficients of each harmonic. For AC signals as in our case, A_0 is zero. The amplitude of each harmonic can be computed from equation (4.12).

$$A_{\gamma} = \sqrt{B_{\gamma}^2 + C_{\gamma}^2} \quad (4.12)$$

The coefficients B_{γ} and C_{γ} for each harmonic are calculated multiplying the corresponding sine and cosine wave to the input signal respectively as in equations (4.13) and (4.14).

$$B_{\gamma} \approx \frac{2\Delta t}{T} \sum_{\gamma=1}^n V_{\gamma} \sin \vartheta \omega t \quad (4.13)$$

$$C_{\gamma} \approx \frac{2\Delta t}{T} \sum_{\gamma=1}^n V_{\gamma} \cos \vartheta \omega t \quad (4.14)$$

where Δt is the time between samples, T is the period and V_{γ} is the input signal.

4.9.3 Fast Fourier Transform

The DFT is very computational intensive and cannot be calculated easily with limited processing power at real-time. As seen from equation (4.10), each frequency component $X(k)$ requires N complex multiplications and N complex additions. There are N frequency components, so an N -point DFT requires $2N^2$ complex operations (N^2 multiplies, N^2 sums). In big O notation, the DFT has a computational complexity of $O(N^2)$. The Fast Fourier Transform uses several optimizations to reduce the number of operations to $O(N\log N)$. The fast Fourier transform (FFT) is a highly efficient method for calculating the discrete Fourier transform (DFT). The DFT is used in signal processing applications for a range of purposes, such as analyzing the frequency components of signals and data compression. In our case the signal is the harmonics in the voltage and current signals. The DFT is a computationally intensive function. A native (non-FFT) implementation of an n -point DFT requires N^2 complex multiplications. The FFT algorithm achieves its efficiency gains by decomposing the DFT into a number of smaller DFTs and exploiting the symmetry and periodicity of the sub stages to reduce the number of calculations. An N -point FFT only requires $N \times \log_2 N$ complex multiplications. Cutting down the number of complex multiplications improves the FFT performance, often by several orders of magnitude, depending on the order of the transform. We have optimized speed by pre-calculating sine and cosine terms used in the butterfly calculations and moving all values to RAM and directly feeding values from RAM to the adders bypassing the processor. These sine and cosine terms are called twiddle factors. A sample soft core implementation of the system is shown in appendix E and the hardware accelerated implementation in appendix F.

The block diagram of hardware accelerated FFT calculations is shown in Figure 4.19 and the actual synthesis shows in figure 4.20. As can be evident from the figure, buffers and RAMs are used to accelerate the calculations. Rather than serially moving each data value from RAM to the Nios processor and performing addition and multiplication, data is moved directly to the multipliers and adders

from the RAM in parallel. This reduces the time by removing the time required to move data from RAM to the Nios registers and also by performing the calculations of all three phases in parallel rather than serially.

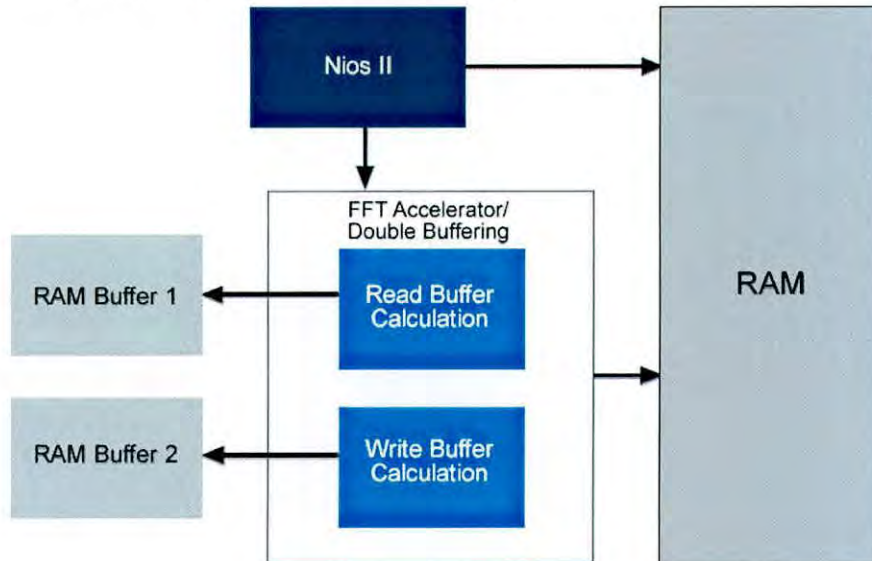


Figure 4.19 Block diagram of hardware accelerated FFT calculation

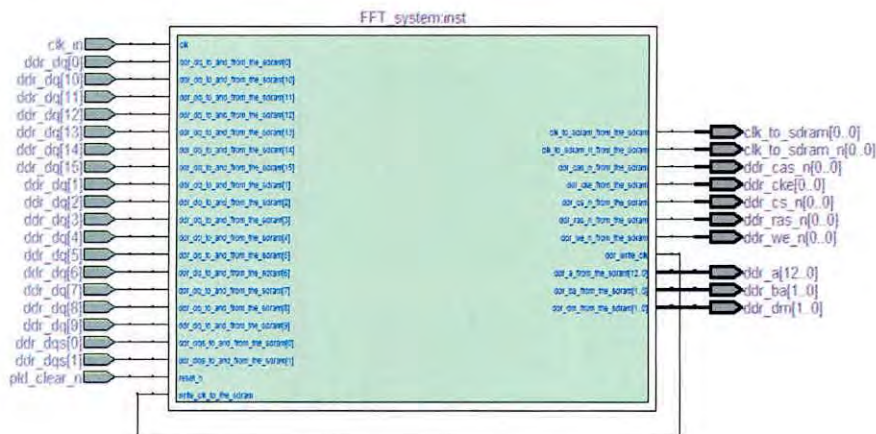


Figure 4.20 FFT block synthesized using Verilog

The overall design and interconnect of the HDL implementation is shown in Figure 4.29. Figure 4.30 shows the actual photo of the implementation in the DE2 FPGA board.

4.10 Acceleration Unit

A three phase metering system requires a lot of calculation and hence it is a processing intensive operation. Normal processors are not capable of performing these calculations in real-time or within an acceptable time interval. So, the main challenge is to either reduce the number of calculations necessary to calculate all the required parameters or somehow accelerate the calculations. Our design achieves its speed by accelerating all calculations. The acceleration methods are:

1. Enhancing design for parallel operation
2. Performing calculations in parallel
3. Using double buffering to increase memory speed
4. Accelerate calculation with pre-calculated values

To achieve these, it was necessary to introduce several key components and make modification to system design. Some of the components added are:

1. SDRAM
2. Dual Port on-chip RAM
3. Hardware Multiplier
4. RAM Arbitrator

4.10.1 SDRAM

The SDRAM is used to store all sampled current and voltage values from the ADC. Since the sampling frequency is 1kHz and there are three current channels and three voltage channels, so there are 6,000 samples per second which are stored in the SDRAM. The RAM has capacity to store another 6,000 sample sets each of which are 16bit values. The SDRAM is composed of input output (IO) cells capable of storing bits (Figure 4.21). However the SDRAM cannot be driven directly as the system bus speed of 100MHz is different than the SDRAM bus speed of 25MHz. A PLL is introduced in the system as shown in Figure 4.22 which interfaces the 25Mhz SDRAM to the 100MHz Avalon bus. The PLL block is comprised of 16bit read and write register and counters.

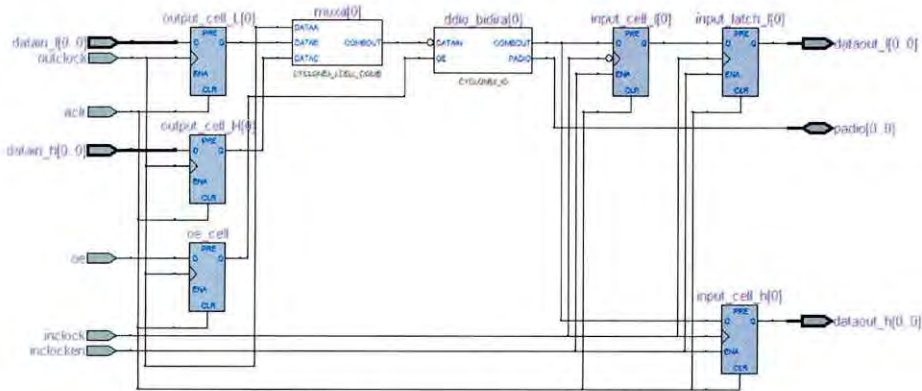


Figure 4.21 SDRAM IO Cell

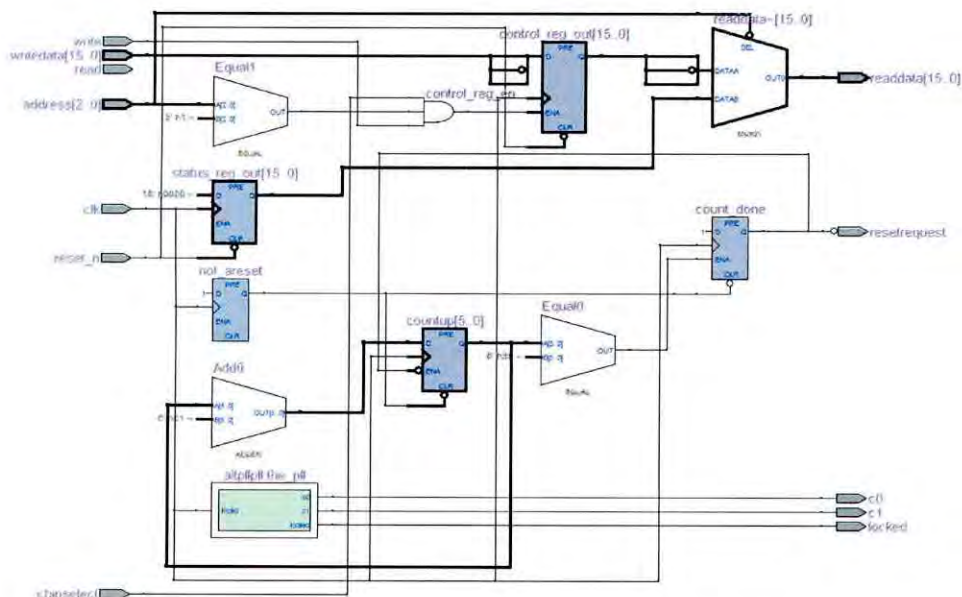


Figure 4.22 PLL required for SDRAM

4.10.2 Dual Port on-chip RAM

The dual port on chip RAMs are required for the ping pong buffers and for storing the pre-calculated twiddle values for the FFT. Since these on-chip memories are dual port it is possible to read and write from the simultaneously using different ports. This has the advantage of accelerating the calculation by using these blocks as ping pong buffers. The block is shown in Figure 4.23. The block comprises on numerous RAM blocks all connected together to form a RAM

chain. The detail of the RAM chip block is available in the Verilog code in Appendix C. The block memory was assigned using SOPC builder and memory address assigned.

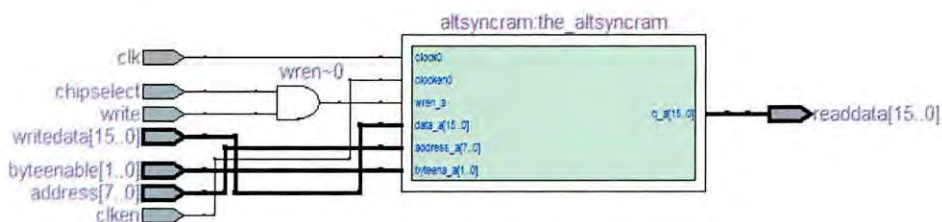


Figure 4.23 Synthesized RAM Block

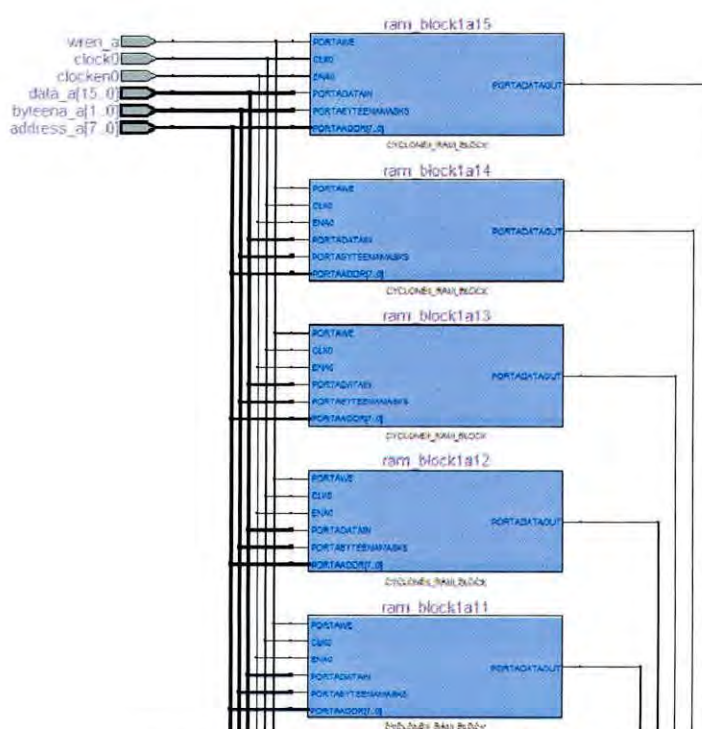


Figure 4.24 Partial section of the on-chip memory

4.10.3 Hardware Multiplier

It is evident from equations (4.8) and (4.13) that both energy and FFT calculation requires multiplication and addition operation. We accelerate this calculation by performing all these multiplications in parallel as well as using hardware multipliers. Since each sampled phase data are saved in a different RAM

block, it is possible to access those blocks simultaneously. Hence it is possible to not only calculate all six channel data simultaneously but also perform the FFT parallel to this operation. Figure 4.25 shows the hardware multiplier in our design.

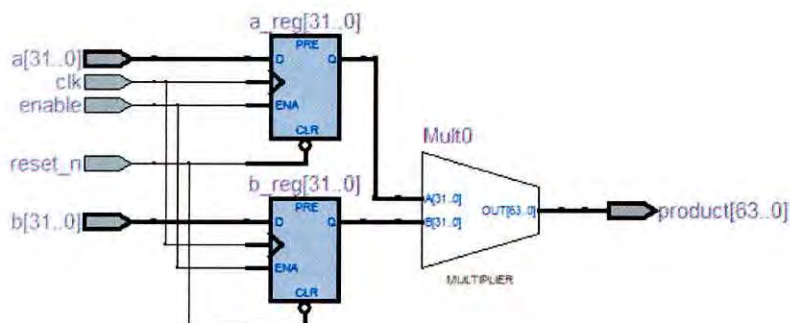


Figure 4.25 Hardware Multiplier

4.10.4 RAM Arbitrator

Since the memory is being accessed by both the data acquiring FSM, the accelerator unit and also the processor, it is necessary to introduce an arbitrator to the RAM to avoid any kind of conflict in RAM access. Figure 4.26 shows the arbitrator which connects different modules to the RAM bus.

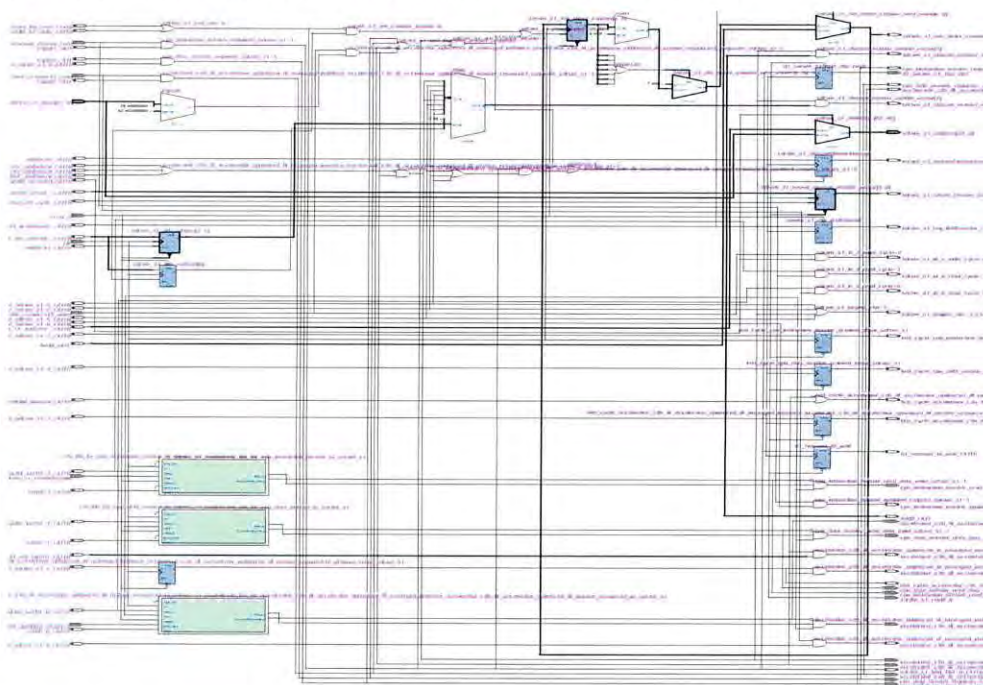


Figure 4.26 RAM Arbitrator

4.10.5 Other blocks

Although the blocks mentioned above comprises the bulk of the accelerator unit, there are also several other blocks and logics present within the system. One such system is the accelerator state machine which runs the accelerator as it goes through different stages. Figure 4.27 shows part of the accelerator subroutine state machine. Input to this block is the accelerator begin, read and select and depending on the various input, the accelerator block advances to various states and performs its operation.

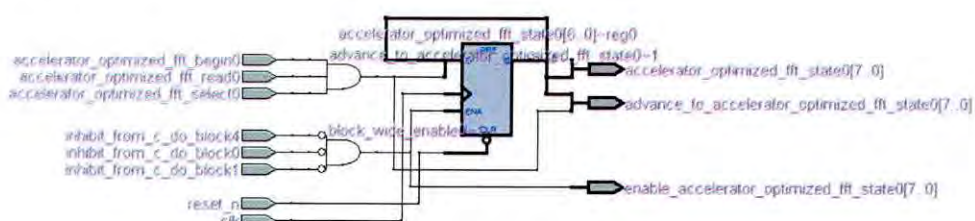


Figure 4.27 Accelerator Subroutine State Machine

Figure 4.28 shows the interface between the accelerator unit and the Nios II processor. It comprised of readdata from the processor, waitrequest to request the processor to wait while it takes over execution from the processor and performs its operation, dataout, processoraddress and other required inputs.

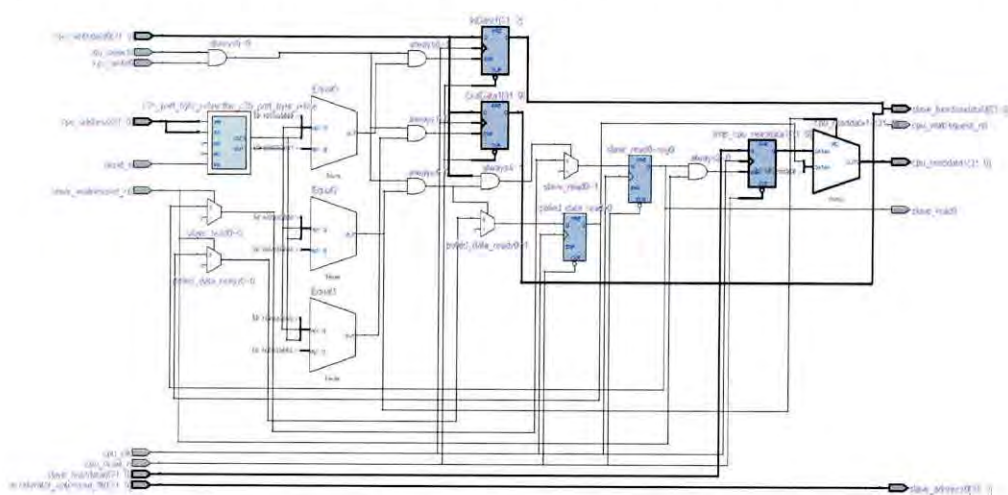


Figure 4.28 CPU Accelerator Interface Instance

4.11 Storage Block

This block is responsible for storing the energy consumption calculated by the meter. There are two separate registers for storing the energy consumption, one stores the watt-hour value and the other kWh value. These are unsigned 32bit registers so they are capable of storing values up to 42,949,67,296 units. After each second of calculation of the consumed energy by the processor this is added to the watt-hour register. If the register value exceeds 1000, then 1 is added to the kWh register and 1000 subtracted from the watt-hour register.

For a typical energy consumption of 1,440 kWh/day (60kW) the meter is capable of storing energy consumption for

$$\frac{4294967296}{1440 \frac{kWh}{day} \times 365 \frac{day}{year}} = 8171 \text{ years}$$

Even with a hypothetical maximum energy consumption of 1,00,000 kWh/day the meter is capable of storing energy consumption for

$$\frac{4294967296}{100000 \frac{kWh}{day} \times 365 \frac{day}{year}} = 117 \text{ years}$$

So the meter can operate and store energy calculation for approximately 8171 years before requiring any kind of reset under typical load. These two register values are also saved to the Flash memory every second. If the meter is reset or loses power at any occasion, these two values are read from the Flash memory during meter power up and the meter continues from the last calculated value without any kind of problem.

4.12 Meter Specification

The proposed meter specification is furnished below.

Power Supply:

- Power supply: Input: 100 to 250VAC. Internal 3.3VDC.
- Supply Current: 300mA (typical)

Accuracy:

- Class 0.2
- $\pm 0.1\%$ at unity power factor.
- $\pm 0.2\%$ at 0.5 power factor.

Input Signal Range:

- Max Voltage Input per phase: 250 rms AC (Phase to neutral)
- AC Voltage (VA, VB, VC): 0 to 270 (rms)5
- AC Current (IA, IB, IC): 0 to 60A.

Frequency:

- 50 Hz.

Energy Measurement:

- 3 Element, 4 Wire, Three Phase Wye.

Interface Connectors:

- Voltage Inputs
- Current Inputs
- Neutral

Programmable Features:

- Programmable unit charge
- Holiday/Weekend variable billing option
- Set Load Control Thresholds.
- CT / PT Ratios.
- Meter ID / Serial Number
- Calibration factors

LCD Display

- Load Profile Data – kWh
- Instantaneous Voltage, Current, Power Factor by Phase.
- Watt Hours, VA
- V Phase A: Phase A voltage.

- V Phase B: Phase B voltage.
- V Phase C: Phase C voltage.
- I Phase A: Phase A current.
- I Phase B: Phase B current.
- I Phase C: Phase C current.
- Energy Consumption and Demand with Time Stamp.
- Harmonic Distortion

Operating Temperature

- 10C to 70 deg C.

Humidity

- 5% to 100%

Standards

- IEC62053-22

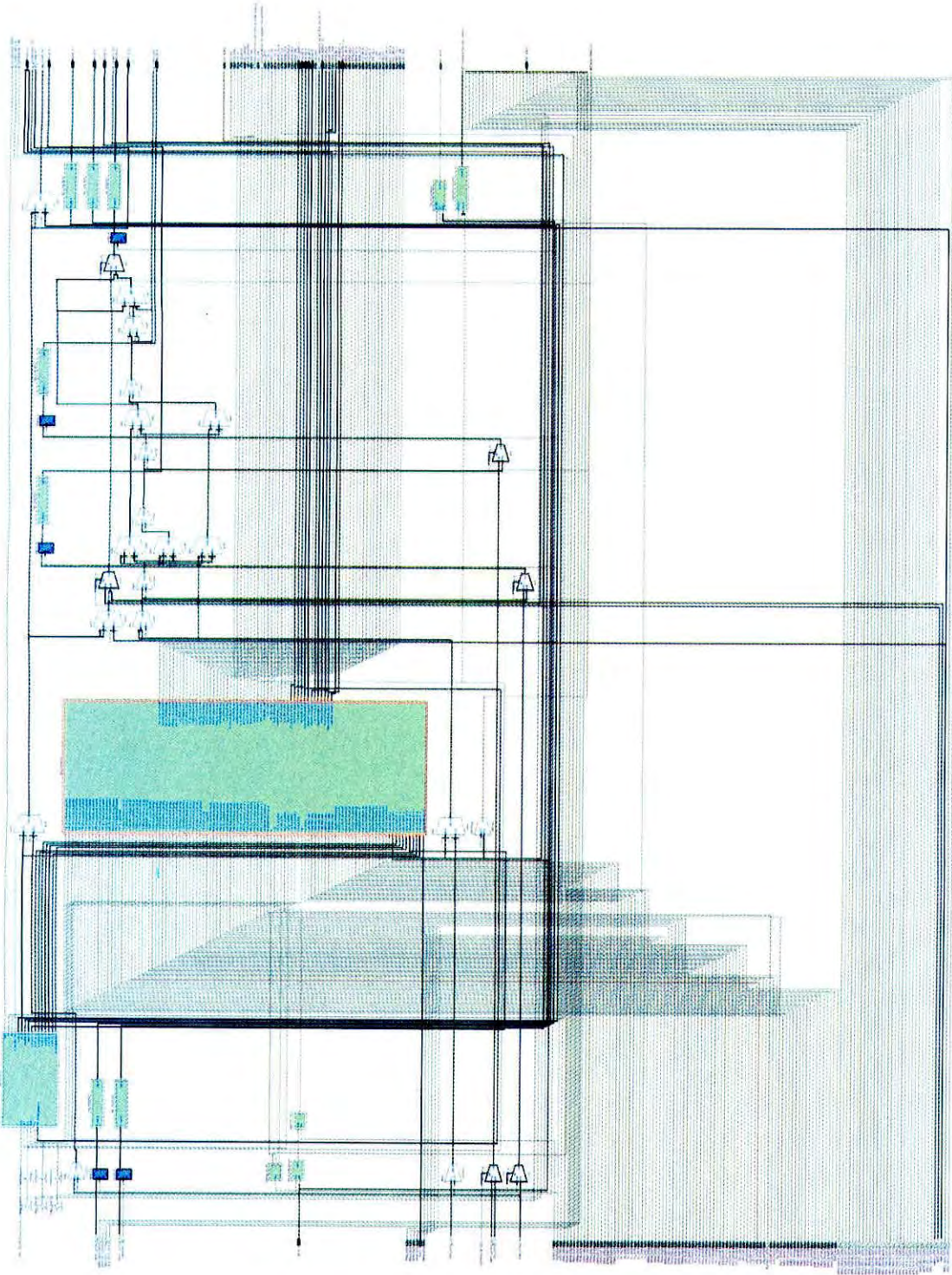


Figure 4.29 Block Diagram of Nios and FSM interconnect

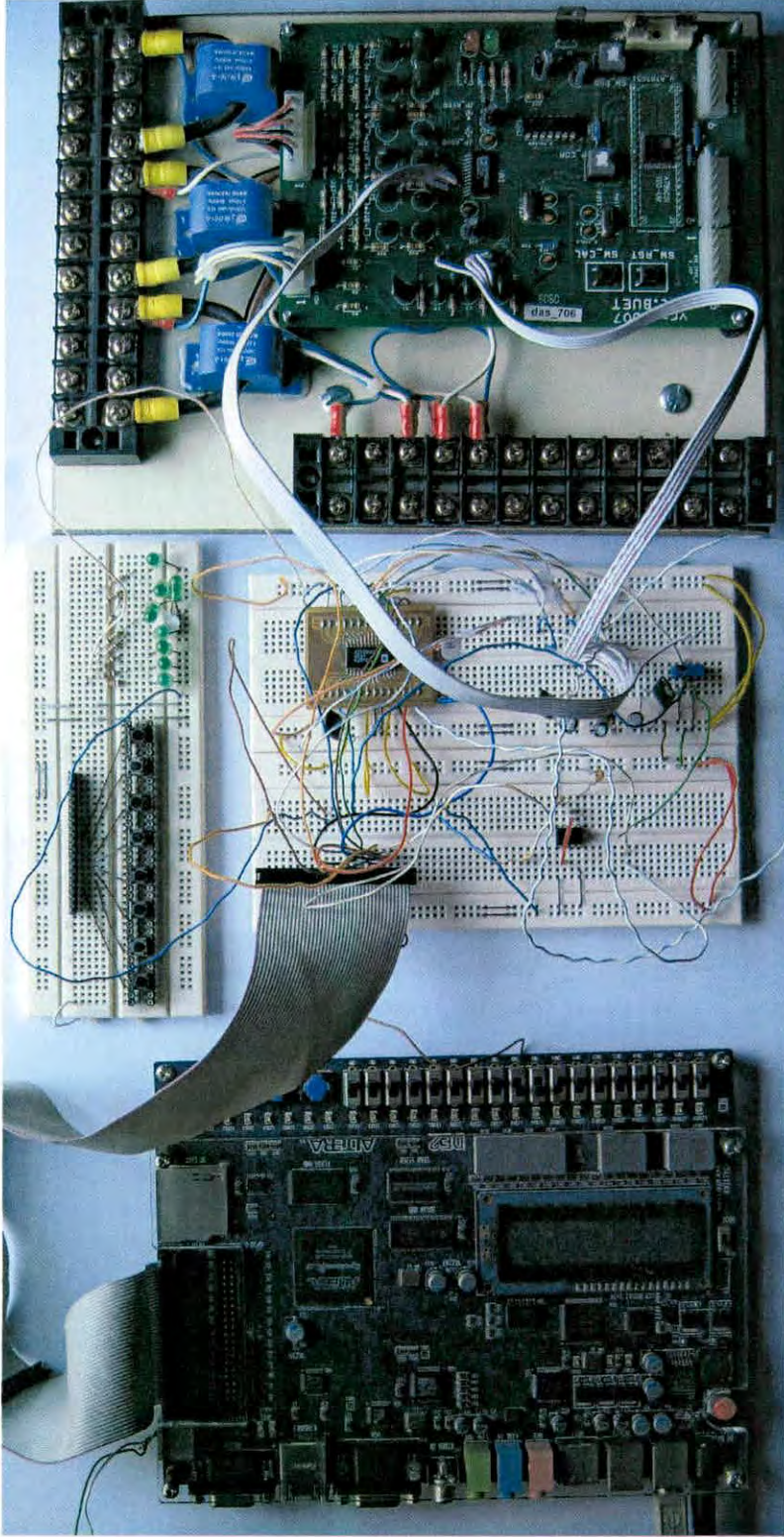


Figure 4.30 Physical implementation of the entire design

Chapter 5

Simulation and Results

5.1 Necessary Files and Tools

The proposed SoC is a single chip solution for a 3 phase energy meter. It operates at 3.3V and draws its power from the power line. The following tools were used for the simulation of the meter:

- Quartus II: Used for schematic drawing, netlist generation and simulation
- SignalTap II: Used for probing

The simulation process is performed on the hardware synthesized from the Verilog code. To simulate the energy meter implementation, first the Verilog code is compiled. The compilation process includes analysis and synthesis, I/O assignment analysis, assembler, fitter placement and finally classics timing analyzer. After successful synthesis, a functional simulation netlist generation is required to actually test and simulate the logic. Finally the logic is simulated with the input waveforms from the vector waveform files and the output matched with the desired waveform outputs for that logic.

Since the simulation of an ADC is beyond the scope of this work, this simulation starts after the digitization process of the ADC. This process starts with simulating the inputs to the FSM from the ADC and inspecting the output. The main signals involved are RESET, SE, SCLK, SDO and SDI. RESET is the reset pin of the ADC, SE is the serial port (SPORT) enable pin and SCLK is the serial clock output of the ADC. The input and output lines of the ADC are SDI (serial data in) and SDO (serial data out) respectively. All of these are single bit lines. As mentioned before the input FSM consists of 11 distinct states. Figure 5.1 illustrates the first state. It can be seen that the RESET and SE pin are low during the first few cycles and then they are brought up to high so that the ADC can perform its operation. Figure 5.2 shows the simulation of state 1. In state 1 binary 1000001110001000 is transmitted by the serial port to the ADC. The “data to send” register contains this 16 bit value. This value is transmitted correct through the SDI output pin to the ADC. This cycle starts

after the SDO pulse has occurred. The bottom of Figure 5.2 shows the output of the internal SCLK positive edge detection circuit. It is evident from the simulation that the correct values are indeed put into the SDI line from FSM at the end of each positive edge of SCLK.

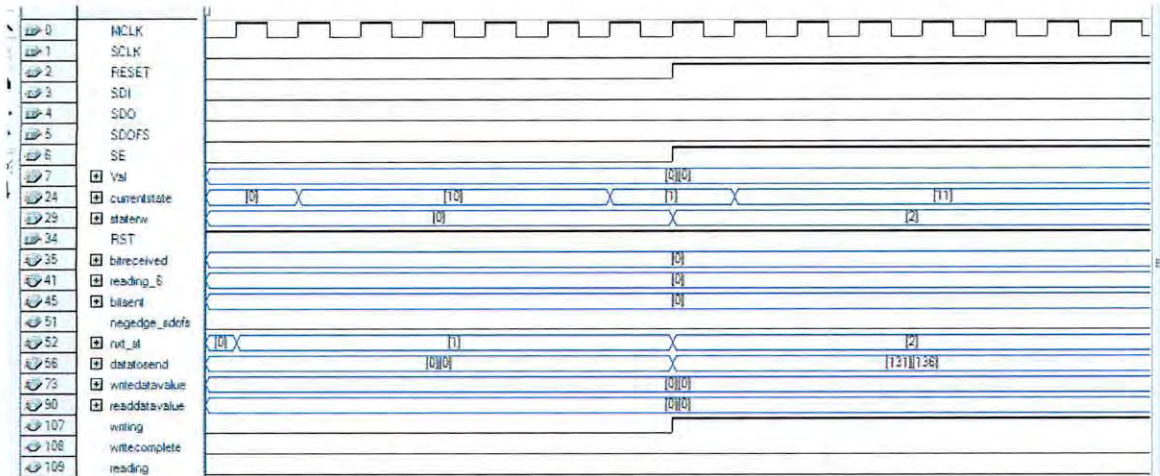


Figure 5.1 State 0. ADC reset.

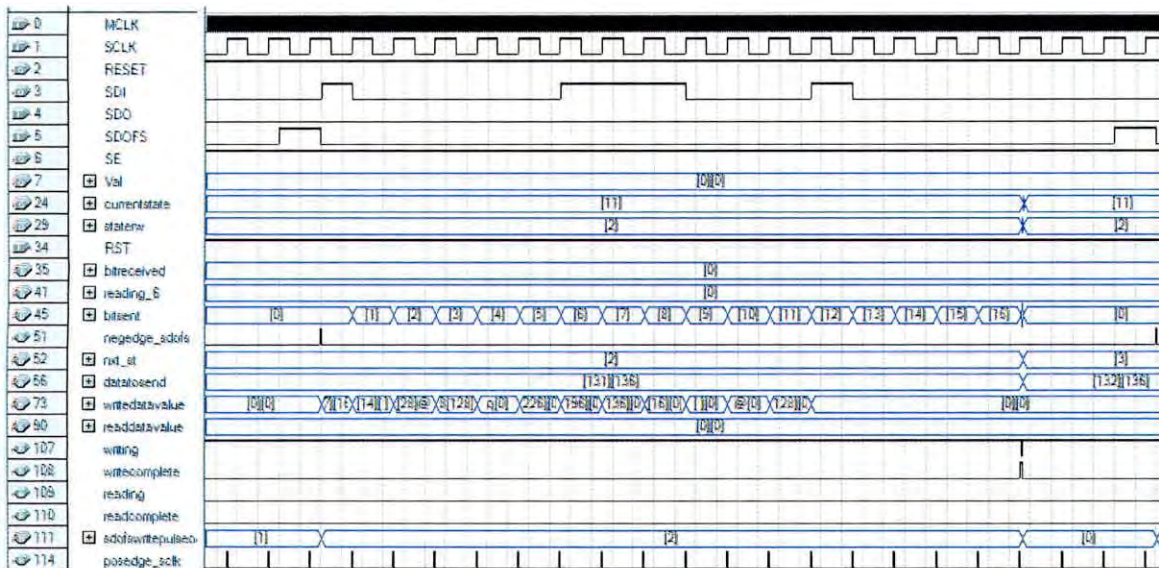


Figure 5.2 State 1. Turn on channel 1 and 2.

In state 2, channel 3 and 4 are turned on by sending “10000**100**10001000” serially through the SDI line. Figure 5.3 shows the simulation of this state. The register `nxt_st` which contains the next state value, correctly shows 3 as the next state. Also the SDI line shows the correct output.

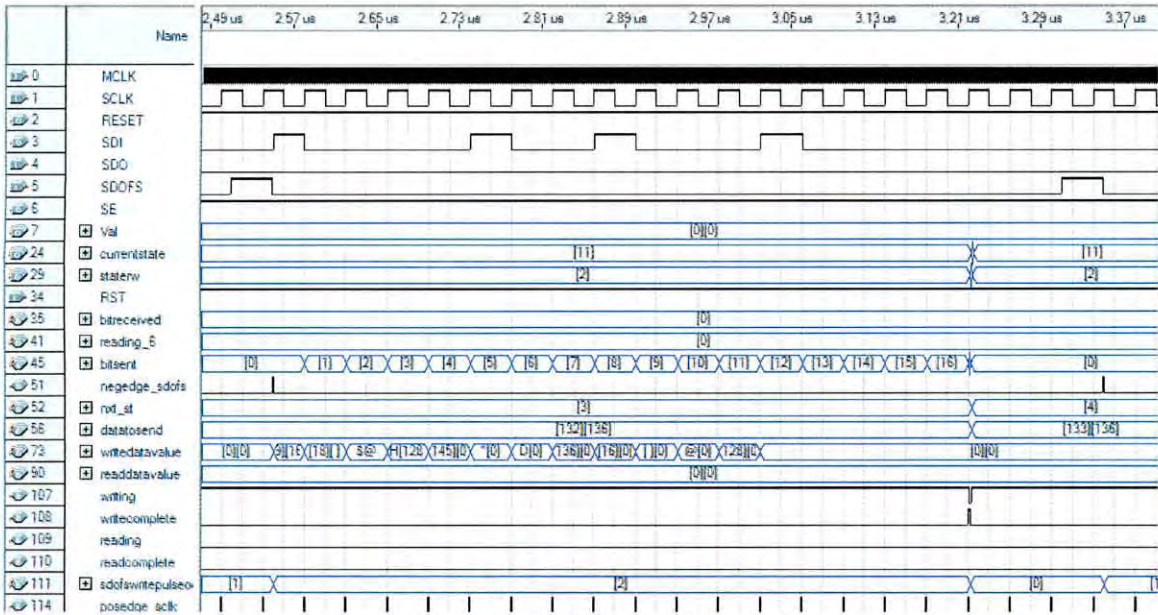


Figure 5.3 State 2. Turn on channel 3 and 4.

The next simulation is for state 3 where channel 5 and 6 are turned on (Figure 5.4). The SDI output is “1000010110001000” as correctly simulated. The next state is also 4 as correctly shown. The data can be seen to be correctly corresponding with the bitsent register which counts how many bits have been sent. The SDI pin is brought to low after all 16 bits have been sent.

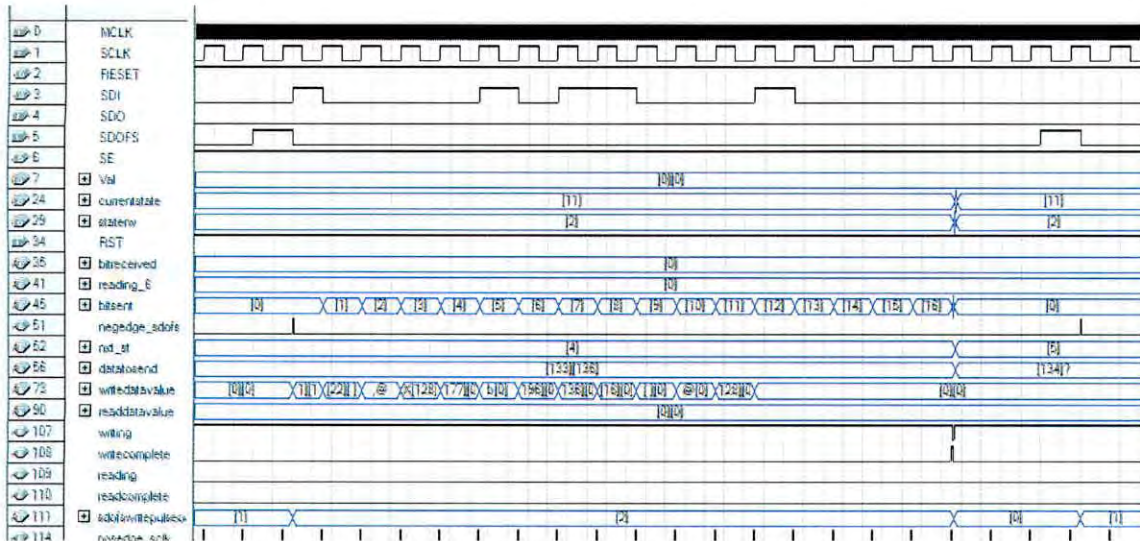


Figure 5.4 State 3. Channel 5 and 6 turned on.

The next state simulation (Figure 5.5) also shows the correct data “1000011000111111” written to the serial output.



Figure 5.5 State 4. Setting sampling rate 1kHz.

In state 5 as shown in Figure 5.6, regular non inverted mode is set. The simulation shows correct state transition as well as correct data output (“1000011100111111”) to SDI pin.

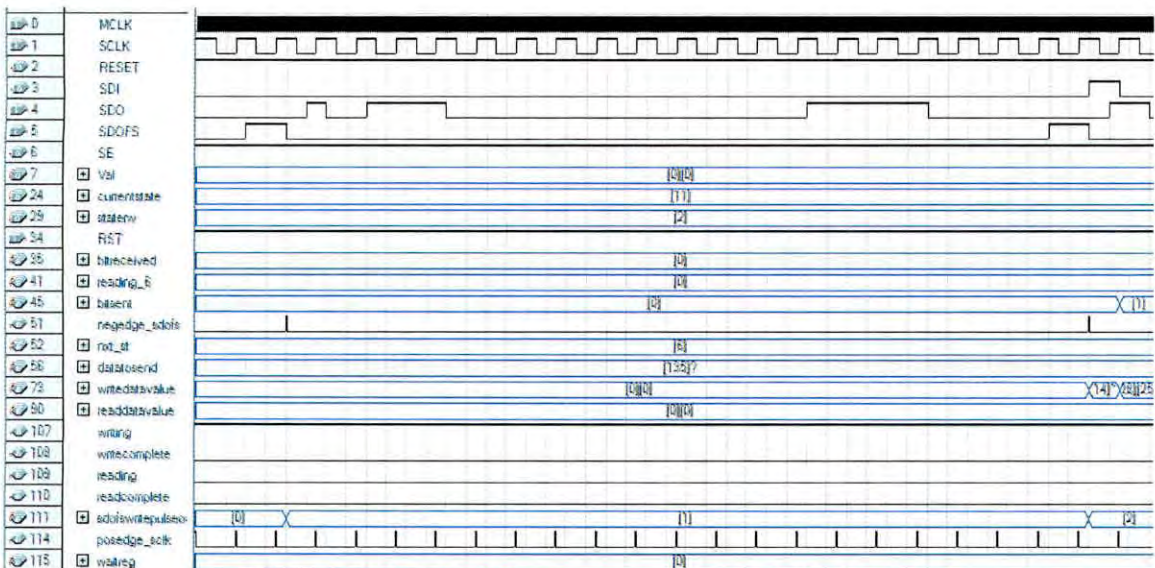


Figure 5.6 State 5. Setting non inverted mode of ADC.

At the end of each data cycle, data is moved to the RAM block where it is later processed by the Nios processor. Instead of serially processing the data as done normally by the processor, a smart block is implemented that processes the data in parallel and reduces the computational time. A simulation of linear non optimized cycle shows:

Data from RAM block to processor register	: 1 cycle *
Addition performed on the data	: 2 cycle
Data moved back to RAM	: 1 cycle
<hr/>	
Total time required	: 4 cycle (addition)
× Total data	: 1000
<hr/>	
Total cycles required for 1 phase	: 4000 cycle (multiplication)
× Total number of phases	: 6
<hr/>	
Total cycles for all 6 phases	: 24000 cycle (multiplication)

We then simulated our smart implementation which performs all additions directly using the hardware adders bypassing the Nios processor registers:

RAM->adders : 0 cycle	RAM->adders : 0 cycle	RAM->adders : 0 cycle
Addition : 2 cycle	Addition : 2 cycle	Addition : 2 cycle
Adder->RAM : 1 cycle	Adder->RAM : 1 cycle	Adder->RAM : 1 cycle
<hr/>		
Total time : 3 cycle	Total time : 3 cycle	Total time : 3 cycle
<hr/>		
Total time required for all 6 phase	: 3 cycle **	
× Total number of data	: 1000	
<hr/>		
Total cycles for all 6 phases	: 3000 cycle (multiplication)	

As can be seen from the above simulation, the time required is reduced 8 times.

* cycle means Nios processor cycle, not clock cycle.

** Only 3 parallel paths shown here. Actual operation is on 6 paths for 6 phases.

Chapter 6

Measured Data

6.1 Calibration

Calibration of a device is necessary before it can be used for any measurement purpose. Our implemented meter calibration was done using phantom loading with a reference calibration meter. In phantom loading, no external load is connected in actual sense and the current and voltage coils are connected separately so that it will consume only less power. In this connection the voltage across device will be supply voltage even if the variac is in minimum position. The meter test results conducted are at 20 points, from 100 mA to 50A, at power factors of 1 and 0.5L as rated in Table 6.1.

A class 0.2 meter, as required by the IEC specification, must not be more than 0.3% error at PF=0.5, and no more than 0.2% error at PF=1. The results from our implementation marginally pass these requirements and can be called class 0.2 compliant. However, in volume production using this same design might yield results outside the limit of those mentioned here due to variance in components and current transformer phase response from meter to meter. For reference, this table shows Class 0.2 limits as stated in the IEC62053-22 document [37].

Table 6.1 Class 0.2 limits as stated in IEC62053-22 document

IEC ACCURACY LIMITS FOR Variation Testing 0.2S ENERGY METERS		
Current	Power Factor	Class 0.2
$0.01 I_N < I < 0.05 I_N$	1	± 0.4
$0.1 I_N < I < I_{MAX}$	1	± 0.2
$0.02 I_N < I < 0.1 I_N$	0.5L	± 0.5
	0.8C	± 0.5
$0.1 I_N < I < I_{MAX}$	0.5L	± 0.4
	0.8C	± 0.4

6.2 Measured Forms

Figure 6.1 shows the sampled waveform of the input voltage. The same signal is also shown in Figure 6.2 as seen in an oscilloscope.

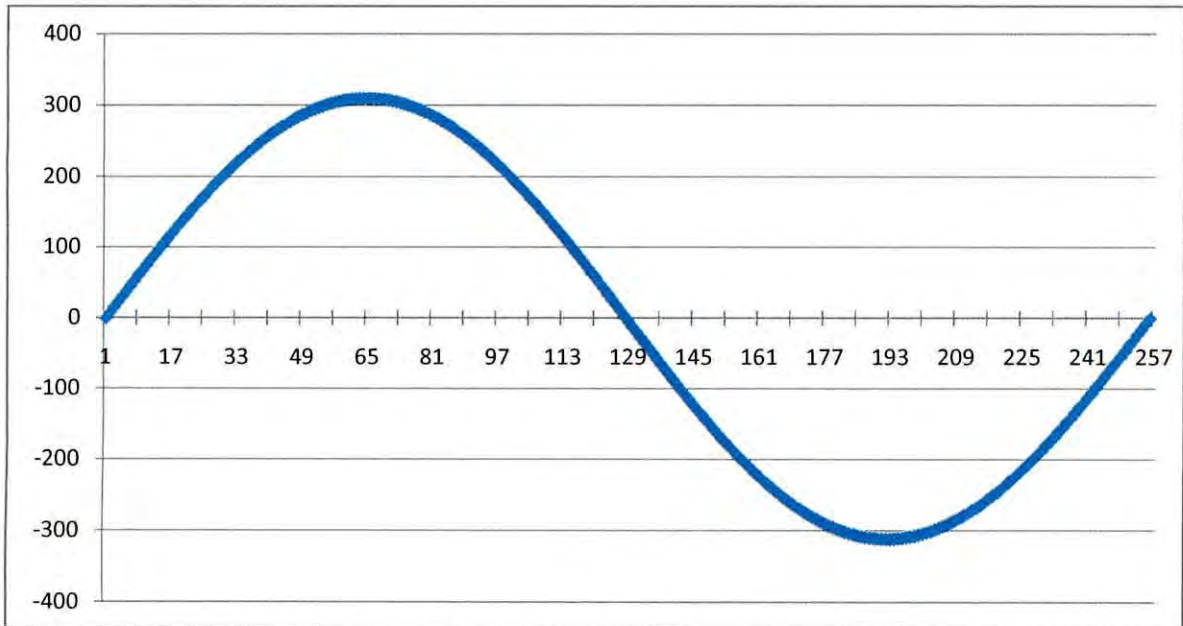


Figure 6.1 Input Voltage Data

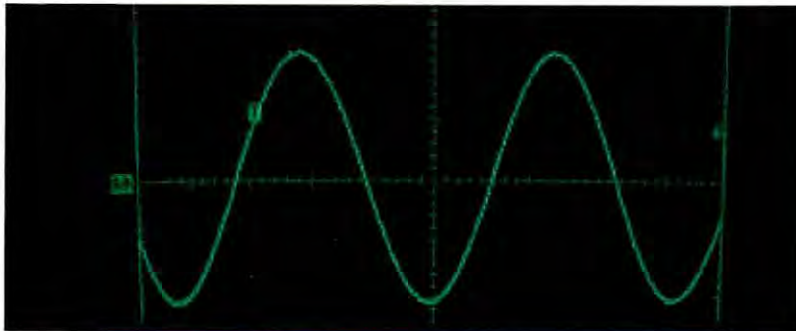


Figure 6.2 Signal at Channel 1 (voltage channel) of the ADC

Figure 6.3 in the next page shows the sampled data received from the ADC by the meter for processing. Figure 6.4 shows only the positive half cycle of the discrete data received.

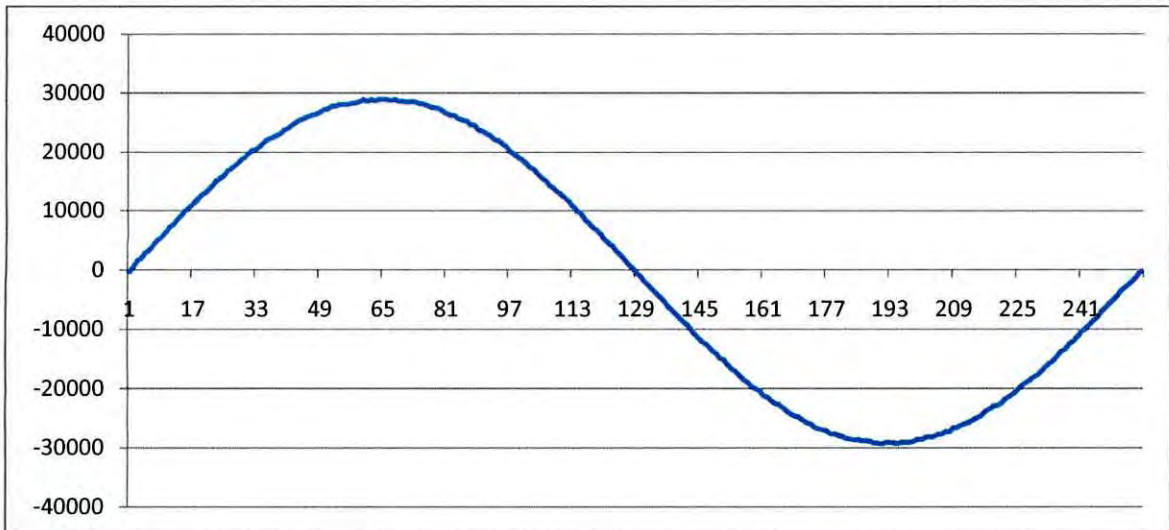


Figure 6.3 Sampled voltage data received from ADC

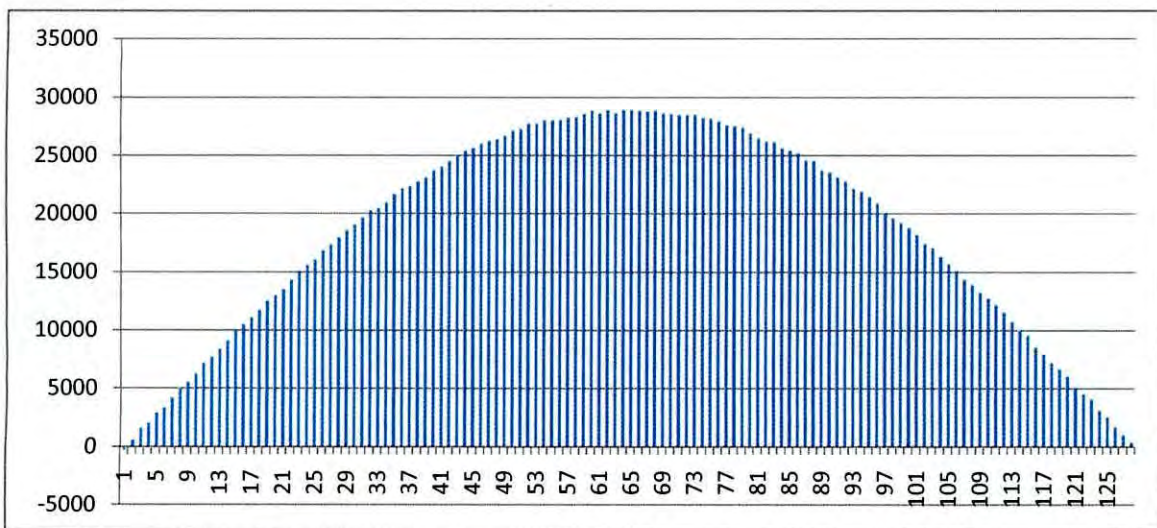


Figure 6.4 Half cycle discrete voltage data as received from ADC

6.3 Measured Data

The meter was connected to phantom load and its response tested against a 0.01 class meter. Voltage variation and power factor tests were performed at the response tested against the IEC specification.

6.3.1 Voltage Variation

Figure 6.5 shows the meter accuracy across the voltage variation tests. Current was varied from 0.1A to 50A while keeping the voltage constant. Data was gathered and graph plotted for both 210V and 230V. At 210V, for very low current the error hovers around the -0.15% mark and gradually reduces as current is increased until finally settling at 0.04% for normal load current. At 230V, the error variation is less and error settles at 0.05% for typical current load.

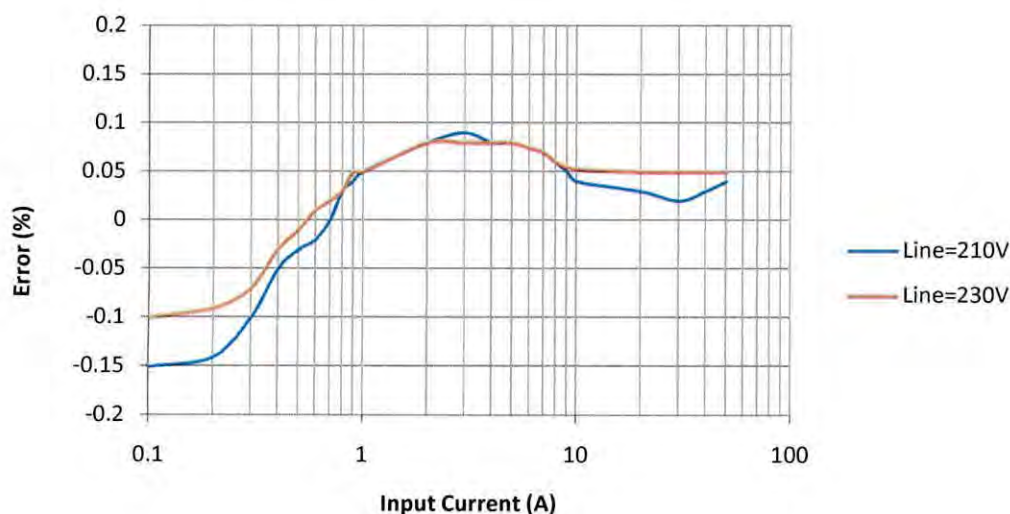


Figure 6.5 Meter Accuracy, voltage variation testing

The graph shows that the error percentage is well within the 0.2% limit set by the IEC standard.

6.3.2 Power Factor response

Figure 6.6 shows the meter accuracy across at PF=0.5. The graph also includes a PF=1 data series for comparison. Again current was varied from 0.1A to 50A and graph plotted. The maximum error percentile at 0.1A was 0.3% and varied from -0.2% to +0.2% for normal operating current. At unity power factor, the error was minimum and almost never crossed 0.1%. Error was around 0% for normal operating current.

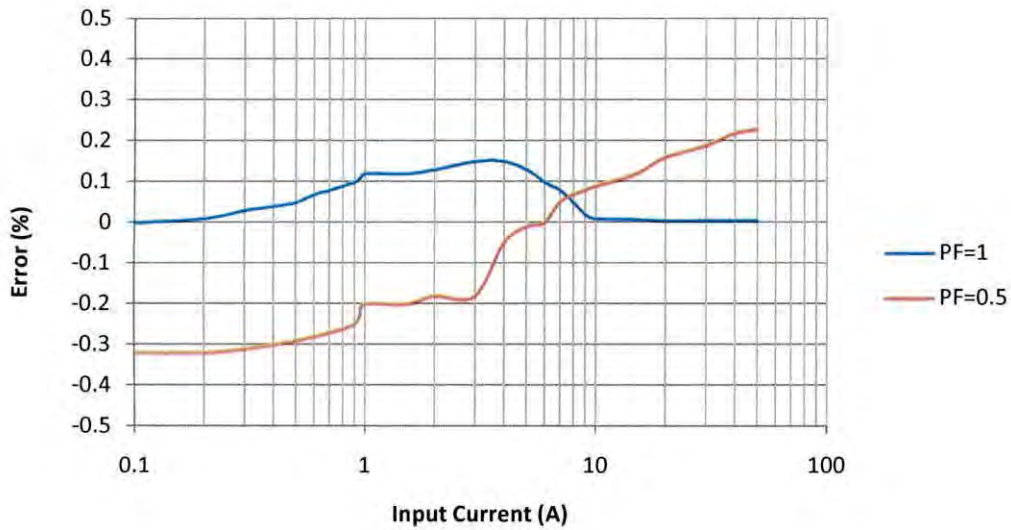


Figure 6.6 Meter Accuracy, power factor testing

Tabulating the above result and comparing with the IEC standard in table 6.2 we can clearly see that the proposed implementation is well within the 0.2 class range specified in the IEC standard.

IEC ACCURACY LIMITS FOR Variation Testing 0.2S ENERGY METERS			
Current	Power Factor	Class 0.2	Proposed Meter
$0.01 I_N < I < 0.05 I_N$	1	± 0.4	0
$0.1 I_N < I < I_{MAX}$	1	± 0.2	0.12
$0.02 I_N < I < 0.1 I_N$	0.5L	± 0.5	0.33
	0.8C	± 0.5	0.27
$0.1 I_N < I < I_{MAX}$	0.5L	± 0.4	0.22
	0.8C	± 0.4	0.21

Table 6.2 0.2 class compliance of the meter implementation

6.3.3 Harmonic Distortion

Harmonic distortion of the input current was measured. Third harmonic was assumed to be the dominant harmonic and HD calculated against that harmonic component. The amplitude of harmonic component of first and third harmonic was measured as 184.8 and 9.6 respectively. Using eqn. 6.1

$$HD = \frac{I_3}{I_1} \times 100 \quad 6.1$$

We get,

$$HD = \frac{9.6}{184.8} \times 100 = 5.1948$$

So THD for third harmonic was calculated and found to be 5.2.



Figure 6.7 THD output from the meter

6.4 Comparison

Only class 1 induction meters were available in the lab for comparison. Figure 6.5 and 6.6 already shows the accuracy of the meter and so those tests were not repeated. The accuracy comparison was performed using harmonic load. Graph 6.8 shows that the accuracy of the proposed implementation is well over a traditional induction meter.

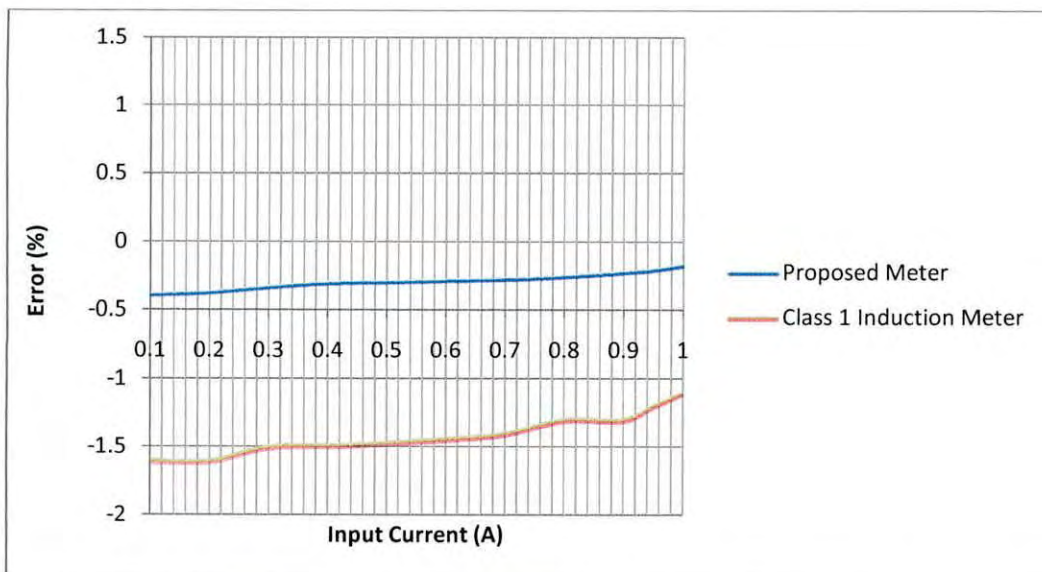


Figure 6.8 Performance comparison of proposed meter and an induction meter

Chapter 7

Conclusion and Future Work

7.1 Conclusion

We have designed and demonstrated single chip implementation of a three phase digital energy meter that is also able to measure the quality of the power. The proposed energy meter measures all the harmonic contents of the power including the fundamental component. As a result the accuracy of the meter is very high even in the presence of harmonics in the power grid. Single chip FPGA implementation makes it cost effective and at the same time achieves additional advantages like less power consumption, less space and components requirements and makes the entire system programmable, reconfigurable and upgradeable based on changing requirements at any point in future. The energy meter is further optimized with fast on-chip memory and parallel path processing so that it is able to perform all calculations including voltage measurement, current measurement, power measurement, phase difference measurement in real-time. An on-chip Fast Fourier Transform (FFT) processor is also implemented to calculate and display the third harmonic distortion. A 16 bit 6 channel simultaneous sampling A/D converter AD73360 is used as the only external component in the meter. The energy meter operates at 3.3 V and draws its power from the power line. Despite the constraint of the availability of high precision external components like resistors and CTs, the meter was able to achieve 0.2 class.

7.2 Future Work

Higher accuracy can be achieved by using high precision component as well as increasing the sampling rate to 64KHz. However, increasing the sampling rate will also increase the number of calculations required several fold but since the whole implementation uses only 25% logic element of the Cyclone chip, multiple NIOS processor can be implemented in the single FPGA chip and they can be dedicated to handle the FFT exclusively or handle each voltage and current channel separately. This will make the implementation more complex but will greatly increase accuracy.

To further add to the flexibility of the meter and ease of use, a VGA interface can be added to the meter. Adding the interface and with some custom programming for graphics, it should be possible to add Fourier analysis functionality output to the NIOS processor. The frequency spectrum of the input AC signals then can be directly seen on any VGA monitor if connected. This will further add the power quality features of the meter.

A PLC chip or RF can be implemented within the meter block. Adding a PLC chip will enable the meter to send data on voltage, current, power factor, load, consumption, power quality and even kWh consumed information to utility provider. This can work as a very important power analysis tool for utility provider as they will not only get consumption usage across his entire distribution network but can also monitor power quality and trace any disruption.

References

- [1] Schwendtner, M.F., "Technological developments in electricity metering and associated fields", *Eighth International Conference on Metering and Tariffs for Energy Supply*, (Conf. Publ. No. 426), pp:240-242, 3-5 Jul. 1996
- [2] Caiceres, R.; Correa, R.; Ferreyra, P.; Cordero, E., "Study of Active Electric Energy Meters Behavior of Induction and Electronic Types", *Transmission & Distribution Conference and Exposition: Latin America*, TDC '06. IEEE/PES pp:1-6, Aug. 2006
- [3] Cataliotti, A.; Cosentino, V.; Nuccio, S., "The Measurement of Reactive Energy in Polluted Distribution Power Systems: An Analysis of the Performance of Commercial Static Meters", *IEEE Transactions on Power Delivery*, Volume 23, Issue 3, pp:1296-1301, Jul. 2008
- [4] P. V. Barbaro , A. Cataliotti , V. Cosentino and S. Nuccio "Behavior of reactive energy meters in polluted power systems," *XVIII Imeko World Congress, Metrology for a Sustainable Development* Rio de Janeiro, Brazil, 17-22 Sep. 2006.
- [5] Alexander H.etc. "Harmonics: Cause, Problem, Solution"-*Part Electrical Construction & Maintenance*. Vol.92 No.5. May 1993.
- [6] Feng Guihong; Zhang Jing; Zhang Bingyi; Zhao Yisong; Ying Yong, "Harmonic Power Detection and Measurement Device Based on Harmonic Power Flow Analysis", *Proceedings of the Eighth International Conference on Electrical Machines and Systems*, 2005. ICEMS 2005. Volume 3, pp: 2262 – 2265, 27-29 Sept. 2005
- [7] Ferrero, A.; Faifer, M.; Salicone, S., "A testing procedure for the new, electronic revenue energy meters", *Instrumentation and Measurement Technology Conference Proceedings, 2008*. IMTC 2008. IEEE, pp:83 – 88, 12-15 May 2008
- [8] R. Arseneau and M. B. Hughes "Selecting revenue meters for harmonic producing loads," *11th Int. Conf. Harmonics and Quality of Power Lake Placid*, NY, 12–15 Sep. 2004.
- [9] A. Din and D. Raisz "What do and what should digital revenue meters measure on distorted networks?," *11th Int. Conf. Harmonics and Quality of Power Lake Placid*, NY, 12–15 Sep. 2004.
- [10] P. S. Filipski and P. W. Labaj "Evaluation of reactive power meters in the presence of high harmonic distortion," *IEEE Trans. Power Del.*, vol. 7, pp. 1793, Oct. 1992.
- [11] M. D. Cox and T. B. Williams "Induction varhour and solid-state varhour meters performances on nonlinear loads," *IEEE Trans. Power Del.*, vol. 5, pp. 1678, Nov. 1990.

- [12] Electricity Metering Equipment (a. c.)—Particular Requirements—Part 23: Static Meters for Reactive Energy (Class 2 and 3), Dec. 2003.
- [13] Electricity Metering Equipment (a.c.)—General Requirements, Tests and Test Conditions—Part 11: Metering Equipment, Nov. 2003.
- [14] Electricity Metering Equipment (a.c.)—Particular Requirements—Part 21: Static Meters for Active Energy (Class 1 and 2), Mar. 2003.
- [15] Electricity Metering Equipment (a.c.)—Particular Requirements—Part 21: Static Meters for Active Energy (Class 0,2S and 0,5S), Mar. 2003.
- [16] Electricity Metering Equipment (AC)—Particular Requirements—Part 24: Static Meters for Reactive Energy (Classes 0,5 and 1), Nov. 2003.
- [17] Boche, H.; Protzmann, M., "Oversampling and boundation of signals", *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, Volume 48, Issue 3, pp:364 - 365, Mar. 2001
- [18] Milosavljevic, D.M.; Milenkovic, V.V.; Radenkovic, V.D., "Precise power and energy measurement", *4th International Conference on Telecommunications in Modern Satellite, Cable and Broadcasting Services, 1999*. Volume 2, pp:637 - 640 vol.2, 13-15 Oct. 1999
- [19] Al-Qatari, S.A.; Al-Ali, A.R., "Microcontroller-based automated billing system", *International IEEE/IAS Conference on Industrial Automation and Control: Emerging Technologies, 1995*, pp:517 – 523, 22-27 May 1995
- [20] Loss, P.A.V.; Lamego, M.M.; Sousa, G.C.D.; Vieira, J.L.F., "A single phase microcontroller based energy meter", *Instrumentation and Measurement Technology Conference, 1998. IMTC/98. Conference Proceedings. IEEE, Volume 2*, pp:797-800, 18-21, May 1998
- [21] Calegari, F., "Electric power/energy measurements for residential single-phase networks", *Industrial International Symposium on Electronics, 2005. ISIE 2005. Proceedings of the IEEE*, Volume 3, Issue, pp: 1087 – 1092, 20-23 Jun. 2005
- [22] Paranhos, I.; Libano, F.; Melchioris, J.; Mano, O.; Roenick, A., "Power energy meter in a low cost hardware/software", *2007 European Conference on Power Electronics and Applications*, pp:1-9, 2-5 Sept. 2007
- [23] Driesen, J.; Van Craenenbroeck, T.; Van Dommelen, D , "The registration of harmonic power by analog and digital power meters", *IEEE Transactions on Instrumentation and Measurement*, Volume 47, Issue 1, pp:195-198, Feb. 1998
- [24] Cataliotti, A.; Cosentino, V.; Nuccio, S., "The metrological characterization of the static meters for reactive energy in the presence of harmonic distortion", *2007 IEEE Instrumentation and Measurement Technology Conference Proceedings*, pp:1 – 6, 1-3 May 2007

- [25] Gallo, D.; Landi, C.; Langella, R.; Testa, A., "On the Accuracy of Electric Energy Revenue Meter Chain Under Non-Sinusoidal Conditions: A Modeling Based Approach", *2007 IEEE Instrumentation and Measurement Technology Conference Proceedings*, pp:1-6, 1-3 May 2007
- [26] Luo zhikun; Xu zhijian; Zheng yucheng; Lu xinjie, "DFT and DSP-based electric energy measurement algorithm of harmonic source load", *International Conference on Power System Technology*, 2002. Proceedings. PowerCon 2002, Volume 4, pp:2487 - 2490 vol.4, 13-17 Oct. 2002
- [27] Ovidiu, P.; Gabriel, C., "DSP's based energy meter", *26th International Spring Seminar on Electronics Technology: Integrated Management of Electronic Materials Production*, 2003. pp:235 – 238, 8-11 May 2003
- [28] Toral, S.; Quero, J.M.; Franquelo, "Power energy metering based on random signal processing", *Proceedings of the 1998 IEEE International Symposium on Circuits and Systems*, 1998. ISCAS '98. Volume 3, pp: 435 - 438 vol.3, 31 May-3 Jun. 1998
- [29] Luo zhikun; Xu zhijian; Zheng yucheng; Lu xinjie, "DFT and DSP-based electric energy measurement algorithm of harmonic source load". *Proceedings. PowerCon 2002. International Conference on Power System Technology*, 2002 Volume 4, Issue , pp: 2487 - 2490 vol.4, 2002
- [30] Dutta, P.; Feldmeier, M.; Paradiso, J.; Culler, D., "Energy Metering for Free: Augmenting Switching Regulators for Real-Time Monitoring", *International Conference on Information Processing in Sensor Networks*, 2008. IPSN '08, pp:283-294, 22-24 Apr. 2008
- [31] Chang Guo-xiang; Wang Cheng-yuan; Guo Dian-ling; Xia Jia-kuan; Liu Xiu-ling, "Design of an FPGA-Based 3-Phase Sinusoidal PWM Controller" *Control Conference, 2006. CCC 2006. Chinese Volume, Issue*, pp:1886 – 1889, 7-11 Aug. 2006
- [32] Zhaoyong Zhou; Guijie Yang; Tiejai Li, "Design and implementation of an FPGA-based 3-phase sinusoidal PWM VVVF controller", *Nineteenth Annual IEEE Applied Power Electronics Conference and Exposition*, 2004. APEC '04. Volume 3, pp: 1703 - 1708 Vol.3, 2004
- [33] Seonil Choi; Govindu, G.; Ju-Wook Jang; Prasanna, V.K., "Energy-efficient and parameterized designs for fast Fourier transform on FPGAs", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2003. Proceedings. (ICASSP apos;03). 2003 Volume 2, Issue, pp: II - 521-4 vol.2, 6-10 Apr. 2003
- [34] Basic Block Diagram of an Electronic Energy Meter. http://en.wikipedia.org/wiki/Image:Block_Diagram.JPG, Dec. 2005
- [35] Analog Devices AD73360 Six channel sigma-delta analog to digital converter. www.analog.com, Apr. 2000

[36] http://en.wikipedia.org/wiki/Field-programmable_gate_array, 19 Oct. 2008

[37] Electricity metering equipment (a.c.) - Particular Requirements - Part 22: Static meters for active energy (classes 0,2 S and 0,5 S), International Electrotechnical Commission, Jan. 2003

Appendix A

PIN configuration of the six channel Analog to Digital Converter AD73360

PIN	Use
VINP1	Analog Input to the Positive Terminal of Input Channel 1.
VINN1	Analog Input to the Negative Terminal of Input Channel 1.
VINP2	Analog Input to the Positive Terminal of Input Channel 2.
VINN2	Analog Input to the Negative Terminal of Input Channel 2.
VINP3	Analog Input to the Positive Terminal of Input Channel 3.
VINN3	Analog Input to the Negative Terminal of Input Channel 3.
VINP4	Analog Input to the Positive Terminal of Input Channel 4.
VINN4	Analog Input to the Negative Terminal of Input Channel 4.
VINP5	Analog Input to the Positive Terminal of Input Channel 5.
VINN5	Analog Input to the Negative Terminal of Input Channel 5.
VINP6	Analog Input to the Positive Terminal of Input Channel 6.
VINN6	Analog Input to the Negative Terminal of Input Channel 6.
REFOUT	Buffered Reference Output, which has a nominal value of 1.25 V or 2.5 V, the value being dependent on the status of Bit 5VEN (CRC:7).
REFCAP	A Bypass Capacitor to AGND2 of 0.1 μ F is required for the on-chip reference. The capacitor should be fixed to this pin. This pin can be overdriven by an external reference if required.
AVDD2	Analog Power Supply Connection.
AGND2	Analog Ground/Substrate Connection.
DGND	Digital Ground/Substrate Connection.
DVDD	Digital Power Supply Connection.
RESET	Active Low Reset Signal. This input resets the entire chip, resetting the control registers and clearing the digital circuitry.
SCLK	Output Serial Clock whose rate determines the serial transfer rate to/from the AD73360. It is used to clock data or control information to and from the serial

Continued on next page

PIN	Use
	port (SPORT). The frequency of SCLK is equal to the frequency of the master clock (MCLK) divided by an integer number—this integer number being the product of the external master clock rate divider and the serial clock rate divider.
MCLK	Master Clock Input. MCLK is driven from an external clock signal. SDO Serial Data Output of the AD73360. Both data and control information may be output on this pin and are clocked on the positive edge of SCLK. SDO is in three-state when no information is being transmitted and when SE is low.
SDOFS	Framing Signal Output for SDO Serial Transfers. The frame sync is one bit wide and it is active one SCLK period before the first bit (MSB) of each output word. SDOFS is referenced to the positive edge of SCLK. SDOFS is in three-state when SE is low.
SDIFS	Framing Signal Input for SDI Serial Transfers. The frame sync is one bit wide and it is valid one SCLK period before the first bit (MSB) of each input word. SDIFS is sampled on the negative edge of SCLK and is ignored when SE is low.
SDI	Serial Data Input of the AD73360. Both data and control information may be input on this pin and are clocked on the negative edge of SCLK. SDI is ignored when SE is low.
SE	SPORT Enable. Asynchronous input enable pin for the SPORT. When SE is set low by the DSP, the output pins of the SPORT are three-stated and the input pins are ignored. SCLK is also disabled internally in order to decrease power dissipation. When SE is brought high, the control and data registers of the SPORT are at their original values (before SE was brought low); however, the timing counters and other internal registers are at their reset values.
AGND1	Analog Ground Connection.
AVDD1	Analog Power Supply Connection.

Appendix B

PIN configuration and use of the FSM Block

PIN	Type	Width	Use
MCLK	Output	1 bit	This is the master clock that drives the ADC. It is usually driven at around 2MHz
RST	Output	1 bit	This is a active low pin that resets the ADC
SCLK	Input	1 bit	1 bit input from the ADC to the FSM. All operations and communications between the ADC and FSM are clocked at the different edges of the SCLK. Usually SCLK is set at a value of MCLK/1024 which is changeable by writing to the registers of the ADC.
SDOFS	Input	1 bit	The SDOFS and SDIFS pin of the ADC are tied together and connected to this pin. Whenever a new data is available the ADC produces a pulse in the SDOFS pin. The FSM detect this pulse and reads the data through the SDO pin serially.
SDO	Input	1 bit	SDO is the serial output from the ADC and input the FSM. Data from the ADC to the FSM are transferred serially through this pin. Whenever a new pulse is detected by the FSM in the SDOFS pin, the FSM starts reading values from the SDO pin at every negative edge of SCLK.
From_Nios_data_address	Input	16 bit	This is a 16 bit input to the FSM from the Nios soft processor. This input is only used when the FSM is used in memory mode and store values from the ADC to memory. This is not used in normal mode when the FSM transfers each set of acquired data immediately to the Nios processor and does not save them to memory.
RESET	Input	1 bit	Reset input for the FSM. Resetting the FSM, resets all counters and states of the FSM. The FSM started

Continued on next page

PIN	Type	Width	Use
			again from state zero and reprograms the ADC again
SE	Output	1 bit	1 bit output from the FSM that enables the serial output pin of the ADC. This is always high expect when the FSM is in state zero and resets the ADC.
SDI	Input	1 bit	Single bit input to the ADC. This line is used to serially transfer data from the FSM to the ADC. All inputs are clocked at the negative pulse of SCLK and is always 16 bit long.
To_Nios_drRdy	Output	1 bit	This pin is generally low. A pulse is created by the FSM on this line whenever all 6 channels data have been read. This signals the Nios process that data is ready to be read from the 6 data channel connected to Nios from the FSM
GLEDS	Output	2 bit	Connected to Green LEDs on the board, used for debugging purposed
Val	Output	16 bit	16 bit output used internally for debugging purpose
Currentstat	Output	4 bit	4 bit register that saves and controls the current state in which the current state machine is in
Statew	Output	4bit	4 bit register that shows the internal state of the state machine. This register signals whether the FSM is in read state, write state or normal state
To_Nios_V1	Output	32 bit	32 bit register directly connected to the Nios processor from the FSM. This register outputs value of the voltage in channel 1 after each set of data are read from the ADC. The NISO processor reads this pin directly as a parallel register.
To_Nios_V2	Output	32 bit	32 bit register directly connected to the Nios processor from the FSM. This register outputs value of the voltage in channel 2 after each set of data are read from the ADC. The NISO processor reads this pin directly as a parallel register.
To_Nios_V3	Output	32 bit	32 bit register directly connected to the Nios processor from the FSM.

Continued on next page

PIN	Type	Width	Use
			This register outputs value of the voltage in channel 3 after each set of data are read from the ADC. The NISO processor reads this pin directly as a parallel register.
To_Nios_I1	Output	32 bit	32 bit register directly connected to the Nios processor from the FSM. This register outputs value of the current in channel 1 after each set of data are read from the ADC. The NISO processor reads this pin directly as a parallel register.
To_Nios_I2	Output	32 bit	32 bit register directly connected to the Nios processor from the FSM. This register outputs value of the current in channel 2 after each set of data are read from the ADC. The NISO processor reads this pin directly as a parallel register.
To_Nios_I3	Output	32 bit	32 bit register directly connected to the Nios processor from the FSM. This register outputs value of the current in channel 3 after each set of data are read from the ADC. The NISO processor reads this pin directly as a parallel register.
To_Nios_VI	Output	32 bit	32 bit register used only when the FSM is responsible for calculating VI instead of the Nios processor. This 32 bit register calculates instantaneous energy of all these phases ($V1 \times I1 + V2 \times I2 + V3 \times I3$)
To_Nios_SampleNo	Input	16 bit	

Appendix C

FSM Verilog Code

This is the verilog code that is used to synthesise the FSM block using Quartus II software and then implemented on the DE2 board. First the inputs and outputs are defined and then the state codes are implemented at each positive edge of the master clock. The read and write operations are performed by the read_fsm and write_fsm state machines which are also synthesized in this module.

```
module adcfsm(MCLK, RST, RESET, SE, SDI, SCLK, SDOFS, SDO, GLEDS, Val,
currentstate, staterw, to_Nios_V1, to_Nios_V2, to_Nios_V3, to_Nios_I1, to_Nios_I2,
to_Nios_I3, to_Nios_VI, to_Nios_SampleNo, from_Nios_data_address,
to_Nios_DtRdy);
```

```
//Define Inputs and Outputs
```

```
input MCLK, RST;
```

```
output reg RESET;
```

```
output reg SE;
```

```
output reg SDI;
```

```
input SCLK;
```

```
input SDOFS;
```

```
input SDO;
```

```
output [1:0] GLEDS;
```

```
output reg [15:0] Val;
```

```
output [3:0] currentstate;
```

```
output [3:0] staterw;
```

```
output reg [31:0] to_Nios_V1, to_Nios_V2, to_Nios_V3, to_Nios_I1, to_Nios_I2,
to_Nios_I3, to_Nios_VI;
```

```
output reg [15:0] to_Nios_SampleNo;
```

```
input [15:0] from_Nios_data_address;
```

```
output reg to_Nios_DtRdy;
```

```
reg old_sclk;
```

```
wire posedge_sclk= ~old_sclk & SCLK;
```

```
wire negedge_sclk= old_sclk & ~SCLK;
```

```
assign currentstate[3:0]=state[3:0];
```

```
reg old_sdofs;
```

```
wire posedge_sdofs= ~old_sdofs & SDOFS;
```

```
wire negedge_sdofs= old_sdofs & ~SDOFS;
```

```
reg SDoline;
```

```
assign GLEDS[1:1]=SCLK;
```

```
assign GLEDS[0:0]=SDOFS;
```

```
//Defines states and variables for the main state machine
```

```
parameter state0=0, state1=1, state2=2, state3=3, state4=4, state5=5, state6=6,
state7=7, writereg=8, readdata=9, statewait=10, donothing=11;
```

```

reg[3:0] state, nxt_st, opcomplete;
reg[3:0] waitreg;
reg[4:0] bitsent;
reg[4:0] bitreceived;
reg[15:0] datatosend /* synthesis keep */;
reg[15:0] datachannel1;
reg[1:0] sdfswritepulseoccured;
reg sdfswritepulseoccured;
reg readcomplete;
reg writecomplete;
reg reading;
reg writing;
assign staterw[3:0]={2'b0,writing,reading};
always @(posedge MCLK)
    begin : fsm_logic

        case (state)
            state0: begin
                SE <= 0;
                RESET <= 0;
                reading<=0;
                writing<=0;
                waitreg <= 4'd4;
                nxt_st <= state1;
            end
            state1: begin
                SE <= 1;
                RESET <= 1;
                waitreg <= 4'd0;
                nxt_st <= state2;
                writing<=1;
                datatosend <= 16'b1000001110001000;
            end
            state2: begin
                SE <= 1;
                RESET <= 1;
                waitreg <= 4'd0;
                nxt_st <= state3;
                writing<=1;
                datatosend <= 16'b1000010010001000;
            end
            state3: begin
                SE <= 1;
                RESET <= 1;
                waitreg <= 4'd0;
                nxt_st <= state4;
                writing<=1;
                datatosend <= 16'b1000010110001000;
            end
            state4: begin

```



```

        RESET <= 1;
        waitreg <= 4'd0;
        nxt_st <= state5;
        writing<=1;
        datatosend <= 16'b1000011000111111;
    end
    state5: begin
        RESET <= 1;
        waitreg <= 4'd0;
        nxt_st <= state6;
        writing<=1;
        datatosend <= 16'b1000011100111111;
    end
    state6: begin
        RESET <= 1;
        waitreg <= 4'd0;
        nxt_st <= state7;
        writing<=1;
        datatosend <= 16'b1000000000000001;
    end
    state7: begin
        RESET <= 1;
        waitreg <= 4'd0;
        nxt_st <= state7;
        reading<=1;
        datachannel1 <= 16'b0000000000000000;
    end
    statewait: begin
        if(waitreg > 0) waitreg <= waitreg-1;
        nxt_st <= nxt_st;
    end
    donothing: begin
        waitreg <= 4'd0;
        nxt_st <= nxt_st;
        if(writecomplete)
            writing<=0;
        if(readcomplete)
            reading<=0;
    end
    default:nxt_st<=state0;
endcase
end

//state machine state change control
always @(posedge MCLK)
begin : state_generation
if(~RST) state=state0;
else if (reading | writing) state = donothing;
else if (waitreg > 0) state = statewait;
else state = nxt_st;
end

```

```

end

//SCLK edge detection code
always @(posedge MCLK)
begin : main_adcfsm
    old_sclk <= SCLK;
    old_sdofs <= SDOFS;
    SDoline <= SDO;
end

//Minor FSM writefsm code
reg [15:0] writedatavalue;
always @(posedge MCLK)
begin : writefsm
    if(writing)
        begin
            if(sdofswritepulseoccured==2'd2 & posedge_sclk)
                begin
                    SDI<=writedatavalue[15];
                    writedatavalue<=(writedatavalue<<1);
                    bitsent<=(bitsent+1'd1);
                    if(bitsent>=5'd16)
                        begin
                            writecomplete<=1;
                        end
                    else
                        writecomplete<=0;
                end
            else
                begin
                    if(negedge_sdofs)
                        begin
                            sdofswritepulseoccured<=(sdofswritepulseoccured+1);
                            if(sdofswritepulseoccured==2'd1)
                                begin
                                    SDI<=datatosend[15];
                                    writedatavalue<=(datatosend<<1);
                                end
                            end
                        end
                end
        end
    else
        begin
            writecomplete<=0;
            SDI <= 0;
            bitsent <=0;
            sdofswritepulseoccured <=0;
        end
end
end

```

```

//Minor FSM readfsm code
reg [15:0] readdatavalue;
reg [15:0] readdatavalue1;
reg [15:0] readdatavalue2;
reg [15:0] readdatavalue3;
reg [15:0] readdatavalue4;
reg [15:0] readdatavalue5;
reg [15:0] readdatavalue6;
reg [2:0] reading_6;
always @(posedge MCLK)
begin : readfsm
    if(reading)
        begin
            if(sdofsreadpulseoccured & posedge_sclk)
                begin
                    readdatavalue<={readdatavalue[14:0],SDO};
                    bitreceived<=(bitreceived+1'd1);
                    if(bitreceived>=5'd15)
                        begin
                            reading_6<=reading_6+1;
                            bitreceived <=0;
                            readdatavalue<=16'b0;
                            sdofsreadpulseoccured <=0;
                            if(reading_6==3'd0)
                                begin
                                    readdatavalue1<={readdatavalue[14:0],SDO};
                                end
                            else if(reading_6==3'd1)
                                begin
                                    readdatavalue2<={readdatavalue[14:0],SDO};
                                end
                            else if(reading_6==3'd2)
                                begin
                                    readdatavalue3<={readdatavalue[14:0],SDO};
                                end
                            else if(reading_6==3'd3)
                                begin
                                    readdatavalue4<={readdatavalue[14:0],SDO};
                                end
                            else if(reading_6==3'd4)
                                begin
                                    readdatavalue5<={readdatavalue[14:0],SDO};
                                end
                            else if(reading_6==3'd5)
                                begin
                                    readdatavalue6<={readdatavalue[14:0],SDO};
                                    to_Nios_V1<=readdatavalue1;
                                    to_Nios_V2<=readdatavalue2;
                                end
                        end
                end
        end
end

```

```

        to_Nios_V3<=readdatavalue3;
        to_Nios_I1<=readdatavalue4;
        to_Nios_I2<=readdatavalue5;
        to_Nios_I3<={readdatavalue[14:0],SDO};
    end
else
    begin
        readdatavalue1<=readdatavalue1;
        readdatavalue2<=readdatavalue2;
        readdatavalue3<=readdatavalue3;
        readdatavalue4<=readdatavalue4;
        readdatavalue5<=readdatavalue5;
        readdatavalue6<=readdatavalue6;
        to_Nios_V1<=to_Nios_V1;
        to_Nios_V2<=to_Nios_V2;
        to_Nios_V3<=to_Nios_V3;
        to_Nios_I1<=to_Nios_I1;
        to_Nios_I2<=to_Nios_I2;
        to_Nios_I3<=to_Nios_I3;
    end
end
else
    begin
        readcomplete<=0;
        Val<=Val;
        reading_6<=reading_6;
        sdofsreadpulseoccured <=sdofsreadpulseoccured;
    end
end
else
    begin
        Val<=Val;
        reading_6<=reading_6;
        readdatavalue<=readdatavalue;

        if(negedge_sdofs)
            begin
                sdofsreadpulseoccured<=1;
            end
        else
            begin
                sdofsreadpulseoccured<=sdofsreadpulseoccured;
            end
        end
    end
end
else
    begin
        readcomplete<=0;

        bitreceived <=0;
    end
end

```

```
readdatavalue<=16'b0;  
sdofsreadpulseoccured <=0;  
Val<=Val;  
reading_6<=0;  
end  
endmodule
```

Appendix D

Pseudo code for energy calculation

This is the pseudo ANSI C code that is used for calculating energy from the sampled data in RAM memory. Six RAM array pointers are passed to the function containing the discrete sampled signal. The function calculates RMS values from the sampled data.

```

void calc_rms(short* V1, short* V2, short* V3, short* I1, short* I2, short* I3,
double* rmsdata, double* powerData)
{
    unsigned long V1t=0, V2t=0, V3t=0, I1t=0, I2t=0, I3t=0; //total summation
    unsigned short V1zc=0, V2zc=0, V3zc=0, I1zc=0, I2zc=0, I3zc=0; //zero crossing
    unsigned short V1lc=0, V2lc=0, V3lc=0, I1lc=0, I2lc=0, I3lc=0; //loop count
    double V1rms=0, V2rms=0, V3rms=0, I1rms=0, I2rms=0, I3rms=0; //rms value
    unsigned short i=0;
    for(i=0; i<=1000; i++)
    {
        //V1
        if(V1[i]>=0 && V1[i-1]<0) //positive zero crossing
        {
            V1zc++;
        }
        if(V1zc==1 && V1[i-1]>0 && V1[i]<=0) //negative zero crossing
        {
            V1zc++;
        }
        if(V1zc==1)
        {
            V1t+=V1[i]*V1[i]; //add voltage until negative crossing arrives
            V1lc++; //count how many samples
        }
        //V2
        if(V2[i]>=0 && V2[i-1]<0) //positive zero crossing
        {
            V2zc++;
        }
        if(V2zc==1 && V2[i-1]>0 && V2[i]<=0) //negative zero crossing
        {
            V2zc++;
        }
        if(V2zc==1)
        {
            V2t+=V2[i]*V2[i]; //add voltage until negative crossing arrives
            V2lc++; //count how many samples
        }
        //V3
        if(V3[i]>=0 && V3[i-1]<0) //positive zero crossing

```

```

    {
    V3zc++;
    }
if(V3zc==1 && V3[i-1]>0 && V3[i]<=0) //negative zero crossing
    {
    V3zc++;
    }
if(V3zc==1)
    {
    V3t+=V3[i]*V3[i]; //add voltage until negative crossing arrives
    V3lc++; //count how many samples
    }
//I1
if(I1[i]>=0 && I1[i-1]<0) //positive zero crossing
    {
    I1zc++;
    }
if(I1zc==1 && I1[i-1]>0 && I1[i]<=0) //negative zero crossing
    {
    I1zc++;
    }
if(I1zc==1)
    {
    I1t+=I1[i]*I1[i]; //add voltage until negative crossing arrives
    I1lc++; //count how many samples
    }
//I2
if(I2[i]>=0 && I2[i-1]<0) //positive zero crossing
    {
    I2zc++;
    }
if(I2zc==1 && I2[i-1]>0 && I2[i]<=0) //negative zero crossing
    {
    I2zc++;
    }
if(I2zc==1)
    {
    I2t+=I2[i]*I2[i]; //add voltage until negative crossing arrives
    I2lc++; //count how many samples
    }
//I3
if(I3[i]>=0 && I3[i-1]<0) //positive zero crossing
    {
    I3zc++;
    }
if(I3zc==1 && I3[i-1]>0 && I3[i]<=0) //negative zero crossing
    {
    I3zc++;
    }
if(I3zc==1)

```

```
{
  I3t+=I3[i]*I3[i]; //add voltage until negative crossing arrives
  I3lc++; //count how many samples
}

//instantaneous power calc
powerData[0]+=V1[i]*I1[i];
powerData[1]+=V2[i]*I2[i];
powerData[2]+=V3[i]*I3[i];
}
//for loop end
rmsdata[0]=V1rms=sqrt((double)V1t/V1lc);
rmsdata[1]=V2rms=sqrt((double)V2t/V2lc);
rmsdata[3]=V3rms=sqrt((double)V3t/V3lc);
rmsdata[4]=I1rms=sqrt((double)I1t/I1lc);
rmsdata[5]=I2rms=sqrt((double)I2t/I2lc);
rmsdata[6]=I3rms=sqrt((double)I3t/I3lc);
}
```


Appendix E

Code for soft FFT generation with predetermined twiddles

The is the ANSI C code for calculating FFT with predetermined twiddles. Predetermined means that the sine and cosine factors are pre calculated and are saved in a data table or array beforehand. This reduces the operational time required for FFT operation

```

void software_fft (
    alt_16 *InData,    // Input Data Buffer
    alt_16 *OutData,   // Output Data Buffer
    alt_16 *TwiddleTable // Interleaved Sine/Cosine Table
)
{
    alt_16 i;
    alt_16 bit_rev_index;
    alt_16 stage_index;
    alt_16 sub_stage_index;
    alt_16 butterfly_index;
    alt_16 twiddle_index;
    alt_16 twiddle_incr;
    alt_16 loop_element,loop_element_div2;
    alt_32 CosReal,SinReal;
    alt_32 TempReal,TempImaginary;
    alt_16 reversed_RealData[NUM_POINTS];
    alt_16 reversed_ImaginaryData[NUM_POINTS];

    // Re-order samples using bit reversal
    for (i = 0; i < NUM_POINTS; i++) {
        bit_rev_index = bitrev(i);
        reversed_RealData[bit_rev_index] = InData[2*i];
        reversed_ImaginaryData[bit_rev_index] = InData[2*i+1];
    }

    // Loop through the stages of the N Point FFT
    for (stage_index = 1; stage_index <= FFT_SIZE; stage_index++) {
        loop_element = 1<<stage_index;
        loop_element_div2 = loop_element/2;

        // Initialize twiddle factor lookup indicies
        twiddle_index = 0;
        twiddle_incr = 1 << (FFT_SIZE-stage_index+1);

        // Loop through each sub stage
        for(sub_stage_index = 0; sub_stage_index < loop_element_div2;
sub_stage_index++) {

            // Calculate sine and cosine values from interleaved twiddle table

```

```

CosReal = TwiddleTable[twiddle_index];
SinReal = TwiddleTable[twiddle_index+1];

// Butterfly calculation
for(butterfly_index = sub_stage_index; butterfly_index < NUM_POINTS;
butterfly_index += loop_element) {
    TempReal = (( CosReal * reversed_RealData[butterfly_index +
loop_element_div2] ) +
    ( SinReal * reversed_ImaginaryData[butterfly_index +
loop_element_div2] )) >> PRESCALE;
    TempImaginary = (( CosReal * reversed_ImaginaryData[butterfly_index +
loop_element_div2] ) -
    ( SinReal * reversed_RealData[butterfly_index + loop_element_div2] ))
>> PRESCALE;

// Perform the butterfly calculation and scale result
for alt_16 type
    reversed_RealData[butterfly_index + loop_element_div2] =
reversed_RealData[butterfly_index] - TempReal;
    reversed_ImaginaryData[butterfly_index + loop_element_div2] =
reversed_ImaginaryData[butterfly_index] - TempImaginary;
    reversed_RealData[butterfly_index] += TempReal;
    reversed_ImaginaryData[butterfly_index] += TempImaginary;

// Write first half of interleaved data to output buffer
if (stage_index == FFT_SIZE) {
    OutData[2*butterfly_index] = reversed_RealData[butterfly_index];
    OutData[2*butterfly_index+1] = reversed_ImaginaryData[butterfly_index];
}
}
twiddle_index += twiddle_incr;
}
}
// Write second half of interleaved data to output buffer
for(i=NUM_POINTS/2;i<NUM_POINTS;i++) {
    OutData[2*i] = reversed_RealData[i];
    OutData[2*i+1] = reversed_ImaginaryData[i];
}
}
}

```

Appendix F

Code for hardware based FFT generation with acceleration

//Main function takes 3 inputs, pointer to input data set, pointer to output data set in RAM and the pointer to pre calculated twiddle factors.

```

void software_fft (
    alt_16 *Input_Data,    // Input Data Buffer
    alt_16 *Output_Data,  // Output Data Buffer
    alt_16 *Pre_Calc_Table // Interleaved Sine/Cosine Table
)
{
    alt_16 i;
    alt_16 bit_rev_index;
    alt_16 stage_index;
    alt_16 sub_stage_index;
    alt_16 butterfly_index;
    alt_16 twiddle_index;
    alt_16 twiddle_incr;
    alt_16 loop_element,loop_element_div2;
    alt_32 CosReal,SinReal;
    alt_32 TempReal,TempImaginary;
    alt_16 reversed_RealData[NUM_POINTS];
    alt_16 reversed_ImaginaryData[NUM_POINTS];

    // Re-order samples using bit reversal
    for (i = 0; i < NUM_POINTS; i++) {
        bit_rev_index = bitrev(i);
        reversed_RealData[bit_rev_index] = Input_Data[2*i];
        reversed_ImaginaryData[bit_rev_index] = Input_Data[2*i+1];
    }

    // Loop through the stages of the N Point FFT
    for (stage_index = 1; stage_index <= FFT_SIZE; stage_index++) {
        loop_element = 1<<stage_index;
        loop_element_div2 = loop_element/2;

        // Initialize twiddle factor lookup indicies
        twiddle_index = 0;
        twiddle_incr = 1 << (FFT_SIZE-stage_index+1);

        // Loop through each sub stage
        for(sub_stage_index = 0; sub_stage_index < loop_element_div2;
sub_stage_index++) {

            // Calculate sine and cosine values from interleaved twiddle table
            CosReal = Pre_Calc_Table[twiddle_index];
            SinReal = Pre_Calc_Table[twiddle_index+1];

```

```

// Butterfly calculation
for(butterfly_index = sub_stage_index; butterfly_index < NUM_POINTS;
butterfly_index += loop_element) {
    TempReal = (( CosReal * reversed_RealData[butterfly_index +
loop_element_div2] ) +
                ( SinReal * reversed_ImaginaryData[butterfly_index +
loop_element_div2] )) >> PRESCALE;
    TempImaginary = (( CosReal * reversed_ImaginaryData[butterfly_index +
loop_element_div2] ) -
                    ( SinReal * reversed_RealData[butterfly_index + loop_element_div2] ))
>> PRESCALE;

// Perform the butterfly calculation and scale result
for alt_16 type
    reversed_RealData[butterfly_index + loop_element_div2] =
reversed_RealData[butterfly_index] - TempReal;
    reversed_ImaginaryData[butterfly_index + loop_element_div2] =
reversed_ImaginaryData[butterfly_index] - TempImaginary;
    reversed_RealData[butterfly_index] += TempReal;
    reversed_ImaginaryData[butterfly_index] += TempImaginary;

// Write first half of interleaved data to output buffer
if (stage_index == FFT_SIZE) {
    Output_Data[2*butterfly_index] = reversed_RealData[butterfly_index];
    Output_Data[2*butterfly_index+1] = reversed_ImaginaryData[butterfly_index];
}
}
twiddle_index += twiddle_incr;
}
}
// Write second half of interleaved data to output buffer
for(i=NUM_POINTS/2;i<NUM_POINTS;i++) {
    Output_Data[2*i] = reversed_RealData[i];
    Output_Data[2*i+1] = reversed_ImaginaryData[i];
}
}
}

```

Appendix G

Code for hardware based FFT generation with acceleration

//Main function takes 2 inputs here (pre-calculated table is no longer needed), pointer to input data set and pointer to output data set in RAM. Based on Altera hardware acceleration code.

```

void hardware_optimized_fft(
    alt_16 * __restrict__ Input_Data, // real part of input data
    alt_16 * __restrict__ Output_Data // real part of output data
)
{
    //Declare all variables at unsigned 16 bit numbers
    alt_u16 twiddle_index;
    alt_u16 twiddle_incr;
    alt_u16 loop_element;
    alt_u16 loop_element_div2;
    alt_u16 l;
    alt_u16 sub_stage_index;
    alt_u16 stage_index;
    alt_u16 butterfly_index;

    //Again declared as 16 bit unsigned number but as restricted pointer
    // Cosine and Sine Tables
    alt_16 * __restrict__ Cos_Table;
    alt_16 * __restrict__ Sin_Table;
    //Real data buffers
    alt_16 * __restrict__ Real_Data_Read;
    alt_16 * __restrict__ Real_Data_Read_by_Port2; //RAMs are declared as dual port
    alt_16 * __restrict__ Real_Data_Write;
    alt_16 * __restrict__ Real_Data_Write_by_Port2; //RAMs are declared as dual port
    //Imaginary data buffers
    alt_16 * __restrict__ Img_Data_Read;
    alt_16 * __restrict__ Img_Data_Read_by_Port2; //RAMs are declared as dual port
    alt_16 * __restrict__ Img_Data_Write;
    alt_16 * __restrict__ Img_Data_Write_by_Port2; //RAMs are declared as dual port
    //Pointers to in and out buffers in main memory
    alt_u32 * __restrict__ Input_Ptr_tmp = (alt_u32 *)Input_Data;
    alt_u32 * __restrict__ Output_Ptr_tmp = (alt_u32 *)Output_Data;

    // Temporary registers for the input stage
    alt_u16 bit_rev_index;
    alt_u32 Input_tmp;

    // Registers for the calculation stage
    alt_16 CosReal;
    alt_16 SinReal;

```

```

alt_16 tRealData;
alt_16 tImagData;
alt_16 temp1, temp2, temp3, temp4;

//Counters for the input and output stages
alt_u16 Input_Counter, Output_Counter;

//Point the ping pong buffers to the appropriate buffer location as defined in SOPC
Real_Data_Read = BufferRAM1;
Real_Data_Read_by_Port2 = BufferRAM1;
Real_Data_Write = BufferRAM2;
Real_Data_Write_by_Port2 = BufferRAM2;

Img_Data_Read = BufferRAM3;
Img_Data_Read_by_Port2 = BufferRAM3;
Img_Data_Write = BufferRAM4;
Img_Data_Write_by_Port2 = BufferRAM4;

// Point the Cosine and Sine Table registers to the appropriate on-chip memory
// as defined in the SOPC builder.
Cos_Table = CosRAM;
Sin_Table = SinRAM;

// Stage 1: Data buffering
// Read data from input buffer and split into read and imaginary part. Buffer write
// operation occurs simultaneously because of acceleration. NUM_POINTS is the
//total number of points as defined globally before.
for (Input_Counter = 0; Input_Counter < NUM_POINTS; Input_Counter++)
{
    bit_rev_index = bitrev(Input_Counter);
    Input_tmp = Input_Ptr_tmp[Input_Counter];
    Real_Data_Read[bit_rev_index] = (alt_16)(Input_tmp & 0x0000FFFF);
    Img_Data_Read[bit_rev_index] = (alt_16)((Input_tmp & 0xFFFF0000)>>16);
}

// Stage 2: FFT Calculations. Go through the various butterfly stages
for (stage_index = 1; stage_index <= FFT_SIZE; stage_index++)
{
    loop_element = 1<<stage_index;
    loop_element_div2 = loop_element/2;
    twiddle_index = 0;
    twiddle_incr = 1 << (FFT_SIZE-stage_index);
    for(sub_stage_index = 0; sub_stage_index < loop_element_div2;
sub_stage_index++)
    {
        CosReal = Cos_Table[twiddle_index];
        SinReal = Sin_Table[twiddle_index];

// Process butterflies with the same twiddle factors

```

```

for(butterfly_index = sub_stage_index; butterfly_index < NUM_POINTS;
butterfly_index += loop_element) {
    l = butterfly_index + loop_element_div2;

    //Assignment registers are executed concurrently in accelerator
    temp1 = Real_Data_Read[l];
    temp2 = Img_Data_Read[l];
    temp3 = Real_Data_Read_by_Port2[butterfly_index];
    temp4 = Img_Data_Read_by_Port2[butterfly_index];

    // Multiplication of 16 bits produce 32 bits so downscaling by bit shifting is
    // necessary
    // All different registers, so all reads occur simultaneously.
    tRealData = (( CosReal * temp1 ) + ( SinReal * temp2 ))>> PRESCALE;
    tImagData = (( CosReal * temp2 ) - ( SinReal * temp1 ))>> PRESCALE;

    // All different registers, so all reads and write occur concurrently.
    Real_Data_Write[l] = temp3 - tRealData;
    Img_Data_Write[l] = temp4 - tImagData;
    Real_Data_Write_by_Port2[butterfly_index] = temp3 + tRealData;
    Img_Data_Write_by_Port2[butterfly_index] = temp4 + tImagData;
}
twiddle_index += twiddle_incr;
}

// Main acceleration occurs by ping-pong buffering. At the end of each iteration
// stage we swap the ping-pong buffers. We do this with a straight assignments.
// We need to have the Real_Data_Write buffer point to the address in memory
// where BufferedRealCalcRead was stored on the previous pass. Since the
// memory buffers are dual port we simply point BufferedRealCalcWrite to the
// location where Real_Data_Read_by_Port2 was pointing to/ in the previous pass.
Real_Data_Read = Real_Data_Write;
Real_Data_Write = Real_Data_Read_by_Port2;
Real_Data_Read_by_Port2 = Real_Data_Read;
Real_Data_Write_by_Port2 = Real_Data_Write;
Img_Data_Read = Img_Data_Write;
Img_Data_Write = Img_Data_Read_by_Port2;
Img_Data_Read_by_Port2 = Img_Data_Read;
Img_Data_Write_by_Port2 = Img_Data_Write;
}

// Stage 3: Write results back to RAM. Read in 32bits and silit.
for(Output_Counter = 0; Output_Counter < NUM_POINTS; Output_Counter++)
{
    Output_Ptr_tmp[Output_Counter] = (((alt_u32)(Img_Data_Read[Output_Counter])
& 0x0000FFFF)<<16) | ((alt_u32)Real_Data_Read[Output_Counter] &
0x0000FFFF);
}
}

```

