TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Syed Muhammad Jawad Hassan IVEM 144956

# ENERGY ESTIMATION OF FPGA ARCHITECTURES FOR A COMPRESSED SENSING ENGINE

Master's Thesis

Supervisor: Yannick Le Moullec

PhD

Tallinn 2018

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Syed Muhammad Jawad Hassan IVEM 144956

# FPGA-ARHITEKTUURIDEL TIHETAJUMISE MOOTORI ENERGIATARBE HINDAMINE

Magistritöö

Supervisor: Yannick Le Moullec

PhD

Tallinn 2018

## Author’s declaration of originality

I hereby certify that I am the sole author of this thesis to all the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented or submitted for examination anywhere else.

Author: Syed Muhammad Jawad Hassan

21.05.2018

# Abstract

Technological advancements in wireless sensor networks (WSN) have enabled and increased the research interest in healthcare applications such as self-fitness, health monitoring and many more. Amongst many such applications, this thesis takes it outset in wireless heart-rate monitoring (HRM) and is mainly focused on estimating the energy consumed by five architectures mapped onto a field programmable gate array (FPGA) platform. The five architectures are full parallel, full parallel without pipeline, generic semi-parallel, semi-parallel, and semi-parallel with clock enable.

After briefly elucidating the extraction of QRS from HRM using an existing system, the thesis moves on to the compressed sensing (CS) engine used for minimizing the size of the transmitted data. Using Altera TimeQuest timing analyzer tool, the aforementioned CS engine architectures are studied comprehensively. The tool starts with performing the timing analysis on the five architectures. In addition, the missing power values of two architectures (semi-parallel with and without clock-enable) are also estimated using Altera Power Analysis tool. Next, the energy consumption has been calculated for the architectures using the estimated values of power and execution time. Finally, the energy consumption of the five architectures has been compared at maximum and operating frequencies.

Conclusively, it has been observed that none of the architectures is a definite ‘winner’ since not all design metrics (execution time, resources, power, and energy) can be optimized simultaneously. Thus, the results illustrate that the five architectures offer different trade-offs that could possibly fit to HRM applications with different requirements in terms of the above design metrics.

This thesis is written in English and is 57 pages long, including 5 chapters, 22 figures, and 5 tables.

# Annotatsioon

## FPGA-arhitektuuridel tihetajumise mootori energiatarbe hindamine

Traadita andmesidevõrkude (WSN) tehnoloogilised edusammud on võimaldanud ja suurendanud teadusuuringute huvi tervishoiu rakenduste vastu, nagu näiteks enesetreeningute ja tervisekontrolliga seoses. Paljude selliste rakenduste hulgast on see töö traadita juhtmevaba südame löögisageduse seirest (SLS) ja keskendub peamiselt viie FPGA-l realiseeritud arhitektuuri osas tarbitud energia hindamisele. Viis arhitektuuri on täisparalleelne, ilma konveierita, üldine poolparalleelne, poolparalleelne ja paralleelne taktkella aktiveerimisega lahendused. Pärast lühikest selgitamist QRSi eraldamise osas südame löögisageduse seirest olemasoleva süsteemiga liigub magistritöö edasi kompresseeritud võendamise (KV) mootori juurde, mida kasutatakse edastatud andmete suuruse minimeerimiseks. Kasutades Altera TimeQuest Ajastusanalüsaatorit, uuritakse ülalmainitud KV mootori arhitektuure igakülgselt. See algab viie arhitektuuri ajalise analüüsi teostamisega. Seejärel hinnatakse Altera jõudluse analüüsimise tööriistaga kahe arhitektuuri osas puuduvaid võimsuse väärtusi (poolparalleelsed kellaajaga ja ilma lahendused). Järgnevalt on energiatarvet arvutatud arhitektuuride jaoks, kasutades eeldatavaid energiakulu- ja täitmisaja väärtusi. Lõpuks on viie arhitektuuri energiatarbimist võrreldud maksimaalsel ja töösagedusel.

Kokkuvõtvalt on täheldatud, et ükski arhitektuur ei ole kindel võitja, kuna kõiki arenduse mõõdikuid (täitmisaeg, ressursid, võimsus ja energia) ei saa üheaegselt optimeerida. Seega näitavad tulemused, et viis arhitektuuri pakuvad erinevaid kompromisse, mis võiksid sobida SLS-i rakendustega ning mis vastavad ülaltoodud arenduse mõõdikutele.

Antud töö on ingliskeelne ja on 57 lehekülge pikk, sisaldades 5 peatükki, 22 joonist ja 5 tabelit.

# List of abbreviations and terms

| | |
|---|---|
| BPM | Beat per minute |
| CS | Compressed sensing |
| EDA | Electronic device automation |
| FPGA | Field programmable gate array |
| FSMD | Finite state machine |
| HR | Heart rate |
| HRM | Heart rate monitoring |
| HRV | Heart rate variability |
| LE | Logic element |
| LUT | Look up table |
| RTL | Register transfer logic |
| TUT | Tallinn University of Technology |
| UUT | Unit under test |
| VHDL | Verilog  hardware language |
| WSN | Wireless sensor networks |
| WBAN | Wireless body area networks |

# Table of contents

# List of figures

# List of tables

# 1 Introduction

The technological advancements over the past several years have pushed forward the research in the new direction of creating cutting-edge applications. The intense use of wireless networks and continuous miniaturization of electronic devices have enhanced the advancement in wireless body area networks (WBAN) [1]. Development in heart rate monitoring (HRM) has received increased interest over the past few years. Heart rate monitoring has initiated a wide range of applications in numerous areas, e.g., it is being used in emergency condition monitoring in hospitals, biomedical, self-fitness, health monitoring for elderly peoples at home and many more [2]. A wireless sensor network (WSN) is a system that collects the measured data from sensor, processes it if needed and sends reports wirelessly to network. A WSN sensor node mainly consists of six fundamental components. These are the sensing unit, processing unit, analog-to-digital converter, power unit, communication unit (radio transceiver), and memory unit. In a WBAN, such WSN nodes are used to compute the functional data from either external or internal parameters of the human body. Examples of sensing include measuring one or more physical parameters like monitoring heartbeat, body temperature or electrocardiogram (ECG).

It is noted that research has especially focused on heart rate variability (HRV) due to increasing demands in sport and self-fitness. HRV is evaluated by examining beat-to-beat variations in standard R-R intervals [2],[3]. Mainly, cardiovascular diseases can be analyzed in view of HRV that can be extracted from heart rate (HR) measurements [4]. Higher HRV has been directly related to the lower mortality rate (which is also influenced by both age and sex). Hospitals and old homes are both feasible venues to use these types of healthcare monitoring systems. Using sensors devices enables continuous monitoring of a patient's heart rate and providing real-time feedback data to the medical personnel, which in turn improves the quality of measured data and gives positive point of no surveillance by medical personnel all the time; they are only alarmed in case of instability sensed in HR [5].

In the sports domain, HRMs are mostly deployed as a training tool to determine the exertion in training spells and sports to gain the maximum benefit to minimize overtraining. During exercise, the output power generated might be an immediate factor, but HR is a more convenient way to measure and monitor it [4]. HRV study is a vital tool to observe the heart ability to react to the regulatory impulse that influences its rhythm. Through studies, it has been proven that during training exercises the trained individuals have higher HRV which decreases gradually until specific intensities, and after that, it stabilizes [6]. Therefore, HR sensing technology is deployed on professional's athletes to control their strength during exercise and workout.

For convenience, such HRM devices are typically and preferably powered by relative small batteries. In consequence, one of the most significant challenges in their design is sparing energy since it is one of the primary factors that influence the lifetime of the whole system

The energy usage in a sensor node can be characterized in three major areas: sensing, data communication and data processing [7]. As of today, the wireless communication part (i.e., radio transceiver) is still considered as the most energy consuming element in a sensor node; it is typically higher as compared to the energy needed to the processing of the data. It has also been shown that security and privacy protection systems increase the energy consumption [8],[9]; however, this aspect is not within the scope of this thesis.

## 1.1 Energy Constraints of WSNs

With the development of the wireless communication, energy resources come out as the primary constraint in WSN. The energy usage in a sensor node can be characterize in three significant areas: sensing, data communication and data processing [8],[10]. The energy available to the sensor nodes is often limited due to the size of the battery, which is the most significant contributor to sensor nodes in term of its weight and dimensions. Although, there is a remarkable improvement in design architectures and data processing significant increase is required in battery technology.

Subsequently, reduction of power consumption and a standard approach to reducing data transmission is the main point of focus for WSN nodes [11],[12]. Therefore, for this purpose, an extra smart layer should be used to accommodate intelligent processing with data compression techniques[13],[14].

## 1.2 Context of this work

This work is part of a research activity conducted at Thomas Johann Seebeck Department of Electronics at Tallinn University of Technology (TTU) and partly in collaboration with Aalborg University (AAU).

Two students from AAU have done the initial work; they proposed an algorithm and architectural designs for an FPGA-friendly compressed sampling engine for WSN-based HRM [15]. Firstly, QRS detection on ECG signals and CS that allows a reduction of data transmitted wirelessly have been studied. Secondly, different hardware architectures have been explored with various parallelism levels for compressed sensing (CS). In their study, two main functional blocks in WBAN node, i.e., QRS detection and compressed sensing (CS), have been designed and simulated for Altera Cyclone III FPGA platform using Quartus design tool. These functional blocks have been tested for various HR signals in the MATLAB-based simulation model. Additionally, hardware architecture solutions for CS engines have been developed at the register transfer level (RTL) to analyze and present how parallelism affects execution time.

Then, in another project conducted by a student from TTU [16], an analysis has been performed for the functional blocks developed in [15]. In [16] the focus was on FPGA resource usage as well as corresponding estimation of power consumption on a Cyclone IV FPGA platform. Synthesis results and power estimates have been presented and compared for three different architectures. The results revealed that pipelining helps to lower the power consumption but requires more resources with higher frequencies.

As mentioned earlier, energy plays a critical role when it comes to the lifetime of battery-powered nodes. Although the results presented in [15] and [16] have very well defined the development of the architectures and power estimates, the most important aspect, i.e., energy consumption, has not yet been undertaken.

Thus, the focus of this MSc thesis is on energy consumption and is expected to provide a valuable contribution to the research activity on WBAN/HRM conducted in the department.

## 1.3 Problem Statement

Expanding the lifetime of the sensors in the system by reducing their energy usage has turned out to be one of the primary challenges in practical applications. In response to this, over the last couple of years, there have been multiple efforts to limit power and energy consumption of FPGAs through new algorithms and architectures.

Contrary to the two previous works that mostly focused on one aspect at a time, this MSc thesis seeks to explore the trade-off between multiple performance metrics, as indicated in what follows.

**Research Statement**: The purpose of the work is to explore the trade-off between execution time, resource usage, power consumption, and most importantly energy consumption among the five architectures proposed in [15]-[16]. Doing so can help designers to understand the relations between the various performance metrics better and make informed design decisions.

To deal with the above research statement, several goals and tasks have been defined as listed below.

- Define and implement the overall flow required to obtain the performance metrics, including:
    - Understand the tool specification of TimeQuest timing analyzer.
    - Exploit it to calculate the execution time at operating frequency (100 MHz) and maximum frequency ($F_{max}$) for each of the five architectures separately.
    - Create test-benches in order to calculate the power at 100 MHz and $F_{max}$ for the five architectures.
    - Calculate energy at 100 MHz and $F_{max}$ for the five architectures.
- Explore the trade-off by comparing the performance metrics for the five architectures.

The primary aim of this thesis consists of literature review, problem statement, the method used in this thesis, analysis and output results, and conclusion with a suggestions for future research.

The remainder of this thesis is organized as follows. Chapter 2 presents a literature review of Electrocardiogram, Field Programmable Gate Arrays (FPGA) and existing architecture. Chapter 3 presents the method used to estimate energy. In the end Chapter 4 presents, the results, and analysis of five architectures. The conclusion is presented in Chapter 5 which summaries the central output theme of this thesis work and recommends some suggestion for the future research work.

# 2 Background

This thesis work investigates the implementation of heart rate (HR) electrocardiogram (ECG) signal processing on FPGA technology; thus, Section 2.1 presents the essence of ECG. Section 2.2 describes the background information about the Intel (Altera) Cyclone IV FPGA. Moreover, section 2.3 discusses the existing architectures used in this work.

## 2.1 Electrocardiogram (ECG)

ECG signals are used as an important tool for the detection and analysis of cardiovascular irregularity. ECG is the signal report of deviation of bioelectric activity of the heart concerning time known as heartbeats, which is calculated in beats per minute (bpm). ECG is used most commonly to record signals for the patient monitoring and examination procedure. The bioelectric activity (contraction and relaxation) of the heart is used to record through the surface electrodes placed on the limbs or chest [17]. The fundamental purpose of ECG is to gather information about the function, structure, and condition of the heart. The anomaly of the ECG shape is called arrhythmia which is a common term used for any cardiac signals that differ from regular (normal) ECG signal [18]. A normal ECG signal is illustrated in Figure 1 [18].

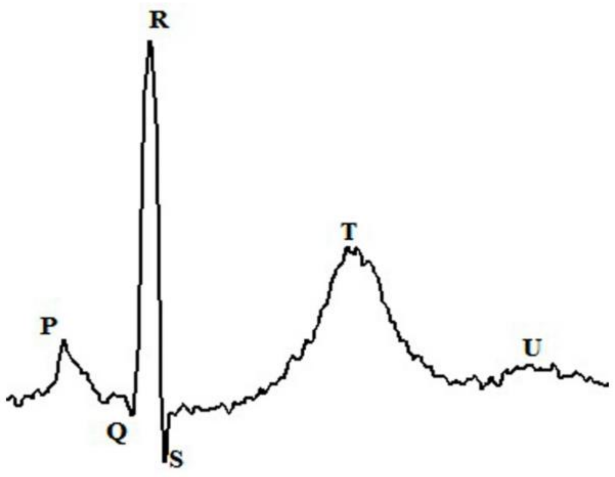


Figure 1. The normal ECG waveform [18].

The pulse of the heart electrocardiogram signal is presented in Figure 1. The typical ECG is described in terms of intervals that are P wave, QRS complex, T wave and U wave. T and U waves consist of signals which contain the vital information about a characteristic of disease affecting heart [19]. P-wave shows the occurrence of impulse due to depolarization of left and right atria. PQ segment initiates a time of spreading impulse from atria to ventricles while depolarization of ventricles marks the origination of QRS complex [11]. Figure 2 illustrates the different waves duration in ECG signal.

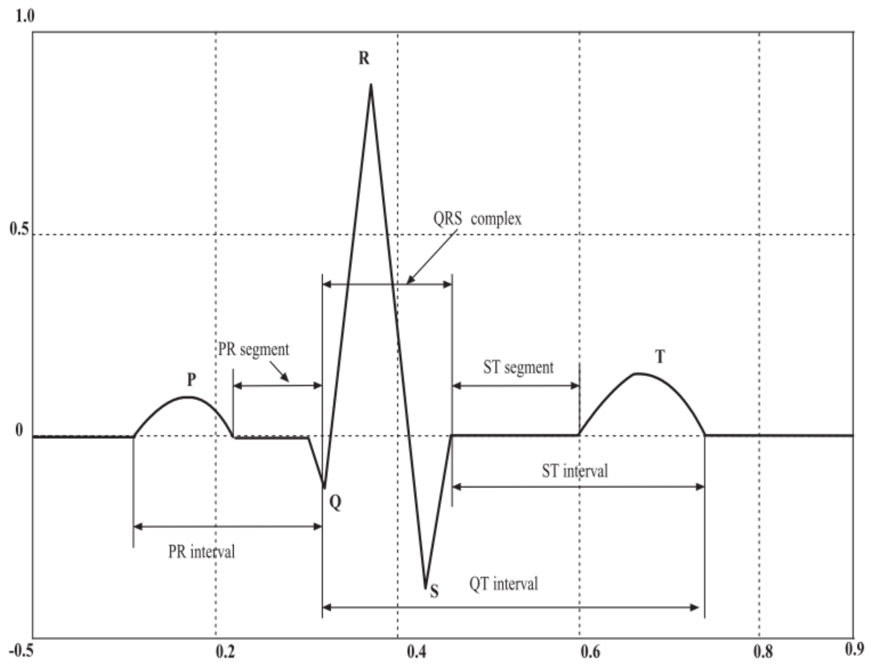


Figure 2. Representation of the different wave durations in a normal ECG signal [15].

P-wave is a slow duration and low amplitude wave of the 60-80ms period and 0.1-0.2mV amplitude [20]. Where the PR-Interval starts from the beginning of P-wave and ends on the beginning of QRS complex, it has a duration of 120-200 ms.

## 2.2 Cyclone IV FPGA

During several years FPGAs have emerged as a fascinating alternative to microcontrollers of commercial sensor nodes because of high processing capability for execution of signal processing applications [11]. The additional latest features, such as increased number of multipliers and adders in modern FPGA’s, have increased their performance regarding latency and throughput of data. Moreover, intensification of technology not only increased the system speed but as well as made the FPGA-based system more energy efficient.

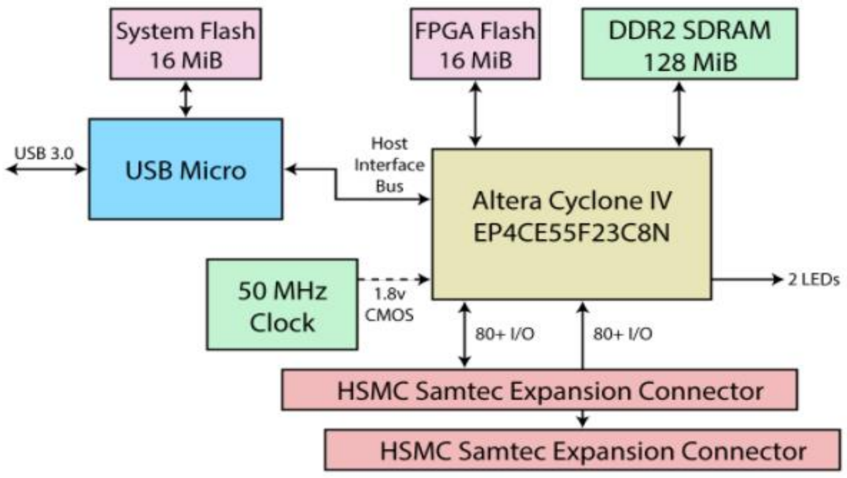


Figure 3. Block diagram of Cyclone IV Board [Opal Kelly][21]

The FPGA, Altera Cyclone IV E P4CE22F17C8L is selected for this thesis has the specification described in Table 1; these values are taken from datasheet [21]. Moreover, the diagram of an example board that uses the Cyclone IV is shown in Figure 3.

Table 1. Cyclone IV E device features description [18].

| Resources | Logic Elements (LEs) | Embedded memory(K bits) | Embedded 18 x18 multipliers | General-purpose PLLs | Global Clock Networks | User I/O Banks | Maximum user I/O |
|---|---|---|---|---|---|---|---|
| **EP4CE22** | 22,320 | 594 | 66 | 4 | 20 | 8 | 153 |

Fabrication of FPGA is designed in a bi-dimensional array with interconnected logical blocks. The logical blocks that stores Boolean functions are composed of look-up tables (LUTs) assembled over simple memories [22]. Cyclone IV devices fabric consists of logical elements, embedded memory blocks, embedded multipliers and 4 input LUTs. The cyclone models are classified according to quality control related to assembling procedures, and it is signified by the characters in model number as shown in Figure 4.

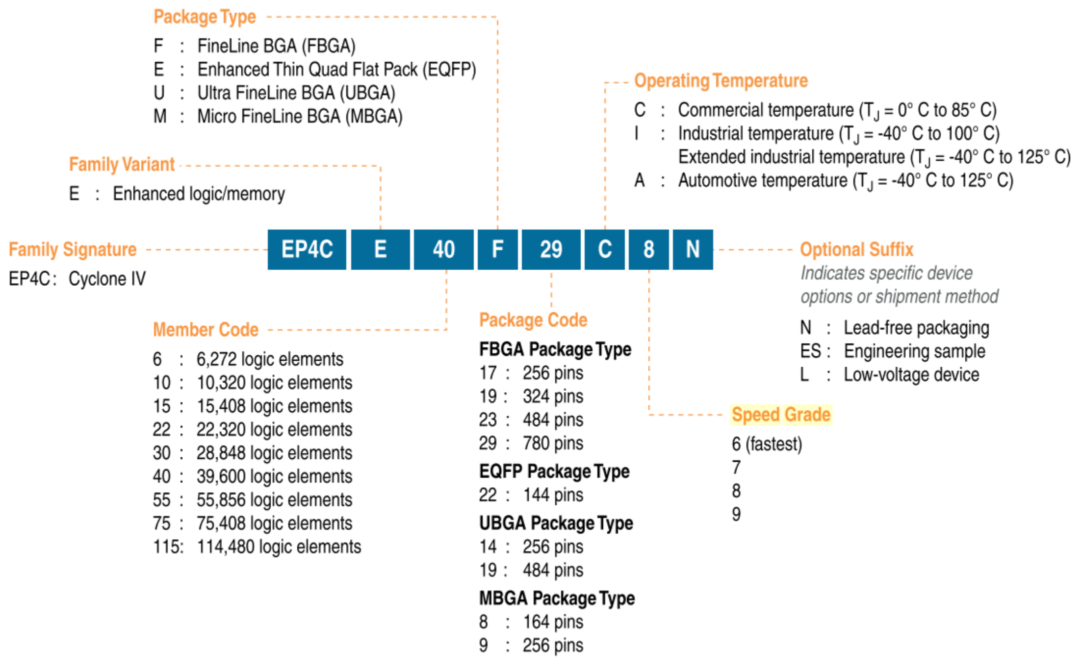

Package Type
F : FineLine BGA (FBGA)
E : Enhanced Thin Quad Flat Pack (EQFP)
U : Ultra FineLine BGA (UBGA)
M : Micro FineLine BGA (MBGA)
Family Variant
E : Enhanced logic/memory
Operating Temperature
C : Commercial temperature ($T_J$ = 0° C to 85° C)
I : Industrial temperature ($T_J$ = -40° C to 100° C)
Extended industrial temperature ($T_J$ = -40° C to 125° C)
A : Automotive temperature ($T_J$ = -40° C to 125° C)
Family Signature
EP4C: Cyclone IV
EP4C
E
40
F
29
C
8
N
Optional Suffix
Indicates specific device options or shipment method
N : Lead-free packaging
ES : Engineering sample
L : Low-voltage device
Member Code
6 : 6,272 logic elements
10 : 10,320 logic elements
15 : 15,408 logic elements
22 : 22,320 logic elements
30 : 28,848 logic elements
40 : 39,600 logic elements
55 : 55,856 logic elements
75 : 75,408 logic elements
115: 114,480 logic elements
Package Code
FBGA Package Type
17 : 256 pins
19 : 324 pins
23 : 484 pins
29 : 780 pins
EQFP Package Type
22 : 144 pins
UBGA Package Type
14 : 256 pins
19 : 484 pins
MBGA Package Type
8 : 164 pins
9 : 256 pins
Speed Grade
6 (fastest)
7
8
9

Figure 4. Packaging Ordering Information for Cyclone IV Device [23]

The Cyclone models first five characters indicates the family signature and variant, member code, package code and type, and speed grade. In this thesis FPGA IV E is considered with member code 22; which means 22,320 logic elements. F 17 Package code and type are FBGA 256 pins, and speed grade C8 is used in model FPGA device. Each of the FPGA devices has its own speed grades rating; it is essential to specify the speed grades because it specifies the maximum frequency at which the given model can operate. The figures which are essential to this study is the number of embedded multipliers and adders used in the FPGA which is selected because as discussed in [11], multipliers might be a limiting factor if the algorithm level design techniques are executed in parallel or semi-parallel. Same critical selection applies to logical elements (LEs), as logical elements are applied to the logic behind the multipliers. This importance of selection is discussed in Section 4, where the results showed how much LEs are used in multipliers, and their impact on the Power consumption and energy estimation for parallel and semi-parallel architecture.

## 2.3 Existing Architecture

Based on the architecture design, RTL implementation makes it possible to describe the design used in this thesis. Different types of architectures are proposed in the previous study [15], the modified semi-parallel, semi-parallel, and full parallel architecture. In high-level synthesis, it is complex and time-consuming task to employ both RR wave and compressed sensing (CS) algorithms onto an RTL architecture [24]. This is because of the non-restricted dimensions of sampling kernels M × N in the CS part, imposing that M and N can be any integer values where M < N [15]. The M × N is matrix-vector multiplication represented as **H** and a vector x of size N, which is illustrated in equation (1) and equation (2)

$$y = Hx \tag{1}$$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} H_{11}x_1 & + & H_{12}x_2 & + & \dots + & H_{1N}x_N \\ H_{21}x_1 & + & H_{22}x_2 & + & \dots + & H_{2N}x_N \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ H_{M1}x_1 & + & H_{M2}x_2 & + & \dots + & H_{MN}x_N \end{bmatrix} \tag{2}$$

Equation (1) shows the execution of all operation can be sequential and, the execution time for hardware is dependent on numerous multiplication and addition in the algorithm (as shown in Equation 2). Here in this study, we assumed fixed vector dimension where the N is considered fixed which can reuse similar sampling kernel H for any signal vector **x**.

In this Master's thesis, Altera Cyclone IV is used; thus the synthesis results obtained in [15] for a Cyclone III have been regenerated for the Cyclone IV using Altera Quartus. The outcome information collected is shown in Table 2.

Table 2: Different functional units synthesis results [15]

| Component Type | Number of LE's | $F_{max\ [MHz]}$ | $T_{min\ [ns]}$ |
|---|---|---|---|
| Adder | 16 | 312.79 | 3.197 |
| Multiplier (combinatorial) | 188 | 85.37 | 11.714 |
| Multiplier (embedded) | 9 | 225.33 | 4.438 |

In the study [15], a plot is used to describe the various degree of parallelism of architecture concerning total execution time in Figure. 5. Where execution time was estimated as a function of N for both multipliers (combinatorial and embedded) illustrated in Table 2. It is noted from Figure. 5 that semi-parallel presents linear growth in clock cycle as a function of N, whereas logarithmic behavior can see for full-parallel. Ultimately, full sequential scheme shows a non-linear increase. Therefore, it is noted that full-parallel perspective is worthwhile to execute large number data in small time vice versa is in case of semi-parallel and full sequential but beneficial to reduce usage of the hardware resource.

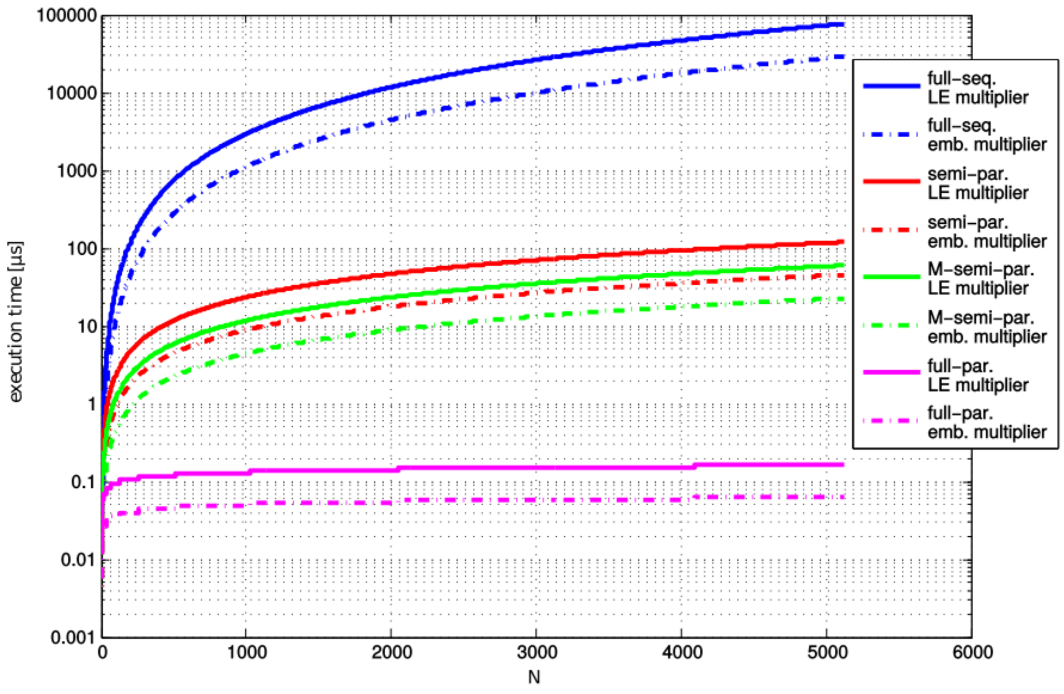


Figure 5. Total execution time over the function of N [15]

FSMD method is used in [15], to describe the RTL implementation of full and modified semi-parallel. Finite State Machine with Data Path (FSMD) is a composed of control path and data path for the design, where Finite State Machine and processing part handle by control path and data path respectively. The concept of FSMD supports the matrix $H \in R^{2\times 4}$ and a vector$x \in R^{4}$. The basic block diagram for FSMD is presented in Figure. 6

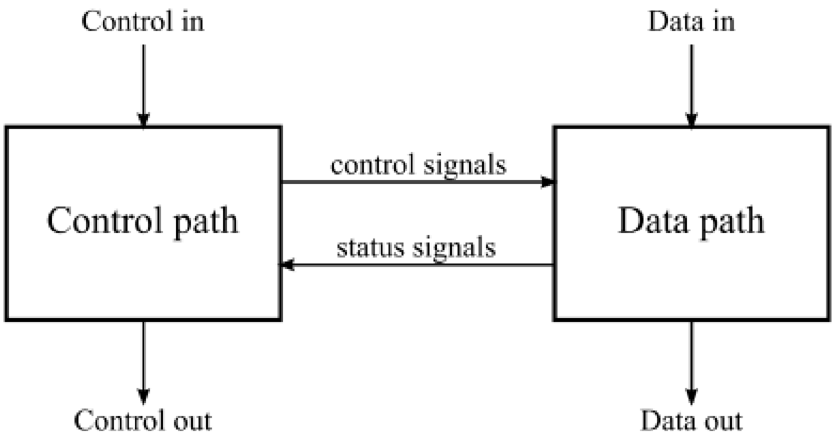


Figure 6. The concept of Finite State Machine with Data Path FSMD [13]

A finite state machine with data path aid to remove the temporary results stored in the register and reset the counter for input integer N and revise the output continually. On account of modified semi-parallel architecture, the multiplication is carried out in parallel, whereas addition is performed in every stage in control path. Here it is crucial to control the clock to achieve reliable energy estimation consumption results. In case of full parallel architecture, the multiplier and adders are both used more parallel and assign for each operation in matrix-vector multiplication. In Figure 7, the modified semi-parallel architecture illustrates multiplication in parallel and addition at every control stage. In contrast to semi-parallel, in full parallel, the multiplication and addition both are executed in parallel at every step (as shown in Figure 8). It can easily be concluded from the figures that with the increase of columns in the matrix will increase the number of functional units within rows of matrix-vector multiplication, where the $X_i$ and $H_{ji}$ are the input registers respectively reserving values in $R_{i,}$ and final output results are stored in $Y_i$.

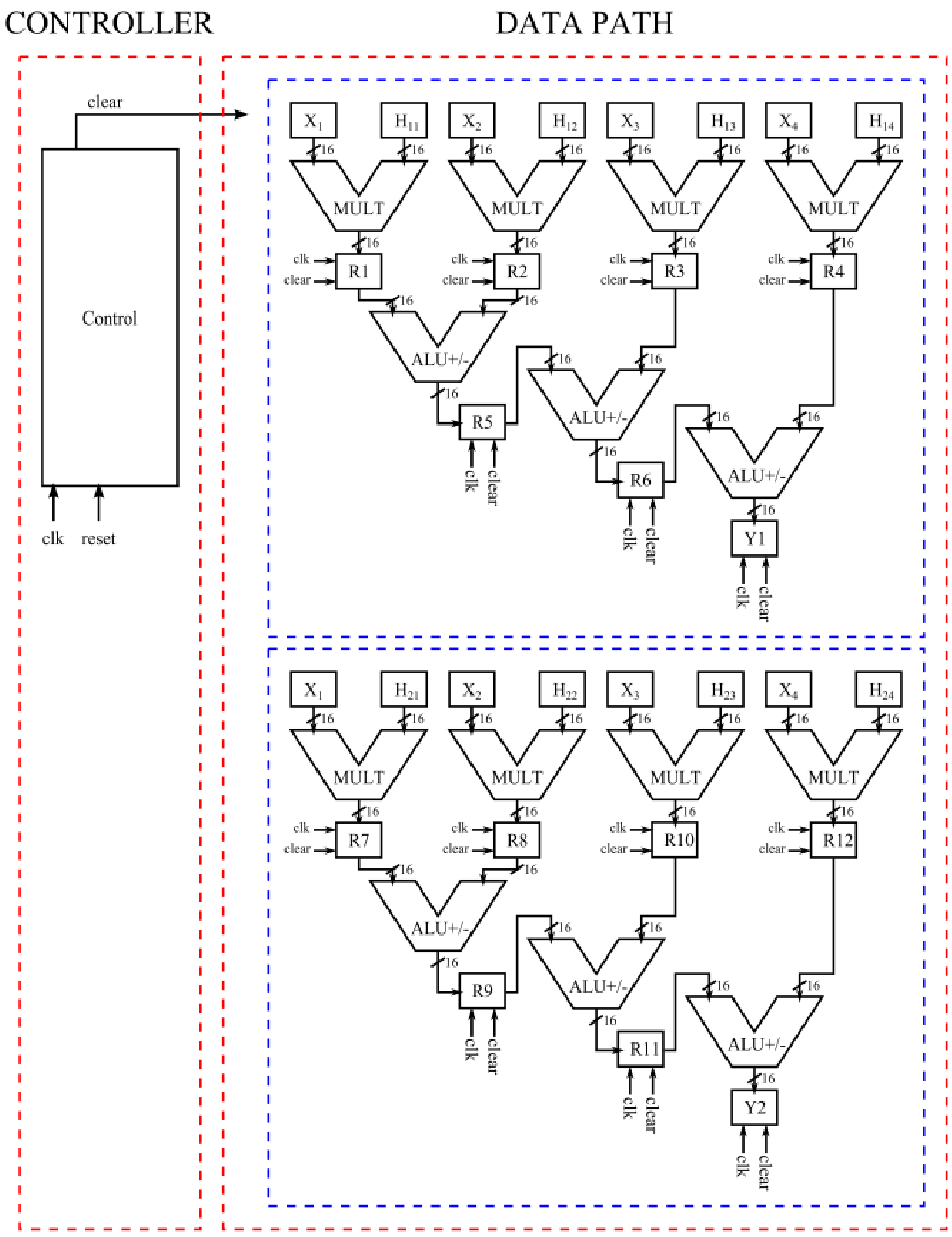


Figure 7. RTL implementation of the modified semi-parallel architecture [13].

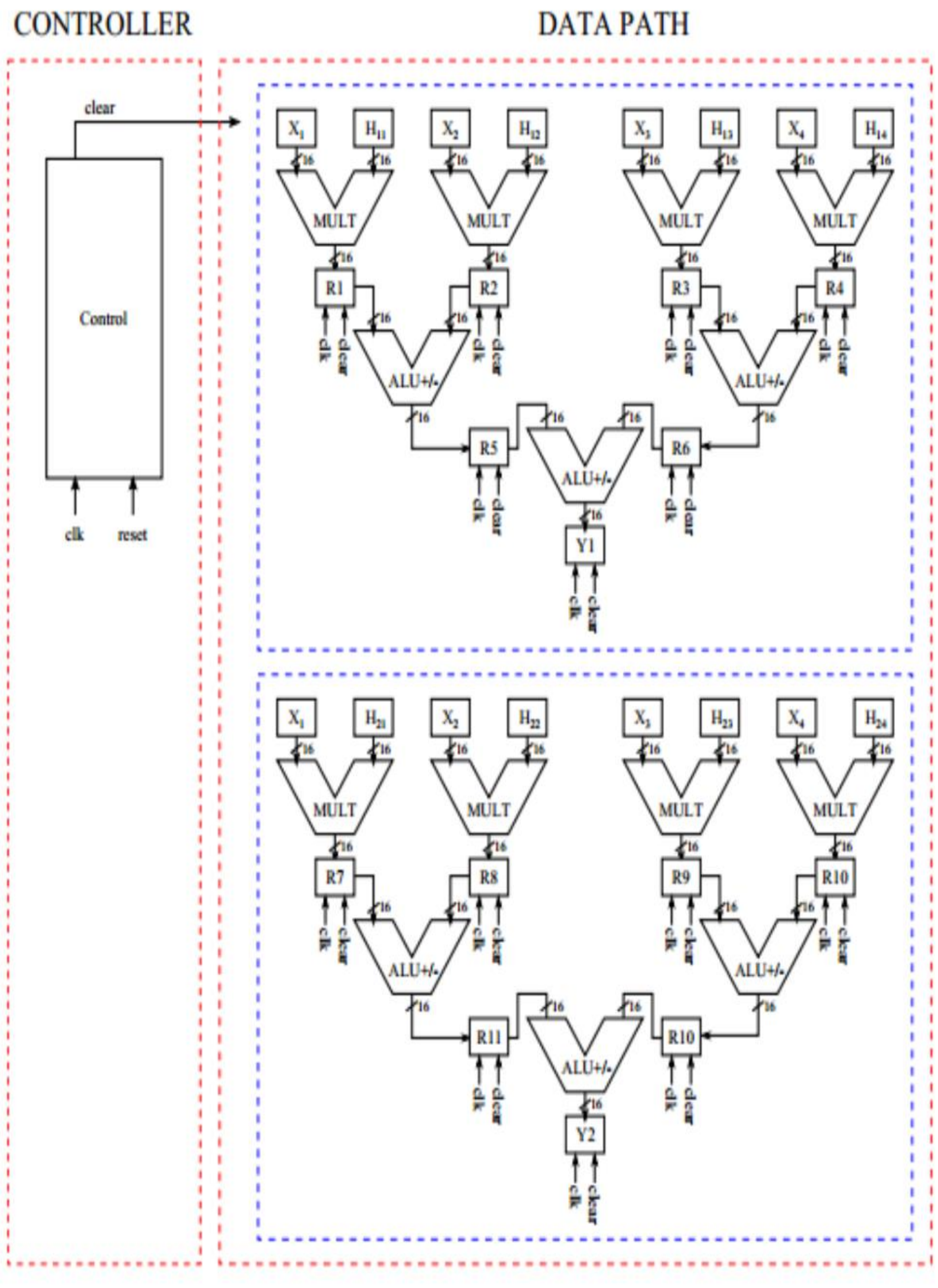


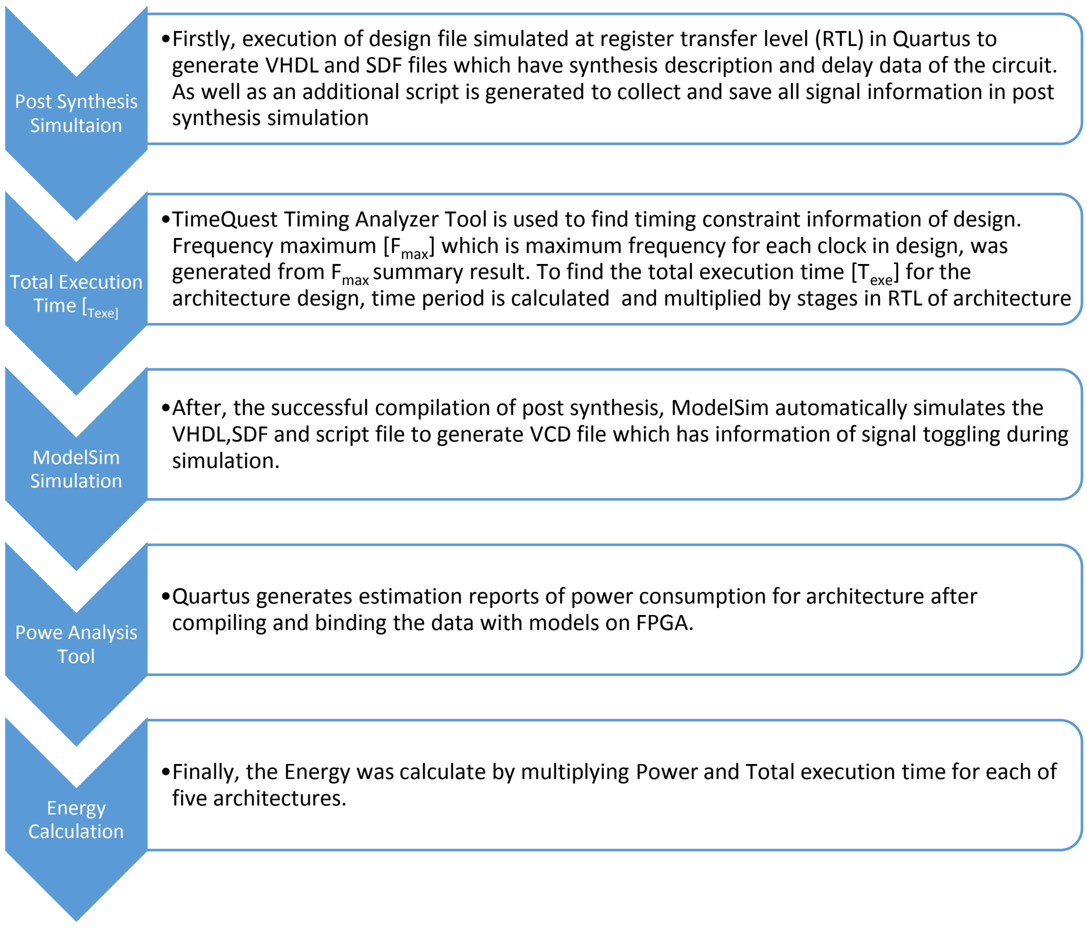
Figure 8. RTL implementation of full-parallel [13]

# 3 Method for Energy Estimation

The previous chapter has introduced the background of the starting point of this thesis work. As discussed in the problem statement (see Section 1), the next step, as well as the main aim of this thesis, is to evaluate and analyze the energy consumption of the main blocks designed and simulated on Altera in [15].

This chapter presents the method which has been used to estimate and analyze the energy consumption by using Altera tools for the current designs. The overall flow of the method using Altera Quartus and ModelSim tools is summarized in Figure 9.

## 3.1 Flowchart



- Firstly, execution of design file simulated at register transfer level (RTL) in Quartus to generate VHDL and SDF files which have synthesis description and delay data of the circuit. As well as an additional script is generated to collect and save all signal information in post synthesis simulation
- TimeQuest Timing Analyzer Tool is used to find timing constraint information of design. Frequency maximum [$F_{max}$] which is maximum frequency for each clock in design, was generated from $F_{max}$ summary result. To find the total execution time [$T_{exe}$] for the architecture design, time period is calculated and multiplied by stages in RTL of architecture
- After, the successful compilation of post synthesis, ModelSim automatically simulates the VHDL,SDF and script file to generate VCD file which has information of signal toggling during simulation.
- Quartus generates estimation reports of power consumption for architecture after compiling and binding the data with models on FPGA.
- Finally, the Energy was calculate by multiplying Power and Total execution time for each of five architectures.

Figure 9. Flow Chart of the method used for energy estimation.

## 3.2 TimeQuest Timing Analyzer Tool

This section provides a brief introduction to TimeQuest Time Analyzer which demonstrates how to set up timing constraints and attain timing information for a logic circuit. Altera handbook tutorial guide describes that timing analysis computes the process of analyzing delay in the logic circuit to determine the reliable circuit operational conditions [25]. Calculating the longest path delay in a circuit and comparing these delays with the clock period is the primary concern of the timing analyzer.

### 3.2.1 Setup the Timing Netlist and Path

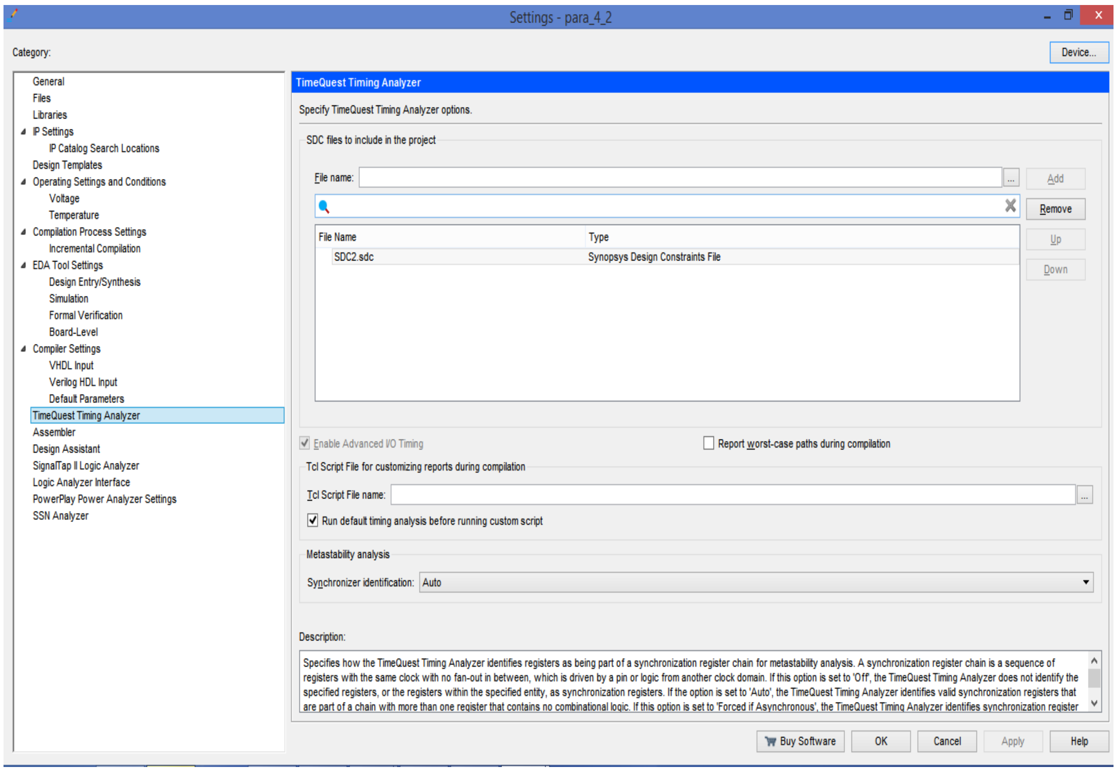
In Quartus Prime software, after successful compilation of the design file [.QPF], we need to define the timing analysis for our design. For performing timing analysis on any design, TimeQuest analyzer requires a timing netlist that shows how different elements are divided into cells, pins, and ports [26]. After generating a timing netlist, data of timing netlist is used by TimeQuest analyzer to drive different design elements and timing

A fundamental principle of timing path is to connect two design nodes, i.e., an output of one register to an input of another register. TimeQuest analyzer uses the following analyzer paths:

- Edge paths are the connections between ports-to-pins and pins-to ports.
- Clock paths are the connections between the clock pin of a register device port or internally generated clock pins.
- Data paths are a connection from a port output pin to input of another port pin or data output pin of a sequential element to data input pin of another sequential element.
- Asynchronous path is the connection established from a port or synchronous pins of another sequential element, i.e. an asynchronous reset or asynchronous clear.

### 3.2.2 Read SDC File

The Synopsys Design Constraint (SDC) file contains the functions used to specify constraints to the TimeQuest timing analyzer. It can also be edited manually by which the timing analyzer re-compile the new timing constraints. In Category list of Quartus tool, TimeQuest Timing Analyzer adds SDC file and run timing analysis by default before running the custom script and set synchronizer identification to auto.



TimeQuest Timing Analyzer

Specify TimeQuest Timing Analyzer options.

SDC files to include in the project

File name:

| File Name | Type |
|---|---|
| SDC2.sdc | Synopsys Design Constraints File |

[✓] Enable Advanced I/O Timing

[ ] Report worst-case paths during compilation

Tcl Script File for customizing reports during compilation

Tcl Script File name:

[✓] Run default timing analysis before running custom script

Metastability analysis

Synchronizer identification: Auto

Description:

Specifies how the TimeQuest Timing Analyzer identifies registers as being part of a synchronization register chain for metastability analysis. A synchronization register chain is a sequence of registers with the same clock with no fan-out in between, which is driven by a pin or logic from another clock domain. If this option is set to 'Off', the TimeQuest Timing Analyzer does not identify the specified registers, or the registers within the specified entity, as synchronization registers. If the option is set to 'Auto', the TimeQuest Timing Analyzer identifies valid synchronization registers that are part of a chain with more than one register that contains no combinational logic. If this option is set to 'Forced if Asynchronous', the TimeQuest Timing Analyzer identifies synchronization register

Figure 10. Addition of  SDC file in Quartus setting.

### 3.2.3 Update Timing Netlist

Update Timing Netlist command then uses the timing constraints and reports about the fail paths. It applies SDC file constraints to currently used timing netlist. The warnings generated during compiling timing netlist report about the undefined clocks, partially defined I/O delays and combinational loops. Finally, reports are generated after updating timing netlist. A successfully executed sequence is illustrated in Figure 11.

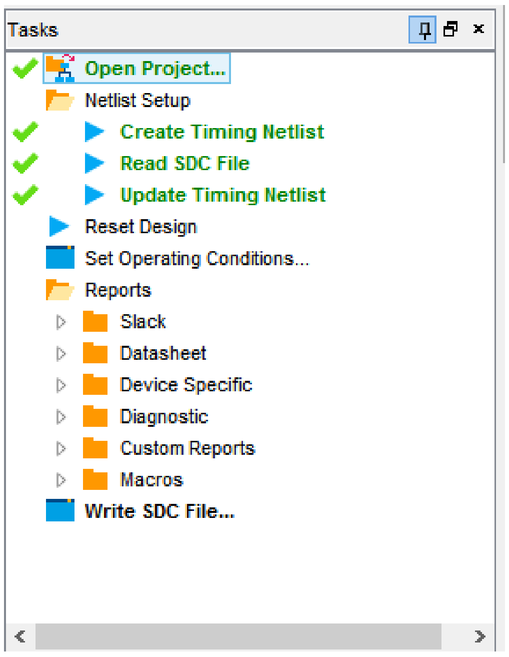
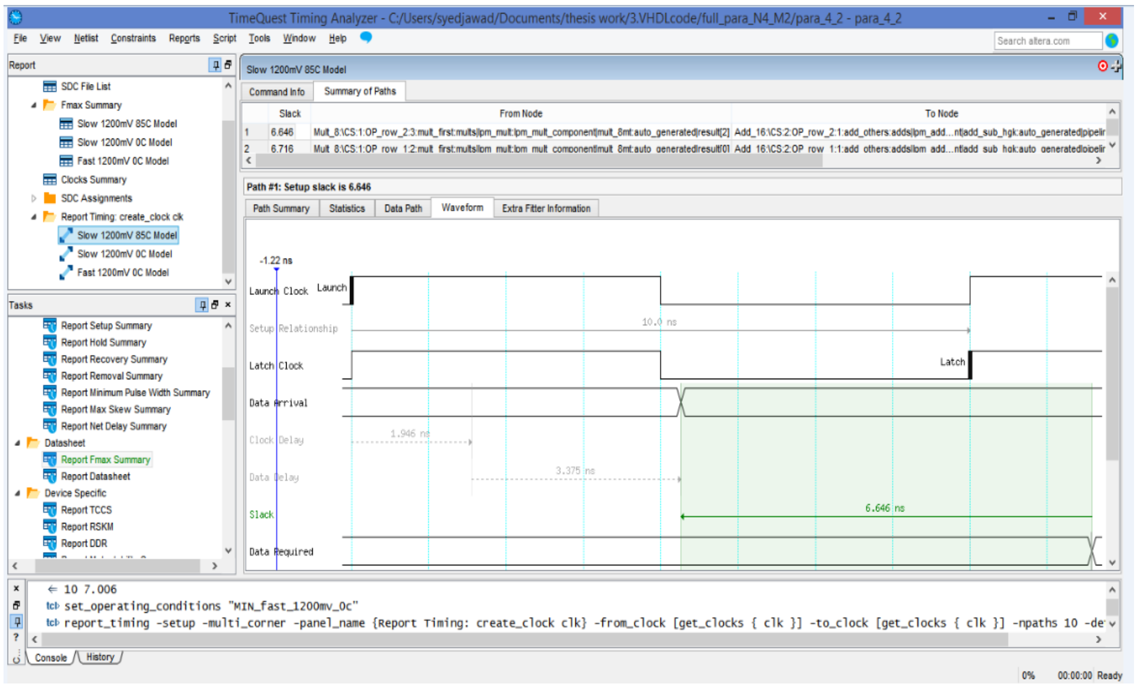


Figure 11. Updating timing netlist

### 3.2.4 Generate Timing Report

Finally, a timing report is generated to verify the timing requirements and locate violations which provide fully constrained design or ignored timing constraints. TimeQuest GUI consists of report, task, results and console pane. After creating timing report, the waveform explains the Latch clock, data arrival time and in results give the clock and data delay with slack value. Slack value represents the dissimilarity between the clock constraint and the path delay [27]. An example is shown in Figure 12.


TimeQuest Timing Analyzer - C:/Users/syedjawad/Documents/thesis work/3.VHDLcode/full_para_N4_M2/para_4_2 - para_4_2

Slow 1200mV 85C Model

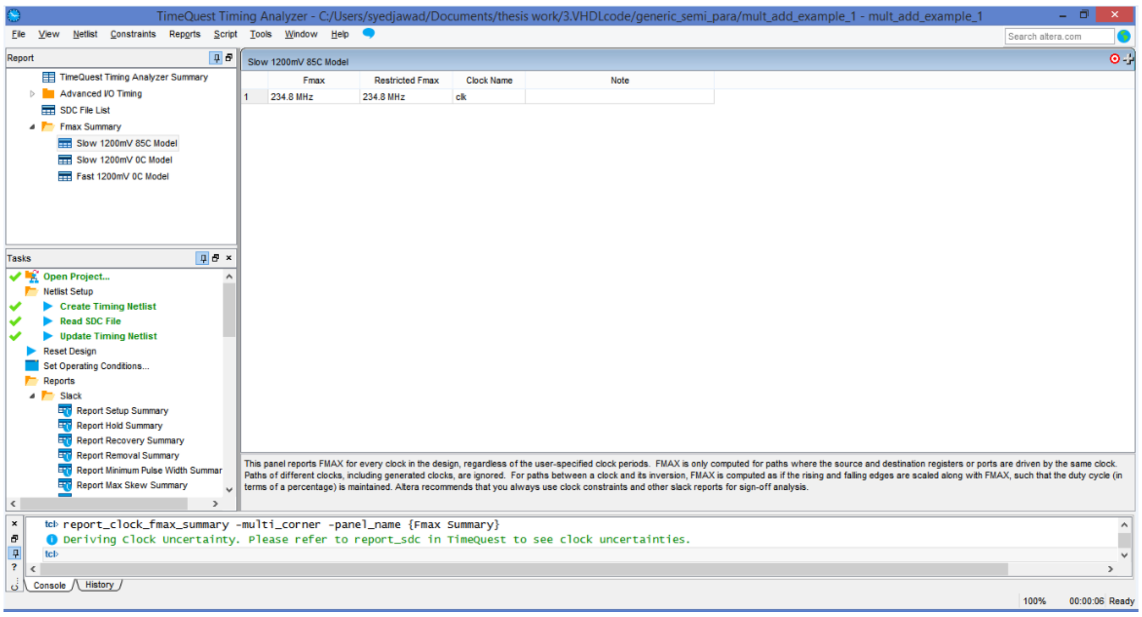
|  | Slack | From Node | To Node |
|---|---|---|---|
| 1 | 6.646 | Mult_8:\CS:1:OP_row_2:3:mult_first:mults\|lpm_mult:lpm_mult_component\|mult_8mt:auto_generated\|result[2] | Add_16:\CS:2:OP_row_2:1:add_others:adds\|lpm_add...nt\|add_sub_hgk:auto_generated\|pipelir |
| 2 | 6.716 | Mult_8:\CS:1:OP_row_1:2:mult_first:mults\|lpm_mult:lpm_mult_component\|mult_8mt:auto_generated\|result[0] | Add_16:\CS:2:OP_row_1:1:add_others:adds\|lpm_add...nt\|add_sub_hgk:auto_generated\|pipelir |

Path #1: Setup slack is 6.646

```
← 10 7.006
tcl▷ set_operating_conditions "MIN_fast_1200mv_0c"
tcl▷ report_timing -setup -multi_corner -panel_name {Report Timing: create_clock clk} -from_clock [get_clocks { clk }] -to_clock [get_clocks { clk }] -npaths 10 -de
```

Figure 12. Clock summary report in TimeQuest Timing Analyzer tool.

Here, with the help of this tool, Fmax is extracted, as shown in Figure 13. Then the period of a clock cycle is calculated with the formula$T = \frac{1}{f}$. The total execution time Texe is calculated by multiplying the period of a clock cycle (T) with the total number of stages. This number of stages can be found with the help of the RTL Viewer, as illustrated in Figure 14 for one of the five architectures, i.e., full parallel architecture.


TimeQuest Timing Analyzer - C:/Users/syedjawad/Documents/thesis work/3.VHDLcode/generic_semi_para/mult_add_example_1 - mult_add_example_1

Slow 1200mV 85C Model

|  | Fmax | Restricted Fmax | Clock Name | Note |
|---|---|---|---|---|
| 1 | 234.8 MHz | 234.8 MHz | clk |  |

This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and falling edges are scaled along with FMAX, such that the duty cycle (in terms of a percentage) is maintained. Altera recommends that you always use clock constraints and other slack reports for sign-off analysis.

```
tcl▷ report_clock_fmax_summary -multi_corner -panel_name {Fmax Summary}
ⓘ Deriving Clock Uncertainty. Please refer to report_sdc in TimeQuest to see clock uncertainties.
tcl▷
```

Figure 13. Summary report for $F_{max}$ for the generic semi-parallel architecture.

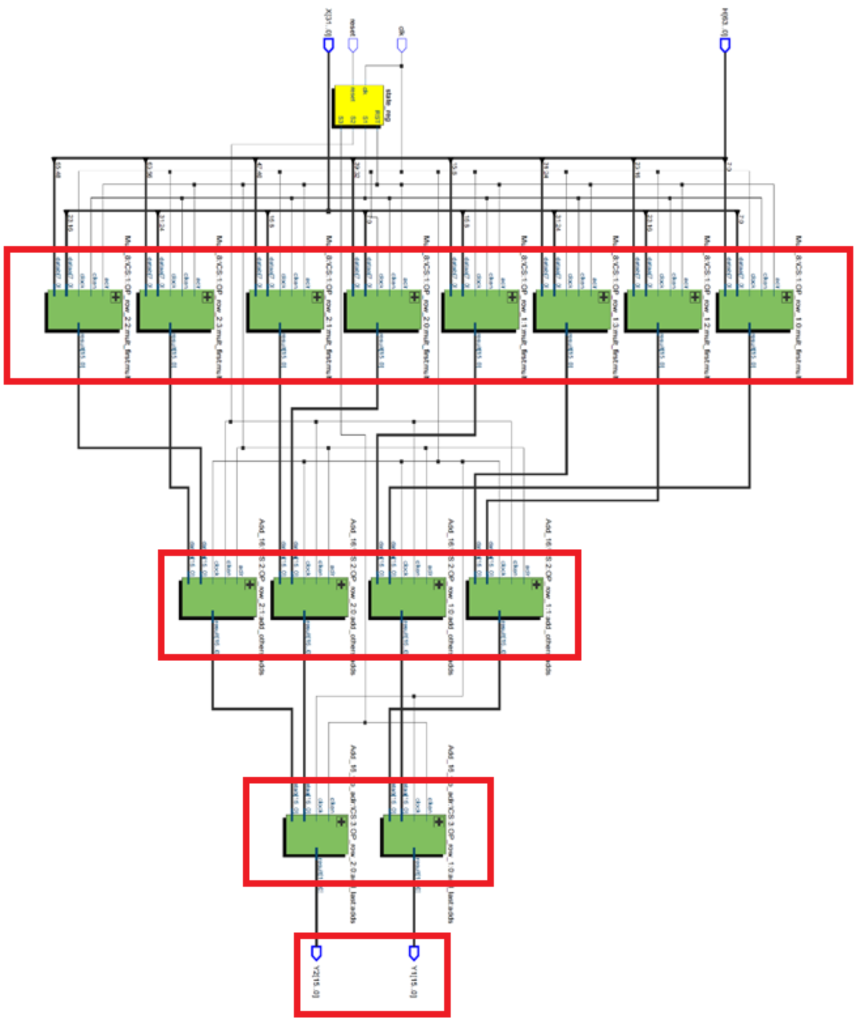
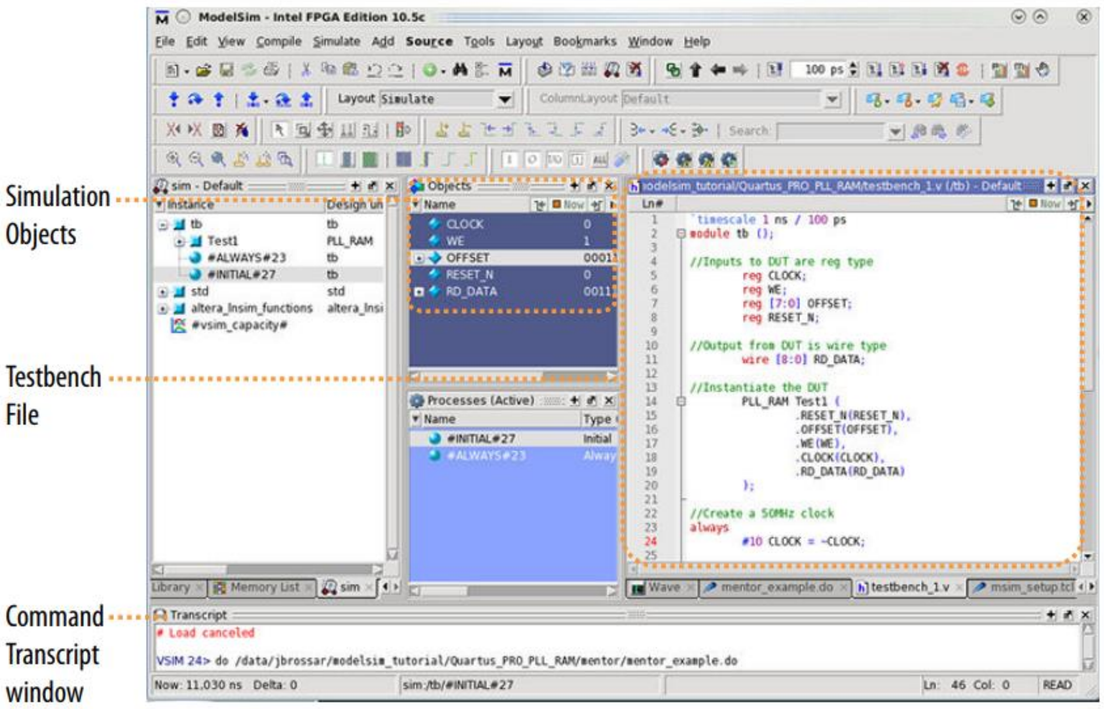


Figure 14. Presentation of stages in architecture design (In red box) for full parallel architecture.

## 3.3 Test Bench

The Altera test bench allows to develop and test design files in a very efficient and rapid way. Test bench files are applied to the design files to test a set of input test signals. These input test signals are generated and applied to the unit under test (UUT) within the test bench [28]. This method allows test benches to create a process to perform self-checking of the results. Finally, the test bench writes outputs to the ModelSim console window (see Figure 15).



| Instance | Design unit |
|---|---|
| tb | tb |
| Test1 | PLL_RAM |
| #ALWAYS#23 | tb |
| #INITIAL#27 | tb |
| std | std |
| altera_lnsim_functions | altera_lnsi |
| #vsim_capacity# | |

| Name | Now |
|---|---|
| CLOCK | 0 |
| WE | 1 |
| OFFSET | 0001 |
| RESET_N | 0 |
| RD_DATA | 0011 |

| Name | Type |
|---|---|
| #INITIAL#27 | Initial |
| #ALWAYS#23 | Alway |

odelsim_tutorial/Quartus_PRO_PLL_RAM/testbench_1.v (/tb) - Default

```
`timescale 1 ns / 100 ps
module tb ();

//Inputs to DUT are reg type
        reg CLOCK;
        reg WE;
        reg [7:0] OFFSET;
        reg RESET_N;

//Output from DUT is wire type
        wire [8:0] RD_DATA;

//Instantiate the DUT
        PLL_RAM Test1 (
                .RESET_N(RESET_N),
                .OFFSET(OFFSET),
                .WE(WE),
                .CLOCK(CLOCK),
                .RD_DATA(RD_DATA)
        );

//Create a 50MHz clock
always
        #10 CLOCK = ~CLOCK;
```

```
# Load canceled

VSIM 24> do /data/jbrossar/modelsim_tutorial/Quartus_PRO_PLL_RAM/mentor/mentor_example.do
```

```
Now: 11,030 ns   Delta: 0    sim:/tb/#INITIAL#27    Ln: 46  Col: 0   READ
```

Figure 15. Test bench

## 3.4 ModelSim

ModelSim is a powerful simulator that can be used to simulate the behavior and performance of logic circuits. The simulator allows the user to apply inputs to design circuit, usually referred to as test vectors, and to observe the output generated in response.

ModelSim performs simulation in the context of the projects, i.e., it simulates one project at a time. The project includes the design files that specify the circuit to be generated simulation files for supporting the EDA simulator during compilation of design [29]. Here the Testbench (.VHD) file has to be linked through EDA Tool settings in Quartus Prime software. After successful compilation of the design project, the simulation process generates and launches in ModelSim from within Quartus Prime software. Subsequently, the script file of ModelSim (.do) file is generated, and Quartus Prime software launches ModelSim; then the Testbench (.VHD) file is simulated according to defined specifications in Simulation settings.

The wave window shown in Figure 16 illustrates the results of the simulation.

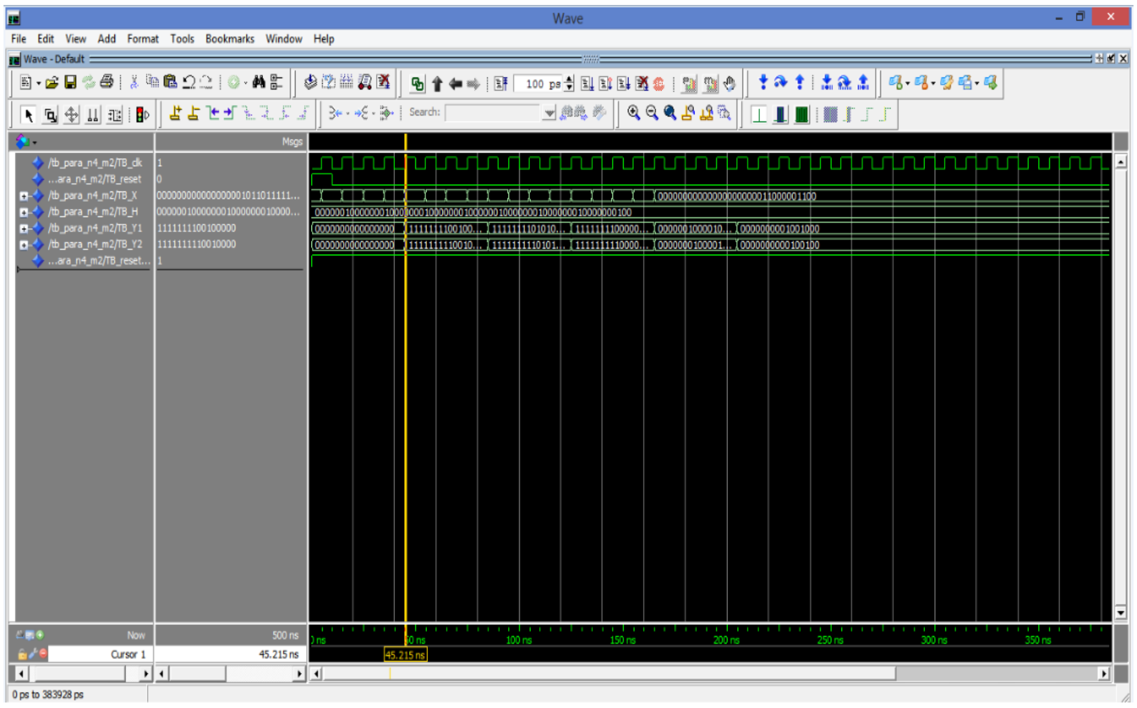

Wave
Wave - Default
/tb_para_n4_m2/TB_clk 1
...ara_n4_m2/TB_reset 0
/tb_para_n4_m2/TB_X 00000000000000001011011111...
/tb_para_n4_m2/TB_H 000000100000001000000010000...
/tb_para_n4_m2/TB_Y1 1111111100100000
/tb_para_n4_m2/TB_Y2 1111111110010000
...ara_n4_m2/TB_reset... 1
Now 500 ns
Cursor 1 45.215 ns
0 ps to 383928 ps

Figure 16. Simulation of full parallel architecture VHDL code waveform in ModelSim.

## 3.5 Power Estimation Analysis

In this thesis, the five different architectures RTL versions have been synthesized in Altera Quartus software and tested through simulation in ModelSim. The power values have been calculated at 100 MHz (for ‘normalized’ comparison purpose) as well as at $F_{max}$ (for maximum performance evaluation)

The following actions were taken to perform the power estimation.

- Each design has been simulated at RTL level with the test bench in order to verify that the functionality of the design was the same as expected.
- In Quartus software, each design has been synthesized which generates annotated VHDL and SDF files that have circuit delays information and generates a script which saves all signals in post-synthesis simulation.
- Then, ModelSim has been configured and used to automatically simulate each design and generate Value Change Dump (VCD) file that stores data of all signal toggled during simulation [30].

- Quartus has then be used to process the VCD file. Finally, Quartus has generated estimation reports of power consumption for each architecture after compiling the data and binding it with the FPGA models.

## 3.6 Calculate Energy

Energy consumption [joules] can be calculated as the product of power and total execution time, i.e., $E = P \times T_{exe}$. The importance of energy consumption reduction in HR monitoring architecture is due to the battery (i.e., energy-limited) driven transmission at each sensor node. Energy has been calculated at 100 MHz and $F_{max}$ for each of the five architectures on the basis of the results obtained in the previous steps. The results are presented in Chapter 4.

Decreasing the energy consumption increases the lifetime of the nodes. However, it is noted that there is a trade-off between energy, power, and execution time. Reducing power benefits in reduction of heat dissipation. This can be achieved by lowering the clock frequency or reducing the level of parallelism, but these approaches result in increased execution time, which in turn may increase energy. The trade-off between these metrics, as well as with the resource usage metrics are discussed in Conclusion.

# 4 Results

This chapter presents the high-level synthesis, power estimates, and energy estimates results of five architectures analyzed using Altera Quartus Prime 15.1 Lite Edition. The energy is estimated over the following five parallelism strategies of architectures:

- Full parallel
- Full parallel without pipeline
- Generic semi-parallel
- Semi-parallel
- Semi-parallel with clock enable.

RTL implementations of these architectures are investigated using Altera Quartus IV for VHDL code synthesis and tested by simulating on ModelSim 10.4b. Table 2 shows the results obtained by synthesizing the VHDL code for the specifications and requirements of different architectures. The objective of this thesis is to evaluate the performance regarding timing analysis, power estimation, and most importantly energy consumption of above architectures. The results are summarized in Table 3 and Table 4. For illustration purpose, the RTL implementations of full parallel with and without pipeline are shown in Figure 15 and Figure 16, respectively.

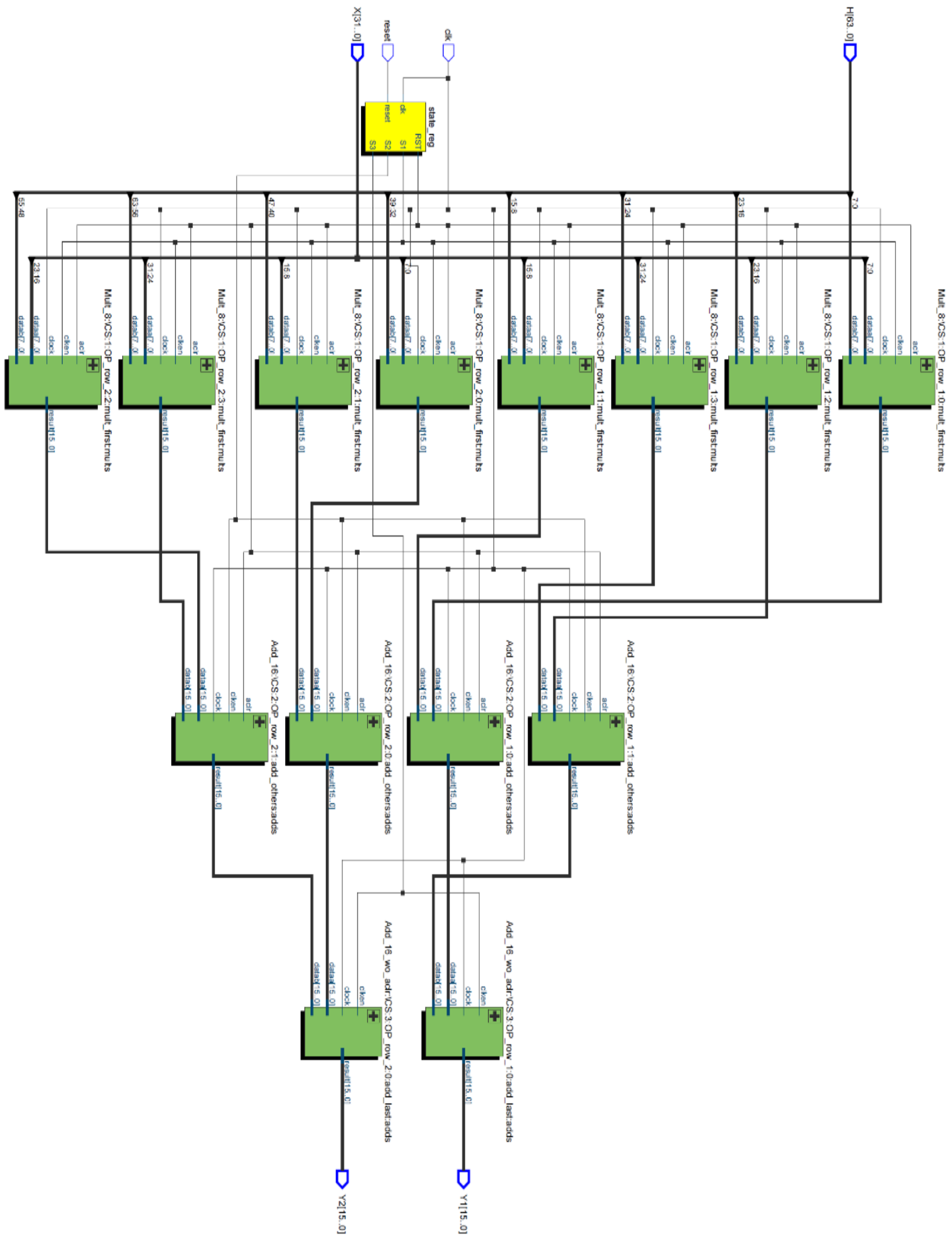


Figure 17.RTL implementation of full-parallel architecture

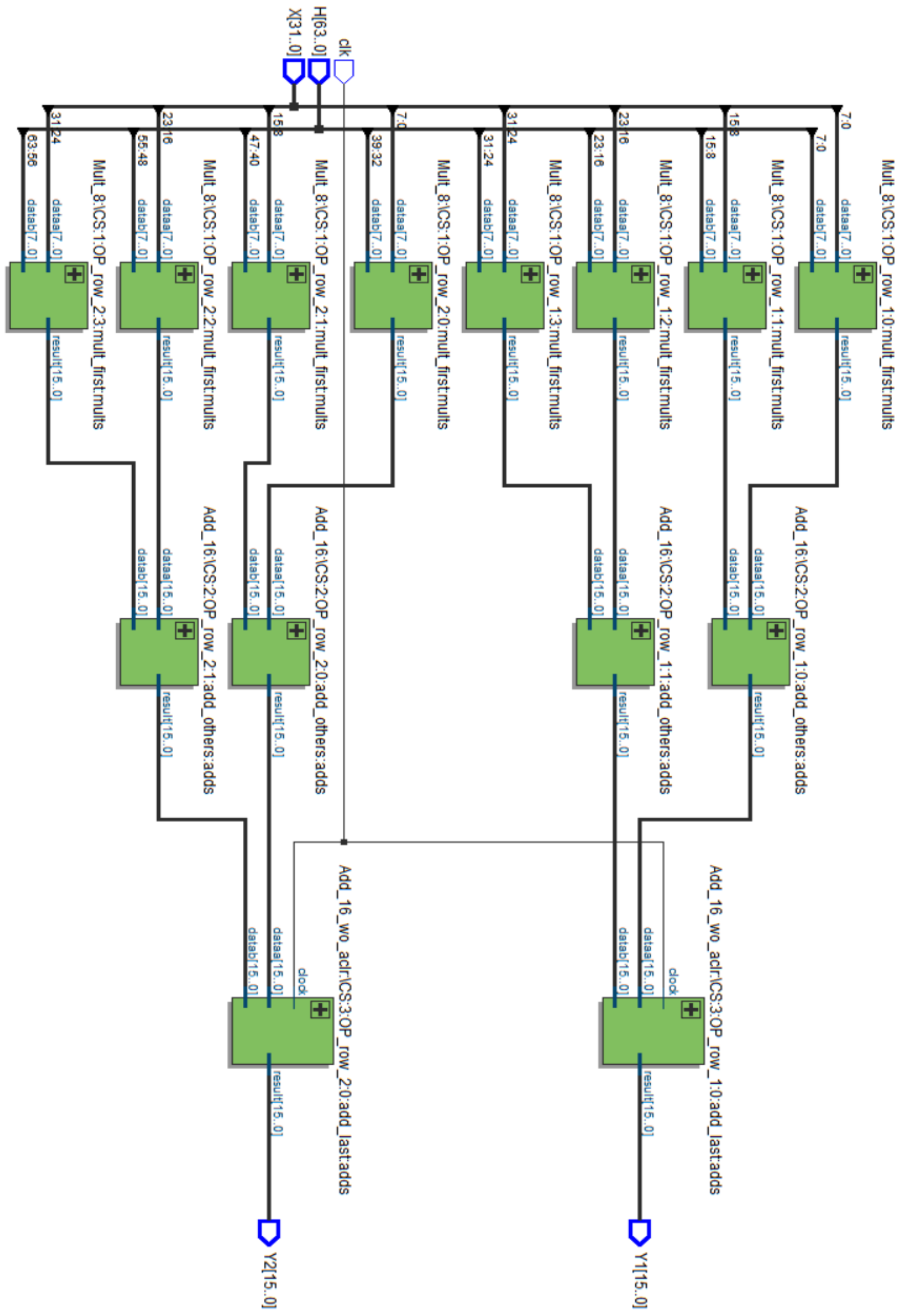
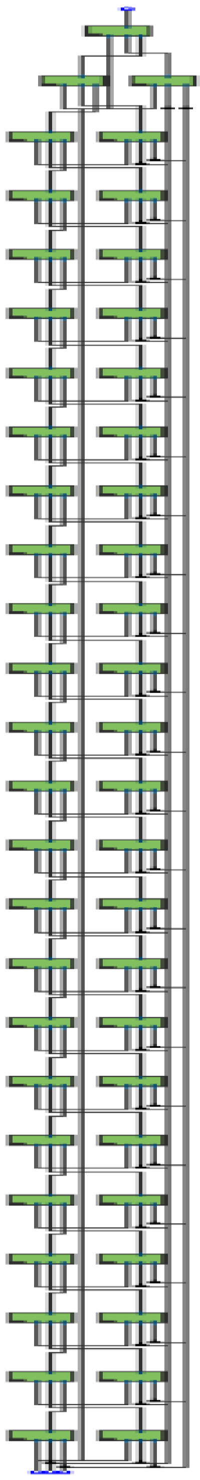

clk
H[63..0]
X[31..0]
Mult_8:\CS:1:OP_row_1:0:mult_first:mults
Mult_8:\CS:1:OP_row_1:1:mult_first:mults
Mult_8:\CS:1:OP_row_1:2:mult_first:mults
Mult_8:\CS:1:OP_row_1:3:mult_first:mults
Mult_8:\CS:1:OP_row_2:0:mult_first:mults
Mult_8:\CS:1:OP_row_2:1:mult_first:mults
Mult_8:\CS:1:OP_row_2:2:mult_first:mults
Mult_8:\CS:1:OP_row_2:3:mult_first:mults
Add_16:\CS:2:OP_row_1:0:add_others:adds
Add_16:\CS:2:OP_row_1:1:add_others:adds
Add_16:\CS:2:OP_row_2:0:add_others:adds
Add_16:\CS:2:OP_row_2:1:add_others:adds
Add_16_wo_adr:\CS:3:OP_row_1:0:add_lastadds
Add_16_wo_adr:\CS:3:OP_row_2:0:add_lastadds
Y1[15..0]
Y2[15..0]

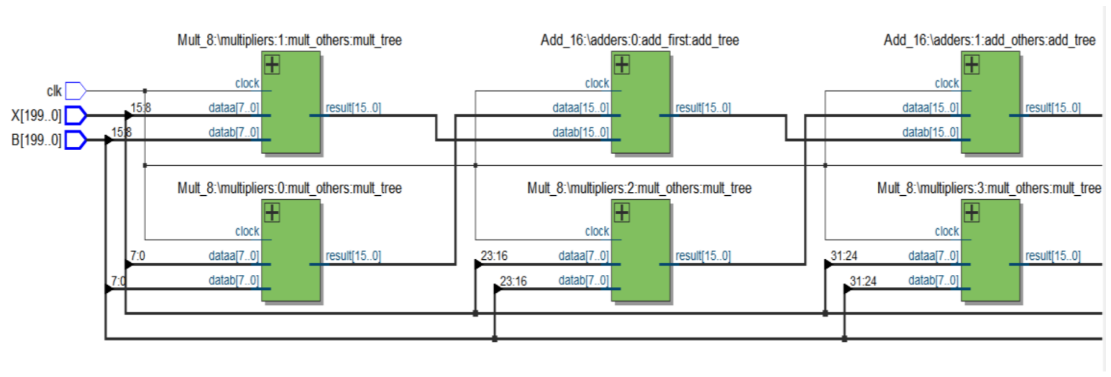
Figure 18. RTL implementation of full-parallel without pipeline.

Table 3. Synthesis results for architectures.

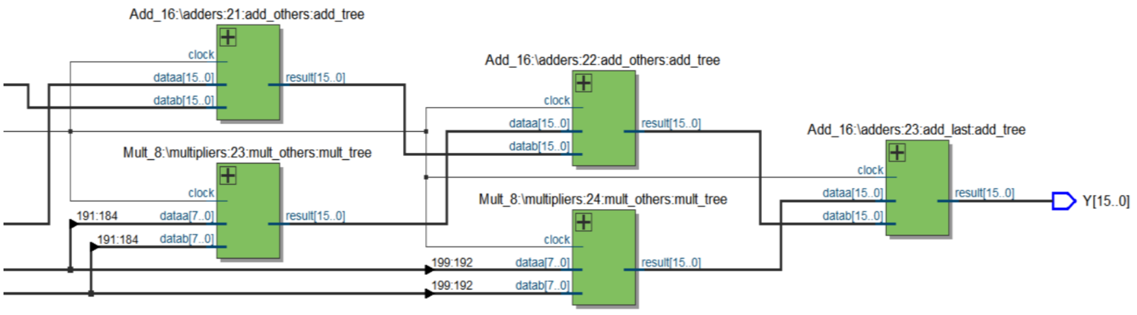
| Architecture | Full Parallel | Full Parallel without Pipeline | Generic Semi Parallel | Semi_Parallel | Semi Parallel_clk |
|---|---|---|---|---|---|
| Top-level Entity Name | para_4_2 | Para_4_2_wo_pipe | Mult_Add_example_1 | seq_8 | Seq_8_clken |
| Family | Cyclone IV E | Cyclone IV E | Cyclone IV E | Cyclone IV E | Cyclone IV E |
| Total Logic elements | 100 | 96 | 384 | 101 | 103 |
| Total combinational functions | 98 | 96 | 384 | 98 | 98 |
| Dedicated logic registers | 100 | 32 | 384 | 101 | 101 |
| Total registers | 100 | 32 | 384 | 101 | 101 |
| Total I/O pins | 130 | 129 | 92 | 129 | 180 |
| Embedded multipliers 9-bit elements | 8 | 8 | 25 | 8 | 8 |
| CS Stage Level | 4 | 1 | 25 | 4 | 5 |
| $F_{max}$ clk frequency [MHz] | 298.15 | 116.9 | 234.8 | 230.68 | 326.69 |
| $Time_{(Exe)}$ (ns) @Fmax | 13.2 (4 Stages) | 8.55 (1 Stage) | 106.47 (25 Stages) | 17.34 (4 Stages) | 15.305 (5 Stages) |
| Operating Frequency [MHz] | 100 | 100 | 100 | 100 | 100 |
| Operating $Time_{(Exe)}$ (nS) | 40 | 10 | 250 | 40 | 50 |

The number of total logic elements in all architectures is relatively similar except the generic semi-parallel architecture that consists of 384 logic elements [16]. This is because the generic semi-parallel architecture requires more functional block to process data, as shown in Figure 19.


(a)


Mult_8:\multipliers:1:mult_others:mult_tree
Add_16:\adders:0:add_first:add_tree
Add_16:\adders:1:add_others:add_tree
Mult_8:\multipliers:0:mult_others:mult_tree
Mult_8:\multipliers:2:mult_others:mult_tree
Mult_8:\multipliers:3:mult_others:mult_tree
clk
X[199..0]
B[199..0]
clock
dataa[7..0]
datab[7..0]
dataa[15..0]
datab[15..0]
result[15..0]
15:8
7:0
23:16
31:24

(b)


Add_16:\adders:21:add_others:add_tree
Mult_8:\multipliers:23:mult_others:mult_tree
Add_16:\adders:22:add_others:add_tree
Mult_8:\multipliers:24:mult_others:mult_tree
Add_16:\adders:23:add_last:add_tree
clock
dataa[15..0]
datab[15..0]
dataa[7..0]
datab[7..0]
result[15..0]
191:184
199:192
Y[15..0]

(c)

Figure 19. (a) RTL implementation of generic semi-parallel architecture. (b) Input side of Generic semi-parallel architecture. (c) Output side of Generic semi-parallel architecture.

It is observed that the numbers of total logic elements and combinational functions are relatively similar for full parallel with and without pipeline, and semi-parallel with and without clock architecture due to least variation in their design structure. Given the internal sequential design of generic semi-parallel architecture, the number of embedded multipliers is higher for the generic semi-parallel architecture (i.e., 25) compared to the other remaining architectures (i.e., 8). Given that the full parallel architecture with pipeline decomposes the large combinational blocks into small blocks, it can reach a higher maximum frequency ($F_{max}$) of 298.15 MHz in comparison to the 116.9 MHz of full parallel without pipeline, as shown in Table 3.

Based on the total execution time $T_{exe}$ calculated at $F_{max}$, and operating frequency for each architecture, the full parallel without pipeline comparatively achieves the best execution times of 8.55 ns and 10 ns at $F_{max}$ (116.9 MHz) and operating frequency (100 MHz), respectively. Since the generic semi-parallel architecture consists of 25 stages of combinational function blocks, it results in the highest execution time of 106.47 ns and 250 ns at $F_{max}$ (234.8 MHz) and operating frequency (100 MHz), respectively.

Looking closer to the frequency and total execution time for each architecture, it can be seen that the total execution time and the throughput do not always increase or descrease with each other. Indeed, the execution time corresponds to how long it takes for a given input data to traverse the design (i.e., its processing time), whereas the throughput corresponds to how many many input and output samples enter and exit the design per time unit. For pipelined architectures, the number of required clock cycles to process one data increases with the number of stages; however, this not only allows increasing the frequency (since each stage is shorter), but also allows starting processing the next incoming data before the previous one has exited the pipeline, thereby enabling a form of parallelism that increases throughput.

Typically, the frequency is limited by the slowest block in the pipeline. When the number of the pipeline stages increases, the frequency can also be increased (since each stage becomes shorter) as can be seen from Table 3. For instance, the frequency of the full parallel architecture with pipeline (consisting of 4 stages) is 298.15 MHz, which is comparatively higher in comparison to 116.9 MHz for the architecture without pipeline consisting of 1 stage only.

However, the total execution time for input one data is higher with 13.2 ns instead of 8.55 ns. This fact makes it clear understanding that pipeling increases the throughput, i.e., the number of samples processed per unit time, but as shown in this case it can be that the latency, i.e., the delay is increased. Choosing between the pipelined and non- pipelined version is a trade-off. If the application requires higher throughput, then the pipelined version is the best option and vice versa if the application requires low latency.

Table 4 shows that the power result is best (i.e., lowest) for the generic semi-parallel architecture using the balanced optimization mode at operating and maximum frequency.

Despite using higher resources, the architecture design and execution of data concludes in lower power consumption among the architectures.

Table 4. Power estimation for design architectures

| | Architectures | | | | |
|---|---|---|---|---|---|
| | **Full parallel** | **Full-Parallel w/o Pipeline** | **Generic Semi Parallel** | **Semi parallel** | **Semi-Parallel clken** |
| **Optimization Mode** | Balanced | Balanced | Balanced | Balanced | Balanced |
| **Total Thermal Power Dissipation @100 MHz** | 85.97 mW | 83.31 mW | 69.98 mW | 85.43mW | 92.24mW |
| **Total Thermal Power Dissipation @ Fmax** | 74.74mW | 101.66 mW | 70.9mW | 95.27mW | 84.79mW |

The results for energy estimation is showed in Table 5. The energy at operating frequency 100MHz is used to compare the efficiency of the architectures. The estimated energy at the maximum frequency is referred to as the maximum performance of the architecture.

Table 5. Energy estimation for design architecture

| | Architectures | | | | |
|---|---|---|---|---|---|
| | **Full parallel** | **Full-Parallel w/o Pipeline** | **Generic Semi Parallel** | **Semi parallel** | **Semi-Parallel clken** |
| Optimization Mode | Balanced | Balanced | Balanced | Balanced | Balanced |
| Energy@100 MHz | 3.438nJ | 0.833 nJ | 17.495 nJ | 3.417 nJ | 0.461 nJ |
| Energy @Fmax | 0.986nJ | 0.869nJ | 0.751nJ | 1.654nJ | 1.297nJ |

## 4.1 Further comparisons of the architectures

The primary focus of this thesis is to estimate the energy. However, there are also other critical design metrics such as a number of resource usage, total execution time and power, which have been examined in this thesis. These factors complicate the comparison for energy estimation. Although comparison of energy itself is relatively straightforward, it is harder when considering the whole system parameters because maybe the most energy economical architecture might have the highest resource usage or even higher power consumption (an issue for heat dissipation) with higher execution time

Thus, in what follows, we compare the five architectures taking into account more than one metric, not only energy. The comparison gets more complicated because now we examine the design in multiple dimensions (Energy, Power, Time and Resources).

### 4.1.1 Comparative analysis of Energy with respect to Resources

Figure 19 illustrates the comparative analysis of architectures regarding resources and energy at operating and maximum frequencies. The comparison reveals that the full parallel without pipeline shows the lowest resources with energy efficiency, and the generic semi-parallel depicted more energy-hungry architecture with higher resource usage. In full parallel without pipeline, the total number of registers is lower than with the pipeline. It means that the pipeline uses more resources to execute data in parallel (a new data can start being processed before the previous one has exited the pipeline), instead of data to execute without pipeline. If for instance, the requirement of the system is focused on resource usage [LUTs], the designer can choose among the best architectures regarding resource usage.

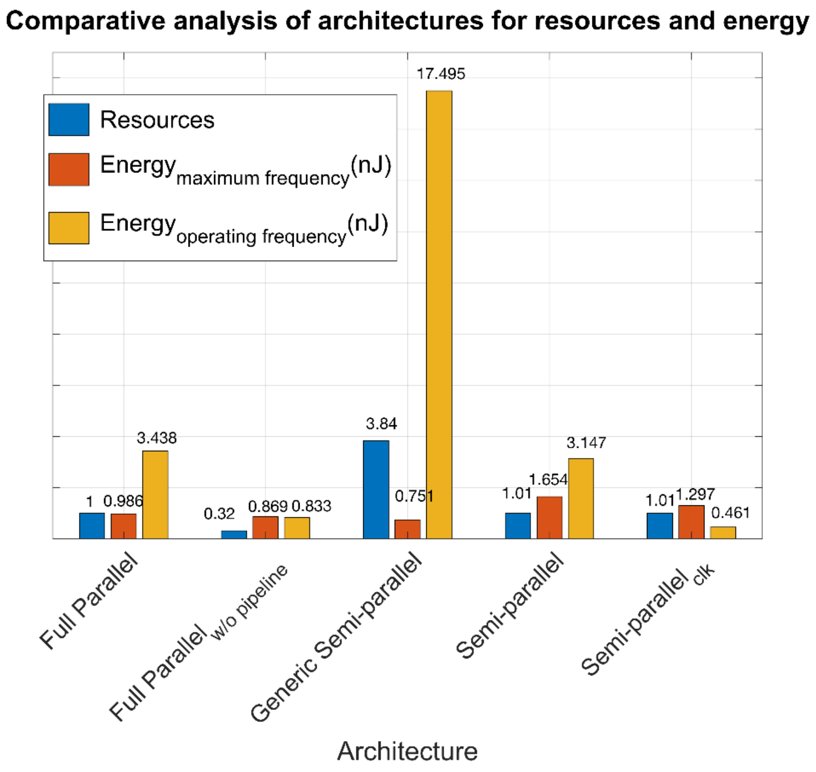


Figure 20. Comparison between resources and energy at maximum and operating frequencies.

## 4.1.2 Comparative analysis of Energy with respect to Execution Time

The comparison regarding energy and execution time for operating and maximum frequencies is shown in Figure 20. Note that the total execution time is divided by 100 to fit in the values in graph illustration. The graph indicates that if we choose energy estimation concerning total execution time at operating and maximum frequency, then full parallel is faster among five architectures. The results for generic semi-parallel shows the slower performance with more energy consumption. This fact can be explained by the organization of the architecture design on which generic semi-parallel is modeled. It is perhaps an unfortunate trade-off since it uses more functional blocks to execute the data while execution is rather slow and with reasonably high energy consumption. In the semi-parallel and semi-parallel with clock enable architectures, the numbers for execution times are nearly alike, but energy values shows different results. Semi-parallel architectures show higher energy consumption on both operating and maximum frequencies.

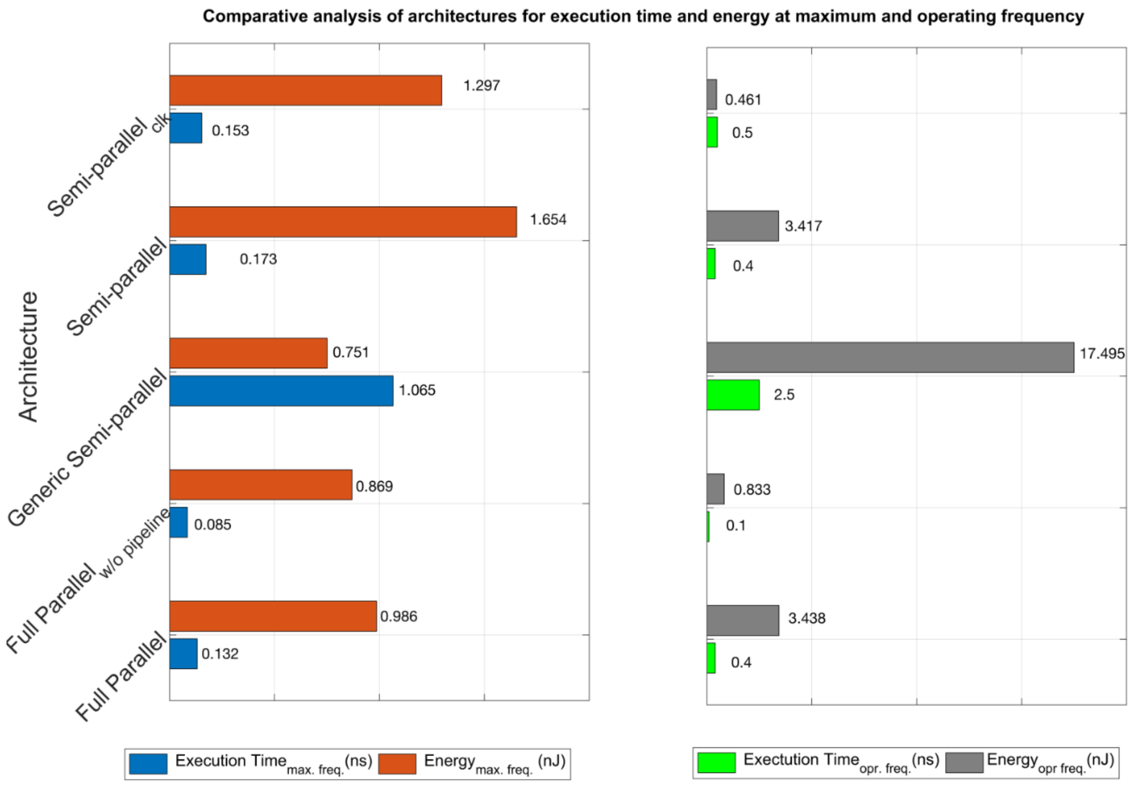


Figure 21. Comparison between total execution time and energy at maximum and operating frequencies.

### 4.1.3 Comparative analysis of Energy with respect to Power

The plot shown in Figure 22 is a comparative analysis in terms of the energy and power for operating and maximum frequencies. The graph indicates that if we choose energy estimation against power consumption values, then the generic semi-parallel architecture shows the best results at $F_{max.}$ At the same time, power numbers remain similar at operating frequency, but energy value changes dramatically. This is due to the total execution time that the generic semi-parallel design takes to process the data. Full parallel and semi-parallel architectures show similar values for power consumption (at 100 MHz and $F_{max}$), but energy consumption decreases significantly. Full parallel without pipeline shows equal number for energy (at 100 MHz and $F_{max}$), but power consumption increases at $F_{max}$ because of the higher frequency.

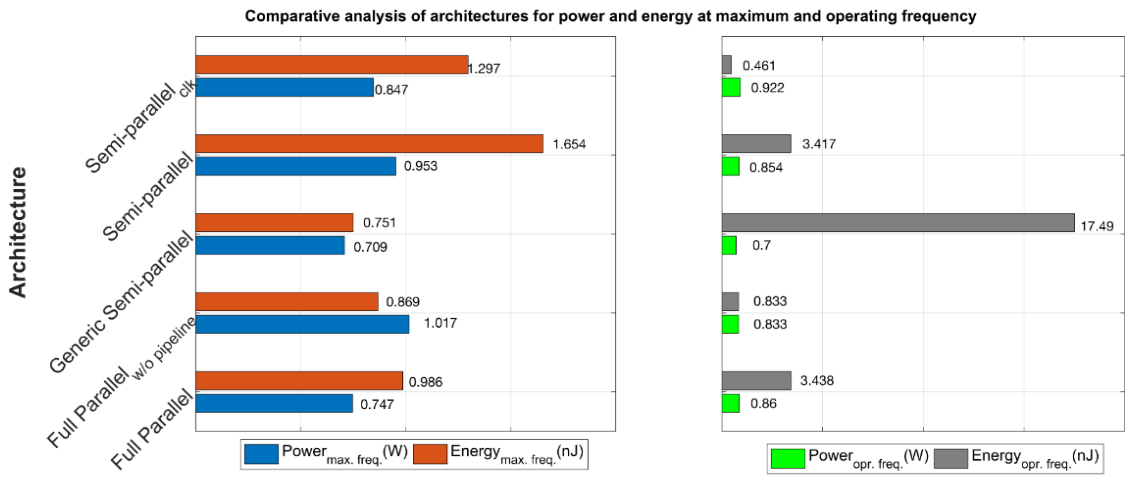


Figure 22. Comparison between power and energy at maximum and operating frequencies

## 4.2 Comments on Comparison

Finally, in light of the comparisons mentioned above, the system designer has to make a trade-off between resources, execution time, power and energy consumption since none of the architecture is a definite 'winner' when all design metrics are considered simultaneously

The full parallel without pipeline architecture demands less resource usage, with twice less total execution time than full parallel architecture. Energy estimated for both systems are nearly equal.

Based on the results, the generic semi-parallel architecture gives the lowest energy consumption values at a maximum frequency; but accompanying this are higher resources and execution time.

# 5 Conclusion

The title of the thesis is "Energy Estimation of FPGA Architectures for a Compressed Sensing Engine" with the research statement as below:

*To explore the trade-off between execution time, resource usage, power consumption, and most importantly energy consumption among the five architectures proposed in the previous studies* [15]-[16].

In this thesis, several tasks have been performed to address the aforementioned research statement. To support this statement firstly, the existing system for WSN has been analyzed. A brief explanation about the extraction of QRS from HRM and how CS can be executed to minimize transmitting data was given. Then the existing architectures for the CS engine were studied in detail to investigate the timing analysis for parallelism within designed architectures via Altera TimeQuest timing analyzer tool. Besides power estimation methods illustrated in [16], this thesis has provided the missing power values calculated for other architectures in Altera power analysis tool. With the values gathered from power and execution time, energy has been calculated for the five architectures. Finally, the energy estimation results have been presented and discussed for maximum and operating frequencies.

Based on the results presented in this thesis, the following findings can be drawn:

- The full parallel architecture with pipeline results in higher maximum frequency without affecting the power and energy consumption at an increased number of registers (LUTs).
- Lowest energy consumption has been observed for the generic semi-parallel architecture at $F_{max}$. However, the architecture has high resource usage and is slowest among other architectures.
- Full parallel without pipeline architectures shows the best results regarding resource usage, execution time and energy.

It is somewhat difficult to state a clear winner for the CS engine when all parameters are considered. However, one striking conclusion that can be drawn is that the best solution for energy estimation is in correlation with architectural frequencies. It implies that the generic semi-parallel architecture is found as the best solution at $F_{max}$, whereas the full parallel architecture without pipeline shows the best results at operating frequency of 100 MHz. Therefore, designers can select a similar architecture based on the requirement of the system.

Given that the energy results have been obtained, and that they have been compared for the five architectures, including in relation to the other design metrics, it can be said that the research statement has been successfully considered in this thesis.

## 5.1 Future work

This subsection briefly presents some ideas for future work that could extend this thesis. Firstly, other target architectural platforms could be considered for estimating the performance regarding execution time, power, and energy consumption. For example, the CS algorithm used in this thesis could be implemented onto other low power FPGAs, DSP processors, or possibly modern embedded GPP featuring some parallelism (e.g., multicore and multithreading).

Secondly, one other possible natural continuation of this work could be to include and evaluate the feasibility to use energy harvesting. This would open new questions such as how to evaluate whether the harvested energy can provide sufficient power for such sensor nodes depending on whether energy harvesting is used to complement or even replace the battery. In the latter case, approaches such as transient computing (implemented on a non-volatile device such as a FRAM-based microcontroller or flash-based FPGA) could provide a suitable mechanism to minimize the impact of power losses when not enough energy is harvested.

## References

[1] P. Demeester, ‘A Survey on Wireless Body Area Networks’, pp. 1–18.

[2] J. Achten and A. E. Jeukendrup, ‘Heart Rate Monitoring Applications and Limitations’, vol. 33, no. 7, pp. 517–538, 2003.

[3] C. Otto and  a Milenkovic, ‘System architecture of a wireless body area sensor network for ubiquitous health monitoring’, *J. Mob. …*, vol. 1, no. 4, pp. 307–326, 2006.

[4] I. P. Panidis and J. Morganroth, ‘Sudden Death in Hospitalized Patients : Cardiac Rhythm Disturbances Detected by Ambulatory Electrocardiographic Monitoring’, *J. Am. Coll. Cardiol.*, vol. 2, no. 5, pp. 798–805, 1983.

[5] R. Kaur, ‘Wireless Body Area Network & ITS APPLICATION’, *Res. Cell An Int. J. Eng. Sci.*, vol. 1, pp. 2229–6913, 2011.

[6] K. C. Chua, V. Chandran, U. R. Acharya, and C. M. Lim, ‘Cardiac state diagnosis using higher order spectra of heart rate variability’, vol. 32, no. 2, pp. 145–155, 2008.

[7] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, ‘A survey on sensor networks’, *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 102–105, 2002.

[8] N. Yenuganti, ‘Authentication in Wireless Body Area Networks ( WBAN )’, no. June, 2016.

[9] I. S. T. He and W. M. Esh, ‘a Ccepted From O Pen C All’, *Ieee Wirel. Commun.*, no. February, pp. 104–111, 2010.

[10] E. Farella, A. Pieracci, L. Benini, L. Rocchi, and A. Acquaviva, ‘Interfacing human and computer with wireless body area sensor networks: The WiMoCA solution’, *Multimed. Tools Appl.*, vol. 38, no. 3, pp. 337–363, 2008.

[11] S. Choi, R. Scrofano, V. K. Prasanna, and J.-W. Jang, ‘Energy-efficient signal processing using FPGAs’, *Proc. 2003 ACM/SIGDA Elev. Int. Symp. F. Program. gate arrays - FPGA ’03*, pp. 225–234, 2003.

[12] R. Joaquinito and H. Sarmento, ‘A wireless biosignal measurement system using a SoC FPGA and Bluetooth Low Energy’, *IEEE Int. Conf. Consum. Electron. - Berlin, ICCE-Berlin*, vol. 2016–Octob, pp. 36–40, 2016.

[13] E. Systems and P. D. Atienza, ‘A Real-Time Compressed Sensing-Based Personal Electrocardiogram Monitoring System’, p. 2.

[14] E. J. Candes and M. B. Wakin, ‘An Introduction To Compressive Sampling’, *IEEE Signal Process. Mag.*, vol. 25, no. 2, pp. 21–30, 2008.

[15] M.El-Sayed and Soeren, ‘An FPGA-friendly Compressed Sampling Engine for WSN-based Heart Rate Monitoring', Report, Aalborg University, 2015.

[16] E. Iee, L. T. Arbind, K. Rimal, and T. Supervisor, ‘POWER ESTIMATION OF FPGA HEART-RATE MONITORING SYSTEM’, 2016.

[17] C. K. S. Kumar, ‘FPGA Implementation for Energy Efficiency in Secure Wireless Sensor Node – A Critical Review Abstract : Research Article February’, no. February, pp. 46–51, 2013.

[18] M. K. Das, ‘Electrocardiogram signal analysis for heartbeat pattern classification’, 2015.

[19] U. Rajendra Acharya, J. S. Suri, J. A. E. Spaan, and S. M. Krishnan, *Advances in cardiac signal processing*. 2007.

[20] R. M. Rangayyan, *Biomedical Signal Analysis*. 2001.

[21] Opalkelly, ‘ZEM4310’, https://www.opalkelly.com/products/zem4310/, 2018.

[22] I. Kuon, R. Tessier, and J. Rose, ‘FPGA Architecture: Survey and Challenges’, *Found. Trends® Electron. Des. Autom.*, vol. 2, no. 2, pp. 135–253, 2007.

[23] Altera, ‘Cyclone IV Device Handbook’, vol. 1, p. 382, 2010.

[24] O. Faust, U. R. Acharya, J. Ma, L. C. Min, and T. Tamura, ‘Compressed sampling for heart rate monitoring’, *Comput. Methods Programs Biomed.*, vol. 108, no. 3, pp. 1191–1198, 2012.

[25] T. T. Analyzer, A. Quartus, I. I. Cad, Q. Ii, U. Timequest, S. Up, and T. Constraints, ‘Using TimeQuest Timing Analyzer’, *Program*, no. May, pp. 1–12, 2011.

[26] A. Corporation, ‘Timing Analysis Overview’, 2014.

[27] S. Feedback and S. Jose, ‘Quartus Prime Standard Edition Handbook Volume 3: Verification’, vol. 3, 2016.

[28] I. Quartus and P. Standard, ‘Simulation Quick-Start for ModelSim * - Intel ® FPGA Edition’, 2017.

[29] Y. S. Kung, J. M. Lin, Y. J. Chen, and H. H. Chou, ‘ModelSim/Simulink Cosimulation and FPGA Realization of a Multiaxis Motion Controller’, *Math. Probl. Eng.*, vol. 2015, 2015.

[30] A. Corporation, ‘PowerPlay Early Power Estimator User Guide Subscribe Send Feedback’, 2015.

# Appendix 1

This appendix contains the VHDL code of test bench used to simulate the full parallel architecture which is presented in Chapter 4.

------------------- Test-Bench VHDL code for full_para_N2_M4 at F$_{max}$--------------------------------

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use std.textio.all; --to read from and write to files
entity TB_para_N4_M2 is
--Testbench, thus no ports
end entity TB_para_N4_M2;
architecture MyTB of TB_para_N4_M2 is
component para_4_2 is
generic (
M: integer := 2;          -- 2^M   Number of And's
b_m: integer := 8
);
port (
clk:            in STD_LOGIC;
reset:   in     std_logic := '0';
X:              in      STD_LOGIC_VECTOR(b_m*(2**M)-1      downto      0)      :=
"00000010000000100000001000000010";
H:              in      STD_LOGIC_VECTOR(b_m*(2**M)*2-1      downto      0)      :=
"0000001000000010000000100000001000000010000000100000001000000100";
Y1:             out STD_LOGIC_VECTOR((b_m*2)-1 downto 0);
Y2:             out STD_LOGIC_VECTOR((b_m*2)-1 downto 0)
);
end component para_4_2;
```

```
constant M: integer := 2;        -- 2^M   Number of And's
constant b_m: integer := 8;
signal TB_clk : std_logic := '0';
signal TB_reset : std_logic := '0';
signal     TB_X     :     STD_LOGIC_VECTOR(b_m*(2**M)-1     downto     0)     :=
"00000010000000100000001000000010";
signal     TB_H:     STD_LOGIC_VECTOR(b_m*(2**M)*2-1     downto     0)     :=
"00000010000000010000000100000001000000010000000100000001000000100";
signal TB_Y1: STD_LOGIC_VECTOR((b_m*2)-1 downto 0);
signal TB_Y2: STD_LOGIC_VECTOR((b_m*2)-1 downto 0);
signal TB_reset_done : std_logic := '0';
-- Change this to control the clock frequency !!!
constant clk_period : time := 3.35 ns; -- 298.15 MHZ, as in .sdc file
begin
uut: para_4_2 PORT MAP (
--M => TB_M,
--b_m => TB_b_m,
clk => TB_clk,
reset => TB_reset,
X => TB_X,
H => TB_H,
Y1 => TB_Y1,
Y2 => TB_Y2
);
-- Reset process
TB_para_N4_M2_GEN_RESET: process (TB_clk) is
begin
-- Change this to control reset duration
if (TB_reset_done = '0') then
TB_reset <= '1', '0' after 10 ns;
end if;
TB_reset_done <= '1';     --not clean style
end process TB_para_N4_M2_GEN_RESET;
-- Clock process definitions(clock with 50% duty cycle)
```

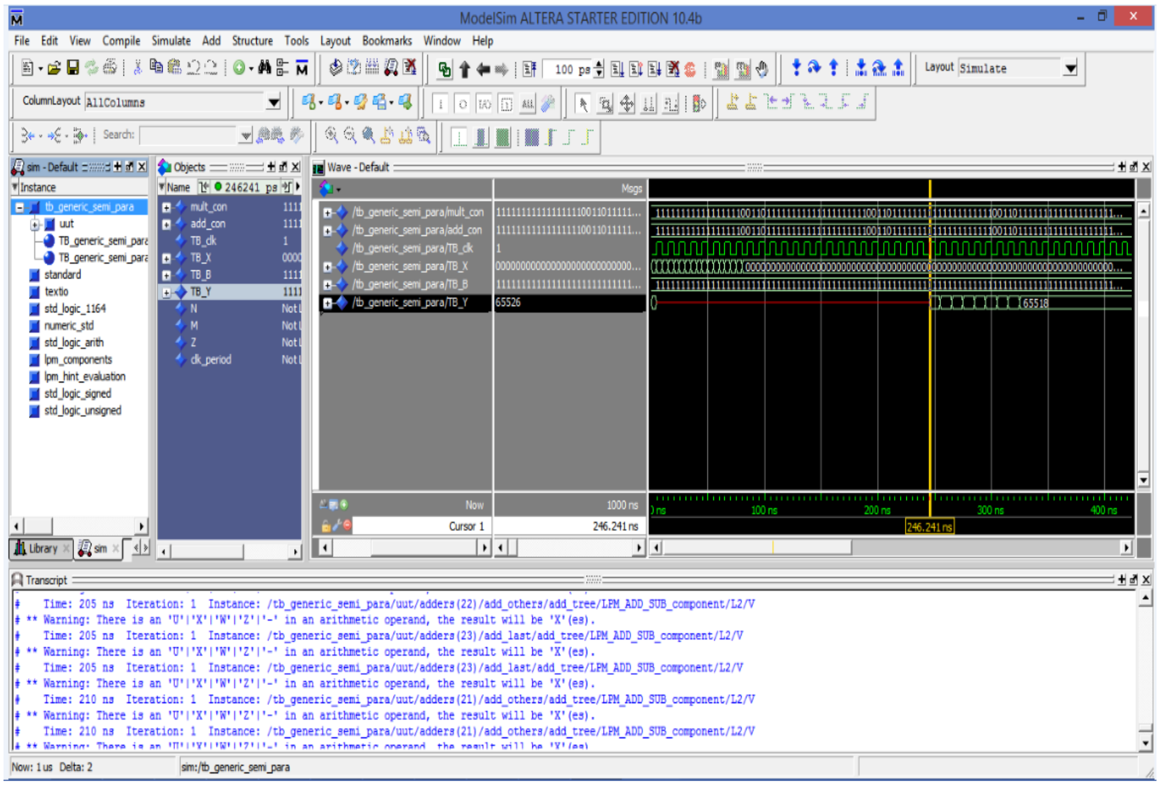
```
TB_para_N4_M2_CLK: process
begin
TB_clk <= '0';
wait for clk_period/2;
TB_clk <= '1';
wait for clk_period/2;
end process TB_para_N4_M2_CLK;
-- Stimuli process
TB_para_N4_M2_GEN_STIMULI: process (TB_clk)
file infile : text is in "C:\Users\syedjawad\Documents\thesis
work\3.VHDLcode\full_para_N4_M2\ECG.txt";   --declare input file  --path needed?
variable inline : line; --line number declaration
variable dataread : integer;
begin
if (TB_clk = '1' and TB_clk'event and TB_reset_done = '1') then
if (not endfile(infile)) then   --checking the "END OF FILE" is not reached.
readline(infile, inline);       --reading a line from the file.
read(inline, dataread); --reading the data from the line and putting it in an integer type
variable.
TB_X <= std_logic_vector(to_unsigned(dataread, 32));
end if;
end if;
end process TB_para_N4_M2_GEN_STIMULI;
end architecture MyTB;
```

--------------Test-Bench VHDL code for semi_para_N4_M2_w_clken at F$_{max}$ -------------

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use std.textio.all; --to read from and write to files
entity TB_semi_para_N4_M2_w_clken is
--Testbench, thus no ports
end entity TB_semi_para_N4_M2_w_clken;
architecture MyTB of TB_semi_para_N4_M2_w_clken is
component seq_8_clken is
        generic (
        N: integer := 4;            -- 2^M   Number of And's
        b_m: integer := 8
        );

        port (
        clk:            in STD_LOGIC;
        reset:   in     std_logic := '0';
        X:              in STD_LOGIC_VECTOR(b_m*N-1 downto 0) :=
"00000010000000100000001000000010";    -- Vector Input

        H:              in STD_LOGIC_VECTOR(b_m*N*2-1 downto 0) :=
"0000000100000001000000011000000100000000100000001000000001000000010";
        -- Matrix Input
        Y1:             out STD_LOGIC_VECTOR((b_m*2)-1 downto 0);
        Y2:             out STD_LOGIC_VECTOR((b_m*2)-1 downto 0)
        );
end component seq_8_clken;
constant N: integer := 4;          -- 2^M   Number of And's
constant b_m: integer := 8;
signal TB_clk : std_logic := '0';
signal TB_reset : std_logic := '0';
signal TB_X : STD_LOGIC_VECTOR(b_m*N-1 downto 0) :=
"00000010000000100000001000000010";    -- Vector Input
signal TB_H: STD_LOGIC_VECTOR(b_m*N*2-1 downto 0) :=
"0000000100000010000000110000010000000010000000100000001000000010";
        -- Matrix Input
signal TB_Y1: STD_LOGIC_VECTOR((b_m*2)-1 downto 0);
signal TB_Y2: STD_LOGIC_VECTOR((b_m*2)-1 downto 0);
signal TB_reset_done : std_logic := '0';
-- Change this to control the clock frequency !!!
constant clk_period : time := 3.061 ns; --326.69 MHZ, as in .sdc file
begin
        uut: seq_8_clken PORT MAP (
                --M => TB_M,
                --b_m => TB_b_m,
                clk => TB_clk,
                reset => TB_reset,
                X => TB_X,
                H => TB_H,
                Y1 => TB_Y1,
                Y2 => TB_Y2
                );
-- Reset process
        TB_semi_para_N4_M2_w_clken_GEN_RESET: process (TB_clk) is
        begin
             -- Change this to control reset duration
                if (TB_reset_done = '0') then
                TB_reset <= '1', '0' after 3.061 ns;
    end if;
        TB_reset_done <= '1';                  --not clean style
        end process TB_semi_para_N4_M2_w_clken_GEN_RESET;
-- Clock process definitions(clock with 50% duty cycle)
  TB_semi_para_N4_M2_w_clken_CLK: process
  begin
                TB_clk <= '0';
    wait for clk_period/2;
    TB_clk <= '1';
                wait for clk_period/2;
  end process TB_semi_para_N4_M2_w_clken_CLK;
-- Stimuli process
  TB_semi_para_N4_M2_w_clken_GEN_STIMULI: process (TB_clk)
  file infile : text is in  "C:\Users\syedjawad\Documents\thesis
work\3.VHDLcode\semi_para_N4_M2_w_clken\ECG.txt";   --declare input file  --path
needed?
  variable inline : line; --line number declaration
  variable dataread : integer;
        begin
                if (TB_clk = '1' and TB_clk'event and TB_reset_done = '1') then
if (not endfile(infile))
then   --checking the "END OF FILE" is not reached.
readline(infile, inline);       --reading a line from the file.
read(inline, dataread); --reading the data from the line and putting it in an integer type
variable.
TB_X <= std_logic_vector(to_unsigned(dataread, 32));
end if;
end if;
end process TB_semi_para_N4_M2_w_clken_GEN_STIMULI;
end architecture MyTB;

%%%%%%%%%%%%%% End of Test bench file  %%%%%%%%%%%%%%%
```

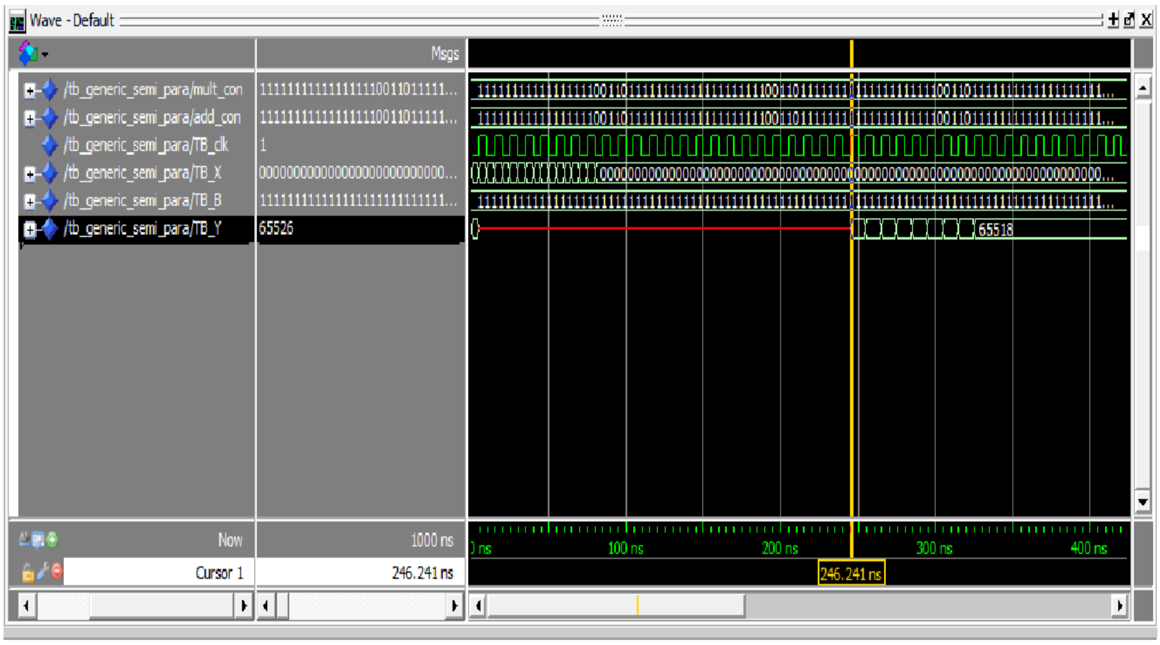
# Appendix 2

The following screenshot shows the output waveform for the generic semi parallel architecture.



```
# Time: 205 ns  Iteration: 1  Instance: /tb_generic_semi_para/uut/adders(22)/add_others/add_tree/LPM_ADD_SUB_component/L2/V
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 205 ns  Iteration: 1  Instance: /tb_generic_semi_para/uut/adders(23)/add_last/add_tree/LPM_ADD_SUB_component/L2/V
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 205 ns  Iteration: 1  Instance: /tb_generic_semi_para/uut/adders(23)/add_last/add_tree/LPM_ADD_SUB_component/L2/V
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 210 ns  Iteration: 1  Instance: /tb_generic_semi_para/uut/adders(21)/add_others/add_tree/LPM_ADD_SUB_component/L2/V
# ** Warning: There is an 'U'|'X'|'W'|'Z'|'-' in an arithmetic operand, the result will be 'X'(es).
#    Time: 210 ns  Iteration: 1  Instance: /tb_generic_semi_para/uut/adders(21)/add_others/add_tree/LPM_ADD_SUB_component/L2/V
```

Now: 1 us  Delta: 2    sim:/tb_generic_semi_para



```
/tb_generic_semi_para/mult_con   11111111111111111100110111111...
/tb_generic_semi_para/add_con    11111111111111111100110111111...
/tb_generic_semi_para/TB_clk     1
/tb_generic_semi_para/TB_X       00000000000000000000000000000...
/tb_generic_semi_para/TB_B       11111111111111111111111111111...
/tb_generic_semi_para/TB_Y       65526
Now                              1000 ns
Cursor 1                         246.241 ns
```