ASIC-IMPLEMENTED MICROBLAZE-BASED COPROCESSOR FOR DATA STREAM MANAGEMENT SYSTEMS

A Thesis

Submitted to the Faculty

of

Purdue University

by

Linknath Surya Balasubramanian

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science in Electrical and Computer Engineering

May 2020

Purdue University

Indianapolis, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF THESIS APPROVAL

Dr. John J. Lee, Chair

Department of Electrical and Computer Engineering

Dr. Lauren A. Christopher

Department of Electrical and Computer Engineering

Dr. Maher E. Rizkalla

Department of Electrical and Computer Engineering

Approved by:

Dr. Brian King

Head of Graduate Program

ACKNOWLEDGMENTS

I would first like to express my gratitude to my advisor Dr. John J. Lee and my thesis committee members Dr. Lauren A. Christopher and Dr. Maher E. Rizkalla for their patience, guidance, and support during this journey. I would also like to thank Mrs. Sherrie Tucker for her patience, help, and encouragement. Lastly, I must thank Dr. Pranav Vaidya and Mr. Tareq S. Alqaisi for all their support, technical guidance, and advice. Thank you all for taking time and helping me complete this study.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

ABSTRACT

Balasubramanian Linknath Surya. M.S.E.C.E., Purdue University, May 2020. ASIC-Implemented MicroBlaze-based Coprocessor for Data Stream Management Systems. Major Professor: John J. Lee.

The drastic increase in Internet usage demands the need for processing data in real time with higher efficiency than ever before. Symbiote Coprocessor Unit (SCU), developed by Dr. Pranav Vaidya, is a hardware accelerator which has potential of providing data processing speedup of up to 150x compared with traditional data stream processors.

However, SCU implementation is very complex, fixed, and uses an outdated host interface, which limits future improvement. Mr. Tareq S. Alqaisi, an MSECE graduate from IUPUI worked on curbing these limitations. In his architecture, he used a Xilinx MicroBlaze microcontroller to reduce the complexity of SCU along with few other modifications.

The objective of this study is to make SCU suitable for mass production while reducing its power consumption and delay. To accomplish this, the execution unit of SCU has been implemented in application specific integrated circuit and modules such as ACG/OCG, sequential comparator, and D-word multiplier/divider are integrated into the design. Furthermore, techniques such as operand isolation, buffer insertion, cell swapping, and cell resizing are also integrated into the system.

As a result, the new design attains 67.9435 μW of dynamic power as compared to 74.0012 μW before power optimization along with a small increase in static power, 39.47 ns of clock period as opposed to 52.26 ns before time optimization.

Keywords: Symbiote Coprocessor Unit, ACG/OCG, sequential comparator, D-word multiplier/divider, FPGA.

1. INTRODUCTION

Due to the exponential increase in number of devices that continuously stream data and the availability of large wireless networks, the demand for systems that have efficient computational power with limited resource has increased. These systems which are designed to process continuously arriving data streams with limited hardware resources are known as Data Stream Management Systems (DSMSes). The streams contain data with high rate and in variable pattern. Aurora [1], Nile [2], Gigascope [3], and Symbiote Coprocessor Unit (SCU) [4] are some of the DSMSes that have been developed.

1.1 Previous Work

Symbiote Coprocessor Unit [4] is a hardware accelerated coprocessor with tuple processing speedup in the range 12.3x to 150x compared to its software-based counterparts. The execution engine of it is a Single Instruction Multiple Data (SIMD) Very-large Instruction Word (VLIW) processor. Stream Controller (SC) is the hardware Finite State Machine (FSM) of SCU which takes care of control and configuration part of the system. However, SCU has many disadvantages. First, the complexity of finite state machine is high such that it limits future expansion. Second, it uses Hyper Transport protocol to interface with host processor which is obsolete. Finally, it uses bus interconnects that are exclusively designed for that version of SCU, which limits flexibility of hardware in future.

Thus, a following study by Mr. Tareq's [5] updates the SCU architecture as follows:

- Xilinx MicroBlaze is used instead of stream controller.
- The Hyper Transport protocol is replaced with PCIe bus to increase the compatibility of SCU.
- AMBA AXI4 bus interconnect is used to communicate within the system.

1.2 Motivation

The SCU is now flexible and has simpler hardware yet it is implemented in an FPGA. It is a known fact that the Application Specific Integrated Circuit (ASIC) of a design is faster than its FPGA implementation. This study aims to implement the most important part of SCU, the execution unit, as an ASIC and improve its timing and power requirements further. When it comes to ASIC, three things are considered as bottlenecks: timing, power, and area. They are termed bottlenecks because the design requirements of an integrated circuit are specified with these attributes. For example, a designer would be given constraints such as maximum power limits, maximum frequency of operation, and maximum area of chip that the designed circuit must satisfy in order to be sent for manufacturing. However, there is always a trade-off between these three attributes of a circuit which must be addressed. This work integrates many features into the hardware of SCU to address them.

1.3 Thesis Outline

This chapter gave the introduction needed for this work. Next chapter discusses a MicroBlaze-based coprocessor for data stream management systems, which is the previous work of this thesis. Chapter 3 explains this thesis work in detail, and chapter 4 explains the experimental setup and results of this study. Chapter 5 concludes the thesis and suggests some future work related to this topic.

2. MICROBLAZE-BASED COPROCESSOR FOR DATA STREAM MANAGEMENT SYSTEMS

2.1 Introduction

A MicroBlaze-based Coprocessor for Data Stream Management Systems (MCDSMS) [5] is an FPGA-based hardware that is designed to perform tuple processing in an efficient way with reduced hardware complexity. Since the MCDSMS [5] is an extension of Symbiote Coprocessor Unit (SCU) [4], it is also sometimes referred to as SCU in this work, since the working principle for both is basically same.

Based on the relatively persistent nature of data stream tuples, the paths of SCU can be divide into two parts, tuple processing path and configuration path. Figure 2.1 shows the tuple processing path consisting of modules that process the continuously varying data and needs to be optimized for high performance. The configuration path consists of components that pass the controlling bits that arrive not so frequently and need not be optimized for performance.

Components such as command/response FIFOs and stream controller that are in the configuration path of the SCU and are called as Stream Management Unit (SMU) as they are responsible for configuring the SCU. The components such as write stripe pipe, read stripe pipe, stream register file, input/output FIFOs, instruction memory, Branch and Stall Unit (BSU) and execution unit are in the tuple processing path and are collectively called as Stream Processing Unit (SPU) as they are responsible for processing the streams. To maintain the performance same as that of the original SCU, changes are only made in the stream management unit.

The interface between host and processor is the Hyper Transport (HT) interface which communicates with read/write stripes to send/receive data and command/response FIFOs to send/receive commands/responses, respectively. The Stream

Register File (SRF) is the storage place of streams and is connected to input/output FIFOs to send data to execution unit and to receive data from execution unit. The SCU at its core has a SIMD-VLIW execution unit which exploits data level parallelism and instruction level parallelism in DSMS operations.

The second version of SCU, that is MCDSMS shown in Figure 2.2, replaces the SC in the first version. It consists of a MicroBlaze controller, interfaces for the MicroBlaze to communicate with other components, and a Dispatch unit. The SPU uses the same SCU components identified in the tuple processing path of previous version. The MicroBlaze has three interfaces, namely command/response FIFOs interface, instruction memory interface, and Kernel run monitor interface. These interfaces are used by MicroBlaze to communicate with command/response FIFOs, instruction memory, and kernel run monitor, respectively. PCIe is used as the interface between

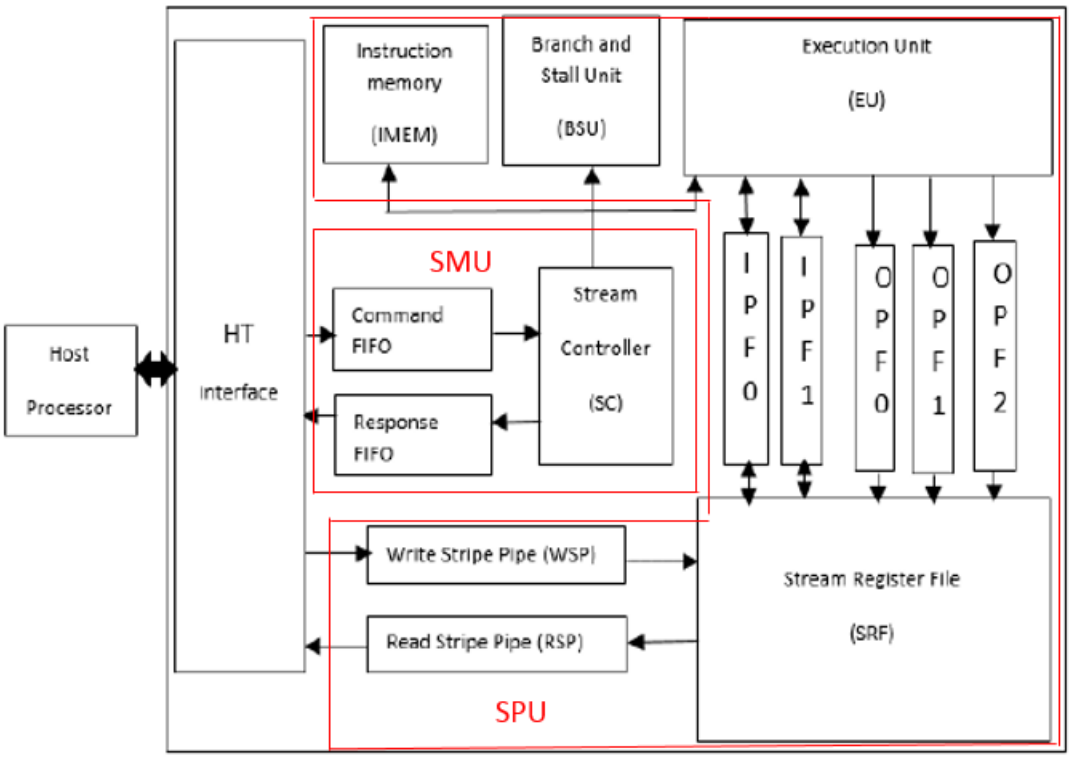

Fig. 2.1. Symbiote Coprocessor Unit Architecture [4].

host and the coprocessor. Therefore, PCIe interacts with dispatch unit of MCDSMS and RAM of the host processor. All data are given to dispatch unit which then segregates the commands from tuples and send them to command/response FIFOs. The tuples are sent to read stripe pipe which in turn puts them into SRF. The following sections discuss the architecture of the MCDSMS in detail.

2.2 Basic Workflow

Whenever the host processor wants to execute a kernel, it performs a sequence of steps. These steps are explained briefly in this section. First, the host processor sends the configuration data such as kernel information, kernel IDs, and tuple information to the management part of SCU. Kernel information tells what operations need to be

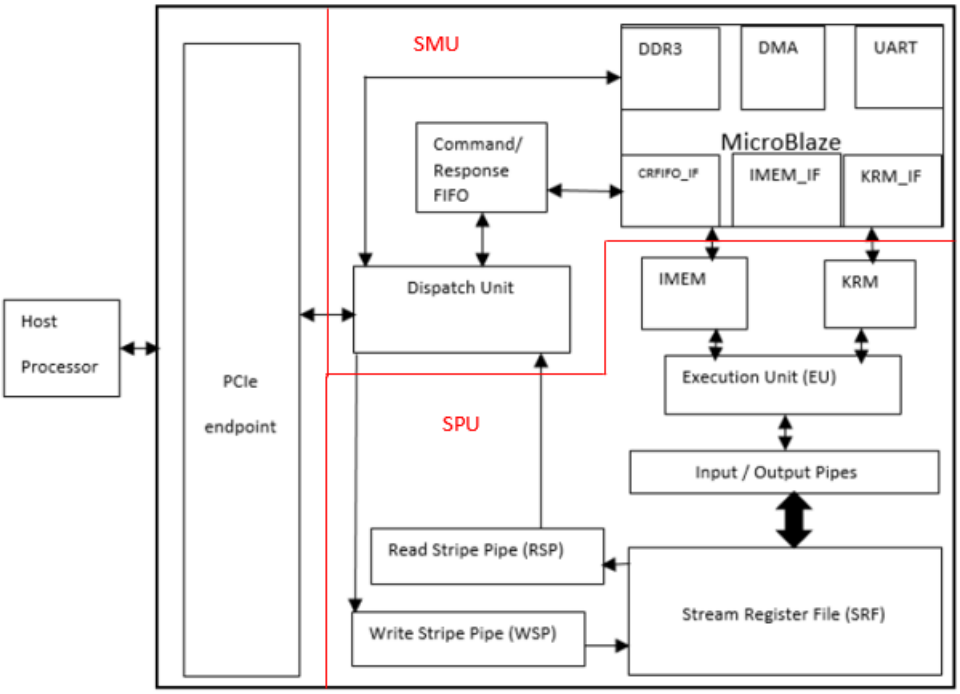

Fig. 2.2. Architecture of MicroBlaze-based Coprocessor for Data Stream Management Systems [5].

done with the given tuples. Tuple information is basically the information of where the tuples corresponding to the kernel are stored in memory and what is their length. After sending this data, the host processor sends the actual tuples upon which the operations must be carried out, to the processing part of SCU. These data are stored in stream register file and acknowledgment signal is sent back to host processor. Once this configuration part is over, the host processor initiates the execution of kernel by sending "RUN" command. The execution unit then processes the tuples and store the results to SRF via output pipes. After all the results have been stored into memory, the module that monitors the execution unit sends an interrupt signal to MicroBlaze which in turn notifies the host processor that the processing is completed. The host processor can then read back the results.

2.3 Sub-systems and Interfaces

As stated earlier, the MCDSMS can be divided into SMU and SPU. MicroBlaze is the main controlling unit of SMU. MicroBlaze uses subsystems as the interface to communicate with other modules that are outside the microcontroller. The MicroBlaze subsystem contains standard and application specific AXI4 memory-mapped peripherals that are explained in the following subsections. Figure 2.3 shows the MicroBlaze subsystem components.

Kernel Run Monitor Interface (KRM-IF)

MicroBlaze communicate with KRM using its interface named Kernel Run Monitor Interface (KRM-IF). KRM-IF contains three types of register files namely Kernel Descriptor File (KDR), Stream Descriptor File (SDR), and Offset Register File (ORF). Figure 2.4 shows the overall architecture of the KRM-IF. Host processor initiates the execution process by sending Run kernel command along with the kernel ID to SMU through KRM-IF. Figure 2.5 shows the states of KRM-IF finite state machine.

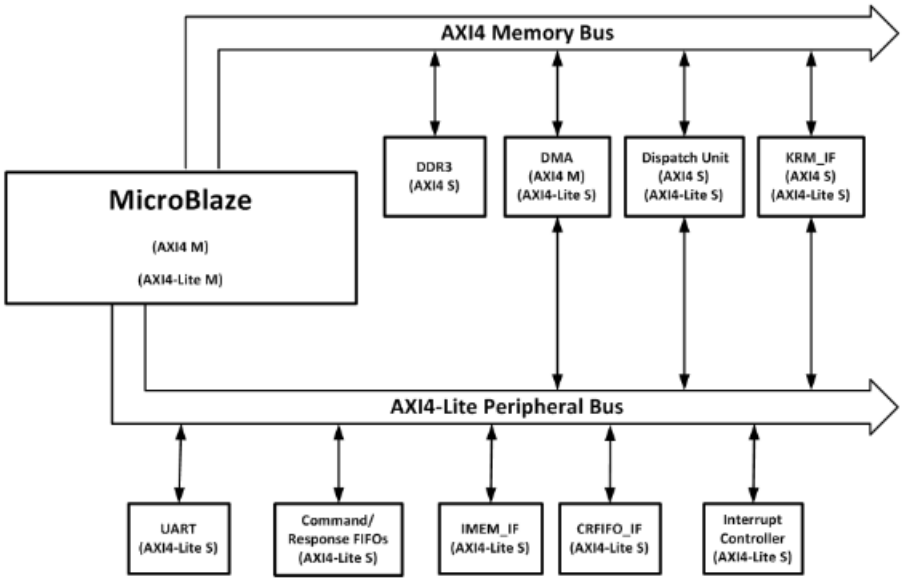

Fig. 2.3. MicroBlaze subsystem [5].

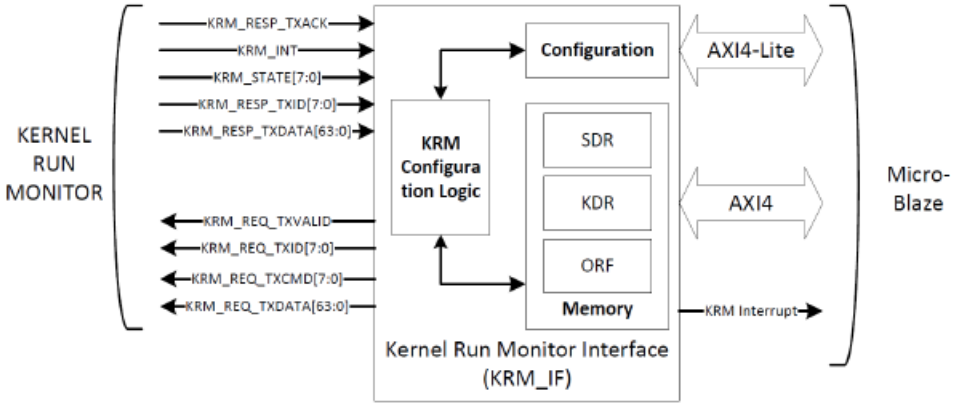

Fig. 2.4. Kernel Run Monitor Interface (KRM-IF) architecture [5].

Instruction Memory Interface (IMEM-IF)

Information regarding the kernel and its IDs are stored in IMEM. MicroBlaze communicates with IMEM through IMEM-IF. IMEM-IF maps Stream Controller Interconnect (SCTX) of SCU used by IMEM to AXI4-Lite.

Figure 2.6 shows the block diagram of IMEM-IF. All transaction requests from MicroBlaze have two components: transaction ID and command. These transactions are sent to interface where it is mapped to respective registers and then to IMEM. After MicroBlaze sends the request, it waits for acknowledgement signal from instruction memory. All acknowledgements have the transaction IDs for which response is sent. However, the response may or may not have response data in it. All responses from instruction memory is stored in registers inside IMEM-IF. MicroBlaze reads

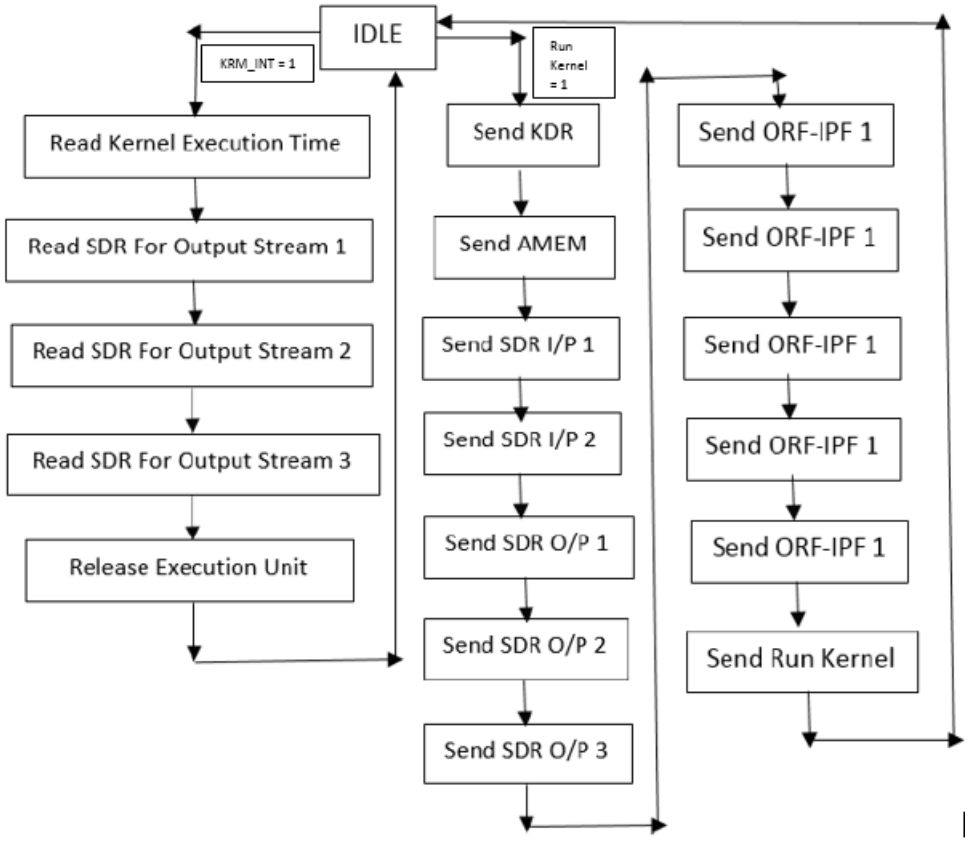

Fig. 2.5. KRM-IF finite state machine [5].

these responses whenever needed by asserting the read enable signal in one of the interface's register. Figure 2.7 shows the IMEM-REQ-CNTRL Register. Figure 2.8 illustrates IMEM-RESP-STAT register which the IMEM-IF must monitor to update the MicroBlaze regarding the status of IMEM.

Command - Response FIFO Interface (CRFIFO-IF)

The commands from host processor are stored in command FIFO, and responses from SCU are stored in response FIFO. MicroBlaze communicates with these FIFOs through CRFIFO-IF. Figure 2.9 shows the block diagram of the CRFIFO-IF. The commands from host processor are of double word length; the CRFIFO-IF breaks down them into two words and stores them into their internal registers for the MicroBlaze to access. Two response words from MicroBlaze are combined in the internal registers and sent to host processor. Furthermore, CRFIFO-IF maps configuration and status signals to the FIFO configuration status register shown in Figure 2.10.

Dispatch Unit (DU)

Dispatch unit does the job of mapping host processor interface with standard interconnect and expose the internal memory of SPU to MicroBlaze. Figure 2.11 shows the block diagram of dispatch unit. When DU receives data from host processor, it segregates tuples from commands and sends each of them to their respective places.

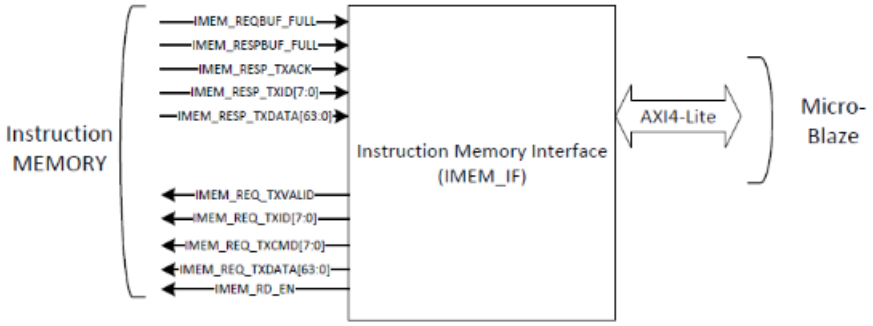

Fig. 2.6. Instruction Memory Interface (IMEM-IF) input and output signals [5].

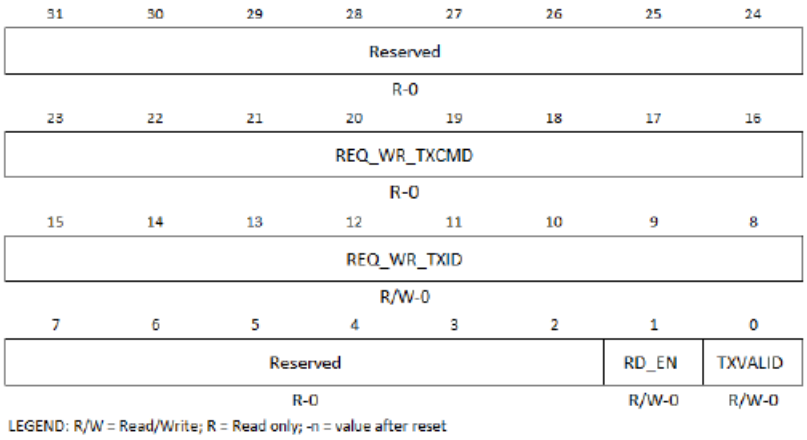

Fig. 2.7. Instruction Memory Request Control (IMEM-REQ-CNTRL) Register [5].

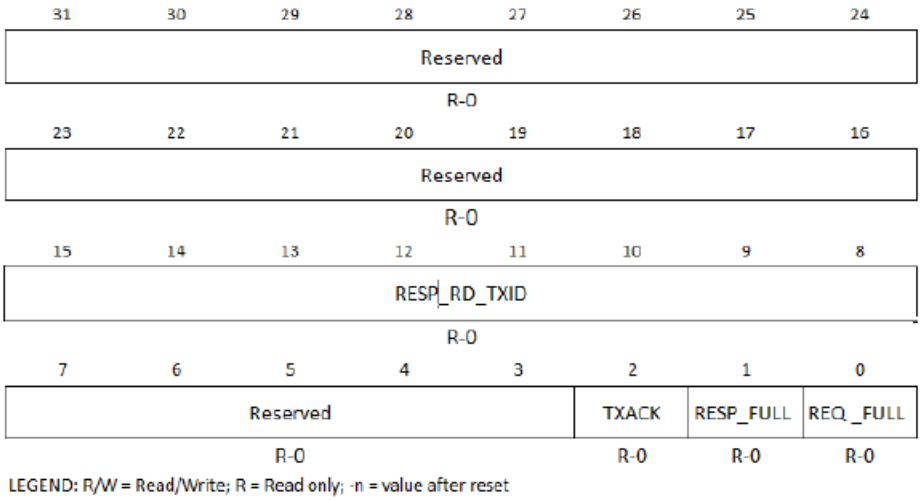

Fig. 2.8. Instruction Memory Response Status (IMEM-RESP-STAT) Register [5].

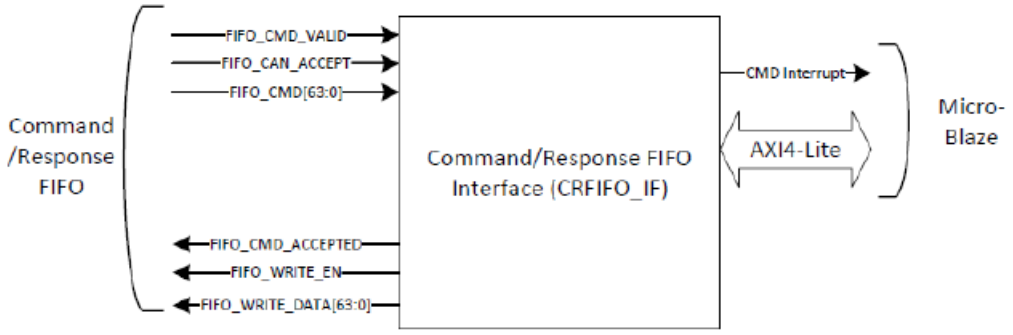

Fig. 2.9. Command/Response FIFO Interface (CRFIFO-IF) signals [5].

Fig. 2.10. FIFO-CNFG-STAT [offset = 10h] Register [5].

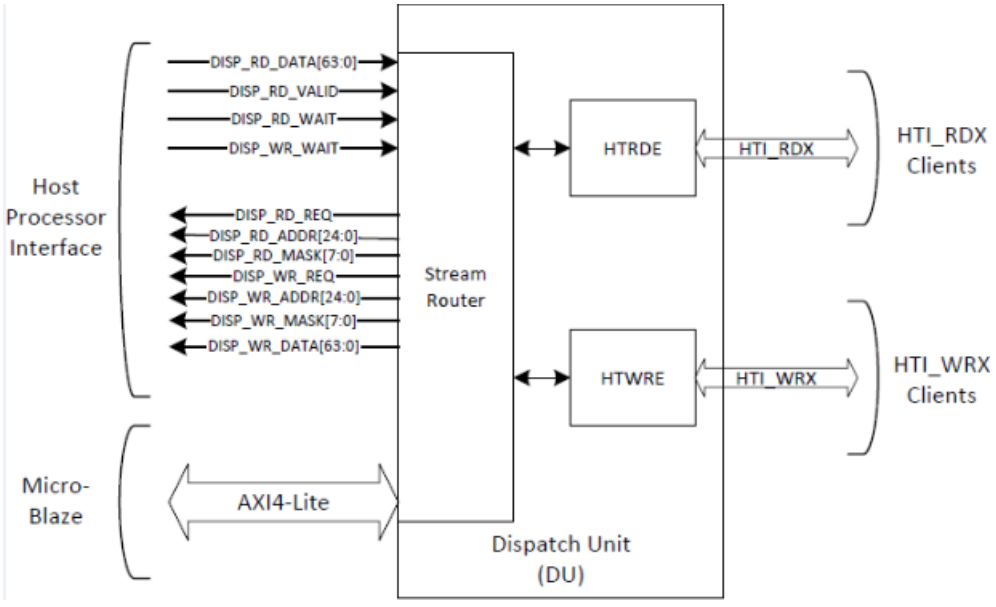

Fig. 2.11. Architecture of Dispatch Unit (DU) [5].

3. POWER AND TIME OPTIMIZATION TECHNIQUES USED

The speedup of Symbiote Coprocessor Unit's (SCU) FPGA implementation is in the range of 12.3 to 150 times than that of other software-based Data Stream Management Systems. It is well known that the Application Specific Integrated Circuit (ASIC) implementation of a module is faster but sometimes power hungrier than the FPGA implementation of the same. The objective of this study is to make SCU suitable for mass production and to improve its timing and power requirements. The first step in this direction is to implement the Execution Unit (EU) of SCU as an application specific integrated circuit. Then, improving the timing and power requirement, which is directly related to increasing the performance and efficiency of the design.

SCU has an execution unit that is VLIW-SIMD-based arithmetic and logic unit. The key to higher performance in DSMS is the ability to exploit fine-grain instruction-level parallelism. One of the methods used for exploiting fine-grain parallelism is to specify multiple independent operations per instruction, which is the idea of very long instruction word (VLIW). Similar to super-scalar processor, VLIW processor has the potential to issue and complete more than one operation at a time. However, there is a difference between them in the mechanism with which the parallel processing is done. The hardware of VLIW processor is not responsible for discovering possibilities to execute multiple operations concurrently as in case of super-scalar processors.

The long instruction word of the VLIW processor already encodes the concurrent operations that are to be executed. This facility of VLIW processors significantly reduces the hardware complexity compared to a super-scalar implementation. Therefore, a VLIW processor has the same effect as a super-scalar RISC or CISC processor, but the later design does so with complex parts of a high-performance super-scalar hardware. It is not required for a VLIW processor to have decoding and dispatch-

ing hardware that tries to reconstruct parallelism from a serial instruction stream since VLIW instruction explicitly specify all possible parallelism within it. The basic diagram of a Very Large Instruction Word (VLIW) processor is shown in Figure 3.12. It is seen clearly from the figure that there exist functional units, namely FU0, FU1, FU2, FU3, . . . , FUn from which the appropriate operation is executed for each instruction.

In order to make an application specific integrated circuit of the explained VLIW processor, the hardware of the design is written in a hardware description language such as Verilog or System Verilog or VHDL. In this work, Verilog language is used to code the hardware of the execution unit. The code must be simulated and debugged for functional errors using industry standard tools. Industry standard tools are software developed by companies such as Synopsys, Cadence, and Mentor Graphics. These are software used in ASIC industry to design, analyze, test, and approve actual integrated circuits which are sold for custom applications.

In this work, the functional verification of pre-synthesis RTL code is done using Mentor Graphics' ModelSim [6]. Once the functional verification is done, the RTL

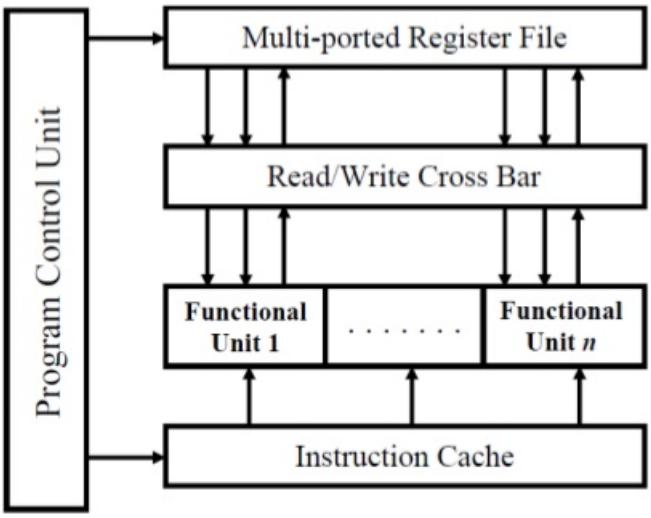

Fig. 3.1. Block diagram of Very Large Instruction Word (VLIW) processor [7].

code is then synthesized with a library file using a software known as Design Compiler from Synopsys [8]. Synthesizing is a process in which the modules written in RTL code are mapped to the actual gates and flip-flops in a library file provided by a vendor. In this thesis work the library files are provided by Synopsys, so the vendor is the Synopsys Corporation. The RTL is synthesized with SAED28RVT0P725C.db standard cell library, and the Verilog file is now turned into a gate level netlist. After the synthesis, various reports of the design such as constraint report, clock gating report, and area report are generated. The netlist is analyzed for synthesis errors, along with the generated reports.

One of the objectives of this study is to reduce the power consumption of ASIC implementation of SCU, which can be achieved using clock gating, operand isolation, and MSB comparison techniques. However, these techniques cannot be directly integrated into the SCU as it was not designed with this intention originally. Therefore, to add these mechanisms into SCU, in this thesis, three modules are designed to suit the hardware. They are AND Clock Gate/OR Clock Gate (ACG/OCG) modules, Double Word (D-word) multiplier/divider, and sequential comparator. These techniques are integrated into the netlist and re-synthesized iteratively until it meets the required power dissipation levels. Along with these modules, operand isolation technique is also used to reduce dynamic power. The power reports of the modules before and after adopting these techniques can be obtained and checked for improvements.

After synthesis is done, the updated netlist is cross-checked with pre-synthesis RTL code for functional equivalence. This process is done by using software named Formality, provided by Synopsys. If the netlist is functionally equivalent to pre-synthesis RTL code, the netlist is analyzed with a timing check software known as PrimeTime [9], provided by Synopsys. Static Timing Analysis (STA) is done in this tool. STA is a process in which a design is checked if it satisfies the setup requirement for a specified frequency along with hold time requirement.

The other objective of the study is to improve the timing, which is to decrease the delays within data paths. To accomplish this, the design module is inserted

with buffers/inverters, cells are up-scaled/down-scaled, and are swapped to make the design meet timing requirements specified in a constraint file. The timing report of the updated maximum and minimum paths are then generated and checked if the objectives are accomplished. Figure 3.2 shows the simplified flow diagram of the work done. The formal verification step explained earlier is an additional work done to confirm that the generated netlist is functionally correct. This step does not come under basic flow of the work, due to which it is not shown in the flow diagram.

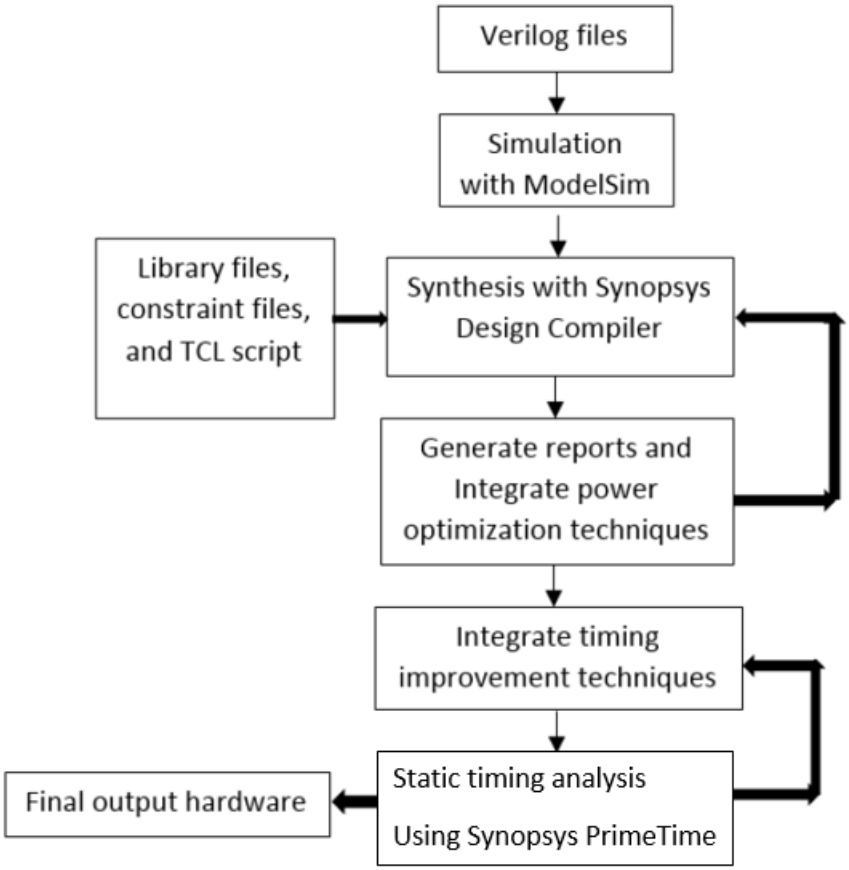

Fig. 3.2. Flow diagram of work done.

3.1 Power Optimization Modules

The increase in desirability of portable devices, demand for reliability and performance, extension of battery life, reduction of package cost, and reduction of green cost have resulted in energy efficiency becoming a critical feature of modern electronic devices. CMOS technology is the key element in the development of VLSI systems since CMOS consumes less power. Moreover, power optimization has become a very important concern in deep sub-micron CMOS technologies as the sizes of transistors become smaller. Reduction in power consumption and overall power management on the chip are the key challenges due to the shrinkage in size of integrated circuits. For many designs, power optimization is important in order to extend battery life. Thus, this work aims to reduce the dynamic power requirement of the SCU's execution unit by introducing various power optimization modules and features into the device.

The modules that were instantiated into the system are discussed in following subsections and the operand isolation technique is discussed in a separate section as there was no module designed for this technique; instead, it was directly integrated into the system.

3.1.1 AND Clock Gate (ACG)/ OR Clock Gate (OCG) Modules

The power consumption in a chip can be due to two factors, namely static power and dynamic power. Static power is the power consumed when the transistors are in a stable state. Dynamic power is the power consumed when the state of transistors changes. The formula for the dynamic power is given as:

$$P_{dynamic} = xcv^2f$$

where, x = activity factor

c = capacitance

v = voltage

f = operating frequency

It is seen from the formula that, factors such as frequency, switching activity, voltage, and capacitance of the device under consideration affect the dynamic power directly. This section describes two types of modules that are integrated into the circuit to reduce dynamic power consumption through reduction in switching activity.

This method is named as clock gating; as the name suggests, this technique is simply gating the clock pulse reaching a Flip-Flop (FF) when there is no need of a clock pulse in the design. Consider an FF that copies input to output at the triggering edge of a clock, the next inactive edge of the clock does not change the output of it but still causes transition in the device. However, for the clock signal to have the next active edge, it is mandatory for it to have that inactive edge. Let's consider the below mentioned cases:

- When the output of an FF is not being read at the current clock cycle.
- When the input of an FF is not changing in the current clock cycle.

In these cases, there is no need for clock signal to reach the terminals of FF. This will prevent unnecessary switching activity of clock terminals. Clock gating can be done in two ways, AND-type clock gating and OR-type clock gating.

AND-Type Clock Gating

Consider an AND gate where one of its input is connected to "ENABLE" and other input connected to clock signal as shown in Figure 3.3.

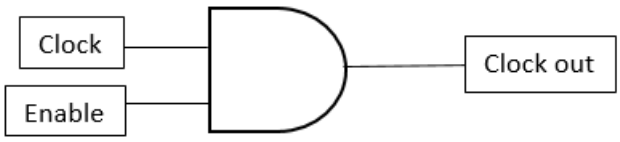

Fig. 3.3. AND gate used in clock gating.

When the enable pin is high, the clock is transferred to the output; however, if the enable is low, the output of AND gate is low. This prevents the clock signal to reach FFs whenever enable is low.

OR-Type Clock Gating

In this case, we use an OR gate with an enable connected to one input and clock connected to other input as shown in Figure 3.4.

When the enable is low; the clock is transferred to gate output. When the enable is high; the gate output also remains high. This characteristic can also be used to prevent clock signals reach FFs when not required.

Problems in Direct Clock Gating and ACG/OCG as a Solution

Considering an AND-type clock gate, to get the same duty cycle at the output of AND gate as of the original clock signal, it must be ensured that the change of state of enable signal does not occur when the clock is high. For this to be achieved, the enable input must toggle only when clock is at '0' state. Else, the clock pulse gets clipped as shown in Figure 3.5. This causes a glitch in clock path. Therefore, AND-typed clock gate outputs the proper duty cycle of a clock only if the enable signal changes state when clock is low. There are two cases that arise when considering the source of 'EN' signal. First, if the source of 'EN' signal is a positive edge-triggered FF, setup and

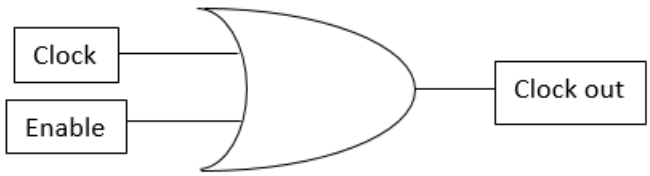

Fig. 3.4. OR gate used in clock gating.

hold timing checks are in the next positive and negative edges respectively from the clock edge where the data is launched from the sending FF. However, this situation is not possible to guarantee the data to pass reliably under all operating conditions. Second, if the source of 'EN' signal is a negative edge-triggered FF, setup checks are performed with respect to the next rising edge and hold check is performed on the same falling edge (zero-cycle) of the clock edge from which the data is sent.

Contrary to the first possibility, the second one assures the data to pass reliably under all conditions. ACG module is the design which captures this functionality and provides solution for pulse clipping in AND-type clock gating. It has a negative level-triggered latch and an AND gate connected to it. Figure 3.6 shows the diagrammatic representation of ACG module.

On the other hand, for an OR-type clock gating, the enable signal must change only when the clock is at '1' state. That is, the clock pulse gets clipped as shown in Figure 3.7 if there is a change in enable signal when clock is low. If 'EN' signal is launched from a positive edge-triggered FF, setup check is with respect to next falling edge and hold check is on the same rising edge as that of the edge from which the data is launched. This situation guarantees correct operation in all operating conditions. This functionality is captured by OCG module shown in Figure 3.8, where it has a positive level-triggered latch and an OR gate connected to it.

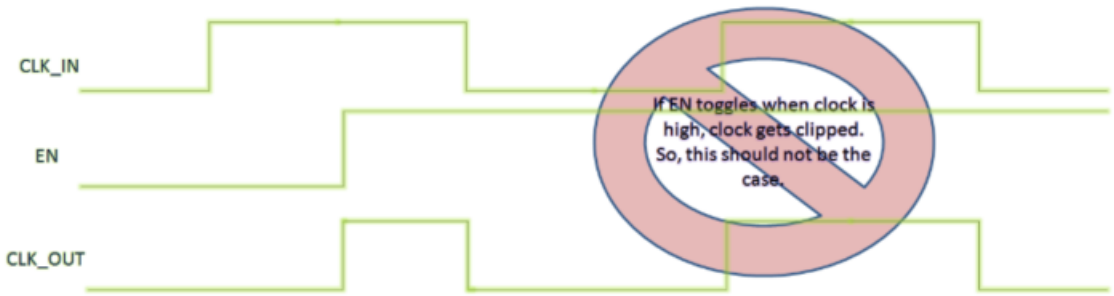

Fig. 3.5. Pulse clipping in AND-type clock gating [10].

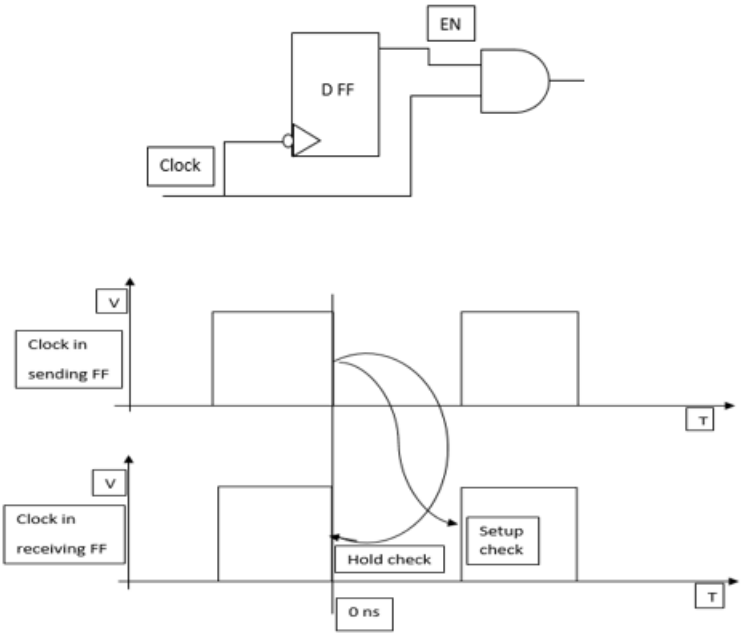

Fig. 3.6. AND-type clock gating with latch to avoid pulse clipping [10].

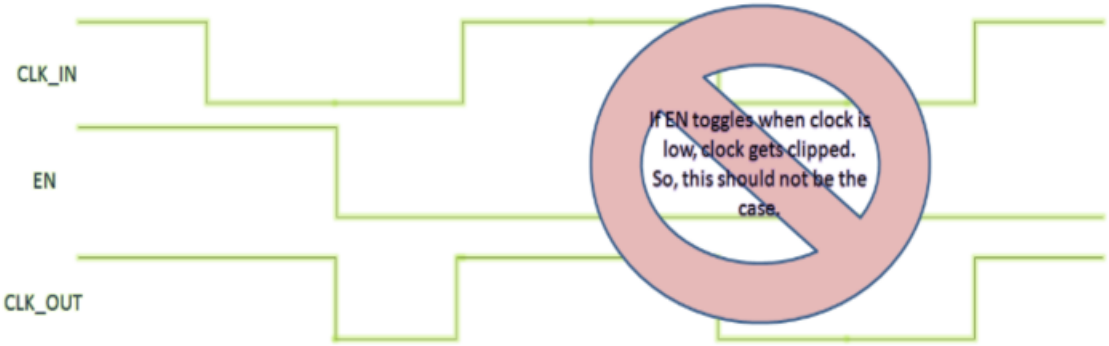

Fig. 3.7. Pulse clipping in OR-type clock gating [10].

3.1.2 D-word Multiplier / Divider

One of the ideas behind reducing dynamic power consumption is to reduce the number of transitions a transistor undergoes. This can be done by using the gates which produce a fewer number of transitions for the changes in input rather than using the gates which produce transitions for all changes in input but with same functionality as the earlier gate. This section deals with one module that is used to reduce dynamic power, known as D-word multiplier / divider. To get the essence of this technique, the truth table of XOR gate shown in Figure 3.9 is first analyzed.

It is understood from the truth table that in an XOR gate, any transition in inputs will always cause a change of state at the output. This property of XOR gate makes it a power-hungry gate. However, if the truth table of AND gate shown in Figure 3.10 and OR gate shown in Figure 3.11 are examined, it is clearly seen that the transition from one of its input to output is blocked by the other input.

For AND gate, if one of its input is '0', the transition from the other input does not reach output at any case. NAND gate follows AND gate, due to which, '0' in one of its input is a blocking state. For OR gate, this blocking state is '1' in one of its inputs, it is known from the truth table that a '1' in one of its inputs does not allow its output to change further until the same input is changed. Similarly, for NOR gate, as it follows OR gate, it has a blocking state of '1'. The above explanation gives enough reasons to exchange an XOR gate with any other gate which at least has one blocking state.

Considering a 4-bit multiplier shown in Figure 3.12 which has 12 full-adders of 2 XOR gates each (as evident from Figure 3.13). Therefore, if just a 4-bit multiplier takes 24 XOR gates, a 64-bit multiplier has at least 192 XOR gates that produce transitions for any change in their input, which causes a large dynamic power dissipation in execution unit. Thus, to curb this power requirement, the need to use a multiplier is reduced by introducing shifters. It is known that, multiplying a number by 2 is just shifting the number to left by one position and this property follows for

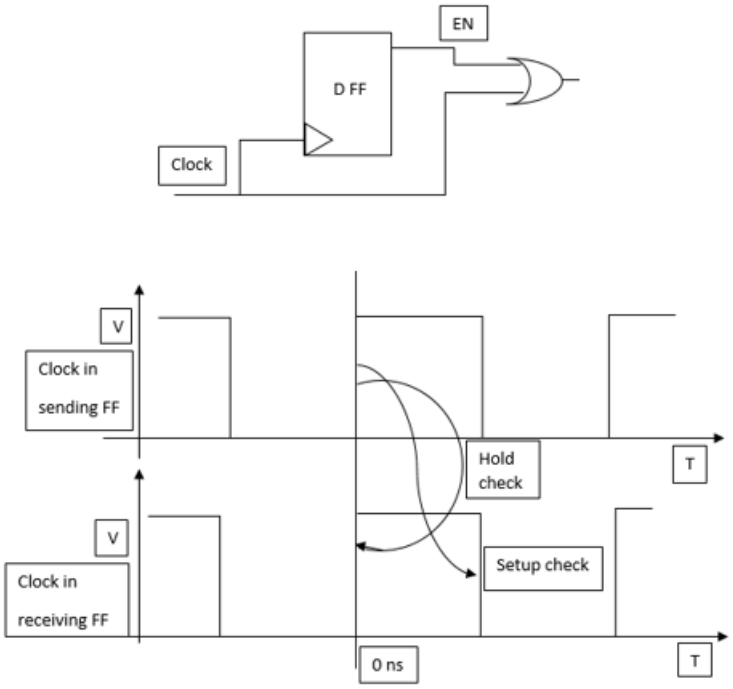

Fig. 3.8. OR-type clock gating with latch to avoid pulse clipping [10].

all numbers in the sequence 2, 4, 8, ... etc. Hence, if a number is multiplied by a number in the sequence 2, 4, 8, ... it means that the multiplicand must be shifted to left, by the log of multiplier to the base 2 number of positions. This action can be captured by a shifter that is made of NAND gates as shown in Figure 3.14, implying lesser dynamic power than XOR-based multiplier.

However, not all multiplications can be done with help of shifter. The multiplications for which the multiplier is not in the mentioned sequence, the operation is carried out with a conventional multiplier only. Therefore, there is a need for comparator which segregates the multiplicative operations which has and does not have a multiplier in the sequence 2, 4, 8, ... and then the operation is carried out by a shifter or a multiplier, respectively. This technique is not only used for multipliers, but it can also be used for dividers as well. Division by any number in the mentioned

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Fig. 3.9. Truth table of XOR gate [11].

Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 3.10. Truth table of AND gate [11].

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Fig. 3.11. Truth table of OR gate [11].

sequence is just shifting the dividend right, by the log of divisor to the base 2 number of positions. It must be noted that the dynamic power reduction by this technique is dependent on the application that it processes. Higher the percentage of multiplicands within the sequence, higher will be the power saved. As the module consists of both shifter and parallel multiplier, area is increased compared to the previous design.

3.1.3 Sequential Comparator

Similar to multipliers and divider, comparators also have a large number of XOR gates as shown in Figure 3.15. It is known that comparators need XOR gates to perform their task, but there are a few situations involving unnecessary transitions going on inside the comparator which may not change the already computed output of the comparator. One such situation is when comparing two operands each of 32 bits; output of the comparator is going to be "not equal" in either case if just the MSBs of operands are not equal or if all 32 bits of the operands are not equal. However, when the operands are not equal in all its bits, there incur unnecessary transitions going on inside the comparator, which increases the power consumption.

As these transitions do not contribute to the result, power consumption during this transition is totally wasteful. The objective of this technique is to eliminate unwanted switching activity, thereby reducing the mentioned wasteful power consumption. Typically, the result for many comparisons can be determined just by comparing the MSBs of both the operands. Hence, the next least significant bit should be involved in the comparison only if its preceding bit comparison was non-diagnostic. Figure 3.16 shows the modified implementation of comparator where the MSB comparison is done first.

If comparing just the MSBs of both operands are enough for arriving at a result, other least significant bits are prevented from comparison. For example, if the MSBs are not equal, the output of XOR gate would be high which in turn is given to OR

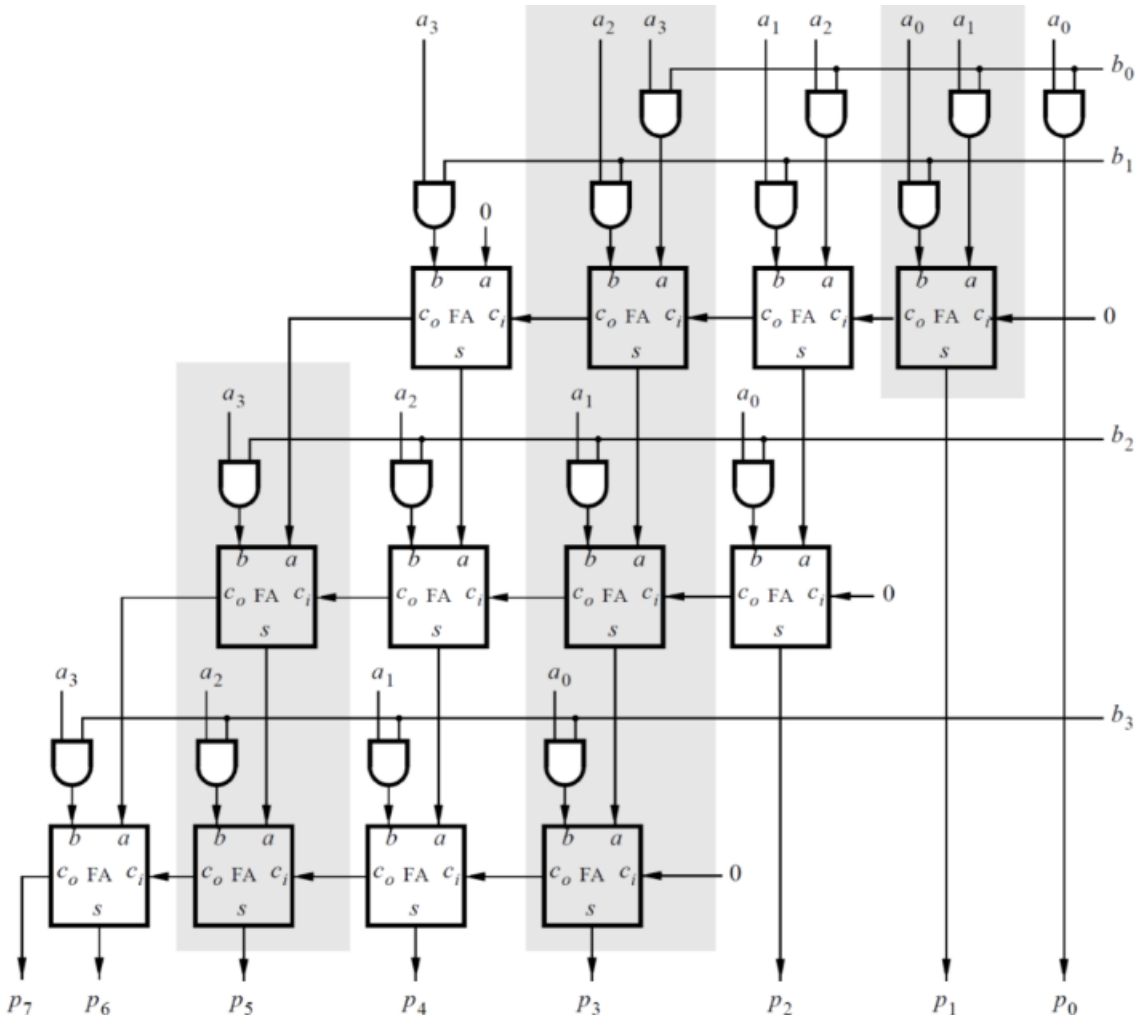

Fig. 3.12. Block diagram of 4-bit multiplier [12].

gates of LSBs. Now since there is a high signal in input of OR gates, the output of them would be just be high irrespective of other LSBs of operands. However, the circuit does not prevent the LSBs from reaching the XOR gates when the XOR output of MSBs is initially in low. This problem can be solved by adding an FF before LSBs and enabling them only after the MSBs are compared.

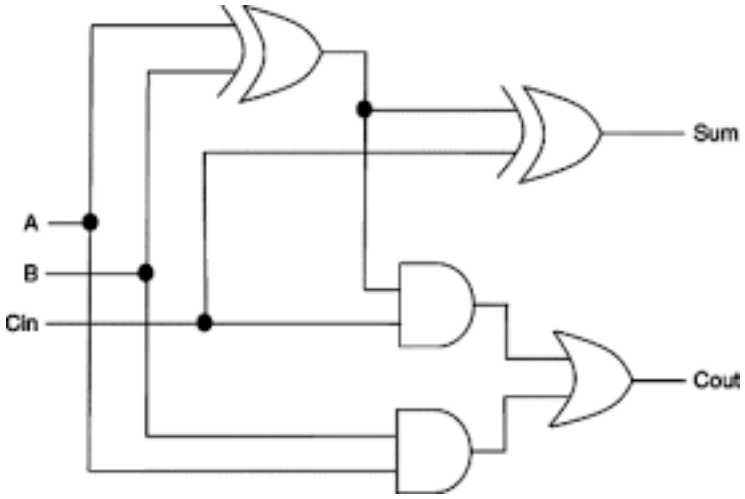

Fig. 3.13. Logic diagram of full adder.

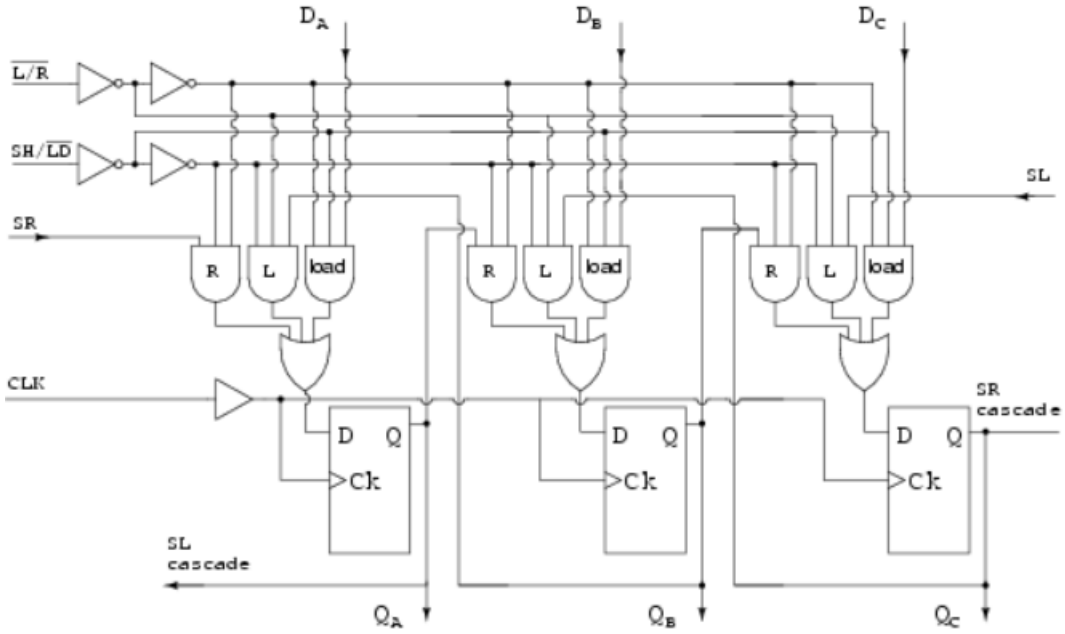

Fig. 3.14. 3-bit shifter [13].

3.2 Power Optimization Technique

3.2.1 Operand Isolation

As the reduction of dynamic power dissipation can be done through reduction of switching activity, this section deals with another method used to reduce unnecessary switching activity. Figure 3.17 shows the line diagram of a 32-bit adder.

Every bit of A and B does not reach the input at the same time. Different bits arrive at the adder's input after passing through different paths. As the value of each bit at the inputs of the adder changes, the adder reevaluates and presents the output for every change. However, not all outputs are considered as the result. The output which is obtained after the last bit changed of all 64 bits is the result that is valid. This action causes unnecessary switching activity that does not contribute to the outcome of the design which in turn causes dynamic power dissipation. When considering multipliers and dividers that have a large number of XOR gates, this power requirement grows much more and contributes to a high percentage of the total power dissipation.

In order to eliminate this unwanted switching activity, there is a need to employ a mechanism that just feeds the correct and final values of operand into the inputs of any functional unit, in this case, adder. To achieve this, a memory element such as latch or an FF can be employed to remember the bits of operands until all bits arrive and then synchronously give them to inputs of the adder when an enable signal is asserted. Figure 3.18 shows the block diagram of operand isolation mechanism. Increased area and increased latency are two factors that are to be considered while adopting this technique.

3.3 Implementation of Power Optimization Modules and Techniques

This section describes how the above discussed power optimization techniques were integrated into the ASIC implementation of SCU's execution engine. Initially,

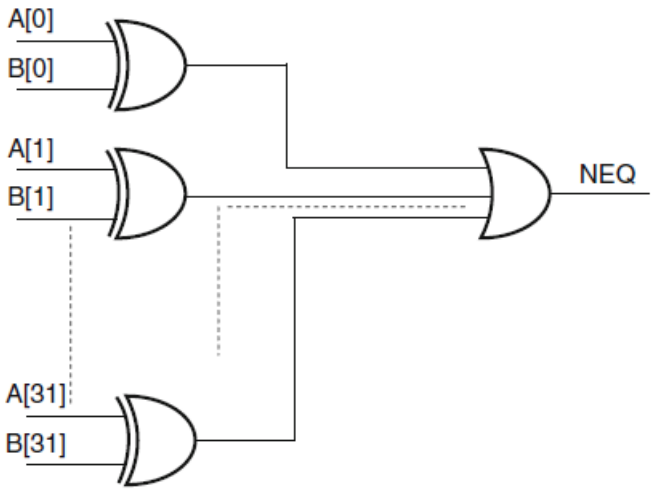

Fig. 3.15. 32-bit comparator [14].

the ASIC converted code of EU was uploaded to Design Compiler [8] software to analyze the code as well as the schematic of it to identify the places where clock gating was needed. After identification, they were divided into two categories based on the source modules of their controlling signal. Then the Verilog codes for ACG and OCG were written and synthesized. The ACG/OCG modules were designed and synthesized separately before integrating into the system to reduce delay.

The ACG and OCG modules were then placed in the identified places where they should belong and the whole system is again compiled. The design of D-word multiplier / divider was first written as separate modules with 64-bit input and output. The module was checked for functional correctness and then it was instantiated inside the EU. For sequential comparator, the ASIC code was first examined for identifying normal comparator and the newly designed sequential comparator was then replaced in the code. The code was then analyzed for input ports of the functional unit and the Verilog code for adding FFs before these ports which drive the functional units was written. Once all modules were simulated and instantiated, the whole system was synthesized, and reports were obtained.

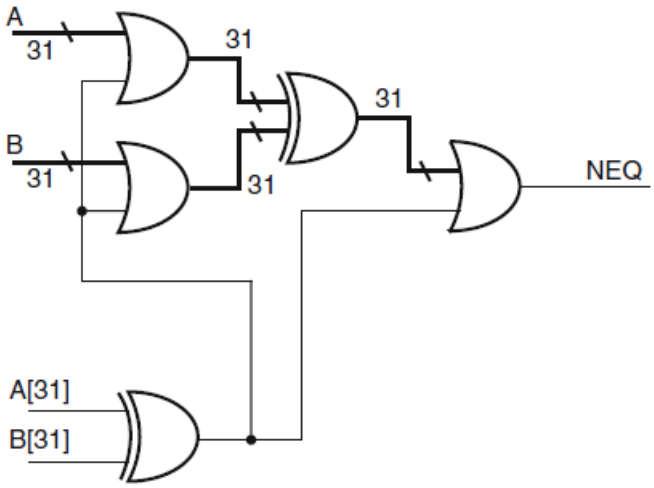

Fig. 3.16. 32-bit comparator with sequential MSB comparison [14].

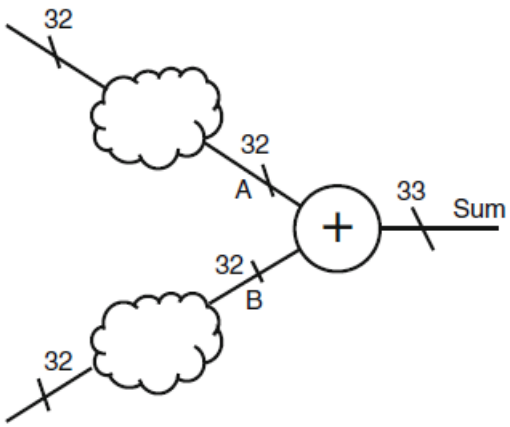

Fig. 3.17. Line diagram of 32-bit adder [14].

3.4 Time Optimization Techniques

This section discusses about the timing optimization techniques that are incorporated in the system. To perform an optimization process, the circuit needs to be analyzed first. The analysis process of a circuit is named as Static Timing Analysis (STA). This process is done using an EDA tool named Synopsys PrimeTime [9]. Data

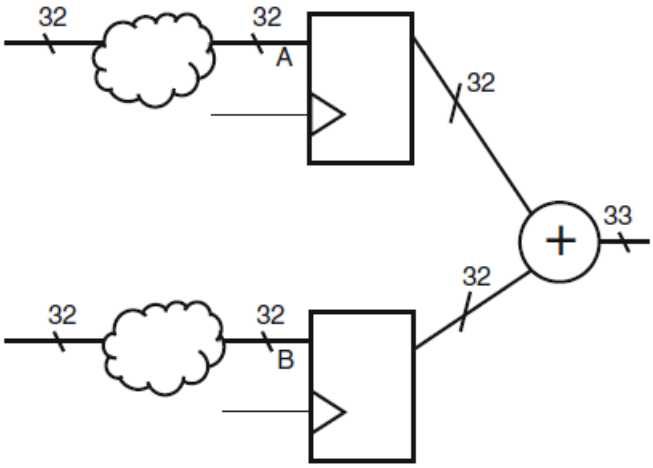

Fig. 3.18. Line diagram of 32-bit adder with latch for operand isolation [14].

are supposed to move in lockstep inside a synchronous digital logic circuit, passing one register to another register design part on each triggering edge of the clock signal. FFs and latches are memory elements that capture this functionality of copying their input to output only at certain values or transitions of clock and remember the previous value at all other times. Two types of timing violations are possible in such systems:

- Setup time violation. Setup time is the minimum time before the active edge of clock signal that the data should be present at the input terminals in order to copy it without any error. There is a setup violation when data arrives after setup time requirement.
- Hold time violation. Hold time is the minimum amount of time after the active edge of clock signal that the data needs to be present at the input in order to copy it without any error. There is a hold time violation if the data changes before this time is reached.

There are many reasons such as input variation, variation in operations performed, variations in manufacturing, variations in temperature, and variations in voltage that cause a signal to arrive at a node in varying time delays. Static Timing Analysis (STA) is an analysis procedure that verifies if all signals arrive at their corresponding

nodes at required time interval, and hence correct circuit operation can be guaranteed at specified frequency. In order to correct the violations, few techniques such as buffer/inverter insertion, cell resizing, and cell swapping were adopted into the circuit. The following subsections discuss these ideas in detail.

3.4.1 Buffer Insertion

As the size of MOSFETs used in chip shrinks, cell propagation delay and interconnect propagation delay change in opposite directions. As the size of wires decreases, there exist increased wire capacitance and resistance. According to Elmore's delay model, the increase in wire resistance and distributed capacitance along the wire cause the overall load capacitance to increase, and hence, the interconnect propagation delay is increased. The characteristics of gate delay and interconnect delay are shown in Figure 3.19.

Therefore, interconnect propagation delay limits the further down scaling of VLSI technology. Buffer (or repeater) insertion is an effective one among other techniques addressing this limitation. The other reason behind increased wire propagation delay is the reduction of signal strength along the wires. If, however, a buffer is inserted, the signal strength can be restored and hence the delay is reduced. Figure 3.20 shows buffer insertion in an interconnect wire. Buffers can be viewed as a shield for capacitive load in the path of a signal.

The metric that is used to measure the buffer requirement of an interconnect is known as critical buffer length. Critical buffer length is defined as the maximum length between two optimally sized and placed buffers in a wire such that the propagation delay in the wire is smaller than when buffers are not inserted.

According to repeater scaling and its impact on CAD [16], when the VLSI technology migrates from 90nm to 45nm, the critical buffer length decreases by 68 percent. Presently in a VLSI chip, there is a need for a large number of buffers which makes

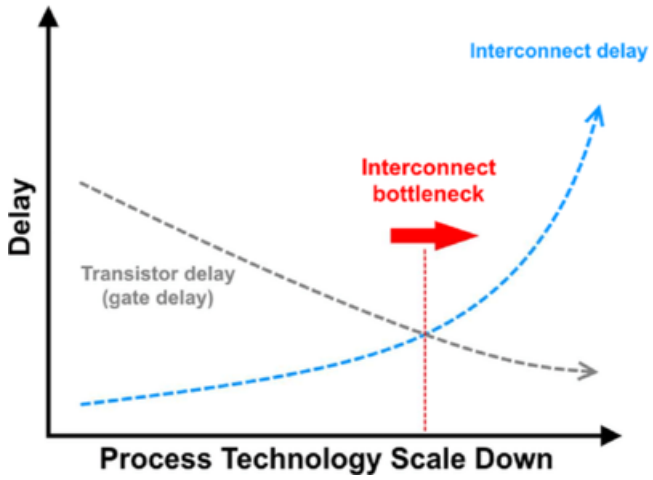

Fig. 3.19. Delay vs process technology graph [15].

buffer insertion a very important process. Therefore, buffer insertion tools need to satisfy various requirements such as:

- High quality of path analysis
- Speed
- Need for accurate delay model
- Ability to multi-task
- Interaction with other back-end tools

EDA tools provide histograms and reports, which makes it easy to identify the interconnects with timing violations. In addition, such tools provide commands and options for searching the right buffer from the technology library to fit corresponding parts of the design.

3.4.2 Cell Resizing

Any chip in electronic design can be assumed as a resistor–capacitor circuit for analysis purposes. This method of analyzing a circuit is known as Elmore model of delay analysis. According to this model, each switching activity of a transistor is

considered to be charging and discharging of a capacitor. In other words, when a transistor is in "ON" position, it is same as charging a load capacitor, and when a transistor is in "OFF" position, it is same as discharging of a load capacitor. Continuing in the same perspective, the delay of a cell is given as the time it takes to charge or discharge a capacitor. The rate at which this charging or discharging takes place is directly proportional to the amount of current that flows through the transistor. From the above explanation, the increase in current through transistor will decrease the delay. A simple MOSFET switch inside a chip looks as shown in Figure 3.21. It is seen from the figure that, to increase current through the transistor, width of the channel under the oxide must be increased. Thus, the delay of a cell can be increased or decreased by varying the channel width of MOSFETs used. Whenever a design part that does not meet the timing requirement is identified, the cells in that part are up-sized to increase the rate at which it charges / discharges and thus reducing the delay. This is shown in Figure 3.22.

The design part of an ASIC consists of combinational circuits sandwiched between registers that are clocked. Propagation delay through the combinational data path with maximum number of logics should be such that valid signals reach the FF input before the setup time of the FF and still satisfy the hold time requirement. The technique of increasing or decreasing the area is also alternatively known as increasing or decreasing the drive strength of a cell. Whenever a cell with larger area is inserted, it can drive a larger load capacitor and vice versa.

A typical standard cell library provided by a vendor has several versions of any given logic gate, each of which has a different gate size and drive strength. EDA tools provide commands and options to search among a list of cells in a library which are of same functionality but have different drive strengths among which a suitable cell can be selected and inserted into the system.

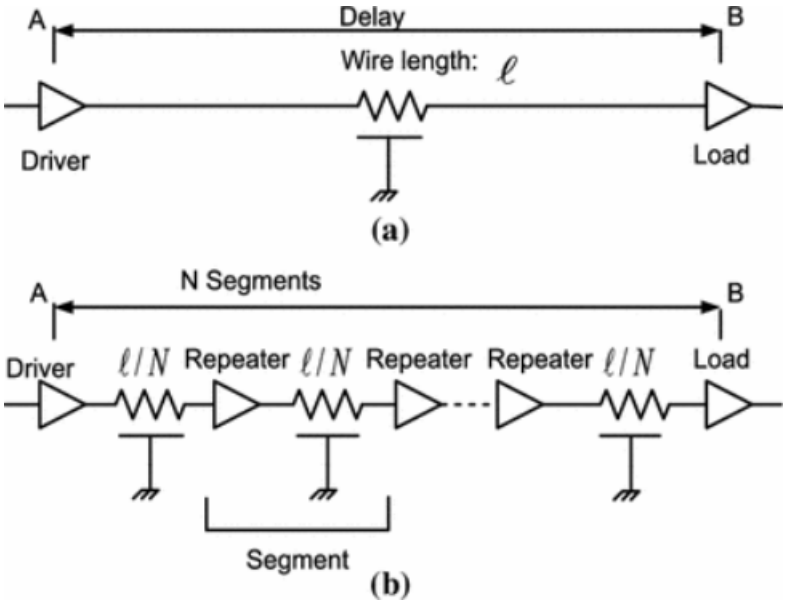

Fig. 3.20. Buffer insertion in an interconnect wire [17].

3.4.3 Cell Swapping

Transistors are switches that are turned on when they are applied appropriate gate-to-source voltages at their gate terminals. Threshold voltage is defined as the minimum gate-to-source voltage that is required to turn "ON" the MOSFET switch. According to threshold voltages, the cells in a library can be divided into two categories namely, high voltage threshold (HVT) cells and low voltage threshold (LVT) cells.

High voltage threshold cell are the cells which have higher fractions of the supply voltage as their gate-to-source threshold voltage. Low threshold cells are the cells that have very low fraction of supply voltage as their gate-to-source threshold voltage. The characteristics of high and low threshold voltage cells are given below:

- Cells with high threshold voltages have low leakage currents. However, the delay of these cells is high. Therefore, HVT cells are used in portions of the chip where high performance is not a priority and the parts that are idle for

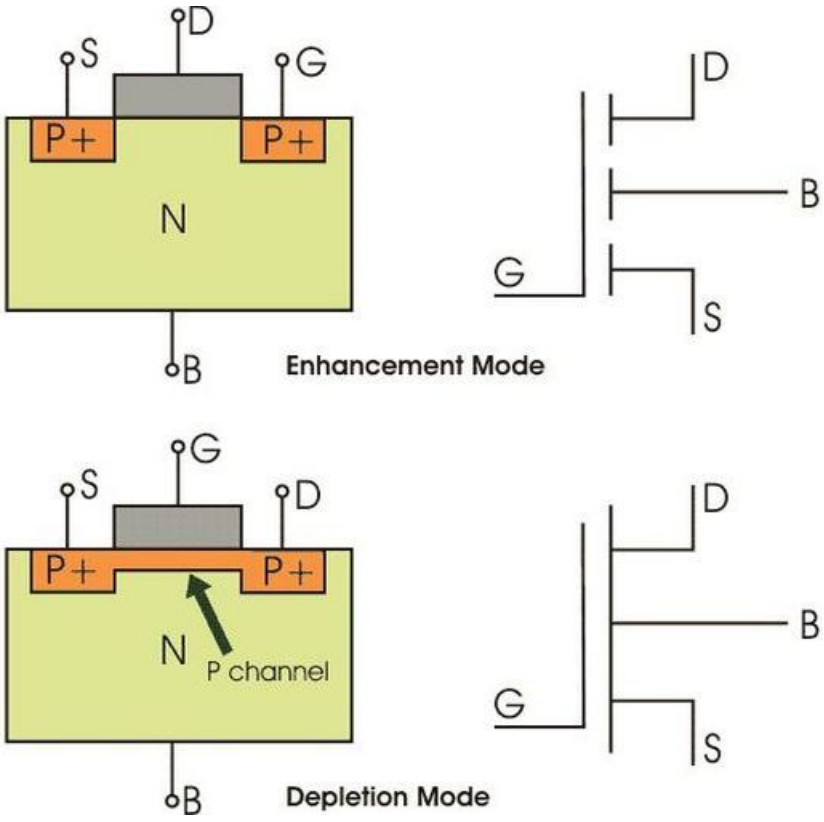

Fig. 3.21. Side View of P-channel MOSFET [18].

most of the times. As leakage component of power dissipation is high in these places, HVT cells are perfect to be placed here.

- Cells with low threshold voltage have higher sub-threshold leakage current. However, delay of these cells is low. The portions of chip which are time critical and require high performance, are designed with this type of cells. These cells have higher leakage power but lower dynamic power than HVT cells.

Table 3.1 shows the correspondence between threshold voltage, leakage power, and dynamic power. The dependencies between the output voltages of HVT/LVT cells and their delays are also depicted in Figure 3.23. It can be seen from the figure that, the LVT cells have higher slope for the same output voltage than the HVT cells, which explains the higher delay of HVT cells. Therefore, it is known that using low threshold voltage cells in place of high threshold voltage cells improves the timing of

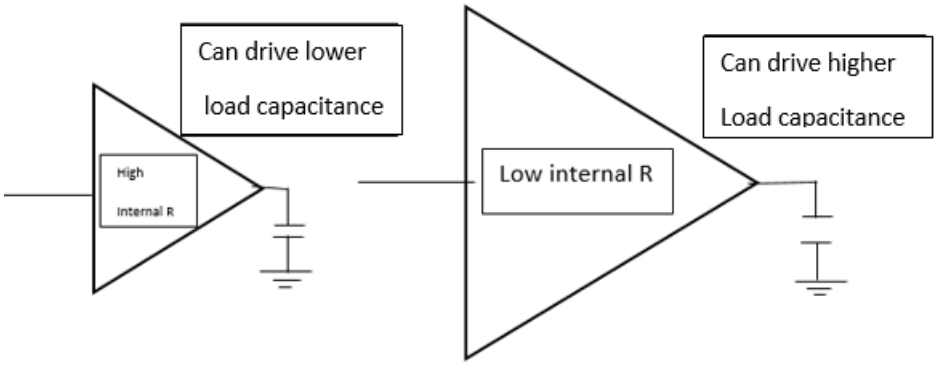

Fig. 3.22. Decrease in cell resistance with increase in area.

the circuit, which is one of the objectives of this work. Using appropriate cells from library to replace the cells in areas where there are setup and hold violations can be done with help of EDA tool's GUI mode or through scripts. There are exclusive commands to search for cells and to swap them with respective cells in the design.

3.5 Implementation of Time Optimization Techniques

This section describes the implementation process of integrating time optimization techniques into the system. After the power optimization modules were instantiated

Table 3.1.
Cell types and their respective power consumption

Cells	**Time**	**Leakage power**	**Dynamic power**
HVT	More	Less	More
LVT	Less	More	Less

and synthesized, the Verilog code of the system was sourced to Synopsys Prime-Time [9] software for timing analysis. The design was analyzed for the data paths that violate worst-case and best-case delay constraints through timing reports and

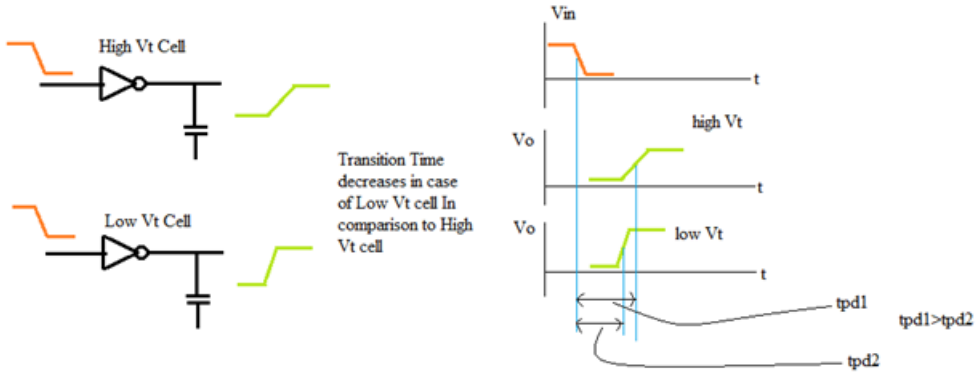

Fig. 3.23. Depiction of relationship between delay and threshold voltages [19].

histograms. The starting and ending nodes of all these paths were listed. These paths were iteratively inserted with buffers one at a time to fit in the interconnects so that the delay requirements are met. In places were the delay needs to be increased to meet the hold time requirements, two inverters were connected one after the other so that the propagation delay is increased but still maintaining the drive strength.

After the buffers were inserted, the synthesized gate level netlist was analyzed for the cells that need to be resized and swapped. Once these cells were identified, the standard cell library was searched for cells with different drive strength and for the cells with different threshold voltage. Each cell was replaced and resized one after the other and timing check was done for each replacement. After completion of cell resizing and cell swapping, the timing reports were obtained. The reports satisfied the objective of reducing the minimum possible clock cycle for the system. The results of power and time optimization is given in the following chapter.

4. EXPERIMENTATION AND RESULT

This chapter discusses the experimental setup and results of the study. In order to understand the setup, it is required to know about the tool command language. The electronic automation tools such as Mentor Graphics ModelSim [6], Synopsys Design Compiler [8], and Synopsys PrimeTime [9] are designed in such a way that they are used with commands from the user. These commands are derived from a language named as Tool Command Language (TCL). All the commands needed to control the tools are put into a file and are sourced by the tools; these files are known as scripts.

The tools read and execute the scripts line by line. Once the analysis and changes are done, the tools can be directed to write the output files, and the reports of power and timing are generated. The following sections explain in detail the process of experimentation and show the results of this work.

4.1 Tool Setup and Experimentation

As explained earlier, the EDA tools are controlled by TCL scripts. This section explains the script similar to the ones used for Design Compiler [8] and PrimeTime [9] in this thesis work. Considering the basic script of Design Compiler [8], it can be divided into sections such as:

- Setting library variables
- Reading design files
- Setting environmental constraints
- Setting design rule constraints
- Setting design optimization constraints
- Compiling the design
- Obtaining reports and writing hardware file of synthesized design

Initially, the technology libraries and link libraries must be mentioned in the script. Technology library is the file that has the information of actual gates and flip-flops that are mapped to the modules in the input Verilog file. Link library is the file that has information of the hardware that is exclusive to this design such as a PLL and is used to resolve the references in the modules of the design. After mentioning the variables for technology and link libraries, the design is read into the tool. The reading process is executed in two steps. First, the design is copied into the memory of the tool and checked for syntax errors. Second, the design is elaborated, and all references are resolved.

After above step is completed, the user can specify constraints for the design. This step has three parts in it. First, the environment constraints are specified. They are the commands that tell the tool about the operating conditions, input load, output load, driving cells, and wire load model of the design under consideration. Second, the design rule constraints are specified that tell the tool about the maximal transition time (low-high and high-low) for a port or a design, maximum fanout value, and maximum capacitance value of the design or any specific part of the design. The standard cell library defines the voltage ranges within which the cells are to be in transition (i.e., 0.1v – 0.9v, 0.2v – 0.8v). Delay of library cells, output transition, and setup / hold times of sequential cells are dependent on these values. In all libraries, a cell input has a fan-out load value. In most cases, it is 1, but can be a different value. The maximum capacitance command limits the allowed capacitance on input, output, or bidirectional ports and/or designs. Third, design optimization constraints are specified. These are the constraints specifying the information of the clock, latency of the clock, uncertainty of the clock, transition of the clock, maximum area of the design, multi cycle paths, and false paths.

The tool tries to optimize the design only after it has achieved the specified design rule constraints. After giving the constrains, the design is compiled. This is the process that maps the design with actual gates and flip-flops from the technology file. The tool maps and optimizes for the constrains in this step. The compile

command is very flexible as it has many variants. Each variant of this command serves different purposes such as power-optimized compiling, clock-gated compiling, iterative compiling, and others. After compiling the design, the reports are generated. The commands for generating reports tell the tool to give a report of power, time, area, constraints, gated clocks, and more. From the reports, the user can analyze the results of the present design and its constraints. Then, the output files are written in any one of the formats among Verilog, system Verilog, and VHDL. Figure 4.1 gives an example of the script used for Design Compiler [8]. Now, to get the idea of script used for Synopsis PrimeTime [9], it can be categorized as follows:

- Read the design data
- Constrain the design
- Specify the environment and analysis conditions
- Check the design and analysis setup
- Perform a full analysis and examine the results

Figure 4.2 shows the flow and commands used in PrimeTime [9]. The commands and their meanings of Design Compiler [8] and PrimeTime [9] are almost similar except for a few of them. Additionally, there are commands for swapping the cells, up-sizing/down-sizing the cells, and for adding buffer/inverters into the design, which were used in this work to optimize the design for timing.

4.2 Experimentation Results

This section presents the results of changes made in the hardware of SCU to reduce power consumption and clock period. The first improvement made to the hardware is integration of ACG/OCG modules. Figure 4.3 shows the screen shot of Design Compiler-generated report of clock gating modules. After clock gating was implemented, multipliers were replaced with Double-word multiplier/divider modules that are capable of doing shift operations and also arbitrary multiplication/division operations. The power analysis report of this hardware is given in Table 4.1. The

module was synthesized with saed28rvt-ss0p7v25c library file, ss0p7v25c operating condition, low analysis effort, and with enclosed wire load model. Then, the sequential comparators were added to the circuit, for which the power report is given in Table 4.2. This module was synthesized with the same library file and operating conditions as before. The wire load model of the MSB comparator is chosen to be segmented type for better delay and power calculation. After the sequential comparator, operand isolation is integrated into the system and overall power optimization is concluded. The total power consumption and reduction due to these techniques as given by synthesis tool are given in Table 4.3. It must be noted that the dynamic power consumption value could reduce further when the module processes applications with higher percentage of multiplicands in the sequence of 2^n and comparison operations that are diagnostic with just MSBs.

It can be seen from the table that leakage power consumption has been increased slightly; the reason behind this is the use of low threshold voltage cells for reducing delay. As explained earlier, there is always a trade-off between power and speed. It is the design-point which decides whether power or speed should be given priority. In this case, the design under consideration is an execution unit, and therefore, speed gets the priority. Therefore, swapping a few HVT cells with LVT cells compromises leakage power, which is reflected in the table. The timing optimization techniques were integrated, and the resulting overall timing report is shown in Table 4.4. It can be seen that the clock period for the path with maximum data path delay is reduced significantly as aimed.

```
# Script file for constraining Adder16
set rpt_file "adder16.rpt"
set design "adder16"

current_design Adder16
source "${script_path}defaults.con"

# Define design environment
set_load 2.2 sout
set_load 1.5 cout
set_driving_cell -lib_cell FD1 [all_inputs]
set_drive 0 $clk_name

# Define design constraints
set_input_delay 1.35 -clock $clk_name {ain bin}
set_input_delay 3.5 -clock $clk_name cin
set_max_area 0

compile

if {[shell_is_in_xg_mode]==0}{
write -hier -o "${db_path}${design}.db"
} else {
write -format ddc -hier -o "${ddc_path}${design}.ddc"

source "${script_path}report.tcl"
```

Fig. 4.1. Snippet of TCL script for Design Compiler [8].

Table 4.1.
Power report of D-word shifter

Power type	**Value**
Cell Internal Power	14.1845 μW
Net Switching Power	1.7982 μW
Total Dynamic Power	15.9827 μW
Cell Leakage Power	26.8152 μW

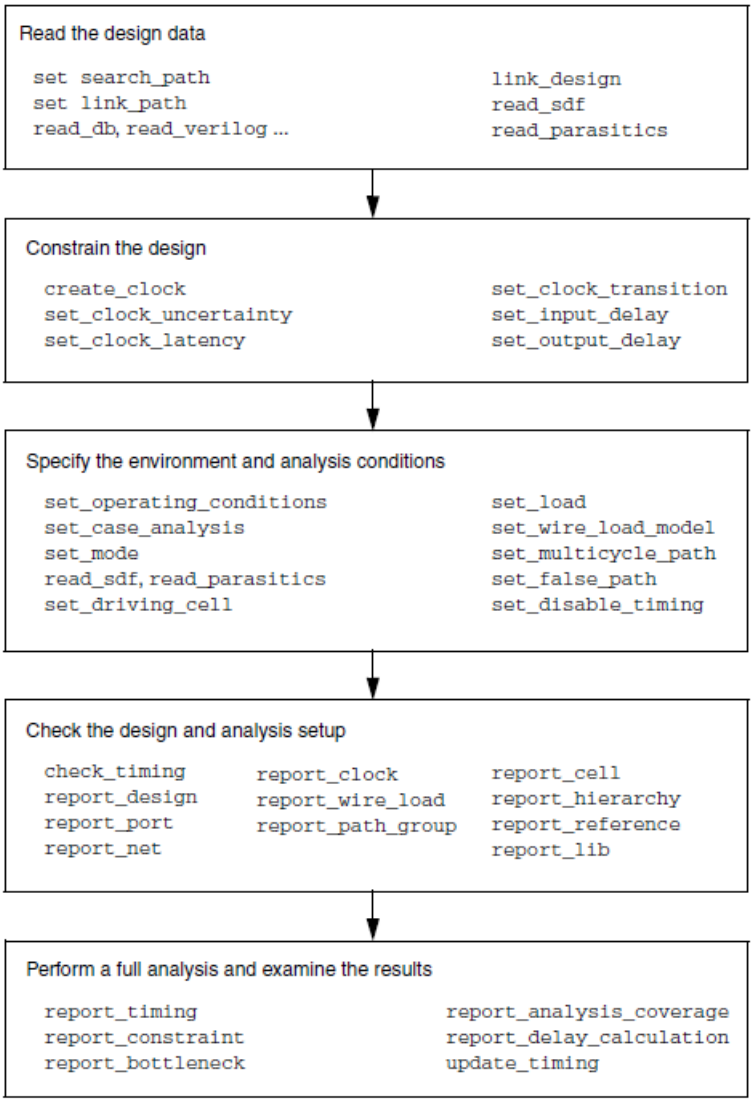

Fig. 4.2. Example and flow of TCL script for PrimeTime [9].

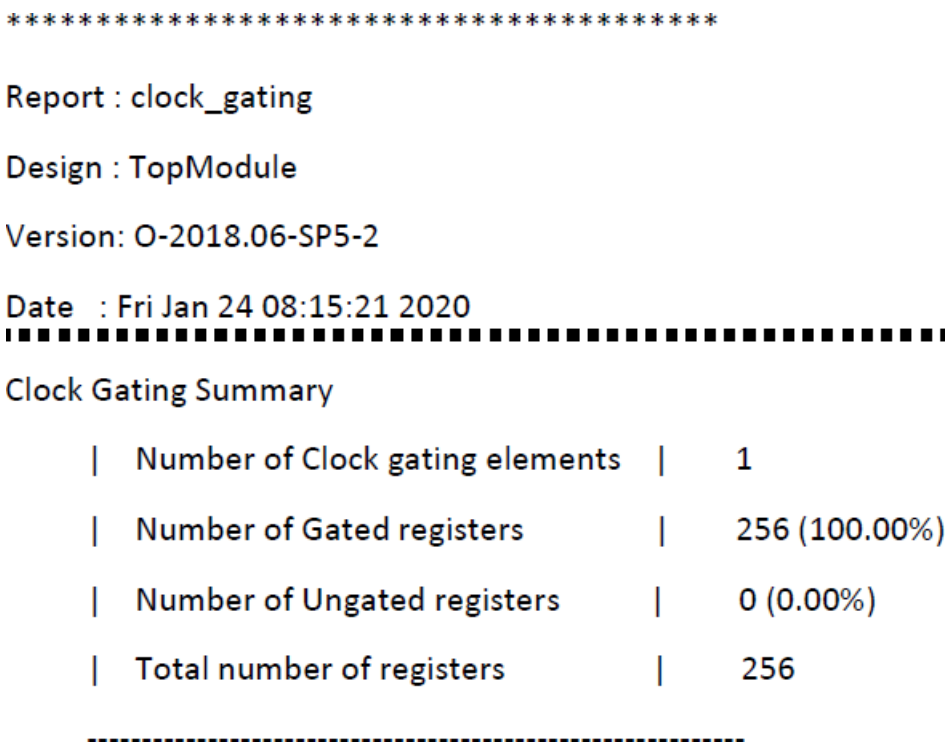

Fig. 4.3. Screen shot of clock gating report.

Table 4.2.
Power report of sequential comparator

Power type	**Value**
Cell Internal Power	27.3736 μW
Net Switching Power	2.1084 μW
Total Dynamic Power	29.482 μW
Cell Leakage Power	32.8137 μW

Table 4.3.
Power report of execution unit before and after optimization

Power type	**Before Optimization**	**After Optimization**
Cell Internal Power	66.0203 μW	62.7846 μW
Net Switching Power	7.9809 μW	5.1589 μW
Total Dynamic Power	74.0012 μW	67.9435 μW
Cell Leakage Power	1.3758e+08 pW	2.1906e+08 pW

Table 4.4.
Timing report of execution unit before and after optimization

Clock Period before Optimization	**Clock Period after Optimization**
55.26 ns	39.47 ns

5. SUMMARY

5.1 Thesis Conclusion

Symbiote Coprocessor Unit is an FPGA-based data stream coprocessor with high performance when compared with other software-based data stream processing methods. The objectives of this thesis study are (i) to pioneer SCU's ASIC implementation and (ii) to improve its timing and power requirements by adopting various optimization methodologies into the design. The following is a list of all implementations and modifications that are applied in this thesis work.

- Application specific integrated circuit implementation of arithmetic and logic unit of SCU. This is to accomplish the objective of pioneering the ASIC implementation of SCU.
- Integration of ACG and OCG modules. Standard cell library has enable FFs which are more efficient than these modules. However, the ACG and OCG modules were developed specifically considering that in the future they could be instantiated in a full custom design where the library used may not have enable FFs.
- Design and integration of double-word shifters instead of multipliers and dividers. Binary multiplication or division by a number in the sequence of 2, 4, 8, ... is just a left or right shift of the other operand by respective number of times. This process does not need multipliers and dividers that have power hungry XOR gates. Therefore, shifters were introduced into the circuit to carry out these processes.
- Design and addition of sequential comparators. Comparators are the modules that have a large number of XOR gates. However, many comparison results can be obtained with just comparing the MSBs of the operands.

- Integration of operand isolation. The signals to ALU do not arrive at the same time, which causes unwanted switching of ALU and provide temporary results that are not of interest. This unwanted switching of input pins of ALU is prevented if the operands are latched and synchronized before they reach the ALU inputs.
- Buffer insertion. Buffers or inverters are added into the interconnects to reduce the propagation delay.
- Alteration of cell sizes. This method decreases cell delay by choosing cells with higher driving strengths.
- Swapping of cells. Cells with low threshold voltage in non-critical paths are being swapped with high threshold voltage cells and vice versa to reduce delay.

The above-listed modifications were performed and analyzed iteratively. As a result, the new design attains 67.9435 μW of dynamic power as compared to 74.0012 μW before power optimization along with a small increase in static power, and 39.47 ns of clock period as opposed to 52.26 ns before time optimization.

The reason that the system could not achieve a very high reduction in power consumption is that, the modifications done in this work do not consider whether the tuples that would be entering the system are of temperature or of pressure or of any other data. If the system is chosen to be operated in a particular environment and expects a particular set of operations and tuples, then, it can be optimized exclusively to that environment and operations may get higher reduction in power consumption. The results of the optimization techniques used in this work were satisfying the objectives of study.

5.2 Future Work

This work is successful in implementing the VLIW execution unit of symbiote coprocessor unit in an application specific integrated circuit hardware. The ASIC perspective of SCU opens door for many future works such as:

- ASIC implementation of whole SCU with an FSM replacing MicroBlaze.
- Incorporation of more optimization techniques to reduce delay, power consumption, and area.
- Integration of Design For Test (DFT) elements such as scan FFs into SCU and testing.
- Placement and routing, clock-tree synthesis, and tape-out of SCU.

Note that the standard cell library used in this study was 28 nm technology. However, in the future, implementing SCU with this library may become obsolete as the current trend in industries is in the levels of 5 and 7 nm technologies. It must be noted that the results for the same modifications would change as the technology scales down. There are possibly many changes for this trend, among which a few are listed below. As the technology scales down:

- Channel length under the oxide layer of MOSFET decreases.
- Gate oxide thickness decreases, due to which, tunneling current increases which in turn increases overall leakage current.
- The width of depletion region between source/drain and the substrate decreases.
- Supply voltage decreases, which helps in decreasing delay of gate.
- Package density of integrated circuit increases.
- Frequency response improves.
- Driving current from sources to drain increases, which helps in decreasing delay.
- The saturation current also increases.

As a result of above-mentioned trends and other physical characteristics of scaled-down technology, the gate delay is expected to be reduced. However, the leakage power and dynamic power may not show a similar trend. Leakage power density increases as the gate is down scaled due to the increase in leakage current in smaller gates. Dynamic power would show a slight increase in its power density as technology is scaled down. These trends must be noted when the same design is being implemented with a different technology library in the future.

REFERENCES

REFERENCES

[1] D. Abadi *et al.*, "Aurora: A data stream management system," *in Proc. of ACM SIGMOD*, p. 666, 2003.

[2] M. A. Hammad *et al.*, "Nile: A query processing engine for data streams," *in Proc. of ICDE*, pp. 851–852, IEEE Computer Society, 2004.

[3] C. Cranor, T. Johnson, and O. Spatascheck, "Gigascope: A stream database for network applications," *in Proc. of ACM SIGMOD*, pp. 647–651, 2003.

[4] P. S. Vaidya, "Hardware-software co-designed data stream management systems," PhD thesis, Purdue University, West Lafayette, IN, USA, 2015.

[5] T. S. Alqaisi, "Microblaze-based coprocessor for data stream management systems," MSECE thesis, Purdue University, West Lafayette, IN, USA, 2017.

[6] MentorGraphics, "Modelsim user guide," 2012.

[7] S. Janwadkar. Very long instruction word processors. [Online]. Available: https://www.slideshare.net/shudhanshu29/vliw-processors. December 2004. Accessed: 04.06.2020

[8] Synopsys, "Design compiler user guide," vol. F-2011.09-SP2, 2011.

[9] SynopsysInc, "Primetime fundamentals user guide," vol. F-2011.12, 2011.

[10] Neeraj. VLSI universe: And/or type clock gating check. [Online]. Available: https://vlsiuniverse.blogspot.com/2014/05/clock-switching-and-clock-gating-checks.html. December 2014. Accessed: 04.06.2020

[11] Abram. Robotshop community: Electronics done quick 7 logic gates. [Online]. Available: https://www.robotshop.com/community/tutorials/show/electronics-done-quick-7-logic-gates. August 2019. Accessed: 04.06.2020

[12] Chegg study: Part 1 gate level 4 bit multiplier design. [Online]. Available: https://www.chegg.com/homework-help/questions-and-answers/part-1-gate-level-4-bit-multiplier-design-please-write-verilog-code-thank-q32638322. 2003. Accessed: 04.06.2020

[13] O. Emmanuel. Shift registers: Introduction, types, working and applications. [Online]. Available: https://circuitdigest.com/tutorial/what-is-shift-register-types-applications. January 2018. Accessed: 04.06.2020

[14] S. Churiwala and S. Garg, *Principles of VLSI RTL Design: A Practical Guide*, ser. SpringerLink: Bücher. Springer New York, 2011. [Online]. Available: https://books.google.com/books?id=gyDS6xiqnakC. Accessed: 04.06.2020

[15] T. Grzela. Trends in transistor gate delay switching time and interconnect delay in current IC, url = https://www.researchgate.net/figure/Trends in transistor gate delay switching time and interconnect delay in current IC.2015. Accessed: 04.06.2020,.

[16] P. Saxena, N. Menezes, P. Cocchini, and D. A. Kirkpatrick, "Repeater scaling and its impact on CAD," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 4, pp. 451–463, 2004.

[17] D. Harris and S. Wimer. VLSI interconnects credits. [Online]. Available: https://slideplayer.com/slide/17905384. December 2010. Accessed: 04.06.2020

[18] Electronicsforu. MOSFET: Basics, working principle and applications. [Online]. Available: https://www.electronicsforu.com/resources/learn-electronics/mosfet-basics-working-applications. April 2019. Accessed: 04.06.2020

[19] VLSIexpert. VLSI concepts: An online information center for all who have interest in semiconductor industry. [Online]. Available: http://www.vlsi-expert.com/2014/01. January 2014. Accessed: 04.06.2020