

# Simulation and Emulation of Digital Circuits and Memory Ports

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS  
FOR THE AWARD OF THE DEGREE  
OF

MASTER OF TECHNOLOGY  
IN  
Control and Instrumentation

Submitted by:

**PERRY MITTAL**

2K17 / C&I / 13

Under the supervision of

**Prof. Madhusudan Singh**  
Electrical Engineering Department, DTU  
&

**Mr. Rohit Goel**  
Staff Engineer and Manager, Mentor Graphics



**DEPARTMENT OF ELECTRICAL  
ENGINEERING**

**DELHI TECHNOLOGICAL UNIVERSITY**

(Formerly Delhi College of Engineering)

Bawana Road, Delhi-110042

2019

**DEPARTMENT OF ELECTRICAL  
ENGINEERING  
DELHI TECHNOLOGICAL UNIVERSITY**  
(Formerly Delhi College of Engineering)  
Bawana Road, Delhi-110042

**CANDIDATE'S DECLARATION**

I, Perry Mittal, Roll No. 2K17/C&I/13 student of M. Tech. (Control and Instrumentation), hereby declare that the thesis titled “**Simulation and Emulation of Digital circuits and memory ports**” which is submitted by me to the Department of Electrical Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology in Control and Instrumentation, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma Associateship, Fellowship or other similar title or recognition.

Place: Delhi

Date: 21.07.2019

**(PERRY MITTAL)**  
**(2K17/C&I/13)**

**DEPARTMENT OF ELECTRICAL  
ENGINEERING  
DELHI TECHNOLOGICAL UNIVERSITY**  
(Formerly Delhi College of Engineering)  
Bawana Road, Delhi-110042

**CERTIFICATE**

I hereby certify that the Project Dissertation titled “**Simulation and Emulation of Digital circuits and memory ports**” by Perry Mittal, Roll No. 2K17/C&I/13 Electrical Engineering Department, Delhi Technological University, Delhi in partial fulfillment of the requirements for the award of the degree of Master of Technology, in Control and Instrumentation is a record of project work carried out by the student under our supervision. To the best of my knowledge this work has not been submitted in part or full for award of any Degree or Diploma to this University or elsewhere.

Place: Delhi

Date: 21.06.2019

**(Prof. Madhusudan Singh)**  
**Supervisor**

Professor

Electrical Engineering Department

Delhi Technological University

**(Mr. Rohit Goel)**  
Staff Engineer, Mentor Graphics

## ACKNOWLEDGEMENT

I would like to express my gratitude towards all the people who have contributed their precious time and efforts to help me, without whom it would not have been possible for me to understand and complete the project.

I would like to specially thank **Prof. Madhusudan Singh**, Department of Electrical Engineering, my project supervisor for his kind, support, motivation and encouragement throughout the period of this work was carried out. His readiness for consultation at all times, his educative comments, his concern and assistance even with practical things have been invaluable.

I would like to thanks **Mentor Graphics** and its staff members for co-operating with me and providing me with the best possible working environment for the projects.

I would like to thank all the faculty members of department of Electrical engineering, DTU, Delhi for their constant encouragement.

I would like to thank all my M.Tech friends.

Place: Delhi

(**PERRY MITTAL**)

Date: 21.07.2019

## ABSTRACT

The Simulation and Emulation of the digital circuits are discussed in this present work in which there is an enhancement to increase the capacity and performance of the digital chips and ICs where it reduces its size with help of the memory inference in complex circuits, the number of multiplexers, adders and subtractors are replaced with the memory block which reduces the area of the digital ICs and the capacity of the digital circuit is getting enhanced.

It also reduces the compilation time of the simulating circuits during gate level and behavioral level synthesis of the circuit. The importance of the work is that all the circuits and ICs can be tested in fraction of seconds using this optimization which reduces a lot of money and time consumption of an organization. The testing of this optimization is done by simulating the module where the analysis of the various scenarios is done on simulator.

The optimization is further tested on emulator where the replica of the module can be created, the emulator can act as the same piece of hardware where the compilation time is reduced.

Methodology adopted is Verilog and VHDL the hardware descriptive language is used. All the scripting of the various scenarios is done with shell scripting. The coding for the optimization of memory ports is written in C/C++ in which all the read and write ports are analyzed.

In memory optimization enhancement various tools for simulation like Questa and precession are used for simulating all the modules of the hardware descriptive Language.

Further, for creating the Real Replica of the hardware, emulator is used where the Register transfer level compiler and Emulator (Veloce) are used of validating all the results in velcomp flow where the quad core processor is used , all the modules are compiled in the parallel manner . It makes the system for informative in which core processor, it is compiling all the Modules with high speed and efficiency. So this optimization reduces the speed of the simulator and emulators in terms of the area and the compilation time of the gate level synthesis where the register transfer level design is transformed into logic gates.

# TABLE OF CONTENTS

## CHAPTER 1: INTRODUCTION

1.0: INTRODUCTION .....	1
1.1: DESIGN OF DIGITAL SYSTEM .....	1
1.2: TEST BENCH IN VERILOG .....	2
1.3: SIMULATION.....	2
1.4: COMPILATION AND SYNTHESIS	
1.4.1: ANALYSIS.....	3
1.4.2: GENERIC HARDWARE GENERATION .....	3
1.5: LOGIC OPTIMIZATION.....	4
1.6: BINDING.....	4
1.7: ROUTING AND PLACEMENT.....	6
1.8: TIMING ANALYSIS.....	6
1.10: HARDWARE GENERATION.....	6
1.11: REGISTER TRANSFER LEVEL DESIGN IN VERILOG.....	7
1.11.1: RTL LEVEL DESIGN.....	7
1.11.2: CONTROL OR DATA PARTITIONING.....	7
1.12: OBJECTIVE OF THE PRESENT WORK.....	8
1.13: OUTLINE OF THE THESIS.....	8

## CHAPTER 2: EMULATION PLATFORM (VELOCE)

2.0: INTRODUCTION .....	9
2.1: EMULATOR.....	9
2.2: DESIGN AND VERIFICATION.....	9
2.4: MENTOR GRAPHICS VERIFICATION PLATFORM.....	10
2.4.1: VIRTUAL PROTOTYPING .....	10
2.4.2: FORMAL VERIFICATION.....	10
2.4.3: SIMULATION .....	11
2.4.4: EMULATION.....	11

2.5: VELOCE THE COMPLETE PLATFORM.....	12
2.6: VELOCE2 EMULATOR.....	13
2.7: VELOCE TEST BENCH XPRESS.....	13
2.8: VELOCE IMPROVES VERIFICATION.....	14
2.8.1 COMPILE.....	14
2.8.2 RUN.....	14
2.8.3 DEBUG.....	15
2.9: VELOCE ACCELERATING SIMULATION.....	16
2.10: CONCLUSION.....	17

### **CHAPTER 3: REGISTER TRANSFER LEVEL COMPILER FLOW**

3.0: INRODUCTION .....	18
3.2: ELABORATION .....	18
3.3: RTLC-VLE SYNTHESIS ENGINE.....	19
3.4: CDFG.....	19
3.5: CDFG EXAMPLE.....	20
3.6: TECHCELL FLOW.....	21
3.7: FLATTENING SUPPORT.....	22
3.8: DESIGNWARE INTEGRATION WITH RTLC.....	23
3.9: EXAMPLE OF DESIGN WARE INTEGRATION.....	24
3.10: EXAMPLE OF DESIGNWARE COMPONENTS.....	25
3.11: TRADIONAL ENCRYPTION TECHNIQUES.....	26
3.12: MEMORY WITH SYNC/ASYNCR SET RESET.....	28
3.13: PRAGMA CONTROL.....	31

## **CHAPTER 4: OPTIMIZATION OF MEMORY PORTS**

4.0: INTRODUCTION .....	32
4.1: CONVERSION .....	32
4.2: ALGORITHM AND APPROACH.....	33
4.3: ACCESSING MEMORY BLOCK.....	34
4.4: VALID CASES WHEN CONVERSION OCCURS.....	35
4.5: HANDLING OF VARIOUS OPERATORS.....	36
4.6: ENROLLING OF FOR LOOP.....	40
4.8: CASES CONVERSION NOT OCCUR.....	43
4.9: HANDLING OF TASK AND FUNCTION.....	49
4.10: CONCLUSION.....	50

## **CHAPTER 5: RESULTS AND CONCLUSION**

5.0: INTRODUCTION.....	51
5.1: FORMATION OF LUTS AND MULTIPLXERS.....	52
5.2: CONVERSION OF FOR TO IF STATEMENT.....	54
5.3: COMPARATIVE ANALYSIS OF QUESTA AND RTL.....	56
5.4: CONCLUSION.....	58

## **CHAPTER 6: MAIN CONCLUSION AND FUTURE SCOPE OF WORK**

6.0: MAIN CONCLUSION .....	59
6.1: FUTURE SCOPE & WORK.....	60
6.2: COMING EMULATORS AND SIMULATORS.....	60

<b>REFERENCES.....</b>	<b>61</b>
------------------------	-----------



## LIST OF FIGURES

Fig. No.	Name of Figure	Page No.
Fig. 1.1	Design Flow	1
Fig. 1.2	Test Bench or a Waveform editor for simulation	3
Fig.1.3	Compilation and synthesis process	5
Fig.1.4	Synthesis Run	6
Fig.1.5	Timing analysis	8
Fig.1.6	Control or data Partitioning	10
Fig.2.1	Verification Time spent	18
Fig.2.2	Verification Infrastructure	20
Fig.2.3	Veloce Verification system	22
Fig.2.4	Co Modelling	24
Fig.2.5	Veloce Flow	26
Fig.2.6	Design Under Test	27
Fig.2.7	Software Debug Solutions for Veloce	28
Fig.2.8	JTAG Probe	29
Fig.3.1	Analysis and Synthesis for RTL Flow	30
Fig.3.2	Elaboration	26
Fig.3.3	CDFG	27
Fig.3.4	CDFG Example 2	28
Fig.3.5	CDFG Example 3	29
Fig.3.6	Techcell Flow	30
Fig.3.7	Example for Design Ware Integration	31
Fig.3.8	Traditional Encryption Techniques	32
Fig.3.9	Memory with sync-async reset	33
Fig.3.10	Cross sharing	34

Fig.4.1	Ram block memory	39
Fig.4.2	Accessing a Memory Block	42
Fig.4.3	CDFG	44
Fig.4.4	The Cdfg enrolling for the n block data	45
Fig.4.5	The Cdfg without enrolling	46
Fig.4.6	Valid Cases for Conversion	47
Fig.4.7	Example of a for if Conversion	48
Fig.5.1	Testcase For Ram Block	49
Fig.5.2	Testcase For Logic Block	50
Fig.5.3	Precession Tool without optimization	51
Fig.5.4	Ram Block	52
Fig.5.5	The Precession tool with optimization	53



# CHAPTER 1

## INTRODUCTION

### 1.0 INTRODUCTION

In the design flow, design specifications are introduced . Specifications provides functionality interface and overall architecture of the designed digital circuit . A behavioral description is made to analyze the design with respect to functionality and performance. Behavioral descriptions are often written in Verilog, VHDL and system Verilog . EDA tools have appeared to simulate behavioral description of digital circuits, these tools have merged the powerful concepts from HDLs and object oriented programming like C/C++. The behavioral description is converted into RTL description in an HDL. The logic synthesis tools transforms the RTL description into gate-level netlist describes the circuit in terms of gates and connections between them. The synthesis tool makes that the gate level netlist meets the timing area and power specifications in Fig 1.1. The gate level netlist is put into an automatic place and route tool, then a layout is created, then the layout is fully verified and fabricates on a digital chip .

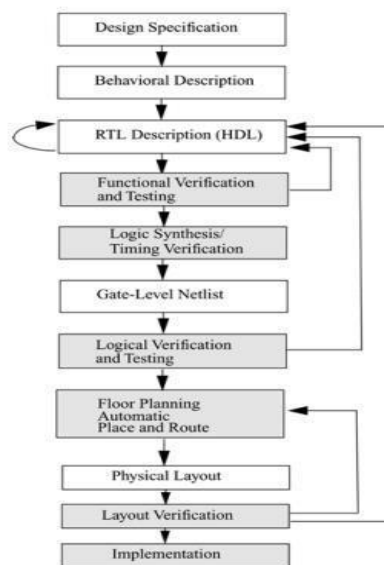


Fig 1.1 Design Flow for various digital circuits

## **1.1 DESIGN OF DIGITAL SYSTEM**

The digital design system is described in Verilog because high level Verilog designs are usually described at the level that consist of system registers and transfer of data between various registers through buses, this level of high level description is called as register transfer level (RTL). Verilog construction used in RTL level design are procedural statements continuous assignments and instantiation statements.

## **1.2 TESTBENCH IN VERILOG**

A digital system designed in Verilog should be simulated and tested for functionality before it is turned out into a hardware while simulation design errors and incompatibility of components in the digital design, all are detected. In simulation, a digital design need generation of test data and observation of all simulation results. This process can be done by use of a Verilog module that is called as a test bench. A Verilog test bench uses HDLs constructs for generation of data monitoring of response and even handshaking of the design. In the test bench, the design is instantiated which is being simulated the test bench together with digital design, forms a model of simulation which is used by Verilog simulation.

## **1.3 SIMULATION**

Mentor Graphics uses “QUESTA” as a simulating tool, simulation of a design requires testing of data, and test data can be generated graphically using editors for wave form or with the help of a test bench. For simulating with the help of a Verilog test bench, the test bench instantiates the design under test (DUT) and as a part of the test bench, it applies test data to the instantiated circuit in Fig 1.2.

Verilog code of a digital circuit and its test bench and the result of simulation are in the form of waveform. Simulation validates the functionality of the digital circuit being tested the timing diagram that the circuit output changed with the rising edge of the clock and no gate delays and the propagation delays are shown in the timing diagram.

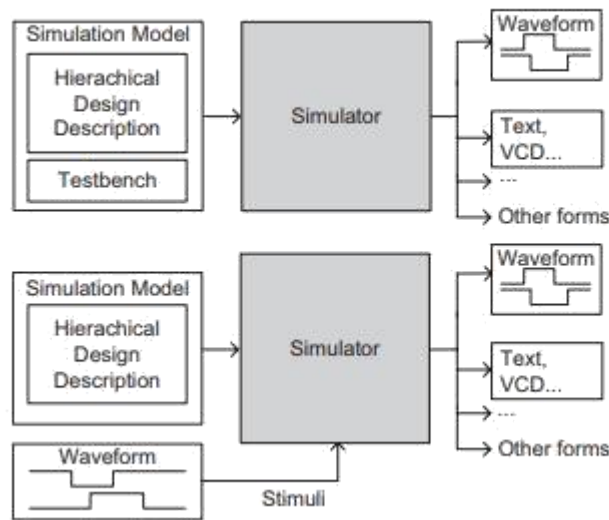


Figure 1.2. Test bench or a waveform editor for simulation

## 1.5 COMPILATION AND SYNTHESIS

Synthesis is the process of transforming the HDL design into gate level netlist, a Verilog representation for synthesis cannot include signal and gate level timing specifications that do not translate to sequential or combinational logic equations, Verilog representation for thesis should follow certain styles of coding for combinational and sequential circuits for example: we have to specify an Application specific integrated circuit(ASIC) or field programmable gate array(FPGA) as our target hardware. When the hardware with detailed timing and specifications become available to the compilation process. The compilation process translates various parts of design to an analysis phase and generates synthesis phase and places and routes components of the target hardware and generates timing details.

### 1.5.1 ANALYSIS

Before the complete design emerged into hardware, the design must be analyzed properly a uniform format must be generated for the complete design, this analysis space also checks the syntax and semantics of the input Verilog code.

### 1.5.2 GENERIC HARDWARE GENERATION

After obtaining uniform representation of all the components of design, the synthesis begins its operation by turning the design into a generic hardware, such as a set of Boolean expressions or a netlist of gates.

## 1.6 LOGIC OPTIMIZATION

This phase of synthesis after a design is converted into a set of Boolean expressions or basic gates is called the optimization phase in Fig 1.3, this phase is responsible for reducing expressions with continuous input and removing redundant logic expressions output of this phase is in the form of Boolean expressions, logic representations or gate netlist.

## 1.7 BINDING

After the logic optimization phase, the synthesis uses information from the given hardware to decide exactly what logic elements are required for the realization of the circuit, this process is called the binding as shown in Fig 1.3. And its output is ASIC or custom IC.

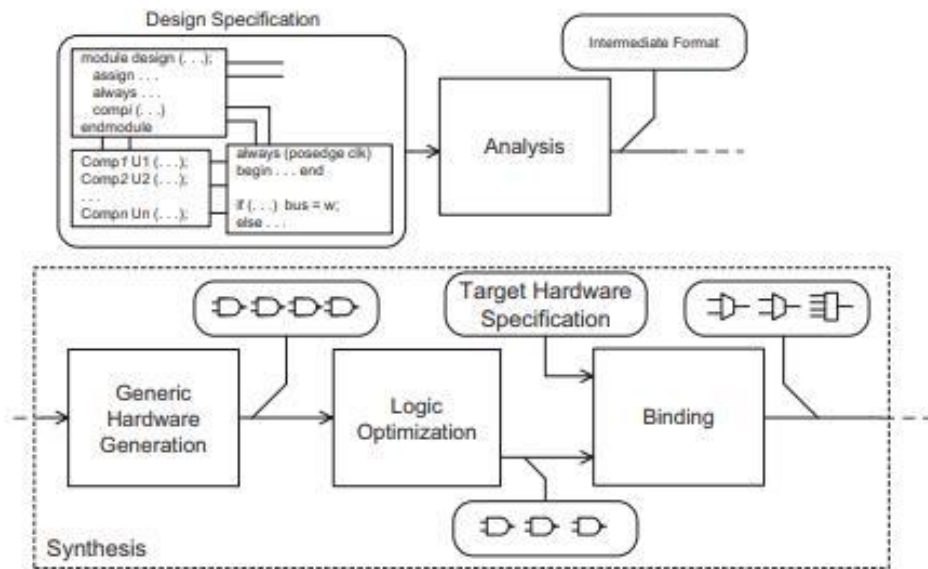


Figure 1.3. Compilation and synthesis process of HDL modules

An example of synthesis the counter circuit used in the simulation is being synthesized, the Verilog description of the design is converted into gates and flip-flops using the synthesis tool, the output of the synthesis is the gates and the flip-flops and their inter connections, this graphical representation of the output, that is generated by the synthesis tool of Altera's Quartus 2 in Fig 1.4 depicts the complete flow for the Synthesis Run to synthesize the module in Verilog..

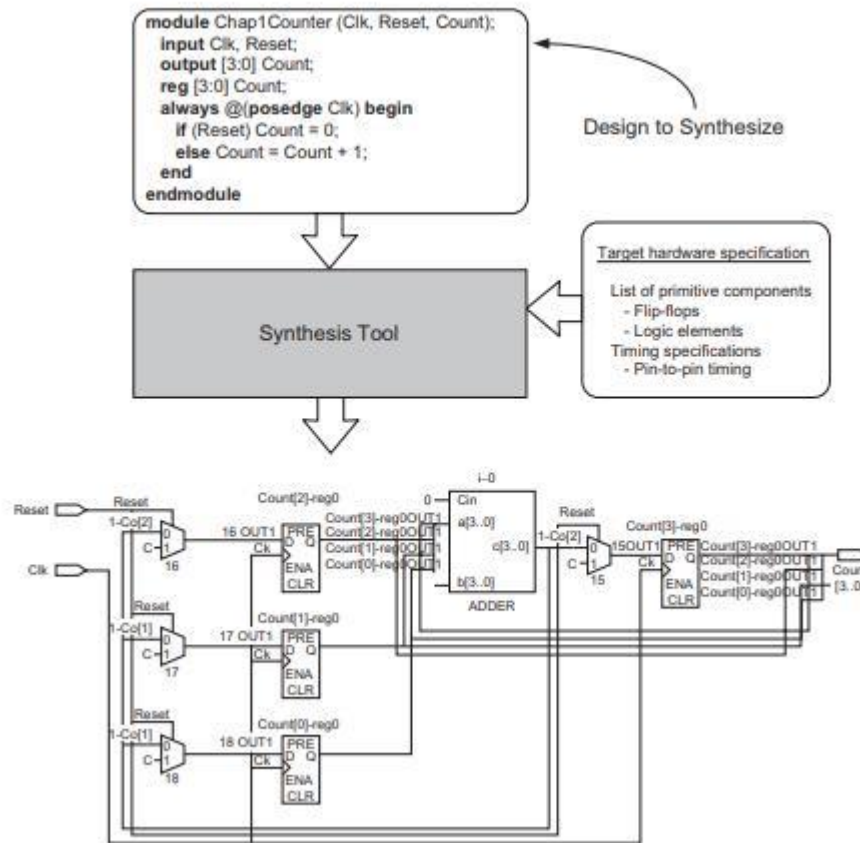


Figure 1.4. Module Compilation and synthesis run of hardware circuits.



## 1.8 ROUTING AND PLACEMENT

Routing and placement decides placement of cells of the target hardware, the inputs and outputs of these cells through wiring and switching areas of the target hardware is given by routing and placement phase in Fig 1.5. The output of this phase is specific to the hardware being used for the manufacturing of an ASIC.

## 1.9 TIMING ANALYSIS

There is a timing analysis phase after the compilation process, this phase generates worst case delays, clocking speed and delay from one gate to another, setup time and require whole time in Fig 1.5. Designers use these information to decide the speed of the clock or more precisely the speed of the circuits.

## 1.10 HARDWARE GENERATION

This is the last stage of the Verilog based design in case of hardware generation, this stage generates a netlist for application specific integrated circuit(ASIC) a program for programming field programmable logic devices(FPLDs) or layout of custom IC cells or layout of custom IC cells in Fig 1.5.

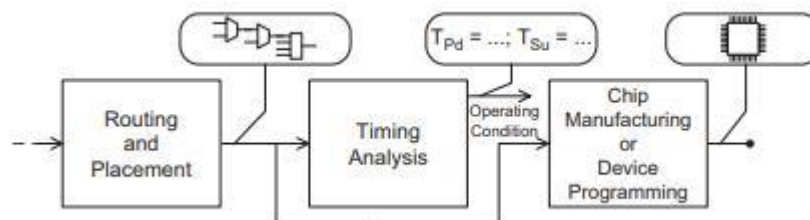


Figure 1.5. Routing and placement of the chip with Timing Analysis.

## 1.11 REGISTER TRANSFER LEVEL DESIGN WITH VERLOG

The register transfer level (RTL) design for digital systems gives us how the Verilog is used for description testing synthesis of various RTL level components of a digital system. The RTL level design and how a complete system is put together at this abstraction level. The basic structure of the Verilog such as modules, ports, utilities for verification of digital components are introduced in this RTL level design.

### 1.11.1 RTL LEVEL DESIGN

Design of hardware components are done by representing the hardware for synthesis and implementing the design by CAD tools. A large design requires planning design and partitioning before its various parts can be represented in Verilog for synthesis in Fig 1.6 shows the RT Level Design.

### 1.11.2 CONTROL OR DATA PARTIONING

The RTL level design is the partitioning of the digital design into control and data part, the control part is a state machine generating control signals that control the flow of data in the data part and the data part consist of the data components in Fig 1.6 shows the Control and data partitioning.

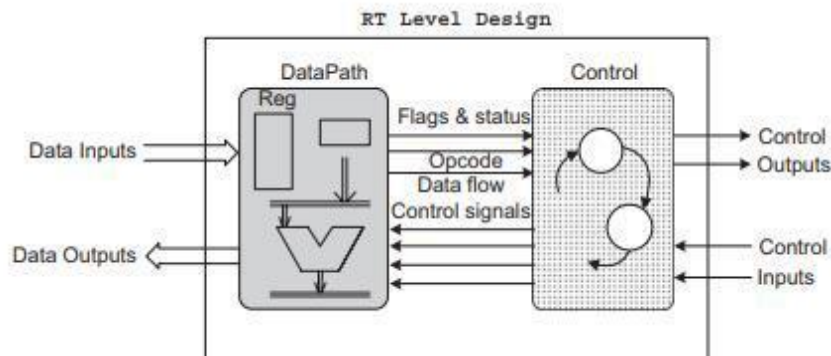


Figure 1.6. Control or data partitioning of RT Level Design

## **1.12 OBJECTIVE OF THE PRESENT WORK**

1. The verification of digital chip and circuits to optimize the area, capacity and performance of the digital chips and ICs
2. The optimization of the memory ports and reducing the area of the digital circuits and ICs is done by inferring memory into it. This optimization also reduces the compilation time and the area of the circuits.
3. In the present work, it is tried to optimize the adders, muxs into the memory declaration.
4. The testing of the optimization of digital circuits and ICs, using the mentor simulation software named questa and the emulation software named veloce.

## **1.13 OUTLINE OF THE THESIS**

This dissertation includes total 6 chapters:

1. Chapter 1 gives the brief introduction about the verification of digital design circuits, the hardware languages like Verilog and VHDL help in generating the testbench by which it can verify the circuits and help in creating the Register Transfer level .
2. Chapter 2 gives the brief introduction about the Mentor graphics Hardware model veloce, it works to test millions of chips in very less time frame and with accurate precision with the use of memory optimization in Emulator.
3. Chapter 3 gives the brief introduction about the Register transfer level compiler flow which generates the graph level optimization where it can mark the memory in order to decrease the size of the chip to increase its capacity and performance.
4. Chapter 4 gives the brief introduction about the optimization of memory ports , sometimes due to large no. of read and write data memory is not marked , but using the for to if optimization the memory is marked and compilation time is reduced.
5. Chapter 5 gives the brief introduction about the comparison between the RTLC and questa like the simulation model and the emulation model both provide the same results or not.
6. Chapter 6 gives the brief introduction about the Future scope and the work, the tasks and the function and the generate block will be supported with memory declaration.

# CHAPTER 2

## EMULATION PLATFORM (VELOCE)

### 2.0. INTRODUCTION

The emulation is basically a hardware which acts exactly like something which want to test by all the rules of the system being emulated. The Veloce is used for the verification of system on chips (SoCs) and is a core technology by Mentor Graphics. It provides hardware architecture innovative operating system and versatile peripheral solutions to provide high speed, high capacity and verification of the design and Veloce accelerates the simulation and used for the hardware debugging.

### 2.1. EMULATOR

Everything in a world of system on chip in which we have processors like embedded CPUs, GPUs and MMUs. Memory like SDRAM, DDRAM, and cache memory. Peripherals like multiple IP blocks and protocols and software like instruction sets and operating systems, drivers and application software's. So these SoCs designs gave the challenges for efficient verification.

### 2.2. DESIGN AND VERIFICATION

System on chips (SoCs) makes existing verification challenges more difficult, a large amount of verification time spent on running stimulation in terms of speed and capacity. In debugging in terms of power and predictability. In test bench development in stimulus coverage and reuse and test planning in terms of metrics analysis and processes in Fig 2.1 shows how much time is spend on Debugging and simulation.

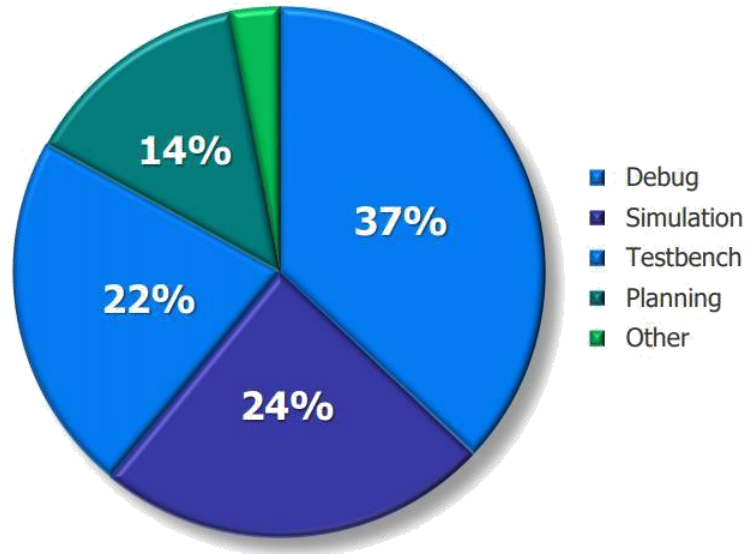


Figure 2.1. Verification time spent on complete design flow

## 2.3. MENTOR GRAPHICES VERIFICATION PLATFORM

It works on the faster, smarter and stronger verification.

### 2.3.1. VIRTULA PROTOTYPING

It involves modelling of a system simulating and visualizing its behavior under real world operating conditions and refining its design through iterative process. Virtual prototyping to build and test prototypes and realistically simulate them on their computers both visually and mathematically. Users can explore multiple design variations testing and refining until system performance is optimized this can help reduce the time and cost of new product development which significantly improves the quality of overall design in Fig 2.2 shows the virtual prototyping.

### 2.3.2. FORMAL VERIFICATION

It is a process of checking the design with respect to certain properties a formal verification tool examines the design to make sure that given properties holds true under all conditions if the input conditions make a property are regarded as property counter examples, property coverage indicates how much of the complete design is exercised by the property in Fig 2.2 there is formal verify.

### 2.3.3. SIMULATION

It is a process of using a simulation software or simulator to verify the functional correctness of the digital design that is modeled using a hardware descriptive language HDL like Verilog in Fig 2.2.

### 2.3.4. EMULATION

Emulation is a system that acts exactly like something else. The emulation model is usually based on hardware description language like Verilog and VHDL as a source code which is compiled into the format used by emulation system. The goal is normally debugging and functional verification the system being designed, an emulator is fast enough to be plugged into a working target system. In place of a yet to be built chip. So that whole system can be debugged with live data. This is a case of hardware emulation in Fig 2.2.

### 2.3.5. FPGA PROTOTYPING

FPGA prototyping is a technique for verifying the functionality and performance of application specific integrated circuit (ASICs) and System on chips by porting their RTL to a field programmable gate array. FPGA, it is being used more widely because hardware complexity is increasing and the amount of related software that needs validating is rising. It gave us the benefit in terms of performance cost, infrastructure, portability and availability in Fig 2.2.

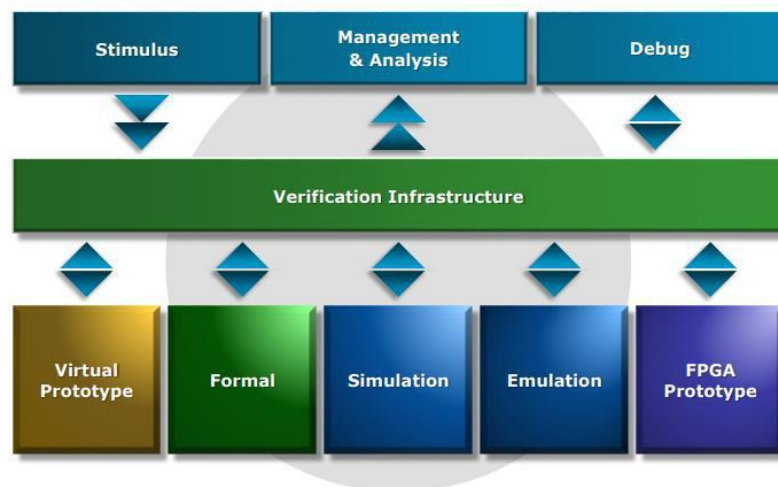


Fig. 2.2 Verification Infrastructure of stimulus and debugging

## 2.4. VELOCE THE COMPLETE VERIFICATION PLATFORM

Veloce strato platform has sufficient execution speed, full visibility, capabilities and ease of use. In model creation and Veloce power application also boost the run time and performance of the power flow upto 4.5x. Veloce virtual peripherals and host devices make the emulator a shared resource for multiple hardware and software engineers. The Veloce emulation platform reduces the risk in the verification of today’s complex SoCs and is the core technology of mentor verification platform. The Veloce emulator accelerates clock and full SoC RTL simulations during all the phases of design process. Test bench xpress(TBX) co modelling software makes Veloce an ultra-fast verification engine up to 10,000 times faster than software simulators significantly reducing development schedule risks.

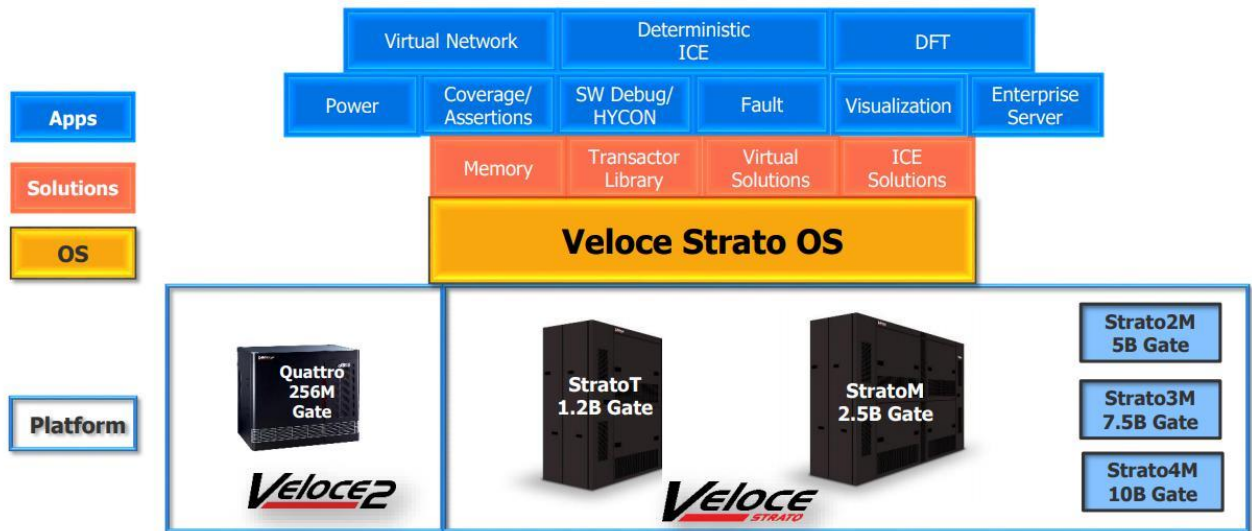


Figure 2.3. Veloce verification system of visualization and assertions

Veloce strato platform is fully scalable, it is engineered to scale to support 15 billion gate designs and capable of verifying the largest chips ever designed in terms of capacity and their design size grow in Fig 2.3 shows the verification complete flow.

## 2.5. VELOCE2 EMULATOR

The Veloce2 Emulator increases the full System on chip (Soc) RTL simulations during all phases of the design. It makes pre silicon testing and debug at hardware speeds, using real world data, both software and software designs. It improves end product quality by increasing the total verification cycles on the design before committing to silicon prototypes. It is a scalable verification platform with capacities from 16 million to 2 billion gates. Veloce2 significantly reduces the verification of the largest hardware and software systems.

## 2.6. VELOCE TESTBENCH XPRESS

The test bench Xpress (TBX) co modelling software makes the Veloce emulator, an ultra-fast transaction level modelling and verification engine. The Veloce emulator and the Veloce TBX reduce the risk while leveraging transaction models used during simulation. In co-modelling, the test benches are interfaced to synthesizable trans actors are put together with the DUT in Veloce TBX automatically generate a direct communication interface between c/c++ or system c environment on a host and the SoC(system on chip) DUT in the Veloce 2 emulator in Fig 2.4.

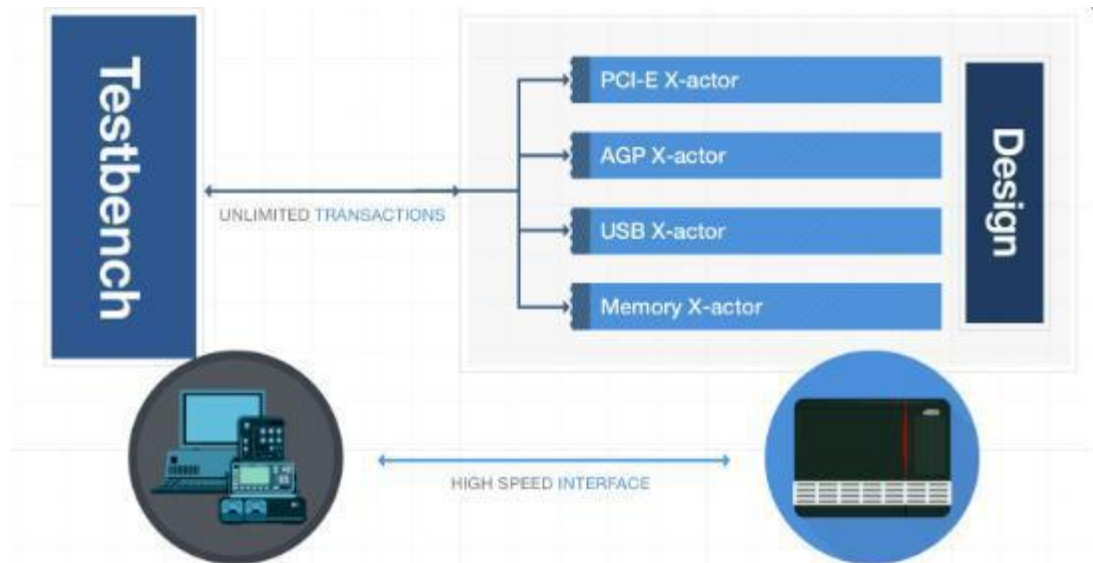


Fig 2.4 comodelling between the testbench and design



## 2.7. VELOCE IMPROVES VERIFICATION AND PRODUCTIVITY

### 2.7.1. COMPILE

It provides fast compiling up to 300 MG/hr. and it also provides fast downloading of the digital design (2min/2BG) in Fig 2.5.

### 2.7.2. RUN

It has very fast and high bandwidth interface between workstation and Veloce, it is up to 40 physical interface links/2BG in Fig 2.5.

### 2.7.3. DEBUG

It provides and 100% visibility, 1M cycle/2BG design in 5 minutes

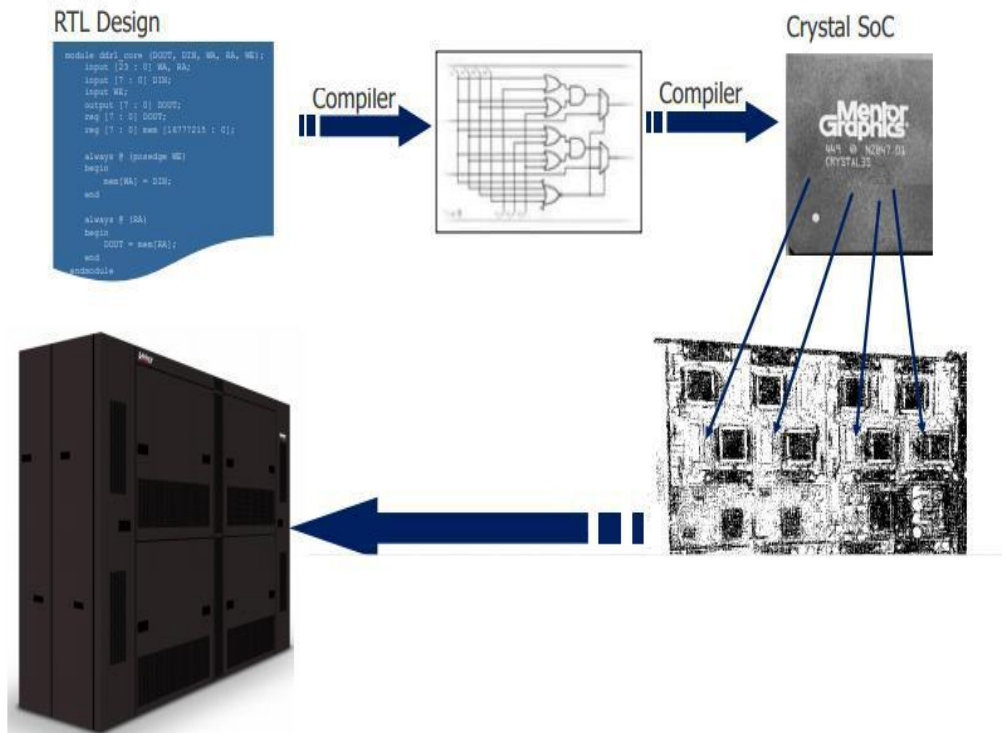


Fig 2.5 Veloce Flow of RTL design and Crystal Soc

## 2.8. VELOCE ACCLERATING BASED SIMULATING DESIGN

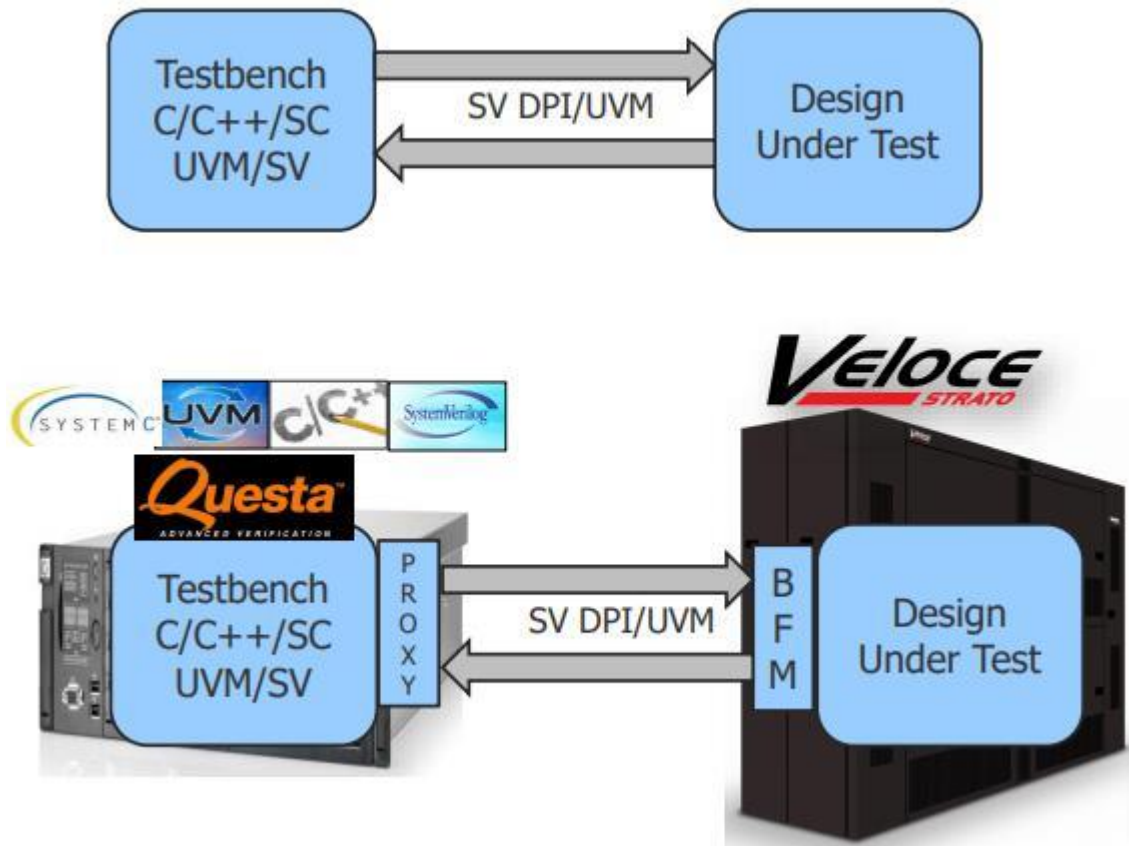


Fig 2.6 Design under Test of the Testbench including C/C++

The Veloce2 Emulator increases the full System on chip (Soc) RTL simulations during all phases of the design. It makes pre silicon testing and debug at hardware speeds, using real world data, both software and software designs. It improves end product quality by increasing the total verification cycles on the design before committing to silicon prototypes. It is a scalable verification platform with capacities from 16 million to 2 billion gates. Veloce 2 significantly reduces the verification of the largest hardware and software systems in Fig 2.6.

## 2.10 SOFTWARE DEBUG SOLUTIONS FOR VELOCE



Fig 2.8 Software debug solutions for Veloce

The emulation is basically a hardware which acts exactly like something which we want to test by all the rules of the system being emulated. The Veloce is used for the verification of system on chips (SoCs) and is a core technology of Mentor Graphics. It provides hardware architecture innovative operating system and versatile peripheral solutions to provide high speed, high capacity and verification of the design and Veloce accelerates the simulation and used for the hardware debugging in Fig 2.8.

System on chips (SoCs) makes existing verification challenges more difficult, a large amount of verification time spent on running stimulation in terms of speed and capacity. In debugging in terms of power and predictability. In test bench development in stimulus coverage and reuse and test planning in terms of metrics analysis and processes.

## 2.11 JTAG PROBE

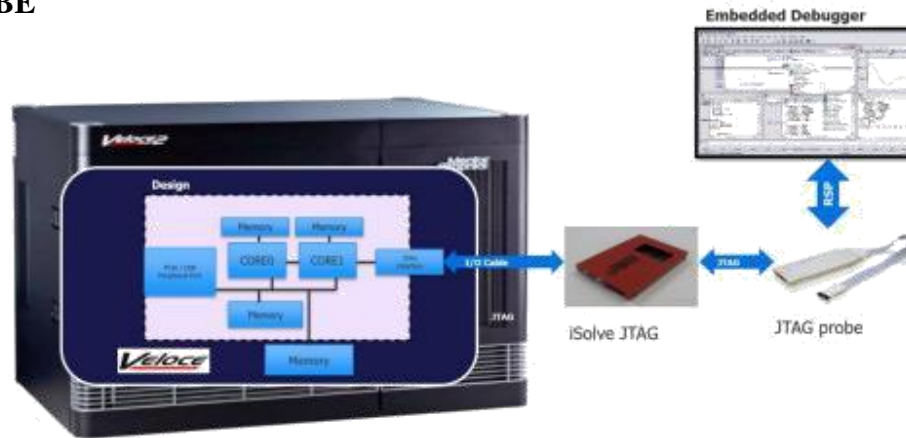


Fig 2.9 JTAG probe and logic analyzer

A boundary –scan (JTAG) based simple logic analyzer and circuit debugging software which test on chip internal logic, e.g debug CPLD firmware .No special knowledge is required to use boundary –scan technology as JTAG Probe in Fig 2.9.

## 2.12 DESIGN OF MEMORY PORTS IN VELOCE

The Veloce is multicore processor in which all the designs and processors work parallelly with 8 times speed then the simulator .the memory ports can be designed with the inference of the memory declaration. With this optimization the memory inference cen be increased and the Memory ports can be decreased using the one hot logic technology.

## 2.13 CONCLUSION

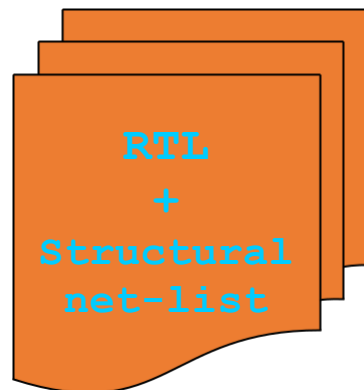
The following chapter discusses the emulator product named veloce, how this emulator work. The specifications of the veloce, how veloce will able to resolve the problems related to simulator and what is need for an emulator. The efficiency and precesion of the emulator. The Emulator is like a replica of the actual hardware and the chip in which we can test the things in real world which is not handled in simulator. So the Emulator is able to decarese the compilation time and improve all the acuaracy which is not achieved by using the simulator.

# CHAPTER 3

## REGISTER TRANSFER LEVEL (RTL) COMPILER FLOW

### 3.0 INTRODUCTION

In the Veloce compile flow, Register Transfer Level Compiler acts as a front end. It reads the RTL description (in verilog/vhdl/systemVerilog) and converts that to a structural netlist with the logic mapped to Veloce FPGAs. RTL has multiple phases – (1) Analyze phase which reads the RTL description and dumps an Object Model of it (2) Synthesis engine which operates on this Object Model and converts that its own representation of the OM (NOM) and synthesizes the netlist. It also accepts netlist description (netlist created by RTL or other tools targeting a different FPGA technology). Apart from dumping the netlist equivalent of RTL, to enable debugging, RTL also dumps the debug database – containing information about signals that will have required name-map(since netlist can't have 2D/3D signals) and information related to signals that were transformed to memories.



**Fig 3. 1 Analysis and Synthesis for RTL FLOW**

### 3.1 THE RTL DESCRIPTION IN VERILOG/VHDL

RTL in Fig 3.1 could also be a verilog netlist description mapped to a different FPGA technology (may be station). In such a case of verilog netlist, the description of the behavior of the technology cells of that FPGA should also be provided to analyzer. RTL compile can be imagined as a three step process – (1) Analyze the RTL (2) Elaborate and partition the design (3) Synthesized RTL in each partition.

on a user option. It takes the user RTL, does syntax checks, and dumps an OM representation of the RTL. Driver reads this OM, elaborates the design, creates a folded graph with nodes as modules, associates cost for each node and partitions the graph into sub graphs and invokes synthesis phase(s) on a sets of sub graphs. NodeExpander reads the velsyn options, and dumps an annotation file for RTLC collating the information in velsyn annotation files. This helps RTLC instrument ceratian nets in a way to allow velsyn to successfully apply the user annotation.

### 3.2. MODULE ELABORATION

The VHL module of upward hier ref path in which top , middle and bottom modules are given

```

..
    module top_ver()
    mid_vhdl m1();
    mid_vhdl m2();
    endmodule
    entity mid_vhdl();
    end;
    architecture rtl of mid_vhdl
    bot: bot_ver();
    leaf: leaf_ver();
    end rtl;
    module bot_ver();
    leaf_ver l1();
    endmodule
    module leaf_ver();
    endmodule

```

For the given RTL description, the folded view created in RTLC-Elaborate will be as shown ...The numbers b/w  $\diamond$  represent the cost of compiling the module. Lets assume that the maximum cost per partition allowed is 400. We start with traversing from the 'top'. We first traverse along the cross language children so that they can be partitioned

out and the rest of the children can be partitioned later. The hier-costs of each of the modules will be sum of cost of that module and hier-costs of all its children. Since top has a vhdl module, we try to partition that first. Now we enter mid\_vhdl. Since this has cross language modules, we try to partition those. We enter bot\_ver. The hier-cost of bot\_ver =  $120 + 500 > 400$ . So we first need to create a partition for leaf\_ver and then worry about bot\_ver. Now we enter leaf\_ver. The cost of this module is  $500 > 400$  but we can't partition this further. So we create partition1 for leaf\_ver. Back to bot\_ver, the hier-cost of bot\_ver will now be just 120. We can put this in a partition – partition 2. This partition has a free space of 280 still. Back to mid\_vhdl. The hier-cost of mid\_vhdl now is 240. We create a vhdl partition – partition3 for this. Now we enter top\_ver. The hier-cost is only 200, since all its children are already partitioned. There is a verilog partition with a free space of 280. Therefore both bot\_ver and top\_ver will be in the same partition.

### **3.3 RTLC-VLE SYNTHESIS ENGINE**

The core synthesis is done by rvlc-vle ,The general flow is divided into various steps as shown .rtlc-vle will work on multiple tops as suggested by rvlc-elaborate during partitioning. It needs to elaborate the part of the design that was allotted to this partition, create an intermediate DS to hold the module information and operate on that DS. First such intermediate DS is the CDFG (CFG + DFG). This is created for each concurrent statement (ex: always block). We traverse along these graphs, partition them at various points, identify the various paths in the graph, the values each operand takes along those paths, perform the data flow analysis – based on which we categorize each net as simple wire/ reg/ latch. Then we allocate resources along the datapath – such as adders/multipliers etc. We perform optimizations on the eventual netlist object model. The eventual optimized NOM is fed to techmap to dump the final netlist.

### 3.4 CDFG (CONTROL AND DATA FLOW GRAPH)

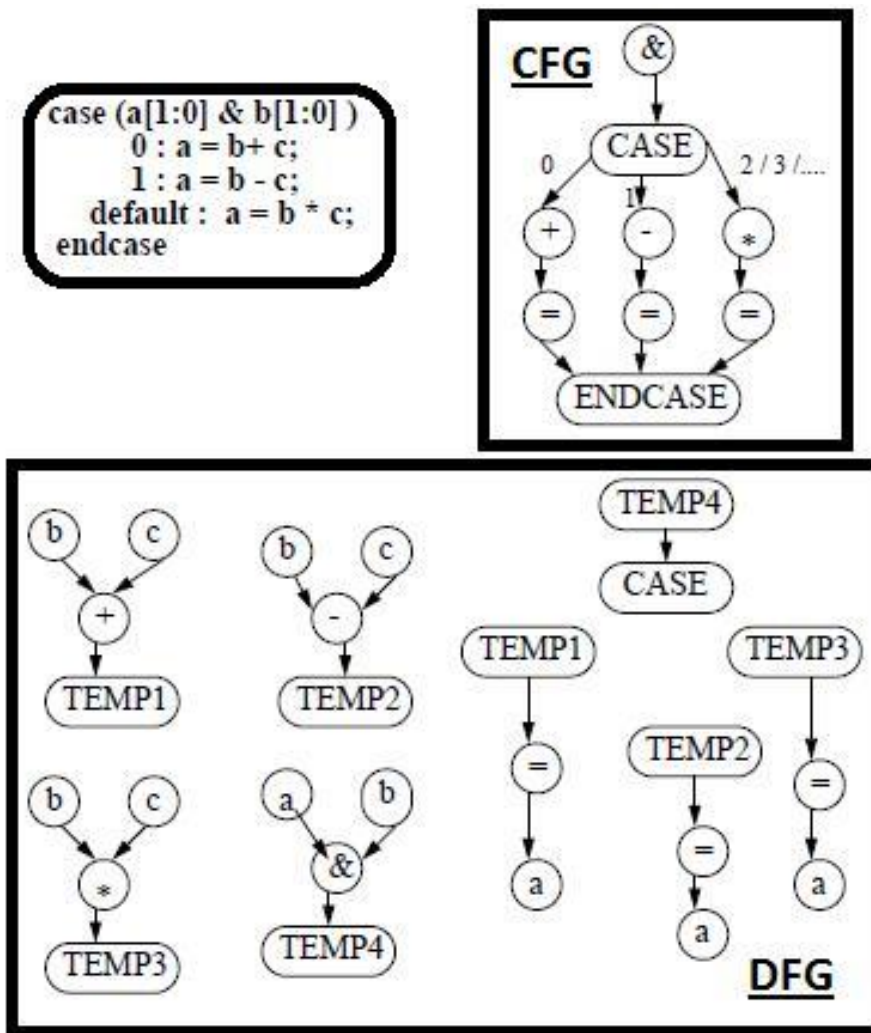


Fig 3.2 Exapnsion of control and data flow graph

This is a simple example to show the Control and data flow graph (CDFG) representation of a simple case statement. Note that, as simple a description as this results in a lot of temporary variables to deal.

It Represents control flow graph of the case statement, how the case statements work internally the conversion of the assignment statement actually move through the control And data flow graph. The case statements which are used in Verilog and VHL coding internally work in this control and data flow Graph (CDFG).



### 3.4 CDFG EXAMPLE 2 (CONTROL AND DATA FLOW GRAPH)

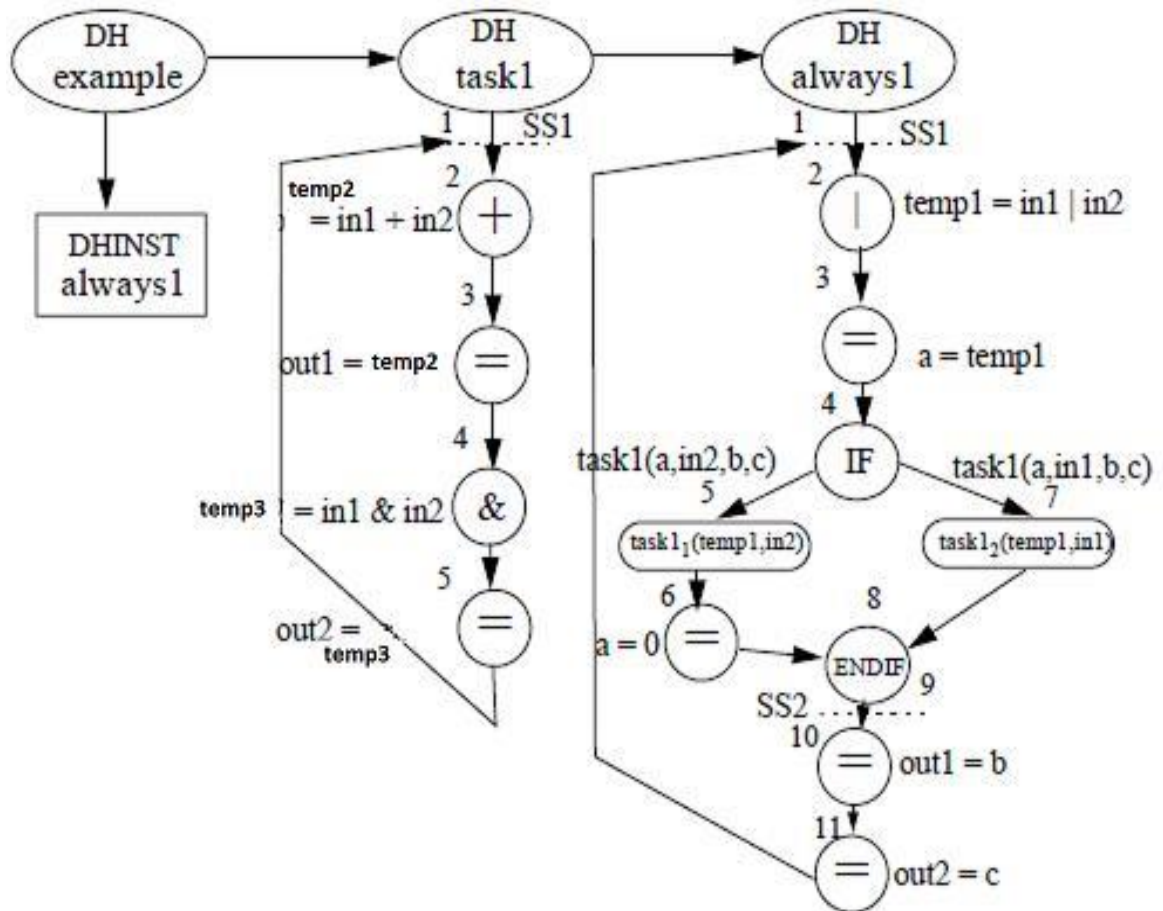


Fig 3.3 Expansion of control and data flow graph in tasks and function

This control and data flow graph in Fig 3.3 expands the tasks and the function code in Verilog with this flow, we can expand the function and the task.

The CFGs of the module contains only the instantiation of always block. The CFG of the always internally has the instance(s) of the task. Based on the cost (some heuristic), the DH\_TASK may be chosen for flattening. Note that Pseudo state cuts are created at the root of each CFG. Pseudo state cuts are also created at each join node (such as end of if condition/ endcase in case statement etc). Based on the statecuts, the CFG is considered to be consisting of various paths containing path segments. A data structure (PVM) containing information about each operand along each path of the CFG is created.

### 3.5 CDFG (CONTROL AND DATA FLOW GRAPH)

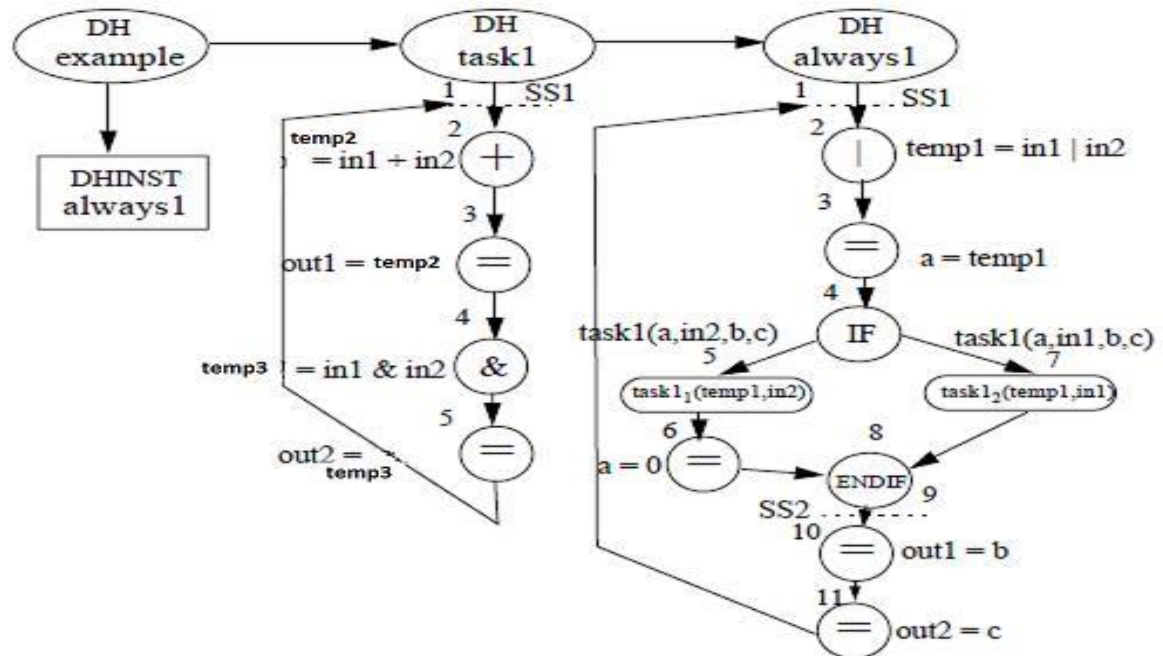


Fig 3.4 Control and data flow graph with always block

The CFGs in Fig 3.4 of the module contains only the instantiation of always block. The CFG of the always internally has the instance(s) of the task. Based on the cost (some heuristic), the DH\_TASK may be chosen for flattening. Note that Pseudo state cuts are created at the root of each CFG. Pseudo state cuts are also created at each join node (such as end of if condition/ endcase in case statement etc). Based on the statecuts, the CFG is considered to be consisting of various paths containing path segments. A data structure (PVM) containing information about each operand along each path of the CFG is created.

1. The CFGs of the module contains only the instantiation of always block.
2. The CFG of the always internally has the instance(s) of the task.
3. Based on the cost (some heuristic), the DH\_TASK may be chosen for flattening.
4. Note that Pseudo state cuts are created at the root of each CFG. Pseudo state cuts are also created at each join node (such as end of if condition/ endcase in case statement etc).
5. Based on the statecuts, the CFG is considered to be consisting of various paths containing path segments.
6. A data structure (PVM) containing information about each operand along each path of the CFG is created.

7. Such a DS for the two states of the always block is shown above.
8. The condition for the paths is also present.
9. Based on the values taken by each operand along various paths of the CFG, resources (such a MUX/ADDER/etc) are allocated.

### 3.6 TECHCELL FLOW

Apart from handling RTL, RTLC can also optimally handle the structural pre-synthesized netlist. The technology cells need to be specified through the “-techlib” option. RTLC remaps this net list to optimal MED\_<LUT/Cell> equivalent.

1. Verilog code for The Techcell Flow how the modules like gtech \_and are instantiated inside the Top Module.

```

module Top(in1, in2, in3, in4, out);
    output out;
    input in1, in2, in3, in4;
    wire temp1, temp2;
    gtech_and i1 (in1, in2, t1);
    gtech_and i2 (in3, in4, t2);
    gtech_and i3 (t1, t2, out);
endmodule

module gtech_and(in1, in2, out);
    output out;
    input in1, in2;
    assign out = i1 & i2;
endmodule

```

This is usually the flow employed while importing netlist(compiled for a certain target Technology). The basic cells of that technology are referred to as techCells. Since the behavioral definitions of techCells is provided to RTLC, they will be treated as being no different from regular modules. However, these basic blocks need to be flattened out to get optimal mapping for Veloce FPGA. To enable this, all the modules that refer to these technology cells, should be provided to RTLC for analysis with the option – techlib. The analysis OM of these modules will be marked as techcells and shall be flattened out by default.

### 3.7 FLATTENING SUPPORT

Motivation for flattening a module

1. To reduce the database size.
2. To Enable cross boundary logic optimizations.
3. To Enable retiming across boundaries to reduce the critical path delay.

Options provided by RTLCC to enable flattening

1. -flatten\_mod <module\_name>
2. All the instances of the specified module get flattened into respective parents where ever possible (e.g. instances will not get flattened into cross-language parents).
3. Flatten\_hier <module\_name>

This option can be used to flatten all the instances under a particular hierarchy (i.e. all modules under <mod\_name>). This reduces user's effort in specifying -flatten\_mod on a number of modules which essentially get covered under a particular hierarchy.

### 3.8 DESIGNWARE INTEGRATION WITH RTLCC

1. Support for seamless integration with Synopsys DW component
2. Support for complicated floating point and FIFO controllers have been added which are extensively used at Intel.
3. DW component support is provided on customer requirement basis.
4. Similar to library integration, where RTLCC selects its definition where no DW definition is given.(i.e. user specified RTL modeling gets higher priority)

Design Ware components are building block IP components. DC/Synopsys provides a library of DW components (which are building block IP components – various commonly used functions/data path resources). Designers can instantiate their components directly without having to worry about the implementation details. It is expected that their implementations shall be optimal(capacity/performance). The RTL written by our customers have these components instantiated and their definitions may not be provided with the assumption that they are part of library. Based on the simulation model provided by Synopsys, RTLCC created (not complete) a library of synthesizable model (fairly optimized).

### 3.9 EXAMPLE FOR DESIGN WARE INTEGRATION

The design ware components behave like macro in which only the name DW01\_addsub is Used for the direct addition and subratakction, It is not needed to write full code for the Subraction and addition only the design ware macro can be used to implement this code.

```
module top(A,B,CI,ADD_SUB,SUM,CO, temp);
output [7 : 0] SUM;
output CO;
input [7 : 0] A, B;
input CI, ADD_SUB;
output [7 : 0] temp;
DW01_addsub #(8)
DW01_addsub_test(.A(A),.B(B),.CI(~CI),.ADD_SUB(ADD_SUB),.SUM(SUM),.CO( CO));

assign temp = A;
Endmodule

module DW_mag_module(a, z);
parameter width=9;
parameter size=8;
input signed [width-1:0] a;
output signed [size-1:0] z;
`include "DW_dp_absval_function.inc"
assign z = DWF_dp_absval(a);
endmodule
```

### 3.10 EXAMPLE OF THE DESIGN WARE COMPONENTS

Here the VHDL code for the Design ware component DW01 is written, Here it is not Needed to write the whole code of the generic mapping , only the macro for the DW Is used for decoding the generic mapping in which the design ware library for the DW01 is used with the IEEE library.

```
library IEEE,DW01;
use IEEE.std_logic_1164.all;
entity top is
port(A : in std_logic_vector(3 downto 0);
B : out std_logic_vector(15 downto 0));
end top;
architecture sim of top is
component DW01_decode
generic(width : POSITIVE);
port(A : in std_logic_vector(width-1 downto 0);
B : out std_logic_vector(((2**width) - 1)
downto 0));
end component;
begin
test : DW01_decode generic map (4)
port map ( A=>A,B=>B);
end sim;
```

### 3.11 TRADITIONAL ENCRYPTION TECHNIQUES

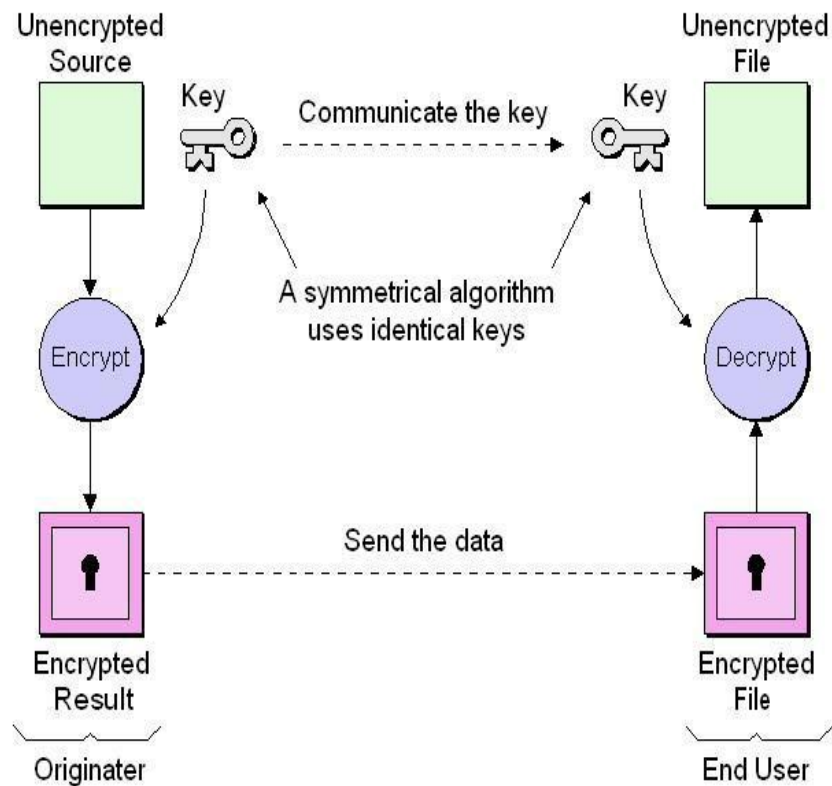


Fig 3.5 The Encryption techniques with originator and end user

Traditional encryption techniques are of two types in Fig 3.5 –

- (1) **Symmetric** – Where the encryption key is shared by the originator with the end user. Only those with this key can decrypt the source.
  - This is fast but less secure
- (2) **Asymmetric** – Where the encryption is done using key1 while decryption is done with key2 i.e. a key pair is used here. The Encryption key is a public key shared by the end user with the originator. The originator encrypts the source and sends the encrypted data to the end user. The end user will use the private key (key2), the access to which is available only with the end user, to decrypt the encrypted source.
  - This is more secure but slow

Automatic memory inference is supported in both vhdl & verilog for

- 2-Dimensional Signal
  - N-Dimensional Signal
  - Byte Enable Memory
  - Array of complex structure/record
- 
- Most important optimization to optimize the capacity of the design.
  - Memory inference is dependent on both declaration and usage.
  - Examples:

The Memory address is the given which is the input to the data and then the data is written which acts like the write data the the data received is given to the output port which acts like the read data is received by the Read port. So the memory has Read and the write data.

```
Usage : RAM[waddress] = data // Memory write port
out = RAM[raddress] // Memory read port
reg [a1:a2] RAM [b1:b2][c1:c3] // N-Dimensional Signal
Usage : RAM[waddr1][waddr2] = data // Memory write port
out = RAM[raddr1][raddr2] // Memory read port
```

Primary advantage of memory inference is improvement in capacity.

Memory is inferred for a signal based on its declaration and usage. Any access to such a signal will be implemented via a readPort/writePort.**Automatic Memory Inference**



### 3.11 MEMORY WITH SYNC\_ASYNC RESET?SET

Here the Memory can be represented with the Asynchronous reset if the the reset is Given in the problem statement , it is represented with synchronous reset if the Reset is not mentioned in the problem statement.

Assigning complete memory in synchronous/asynchronous reset/set condition  
reg [0:7] RAM [0:255];

always @ (posedge clock or posedge reset)

```
begin
    if ( reset )
        for(int I = 0; I < 256; I++)
            RAM[I] = 0;
    else
        begin
            RAM [waddr] = din;
            Dout = RAM [raddr];
        end
    end
end
```

Subtype slv8 is std\_logic\_vector (0 to 7)

Type RAM is array (0 to 255) of slv8

Signal memory : RAM;

Process (clock, reset)

Begin

```
if ( reset ) then
    for I in 0 to 255 loop
        RAM(I) <= (others => '0');
    end loop;
elsif ( clock' event && ( clock = '1' ) ) then RAM (waddr) <= din; Dout
    <= RAM (raddr);

end if;
```

End process;

### 3.13 PRAGMA CONTROL

#### VERILOG PRAGMA ATTRIBUTE

**Verilog:-** // pragma attribute <signal-name> <pragma-name> 0/1

<signal-name> = name of register array

<pragma-name> = ram\_block = force implementation as built-in memory

core logic\_block = force implementation as register array gates 0 = disable, 1 = enable

The Verilog attribute is represented in the following manner where the Ram Block 1 is mentioned.

Ex: reg [31:0] core [63:0]; // 64x32 bit memory array

// pragma attribute core ram\_block 1

#### VHDL PRAGMA ATTRIBUTE

**Vhdl:-** attribute <pragma name> : boolean;

attribute <pragma name> of <signal-name> :

<signal/variable> is <true/false>

false = disable, true =

enable Ex:

VHDL attribute is

represented in the following

way in which RAM

BLOCK of type is true

signal core : ram\_type; -- memory array

attribute ram\_block : boolean;

attribute ram\_block of core : signal is true;

## 14 CROSS SHARING

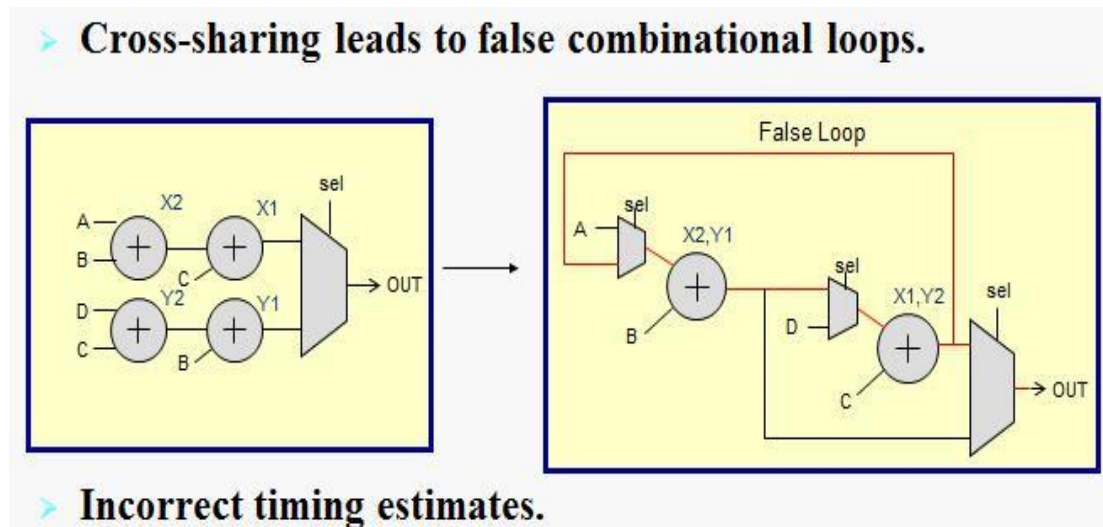


Fig 3.7 Cross –Sharing between the combinational loops

Caution must be taken while sharing resources. Sharing must be done in such a way that it does not result in any combination loop (albeit a false loop). In the above example in Fig 3.7, since only two adders are required essentially, we can try to optimize the logic by sharing the Adders. If Adders X2, Y1 are shared (owing especially to the shared input ‘B’) and Y2, X1 are shared (owing to the shared input ‘C’), it would result in a false combinational loop. Note that its not a loop in the strict sense –

Case (i) – When  $sel == 0 \Rightarrow$  The loop is broken at 1<sup>st</sup> Mux

Case (ii) – When  $sel == 1 \Rightarrow$  The loop is broken at 2<sup>nd</sup> Mux

Such a false combination loop may not be solvable for a timing estimation tool resulting in incorrect estimates.

### 3.15 CONCLUSION

This chapter gives brief introduction about the Register transfer level compile flow in which it has analyzed the control and data flow graph in which it has many traversals. The system needs to pick the most optimized control and data flow graph. This graph will optimize all the control and data related issues. The control and data flow graph will analyze all the nodes of a register transfer level module and then it has to optimize the path so this compile flow will tell where to optimize all the RTL level.

# CHAPTER 4

## OPTIMIZATION OF MEMORY PORTS

### 4.0. INTRODUCTION

This Optimization is done in case of there are large Number of ports which can hamper the memory during synthesis. So with this conversion we are converting it into single port and also we are reducing the compilation Time.

In case enrolling of the loop, it has the constant address in that case they are not declaring it as memory, it can be declared as logic Block in which we have many adders, Multiplexers and LUTs which takes a lot of area for the chip. Thus memory Inference is really important and helpful in chip designing, In this to infer memory through this optimization in this case for and if statement is converted into simple if statement, thus that only one write port works at a time. This makes the memory to be inferred causing the small area and better performance of the chip.

Compilation time is also got reduced with this optimization of ports of memory. In which only the one write port is active at a time.

### 4.1 FOR IF LOOP IS CONVERTED ITO SIMPLE IF STATEMENT

The For loop and the if statement are converted into single if statement using this conversion.

```
For ( i=C1 ; i< C2 ; i++ )
begin
  if( i == named Object Use)
    <Assignments(only)>
end
```

( **conversion** )

```
if( (named Object Use >= C1) && (named Object Use < C2) )
  <Assignments>
```

```

1
2 module top(clock, addr, din, dout);
3   input clock ;
4   input [9:0] addr ;
5   input [7:0] din ;
6   output [7:0] dout;
7   // pragma attribute mem ram_block 1
8   integer i;
9   reg [7:0] mem[1023:0];
10
11  initial
12  begin
13    for(i = 0 ; i < 1024 ; i = i+1)
14    begin
15      mem[i] = 0 ;
16    end
17  end
18
19  assign dout = mem[addr] ;
20  always @(posedge clock)
21  begin
22    for(i = 0; i < 50; i=i+1)
23    begin
24      if(i == addr)
25        mem[i] <= din ;
26    end
27  end
28 endmodule

```

top.v\* 28L, 509C 1,0-1 AL

Fig 4.1 the for if enrolling of the loop with memory declaration.

## 4.2 ALGORITHM AND APPROCH

Conversion is done while pre Traversal before CDFG conversion in Fig 4.1.

1. Check FOR Loop whether conversion is possible or not. (Eg check generic for loop , if condition )
2. If conversion is not possible then return.
3. If conversion is possible save the for loop in a list.
4. Iterate over this list, once the traversal is done. Check for if conversion is possible (Eg memory element. Mem access type pattern on that dimension.
5. If conversion is not possible, then traverse the non converted with different object.
6. If conversion is possible, then traverse the converted loop with the different object. Store it into a table then finally we use it for the conversion.

---

### 4.3 ACCESSING THE MEMORY BLOCK

Logic [7:0] memory [255:0] [3:0]

Logic block	Memory block	Logic block
1	2	3
ADDRESS ACCESS TYPE	ADDRESS ACCESS TYPE	DATA ACCESS TYPE
Static access	Non static access	Static access
[255:0]	[3:0]	[7:0]

Memory is declared which is really important to declare otherwise it causes the logic block to be made which consists of adders, Multiplexers and logic blocks which increases the capacity of chip.

By memory inference we are able to mark memory. In this memory declaration we have 256 address depth and 4 bits we have 3D memory and 8 bits the data width.

There is packed and unpacked array in which we have the data width is considered in case of packed array and the unpacked array we have the address depth. The first bit is considered to be the no. of memory blocks and second bit decides the no. of bits in the memory depth.

In unroll able for loop we have constant address in that case memory is not declared, it is assumed as a logic block which makes the area of the chip to exponentially increases and memory is not inferred where in case of optimization the conversion of for loop makes it non - static address corresponding to it memory is inferred, this cause the chip size to decrease. Hence the optimization of device is there.

So this optimization is really helpful in maintaing the balance between the area and the performance. With the data out comparision between the questa and Register Transfer level compiler, we can get the results of memory inference with simulation and emulation. So the Questa and The Register Transfer level compiler are very important in case of these simulation and Emulation.

```

1
2 module top(clock,addr,din,dout);
3   input clock ;
4   input [9:0] addr ;
5   input [7:0]din ;
6   output [7:0] dout;
7 // pragma attribute mem ram_block 1
8   integer i;
9   reg [7:0] mem[1023:0];
10
11   initial
12   begin
13     for(i = 0 ; i < 1024 ; i = i+1)
14       begin
15         mem[i] = 0 ;
16       end
17   end
18
19   assign dout = mem[addr] ;
20   always @(posedge clock)
21   begin
22     for(i = 0; i < 50; i=i+1)
23     begin
24       if(i == addr)
25         mem[i] <= din ;
26     end
27   end
28 endmodule

```

top.v\* 28L, 509C 1,0-1 AI

Fig 4.2 The Read and Write Ports of The Memory (There 50 write ports and one read port)

In this Verilog Program we have 50 write ports and one Read port with this optimization of memory block in Fig 4.2 at a time only one Read port is active which is actually needed to compile the testcase. Hence this optimization is good, because it helps in inference of memory causes this enrolling of the for loop to convert it into the single if statement.

This Verilog program represent how we need to use the chain of muxes in case of the output. But with this optimization we can stop this unrolling abd convert this for loop into the single if statement.

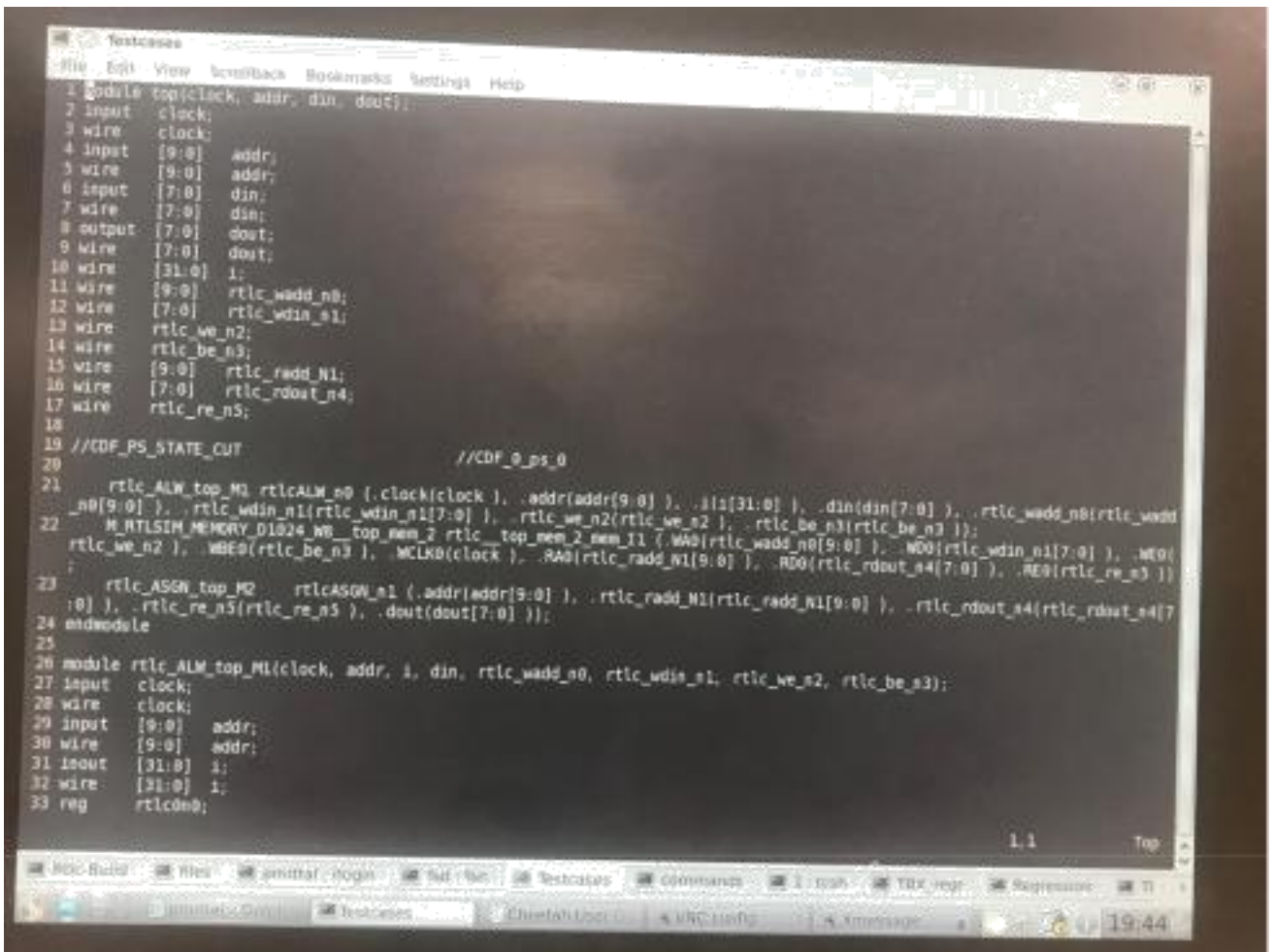


Fig 4.3 Questa output of the memory representation. The simulation output of the results

Here in this CDFG representation in Fig 4.3, we have the Questa output, how the VMW PARAMEM is formed. Here the different wires are declared like we have to represent it in the form of wires. Where we have only inputs and outputs, there is no storage elements.

In Verilog the storage element is represented by register. So in this module we have clock, address bits, data bits to represent it in the form of memory block. So this CDFG representation is helpful in pre-analyzing the data and control flow of the loop.

This Module is represented in the form of continuous enrolling of the For loop such that it helps in the generating the control and data flow graph without optimization.



## 4.4 VAILD SCENARIOS WHERE CONVERSION OCCURS

Here there is conversion of the for if loop in which I is used in dimension with Static , Non\_ Static , Static + Non\_Static Access Type and Default Access Type.

### **The Memory read and write ports**

```
module top(clock,addr,din,dout);

    input clock ; input
    [9:0] addr ; input
    [7:0]di
    output[7:0] dout;
    integer i;
    reg [7:0] mem[1023:0]
assign dout = mem[addr] ;
always @(posedge clock) begin
for(i = 0; i < 50; i=i+1)
begin
if(i == addr)
mem[i] <= din ;
end
end
endmodule
```

Here The Mem Access Type of I is Static so we are converting this For If loop.

Here in This loop for conversion, we have to check the access type of the variable to be converted. Hence it is determined with the help of register Transfer level Language, how this RTL language is converted into the C language code. This is all about the conversion, there is the conversion of these for loops into the if statement. Thus with the help of this code, there is the optimization which is related to all the for loop conversion. Hence all the conversion is done according memory access pattern. We should carefully observe all the memory patterns of this loop is statements which helps in the conversion of if statement.

## 4.5 HANDLING OF DIFFERENT OPERATORS

The statement is the FOR loop conversion without memory block

The cases where the logical (And) (&&) operator is used we are checking whether there is logical equal (==) operator is also used in that case we are converting. always @ \*

Here Different operators are handled with certain limits and integer are handled and operator and or operator, then have to merge all the operators if logical equal to operator is there. So the Enable switches will not impact the optimization since they logical equal to operator is given preference due to the preference of the logical equal to the OR operator and the AND operator is not taken into consideration.

Handling of the operators in if condition is really important. Because if condition will decide when we have to choose the differences in case of the operator handling.

All the Verilog program begins with the positive edge of the clock which decides at what edge the clock, begins to work. The clock will stabilize and then will get the output in case of every positive edge of the clock. Then the for loop starts to enroll, Thus this changes will cause the positive edge of the clock to occur.

Since first of all the logical equal to operator is compared in case of the loops. Then will check the logical and operator, so need to check the logical equal to operator and then need to check the logical and operator. These are the comparisons which we need to made in case of the logical and and equal to operator.

The For and if loop without memory Block.

```
Always@(posedge clk)

begin
  out1 = 0;
  for(integer i = 0; i < 8; i++)
    if(i == in2 && (enable1 || enable2))
      out1[i] = in1[i];
end
```

## 4.6 Enrolling of the For Loop

Here to discuss the internal working of the loops while expansion through control and data flow graph (CDFG)

In case they are converting we will write on CDFG otherwise we will not.

```
//NOP_1;      //CDF_1

[-] i[31:0] [+] = [+] 1'd0[0]{0};      //CDF_2

[+] rtlc0n0 = ([+] 32'd00000000000000000000000000000000[31 :

{00000000000000000000000000000000} [+] == [+] addr[9 : 0]);      //CDF_7

//NOP_8;      //CDF_8

//NOP_1260;    //CDF_1260

[+] rtlc0n102 [-1:-1] [+] = [+] rtlc0n0;      //CDF_1261

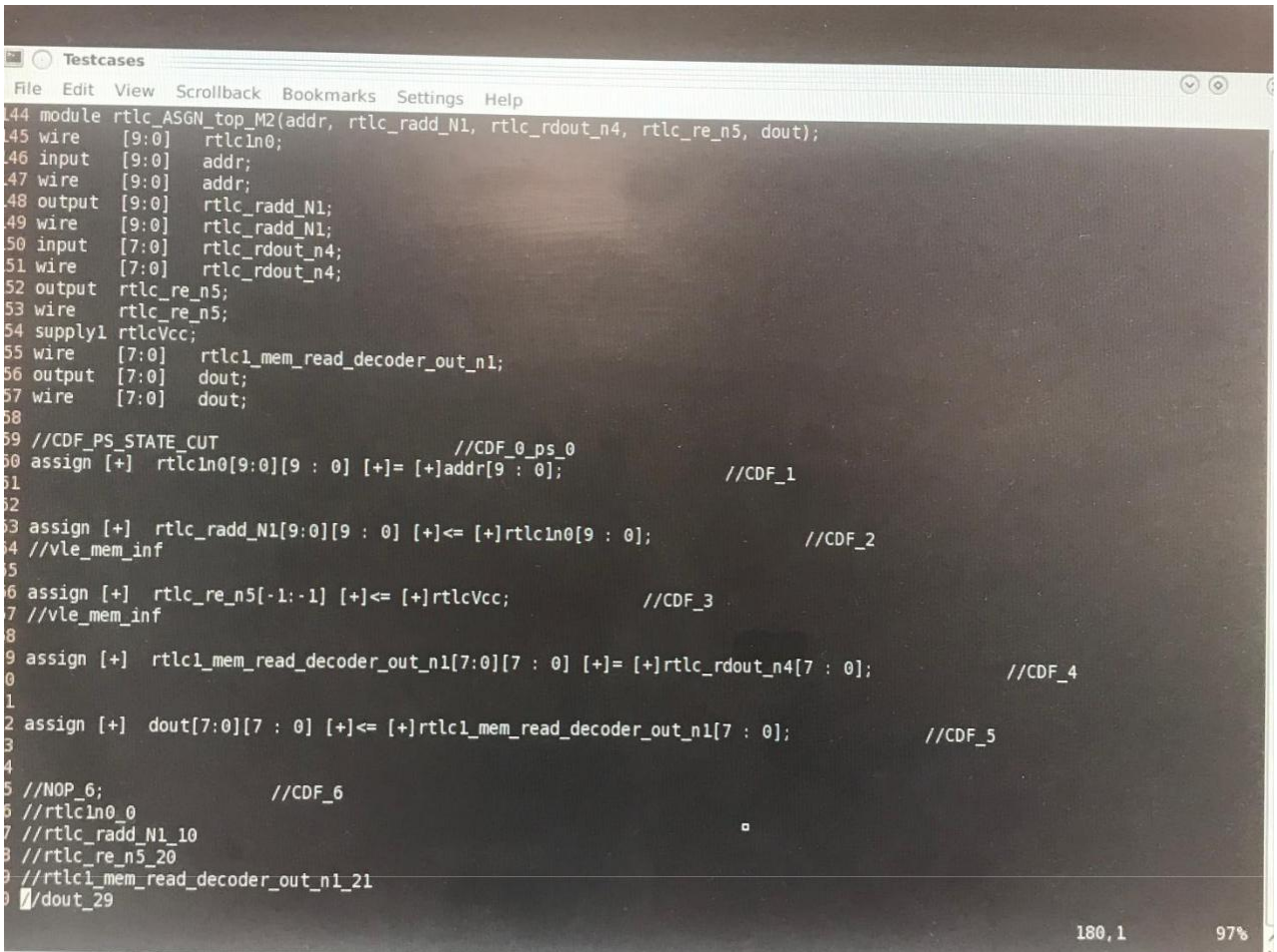
[+] rtlc0_mem_write_datain_n3 [7:0][7 : 0] [+] = [+] din[7 :      //CDF_11
0];
```

This how it can seen the enrolling of the for loop at the CDFG level. The control the data flow is that step in design flow when there is not optimization occurs.

A general graph is formed where we can decide whether this graph like structure in which we have control and the data flow in which there can be enrolling the for loop which consist of the graph like structure in terms of the control and the data flow ie CDFG level . Here we can see that there is enrolling of the for loop occurs where we can see that no optimztion occurs at the CDFG i.e control and data flow graph.

The Memory Inference is handed for Hard constraints.

Path for all the test cases /in/innrtlc15/perry/testcase/memory/ which have been tested during my testing.



```
Testcases
File Edit View Scrollback Bookmarks Settings Help
44 module rtlc_ASGN_top_M2(addr, rtlc_radd_n1, rtlc_rdout_n4, rtlc_re_n5, dout);
45 wire [9:0] rtlcIn0;
46 input [9:0] addr;
47 wire [9:0] addr;
48 output [9:0] rtlc_radd_n1;
49 wire [9:0] rtlc_radd_n1;
50 input [7:0] rtlc_rdout_n4;
51 wire [7:0] rtlc_rdout_n4;
52 output rtlc_re_n5;
53 wire rtlc_re_n5;
54 supply1 rtlcVcc;
55 wire [7:0] rtlc1_mem_read_decoder_out_n1;
56 output [7:0] dout;
57 wire [7:0] dout;
58
59 //CDF_PS_STATE_CUT //CDF_0_ps_0
60 assign [+] rtlcIn0[9:0][9 : 0] [+] = [+] addr[9 : 0]; //CDF_1
61
62
63 assign [+] rtlc_radd_n1[9:0][9 : 0] [+] <= [+] rtlcIn0[9 : 0]; //CDF_2
64 //vle_mem_inf
65
66 assign [+] rtlc_re_n5[-1:-1] [+] <= [+] rtlcVcc; //CDF_3
67 //vle_mem_inf
68
69 assign [+] rtlc1_mem_read_decoder_out_n1[7:0][7 : 0] [+] = [+] rtlc_rdout_n4[7 : 0]; //CDF_4
70
71
72 assign [+] dout[7:0][7 : 0] [+] <= [+] rtlc1_mem_read_decoder_out_n1[7 : 0]; //CDF_5
73
74
75 //NOP_6; //CDF_6
76 //rtlcIn0_0
77 //rtlc_radd_n1_10
78 //rtlc_re_n5_20
79 //rtlc1_mem_read_decoder_out_n1_21
80 //dout_29
180,1 97%
```

Fig 4.4 the various read and the write ports in terms of control and data flow graph.

The various Address bits and data bits in Fig 4.4 are represented interms of CDFG level. The Model is converted into a vele memory such that memory is inferred at this level. So that various data out and memory is represented in the form of decoder the memory read decoder the memory write decoder. All the input data read and output data write is represented interms of the decoder The Vcc and address depth bits and data width all is represented interms of the decoder.

The Netlist conversion of the Verilog Module. How the various LUTs and Modules are formed in case of the Optimization of the loop.

```

15 wire [7:0] rtlcmn2;
16 supply0 rtlcVss_m_n3;
17 supply1 rtlcVcc_m_n4;
18
19 wire rtlc_int_n81;
20 MED_LUT4_0100 rtlclut_n171 (.0(rtlc_int_n81), .I0(addr[1]), .I1(addr[2]), .I2(addr[3]), .I3(addr[4]));
21 wire rtlc_int_n82;
22 MED_LUT3_F7 rtlclut_n172 (.0(rtlc_int_n82), .I0(addr[4]), .I1(addr[5]), .I2(rtlc_int_n81));
23 wire rtlc_int_n83;
24 MED_LUT3_01 rtlclut_n173 (.0(rtlc_int_n83), .I0(addr[6]), .I1(addr[7]), .I2(addr[8]));
25 MED_LUT3_40 rtlclut_n174 (.0(rtlc_we_n2), .I0(addr[9]), .I1(rtlc_int_n82), .I2(rtlc_int_n83));
26 MED_BUF rtlcbuf_n175 (.Z(rtlc_re_n5), .A(1'b1));
27 VMW_PARAM_MEM_rtlc0_10_8_5_0_1_1 #( 10,8,5,0,1,1) mem (.BA0(addr[9:0]), .RD0(dout[7:0]), .WA0(rtlcmn0[9:0]), .WD0(rtlcmn2[7:0]), .WE0(rtlcmn1));
28 MED_FD rtlcreg_rtlcmn0_0 (.Q(rtlcmn0[0]), .D(addr[0]), .CP(clock));
29 MED_FD rtlcreg_rtlcmn0_1 (.Q(rtlcmn0[1]), .D(addr[1]), .CP(clock));
30 MED_FD rtlcreg_rtlcmn0_2 (.Q(rtlcmn0[2]), .D(addr[2]), .CP(clock));
31 MED_FD rtlcreg_rtlcmn0_3 (.Q(rtlcmn0[3]), .D(addr[3]), .CP(clock));
32 MED_FD rtlcreg_rtlcmn0_4 (.Q(rtlcmn0[4]), .D(addr[4]), .CP(clock));
33 MED_FD rtlcreg_rtlcmn0_5 (.Q(rtlcmn0[5]), .D(addr[5]), .CP(clock));
34 MED_FD rtlcreg_rtlcmn0_6 (.Q(rtlcmn0[6]), .D(addr[6]), .CP(clock));
35 MED_FD rtlcreg_rtlcmn0_7 (.Q(rtlcmn0[7]), .D(addr[7]), .CP(clock));
36 MED_FD rtlcreg_rtlcmn0_8 (.Q(rtlcmn0[8]), .D(addr[8]), .CP(clock));
37 MED_FD rtlcreg_rtlcmn0_9 (.Q(rtlcmn0[9]), .D(addr[9]), .CP(clock));
38 MED_FD rtlcreg_rtlcmn1 (.Q(rtlcmn1), .D(rtlc_we_n2), .CP(clock));
39 MED_FD rtlcreg_rtlcmn2_0 (.Q(rtlcmn2[0]), .D(din[0]), .CP(clock));
40 MED_FD rtlcreg_rtlcmn2_1 (.Q(rtlcmn2[1]), .D(din[1]), .CP(clock));
41 MED_FD rtlcreg_rtlcmn2_2 (.Q(rtlcmn2[2]), .D(din[2]), .CP(clock));
42 MED_FD rtlcreg_rtlcmn2_3 (.Q(rtlcmn2[3]), .D(din[3]), .CP(clock));
43 MED_FD rtlcreg_rtlcmn2_4 (.Q(rtlcmn2[4]), .D(din[4]), .CP(clock));
44 MED_FD rtlcreg_rtlcmn2_5 (.Q(rtlcmn2[5]), .D(din[5]), .CP(clock));
45 MED_FD rtlcreg_rtlcmn2_6 (.Q(rtlcmn2[6]), .D(din[6]), .CP(clock));
46 MED_FD rtlcreg_rtlcmn2_7 (.Q(rtlcmn2[7]), .D(din[7]), .CP(clock));
47 rtlc_top_GLOBAL_SIGNAL rtlc_top_GLOBAL_SIGNAL_INST ();
48 rtlc_tbx_HR_INSTR_MOD rtlc_tbx_HR_INSTR_INST ();
49 endmodule
50

```

Fig 4.5 NETLIST and the memory declaration with the Look up Table (LUTS) Formation

In this module in Fig 4.5 we have various LUTs and Flip Flops are there as storage elements which help in the formation of various components of the chip. Since the MED\_FD which means the Mentor Emulation Division D flip flops means the data flip flops which are able to store the data information. These Look up tables can also be represented in any form in which there has to represent the logic so the VMW\_PARAM\_MEM is really important to optimize the area of the chip.

## 4.6 COVERSION CASES

The memory can be respresented in the form of part select :

```
for(i = 0; i < 50; i=i+1)
begin
  if(i == addr)
    mem[i+: 8] <= din ;
```

Here the memory is represented in the form of the part select, in which it takes a only the part of the memory

The digital design system is described in Verilog because high level Verilog designs are usually described at the level that consist of system registers and transfer of data between various registers through buses, this level of high level description is called as register transfer level (RTL). Verilog construction used in RTL level design are procedural statements continuous assignments and instantiation statements.

Since The part select conversion can include only the portion of the memory. Thus the complete flow of the memory is not able to include the complete flow of the memory which causes the memory to be optimally utized such that the part selection of the memory is not included interms of the optimization. Thus the optimally inclusion of the memory needs the complete declaration. Not the part selection of the memory.

A digital system designed in Verilog should be simulated and tested for functionality before it is turned out into a hardware while simulation design errors and incompatibility of components in the digital design, all are detected. In simulation, a digital design need generation of test data and observation of all simulation results. This process can be done by use of a Verilog module that is called as a test bench.

A Verilog test bench uses HDLs constructs for generation of data monitoring of response and even handshaking of the design. In the test bench, the design is instantiated which is being simulated the test bench together with digital design.

```
Testcases
File Edit View Scrollback Bookmarks Settings Help
40 wire rtlc_we_n118;
41 wire rtlc_we_n122;
42 wire rtlc_we_n126;
43 wire rtlc_we_n130;
44 wire rtlc_we_n134;
45 wire rtlc_we_n138;
46 wire rtlc_we_n142;
47 wire rtlc_we_n146;
48 wire rtlc_we_n150;
49 wire rtlc_we_n154;
50 wire rtlc_we_n158;
51 wire rtlc_we_n162;
52 wire rtlc_we_n166;
53 wire rtlc_we_n170;
54 wire rtlc_we_n174;
55 wire rtlc_we_n178;
56 wire rtlc_we_n182;
57 wire rtlc_we_n186;
58 wire rtlc_we_n190;
59 wire rtlc_we_n194;
60 wire rtlc_we_n198;
61 wire rtlc_re_n201;
62 supply0 rtlcVss;
63 wire rtlcn1635;
64 wire rtlcn1636;
65 wire rtlcn1637;
66 wire rtlcn1638;
67 wire rtlcn1639;
68 wire rtlcn1640;
69 wire rtlcn1653;
70 wire rtlcs100;
71 wire rtlcs105;
72 wire rtlcs108;
73 wire rtlcs110;
74 wire rtlcs113;
75 wire rtlcs115;
76 wire rtlcs119;
40, 1
pmittal : rlogin %d : %n Testcases commands 1 : tcsh TBX_regr Regression TBX-For_IF DW_fun
```

Fig 4.6 various parts of the memory the write enable and the input of the memory.

In Memory Block there are write enables and the read enable and the Byte enable as well how much write and byte enable memory is required. Since the RTL memory required for the write enable as well as byte enable. There needs the memory to be declared with address read and write ports. The data read and write port, the byte enable read and write port. The memory have various enable which helps to write the address bit in memory in Fig 4.6.

In Verilog the storage element is represented by register. So in this module there is a clock, address bits, data bits to represent it the form of memory block. So this CDFG representation is helpful in preanalyzing the data and control flow of the loop

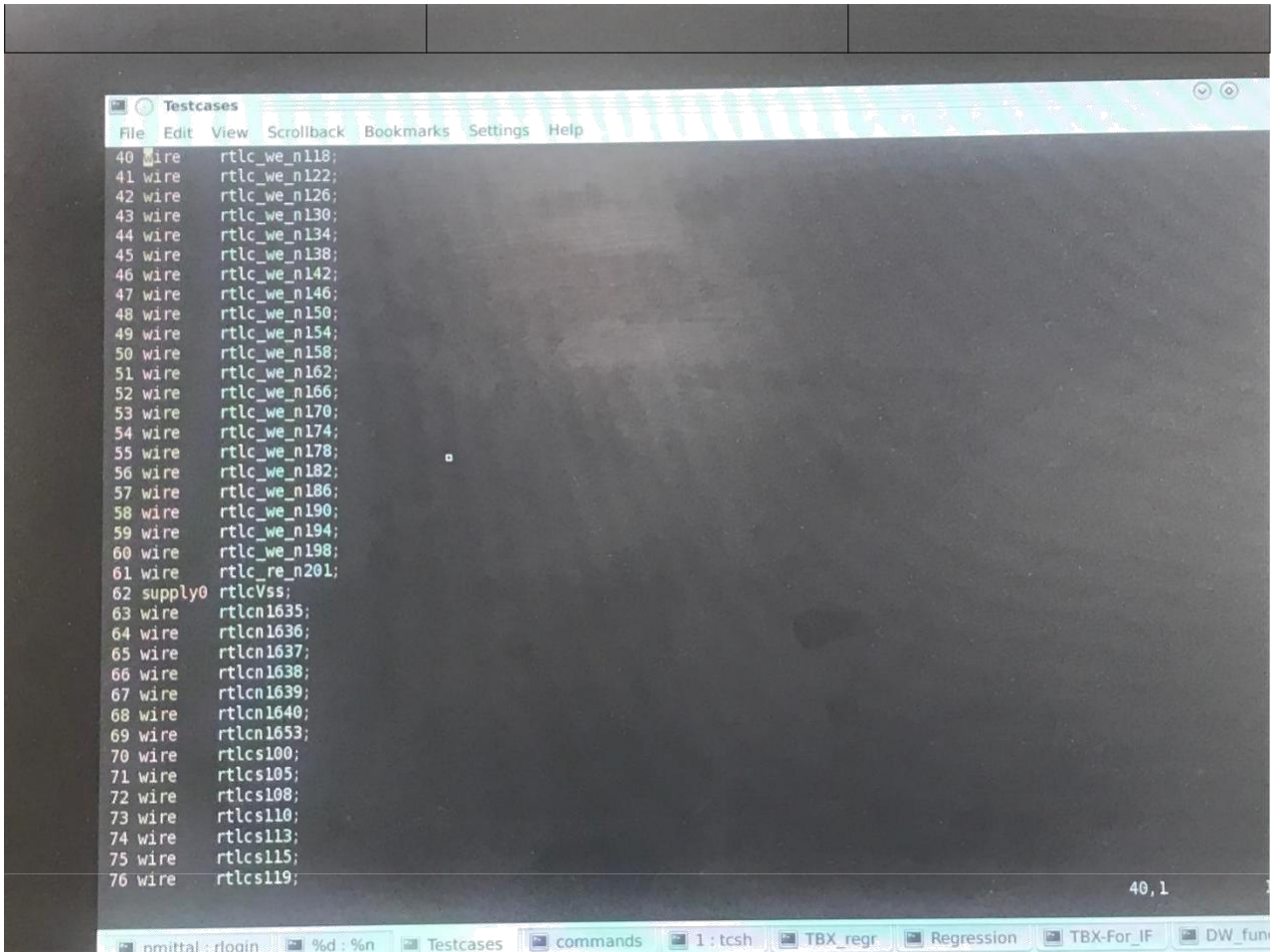


Fig 4.7 The memory formation also represents the write enables

In unroll able for loop in Fig 4.7 we have constant address in that case memory is not declared, it is assumed as a logic block which makes the area of the chip to exponentially increases and memory is not inferred where in case of optimization the conversion of for loop makes it non - static address corresponding to it memory is inferred, this cause the chip size to decrease. Hence the optimization of device is there.

So this optimization is really helpful in maintaing the balance between the area and the performance. With the data out comparsion between the questa and Register Transfer level compiler, we can get the results of memory inference with simulation and emulation





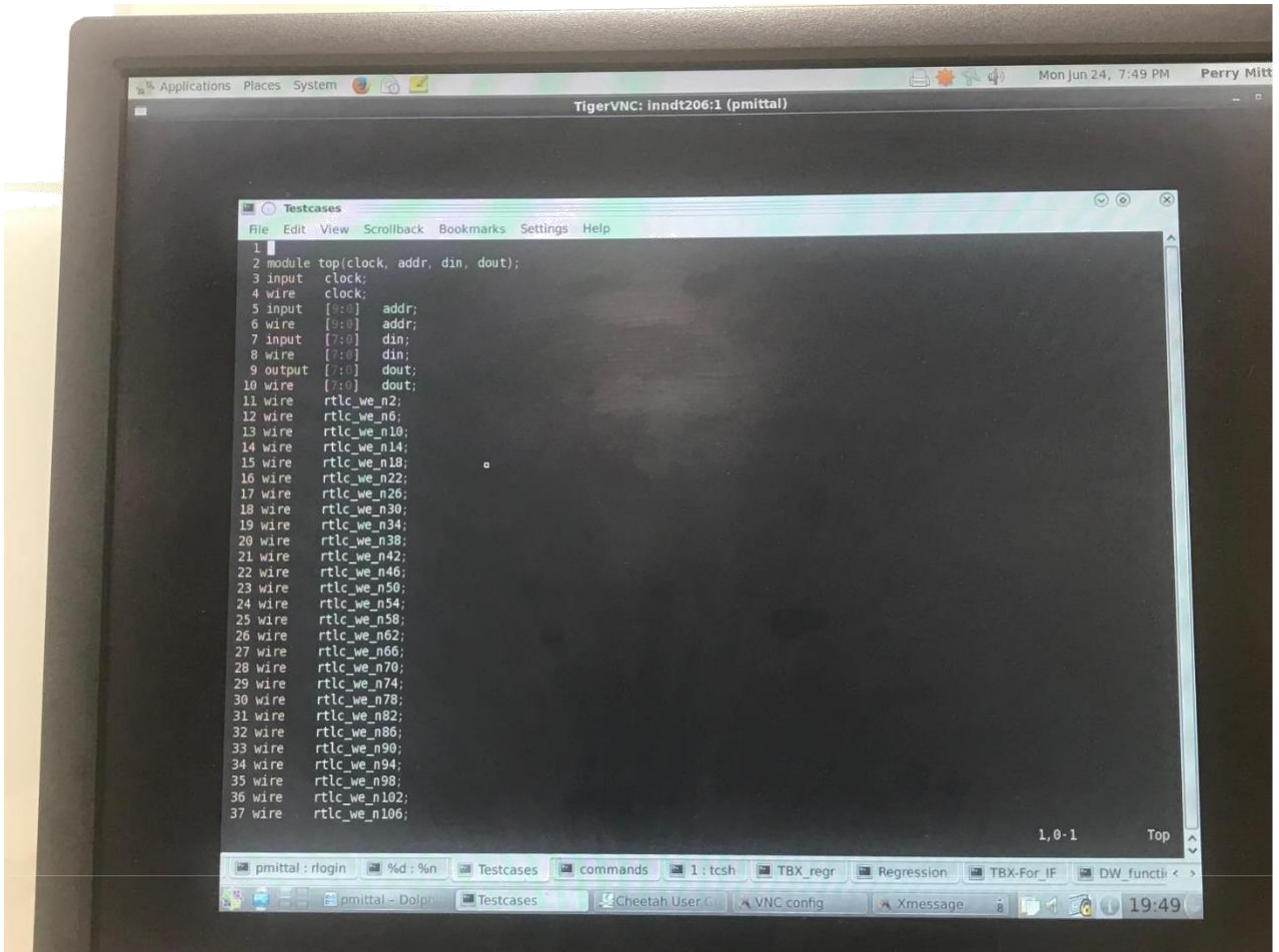


Fig 4.9 Testcases related to the For If Optimization.

In case enrolling of the loop in Fig 4.9, we have the constant address in that case we are not declaring it as memory, it can be declared as logic Block in which we have many adders, Multiplexers and LUTs which takes a lot of area for the chip. Thus memory Inference is really important and helpful in chip designing, we are able to infer memory through this optimization in this case for and if statement is converted into simple if statement, thus that only one write port works at a time. This makes the memory to be inferred causing the small area and better performance of the chip.

All the Verilog program begins with the positive edge of the clock which decides at what edge the clock, begins to work. The clock will stabilize and then it will the output in case of every positive edge of the clock. Then the for loop starts to enroll, Thus this changes will cause the positive edge of the clock to occur.

## 4.7 HANDLING OF THE TASK AND FUNCTION

We are not handling the cases inside the task and function.

We also not handling the Initial block.

```
for (i=0; i<d1 ; i=i+1)
begin
if(i%3 == 1)
    memory[i] = (2**word_size1)-i-
1; else if(i%3 == 2)
    memory[i] = (2**(word_size1/2))-i-
1; else
    memory[i] = i;
end
```

In case of else condition we also not converting the for if loop.

Here in This loop for conversion, we have to check the access type of the variable to be converted. Hence it is determined with the help of register Transfer level Language, how this RTL language is converted into the C language code. This is all about the conversion, we have converted these for loops into the if statement.

Thus with the help of this code, we can have the optimization which is related to all the for loop conversion. Hence all the conversion is done according memory access pattern. We should carefully observe all the memory patterns of this loop is statements which helps in the conversion of if statement.

There is packed and unpacked array in which we have the data width is considered in case of packed array and the unpacked array we have the address depth. The first bit is considered to be the no. of memory blocks and second bit decides the no. of bits in the memory depth.

## 4.9 MEMORY ACCESS TYPE OF THE MEM BLOCK

```
reg [7:0]mem[1024:0][101:0]
always @(posedge clock)
    for(i = 0; i < 101; i=i+1)
        if(i == sel)
            mem[addr][i] = d1 ;
```

In this case also we are not converting because we have access pattern of I is not from the Static, Non\_static, Static + Non\_static and Default Access Type.

A general graph is formed where we can decide whether this graph like structure in which we have control and the data flow in which we can enroll the for loop which consist of the graph like structure in terms of the control and the data flow ie CDFG level .

Here we can see that there is enrolling of the for loop occurs where we can see that no optimztion occurs at the CDFG i.e control and data flow graph.

Here in This loop for conversion, we have to check the access type of the variable to be converted. Hence it is determined with the help of register Transfer level Language, how this RTL language is converted into the C language code. This is all about the conversion, we have converted these for loops into the if statement.

Thus with the help of this code, we can have the optimization which is related to all the for loop conversion. Hence all the conversion is done according memory access pattern. We should carefully observe all the memory patterns of this loop is statements which helps in the conversion of if statement.

Compilation time is also got reduced with this optimization of ports of memory. In which only the one write port is active at a time. This is done with the help of Register Transfer level (RTL) compiler and the Questa simulator. We compared the results with the simulator as well as emulator

## **4.10 EXAMPLE OF THE FOR\_IF CONVERSION WITHOUT MEMORY BLOCK**

The Module with input and output ports without memory block.

```
module top( clock, addr, din, out1);
    input clock ;
    input [4:0] addr ;
    input logic [7:0] din ;
    integer i;
    output logic [7:0] out1[31:0] ;
    always @(posedge clock)
    begin
        for(i = 0; i < 7; i=i+1)
        begin
            if(i == addr)
                out1[i] <= din ;
        end
    end
endmodule
```

The digital design system is described in Verilog because high level Verilog designs are usually described at the level that consist of system registers and transfer of data between various registers through buses, this level of high level description is called as register transfer level (RTL). Verilog construction used in RTL level design are procedural statements continuous assignments and instantiation statements.

## **4.11 CONCLUSION**

This chapter gives the brief introduction about the Memory. The declaration of memory, how to decrease the size of chip using the for if optimization, this chip introduces the for if optimization.

In which the memory inference take place by reducing the no. of ports which cause the addres , subtractors and multiplxers are not formed in place of it memory is formed which causes the size of the chip to reduce so it able to reduce the capacity of the ICs and chip . It also reduces the compilation time which further inhances the performace of the circuit. So overall it able to enhance the capacity, performance and speed of the digital circuits which causes our Verfication technology to excel in the market.



# CHAPTER 5

## RESULTS AND CONCLUSION

### 5.0INTRODUCTION

The below Table shows the comparison between the compilation Time with and without optimization. The memory is inferred in the case of optimization where in other case, the logic Block is formed.

SIMULATION RESULTS		EMULATION RESULTLS	
Compilation Time without Optimization	Compilation Time with Optimization	Compilation Time without Optimization	Compilation Time with Optimization
8 sec	1 sec	7 sec	1 sec
12 sec	3 sec	10 sec	3 sec
15 sec	4 sec	12 sec	4 sec
17 sec	5 sec	15 sec	5 sec

Compilation time is also got reduced with this optimization of ports of memory. In which only the one write port is active at a time. This is done with the help of Register Transfer level (RTL) compiler and the Questa simulator. We compared the results with the simulator as well as emulator.

Memory is not inferred where in case of optimization the conversion of for loop makes it non - static address corresponding to it memory is inferred, this cause the chip size to decrease. Hence the optimization of device is there.

In Memory Block there are write enables and the read enable and the Byte enable as well how much write and byte enable memory is required. Since the RTLC memory required for the write enable as well as byte enable. We need the memory to be declared with address read and wrte ports. The data read and write port, the byte enable read and write port. The memory have various enable which helps to write the adrees bit in memory.

```
1
2 module top(clock,addr,din,dout);
3   input clock ;
4   input [9:0] addr ;
5   input [7:0]din ;
6   output [7:0]dout;
7 // pragma attribute mem ram_block 1
8   integer i;
9   reg [7:0] mem[1023:0];
10
11  initial
12  begin
13    for(i = 0 ; i < 1024 ; i = i+1)
14      begin
15        mem[i] = 0 ;
16      end
17  end
18
19  assign dout = mem[addr] ;
20  always @(posedge clock)
21  begin
22    for(i = 0 ; i < 50; i=i+1)
23      begin
24        if(i == addr)
25          mem[i] <= din ;
26      end
27  end
28 endmodule
```

"top.v" 28L, 509C

1,0-1 All

Fig 5.1 The Verilog Module for the initialization of the memory.

## 5.1 FORMATION OF LUTS AND MULTIPLEXERS

In case enrolling of the loop in Fig 5.1, it has the constant address in that case we are not declaring it as memory, it can be declared as logic Block in which it has many adders, Multiplexers and LUTs which takes a lot of area for the chip. Thus memory Inference is really important and helpful in chip designing, this is able to infer memory through this optimization in this case for and if statement is converted into simple if statement, thus that only one write port works at a time.

So this optimization is really helpful in maintaing the balance between the area and the performance. With the data out comparsion between the questa and Register Transfer level compiler, we can get the results of memory inference with simulation and emulation



```

1
2 module top(clock,addr,din,dout);
3   input clock ;
4   input [9:0] addr ;
5   input [7:0]din ;
6   output [7:0] dout;
7 // pragma attribute mem ram_block 1
8   integer i;
9   reg [7:0] mem[1023:0];
10
11  initial
12  begin
13    for(i = 0 ; i < 1024 ; i = i+1)
14      begin
15        mem[i] = 0 ;
16      end
17  end
18
19  assign  dout = mem[addr] ;
20  always @(posedge clock)
21  begin
22    for(i = 0; i < 50; i=i+1)
23      begin
24        if(i == addr)
25          mem[i] <= din ;
26      end
27  end
28 endmodule

```

"top.v" 28L, 509C

1,0-1 All

RTLc-Build files pmittal : rlogin %d : %n Testcases commands 1 : tcsh TBX\_regr Regression TI

Fig 5.2 Pragma Representation of the mem RAM-BLOCK 1

## 5.2 CONVERSION OF FOR TO IF STATEMENT

Here in this loop for conversion in Fig 5.2, it has to check the access type of the variable to be converted. Hence it is determined with the help of register Transfer level Language, how this RTL language is converted into the C language code. This is all about the conversion, it has converted these for loops into the if statement. Thus with the help of this code, it can have the optimization which is related to all the for loop conversion. Hence all the conversion is done according memory access pattern.

In unrollable for loop it has constant address in that case memory is not declared, it is assumed as a logic block which makes the area of the chip to exponentially increases and memory is not inferred where in case of optimization the conversion of for loop makes it non - static address corresponding to it memory is inferred, this cause the chip size to decrease.

## **5.3 A COMPARITIVE ANALYSIS OF QUESTA AND REGISTER TRANSFER LEVEL**

The digital design system is described in Verilog because high level Verilog designs are usually described at the level that consist of system registers and transfer of data between various registers through buses, this level of high level description is called as register transfer level (RTL). Verilog construction used in RTL level design are procedural statements continuous assignments and instantiation statements.

In unrolling for loop it has constant address in that case memory is not declared, it is assumed as a logic block which makes the area of the chip to exponentially increases and memory is not inferred where in case of optimization the conversion of for loop makes it non - static address corresponding to it memory is inferred, this cause the chip size to decrease. Hence the optimization of device is there.

Here in This loop for conversion, it has to check the access type of the variable to be converted.Hence it is determined with the help of register Transfer level Language, how this RTL language is converted into the C language code. This is all about the conversion, it has to be converted these for loops into the if statement.Thus with the help of this code, we can have the optimization which is related to all the for loop conversion. Hence all the conversion is done according memory access pattern. It should carefully observe all the memory patterns of this loop is statements which helps in the conversion of if statement.

In unroll able for loop it has constant address in that case memory is not declared, it is assumed as a logic block which makes the area of the chip to exponentially increases and memory is not inferred where in case of optimization the conversion of for loop makes it non - static address corresponding to it memory is inferred, this cause the chip size to decrease. Hence the optimization of device is there.

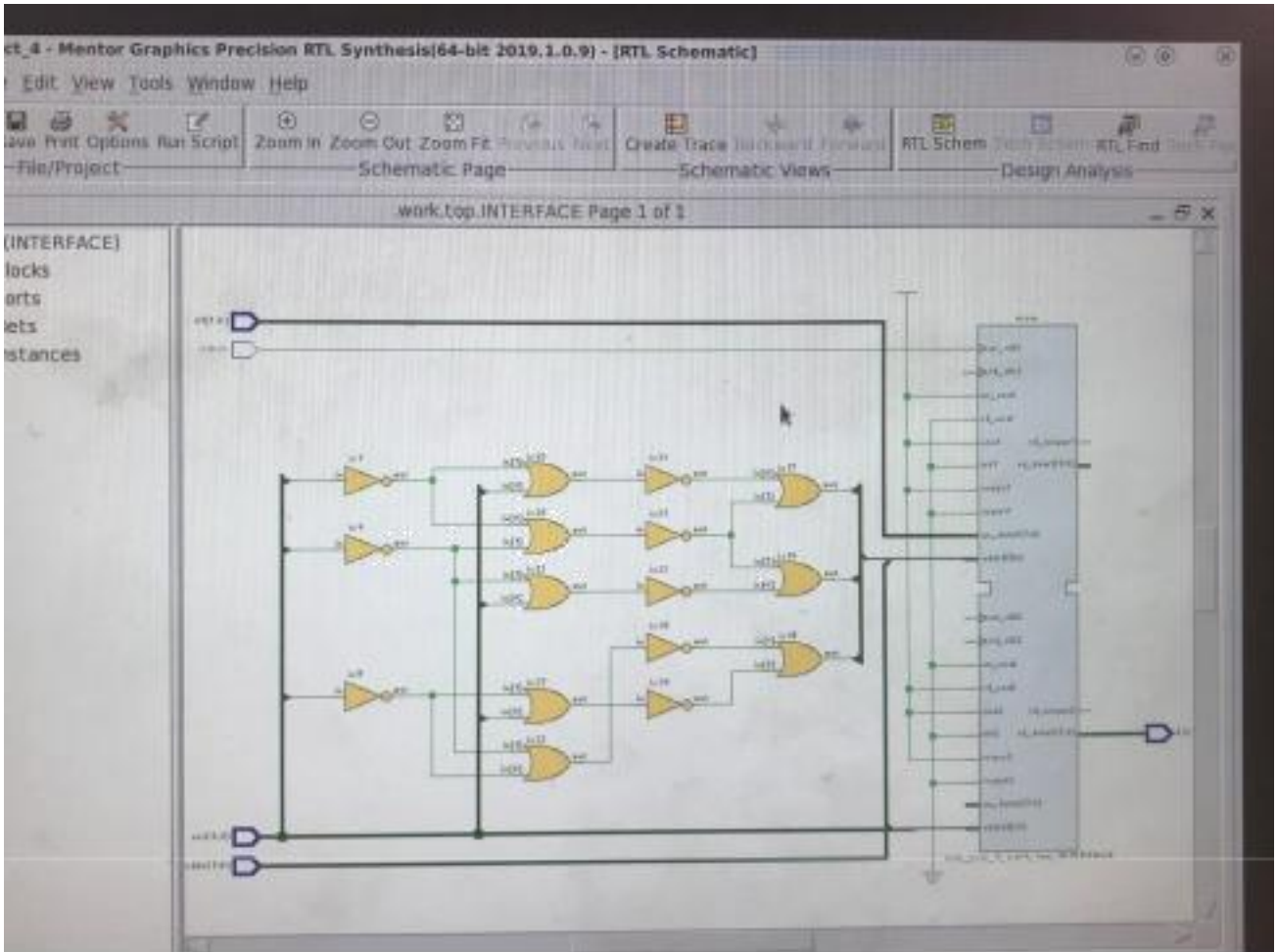


Fig 5. 3 Precesion tool with for loop optimization

#### 5.4 READ AND WRITE ENABLE IN MEMORY BLOCK

In Memory Block there are write enables and the read enable and the Byte enable as well how much write and byte enable memory is required. Since the RTLC memory required for the write enable as well as byte enable. We need the memory to be declared with address read and write ports. The data read and write port, the byte enable read and write port. The memory have various enable which helps to write the adress bit into the memory.

The precesion tool in Fig 5.3 is the tool for the formal verification with use of all the LUTs, multiplexers with this tool which is very helpful for the gate level synthesis all the gate level synthesis is done with the help of this precesion tool.

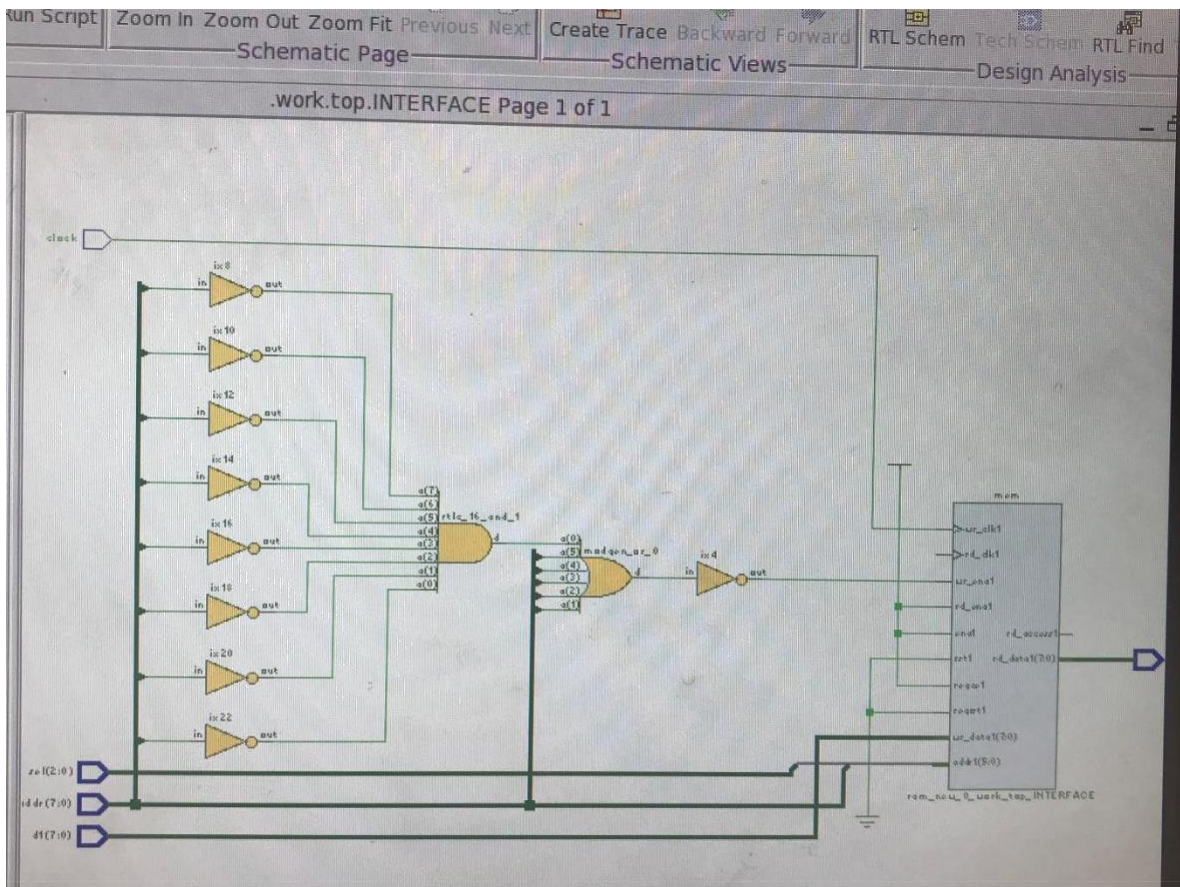


Fig 5.5 Precesion tool conversion of the FOR IF Optimization

This Precesion tool in Fig 5.5 makes the optimization is done in case of it has large Number of ports which can hamper the memory during synthesis. So with this conversion it is converted into single port and it is also reducing the compilation time.

## 5.4 CONCLUSION

In case enrolling of the For loop, it has the constant address in that case it is not declaring it as memory, it can be declared as logic Block in which it has many adders, Multiplexers and LUTs which takes a lot of area for the chip. Thus memory Inference is really important and helpful in chip designing, it is able to infer memory through this optimization in this case for and if statement is converted into simple if statement, thus that only one write port works at a time. This makes the memory to be inferred causing the small area and better performance of the chip.

# CHAPTER 6

## MAIN CONCLUSION AND FUTURE SCOPE OF WORK

### 6.0 MAIN CONCLUSION

In this dissertation while working on the verification of digital design chips, the optimization of time and space complexity is solved by using the memory inference and decreasing the number of ports of memory. Various logic block like adders, subtractors and multiplexers are solved by using this optimization, further more all the results have been verified using the questa simulator of mentor graphics and the Register transfer level compiler in which first of all it creates a library and then analyze that file in disk and then compile the data, these both are the procedures for simulations and then finally it also verified the results using emulator i.e VELOCE. The Emulation process consists of the Velcomp flow in which it has a static graph which maintain all the dependencies between each phase. In this flow in Fig 6.0 user needs to create the configuration file named Veloce.config to provide all the options. So this present work concludes the optimization of space of the chip and verification of the chip using the questa and register transfer level compiler and the Emulator Veloce, where all the testing is done using the Velcomp flow which contains all the PRERTLCFORDFT, PRERTLCFORUPF.

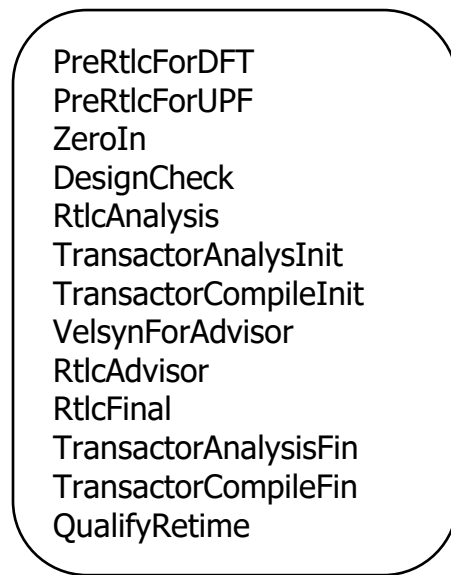


Fig 6.0 Velcomp flow for the emulator

## **6.1 FUTURE SCOPE OF WORK**

This work can be extended in tasks and functions and other complex algorithms where every thing works in repetitive fashion.

1. The process of synthesis and compilation is also reduced by decreasing the no. of flip flops with enhancements in timing analysis.
2. A new logic can be developed which will enhance the optimizations for the no. of flip flops and the retimmmg which can also play a role in generating a new flow.
3. Retimming is also another technique which can decrease the timing analysis between two flip flops which can be implemented in future in this project.
4. The verification of millions of IC and chips can be done with high precision and with less amount of time. It can be supported for various other modules that can perform with larger capacity.
5. Memory ports can be further decreased with more enhancements like creating the one hot logic technique which can be implemented in future.
6. There can also be check for dissolving the bad memories as it can impact at the testcases where bad memories can be dissolved.

## **6.2 COMING EMULATOR AND SIMULATOR**

The recent Emulators VeloceX and VeloceY can be used in further research work in which they can boost the runtime and performance by 4.5 X , so all the testing of chips and ICs are further enhanced by 2 X speed . By using these emulators, the reseach work can be further improvised by using the millions of chips to test with high optimization. The Emulators provide the complete verification by using the accurate performance analysis and the various advanced verfications methodologies. It provides the compele verification such that advanced the testing of all the chips and IC with advanced accuracy and with high optimization.



## REFERENCES

- [1] Li H, Steurer M, Shi K L, et al. "development of a unified design, test, and research platform for wind energy systems based on hardware-in the-loop real-time simulation." *Industrial Electronics, IEEE Transactions on* 53.4 (2006):1141-1151
- [2] Rakopoulos, C. D., and E. G. Glakoumis. "Sensitivity analysis of transient diesel engine simulation." *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering* 220.1 (2006): 89-101.
- [3] Bayat Sarmadi. S.. Miremadi. S.G.. Asadi. G., Ejlali. k, "Fast Prototyping with Co-operation of Simulation and Emulation." in *Proceedings of 12<sup>th</sup> International conference on Field Programmable Logic and Applications, 2002*
- [4] Canellas, N.. Moreno. J. M.. "Speeding up hardware prototyping by incremental Simulation and Emulation". in *Proceedings of 11<sup>th</sup> International Workshop on Rapid System Prototyping, 2011.*
- [5] Clement. B., Herremeulc, R.. Lmtreihccq. E.. Ramanadin. B., Coulomb. p., Pogodalla, F.. "Fast Prototyping: A System Design Flow Applied to a Complex System-On-Chip Multiprocessor Design". in *Proc. Of the ACM IEEE Design Automation Conference [DAC]. 1999*
- [6] De Micheli. G., *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, 1994.
- [7] Evans, A.. Silburl, A., Vrckovnik, G., Brown. T.. Dufresne. M.. Hall, G.. Ho, T.. Liu., Y.. "Functional Veri-cation of large ASICS". In *Proc. Of the ACM/ IEEE design Automation Conference (DAC). 1998.*
- [8] IEEE Std 1076-1993: *IECE Standard VHDL Language Reference Manual.*
- [9] IEEE Std 1364-1995: *IECE Standard Verilog HDL Language Reference Manual.*
- [10] Kim, N.. Choi. H.. Lee, S., Lee. S., -C. Park. I.. -M. Kpn. C.. "Virtual Chip: Making Functional Models Work On Real Target Systems", in *hoc. Of the ACM/IEEE Design Automation Conference (DAC), 1998.*
- [11] Kudlugi. M. , Hassoun, S.. Selvidge. C.. Pryor. D.. "A Transaction-Bared Unified Simulation and Emulation Architecture for Functional Verification," in *Proceedings of 38th ACM/IEEE Design automation conference, 2001, pp. 623.628.*
- [12] Rowson, J. A.. "Hardware Software Co-Simulation", in *Proceedings of 31st ACM/IEEE Design Automation Conference. 1996. pp. 439440.*



- [13] Varghese, J., Butts, M., and Batcheller, J., "An efficient logic emulation system," IEEE Trans. on VLSI Syst., vol. I, June 1993, pp. I7 I. 174.
- [14] ] Y. Zheng and D. Nicol, "A virtual time system for openvz-based network emulations," in 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation (PADS). IEEE, 2011, pp. 1–10.
- [15] D. Nicol, D. Jin, and Y. Zheng, "S3F: The Scalable Simulation Framework Revisited," in Proceedings of the 2011 Winter Simulation Conference, Phoenix, AZ, December 2011.
- [16] M. Liljenstam, J. Liu, D. Nicol, Y. Yuan, G. Yan, and C. Grier, "Rinse: the real-time immersive network simulation environment for network security exercises," in Proceedings of Workshop on Principles of Advanced and Distributed Simulation, 2005. PADS 2005. IEEE, 2005, pp. 119–128.
- [17] R. Fujimoto, "Parallel discrete event simulation," in Proceedings of the 21st conference on Winter simulation. ACM, 1989, pp. 19–28.
- [18] Y. Zheng, D. Nicol, D. Jin, and N. Tanaka, "A Virtual Time System for Virtualization-Based Network Emulation and Simulation," Journal of Simulation, 2011, To appear
- [19] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu et al., "Advances in network simulation," Computer, vol. 33, no. 5, pp. 59–67, 2002.
- [20] ] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and ´ D. Becker, "Scalability and accuracy in a large-scale network emulator," ACM SIGOPS Operating Systems Review, vol. 36, no. SI, pp. 271–284, 2002.
- [21] T. Benzel, R. Braden, D. Kim, C. Neuman, A. Joseph, K. Sklower, R. Ostrenga, and S. Schwab, "Experience with DETER: A testbed for security research," in 2nd International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2006. TRIDENTCOM 2006. IEEE, 2006, pp. 10–388.
- [22] Co-Verification Debugger Enables Hardware and Software Communication for SoC Verification. [Online] Available. <http://www.axiscorp.com/products/coverification.html>
- [23] (2003, Feb.) Aptix and Zaiq Reseller Agreement to Improve Communication System Design Validation. [Online] Available. <http://www.aptix.com/news/news.htm>
- [24] D. Brahme, S. Cox, J. Gallo, M. Glasser, W. Grundmann, C. Ip, W. Paulsen, J. Pierce, J. Rose, D. Shea, and K. Whiting, The Transaction-Based Verification Methodology. Berkeley, CA: Cadence Berkeley Labs, 2000.
- [25] F. Carbognani, C. Lennard, C. Ip, A. Cochrane, and P. Bates, "Qualifying precision of abstract systemC models using the systemC verification standard," in Proc. Design, Automation, Test in Europe, 2002

- [26] F. Casaubielilh, A. McIssac, M. Benhamin, M. Barttley, F. Pogodalla, F. Rocheteau, M. Belhadj, J. Eggleton, G. Mas, G. Barrett, and C. Berthet, “Functional verification methodology of chameleon processor,” in Proc. ACM/IEEE Design Automation Conf., 1996, pp. 421–426.
- [27] B. Clement, R. Hersemeule, E. Lantreibecq, B. Ramanadin, P. Coulomb, and F. Pogodalla, “Fast prototyping: a system design flow applied to a complex system-on-chip multiprocessor design,” in Proc. ACM/IEEE Design Automation Conf., 1999, pp. 420–424.
- [28] A. Clouard, G. Mastrococco, F. Carbognani, A. Perrin, and F. Ghenassia, “Toward bridging the precision gap between SoC transactional and cycle accurate levels,” in Proc. Design, Automation, Test in Europe Conf., 2002.
- [29] A. Evans, A. Silburt, G. Vrckovnik, T. Brown, M. Dufresne, G. Hall, T. Ho, and Y. Liu, “Functional verification of large ASICS,” in Proc. ACM/IEEE Design Automation Conf., 1998, pp. 650–655.
- [30] G. Ganapathy, R. Narayan, G. Jordan, and D. Fernandez, “Hardware emulation for functional verification for K5,” in Proc. ACM/IEEE Design Automation Conf., 1996, pp. 315–317
- [31] P. Hardee. Transaction-Level Modeling and the ConvergenSC Accelerated Transaction Based Co-Simulation Products. [Online] Available. <http://www.coware.com>
- [32] C. Ip and S. Swan. (2003) A Tutorial Introduction on the New SystemC Verification Standard. [Online] Available. <http://www.systemC.org>
- [33] M. Kantrowitz and L. Noack, “I’m done simulating: now what? Verification coverage analysis and correctness checking of the DEC-chip21164 alpha microprocessor,” in Proc. ACM/IEEE Design Automation Conf., 1996, pp. 325–330.
- [34] N. Kim, H. Choi, S. Lee, S. Lee, I.-C. Park, and C.-M. Kyun, “Virtual chip: making functional models work on real target systems,” in Proc. ACM/IEEE Design Automation Conf., 1998, pp. 170–173.
- [35] A. Meyer. A Loosely Coupled C/Verilog Environment for System Level Verification. [Online] Available. <http://www.zaiqtech.com>
- [36] ] J. Monaco, D. Holloway, and R. Raina, “Functional verification methodology for the powerPC 604 microprocessor,” in Proc. ACM/IEEE Design Automation Conf., 1996, pp. 319–324
- [37] I. Moussa, T. Grellier, and G. Nguyen, “Exploring SW performance using SoC transaction-level modeling,” in Proc. Design, Automation, and Test in Europe Conf., 2003, pp. 120–125.
- [38] M. Newman, “Test benches in C speed verification by unifying emulation and simulation,” Integrated Syst. Design, pp. 34–40, 1999
- [39] Cadence Application Note. (2003) Accelerated transaction based co-simulation. [Online] Available. <http://www.cadence.com>

- [40] V. Popescu and B. McNamara, “Innovative verification strategy reduces design cycle time for high-end sparc processor,” in Proc. ACM/IEEE Design Automation Conf., 1996, pp. 311–314
- [41] B. Schnaider and E. Yogev, “Software development in a hardware simulation environment,” in Proc. ACM/IEEE Design Automation Conf., 1996, pp. 684–689.
- [42] R. Stevens, UNIX Network Programming, and Networking APIs: Sockets and XTI, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 1997, vol. 1
- [43] S. Swan. (2001) An Introduction to System Level Modeling in SystemC 2.0. [Online] Available. <http://www.systemC.org>
- [44] M. Wannemacher, M. Munteanu, S. Perret, R. Singer, "Taking the best out of two worlds: prototyping and hardware emulation," Seventh IEEE International High-Level Design Validation and Test Workshop (HLDVT'02), 2002, pp.156-161.
- [45] P. Rashinkar, P. Paterson, and L. Singh, System-On-AChip Verification: Methodology and Techniques, Kluwer, 2000, p. 153.
- [46] I. Mavroidis and I. Papaefstathiou. “Efficient testbench code synthesis for a hardware emulator system,” Proceedings of the Conference on Design, Automation and Test in Europe (DATE '07). EDA Consortium, San Jose, CA, USA, pp. 888-893.
- [47] Mentor Graphics, “The Target Platform Methodology for HW/SW Debugging before Silicon”, White Paper, url: [www.mentor.com](http://www.mentor.com).