Implementation of a 32-bit MIPS Based RISC Processor using Cadence

[1]Mohit N. Topiwala, [2]N. Saraswathi
[1]M.Tech Student, [2]Asst. Professor S.G
[1,2]Department Of Electronics and Communication,SRM University,Chennai, India
[1]mohit.topiwala@gmail.com,[2]saraswathy.n@ktr.srmuniv.ac.in

***Abstract-* This paper presents implementation of a 5-stage pipelined 32-bit High performance MIPS based RISC Core. MIPS (Microprocessor without Interlocked Pipeline Stages) is a RISC (Reduced Instruction Set Computer) architecture. A RISC is a microprocessor that had been designed to perform a small set of instructions, with the aim of increasing the overall speed of the processor. MIPS have 5 stages of pipeline viz. Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Access (MEM) and Write Back (WB) modules. The various modules being used are Instruction Memory, Data Memory, ALU, Registers etc. The aim of this paper is to include Hazard detection unit and Data forwarding unit for efficient implementation of the pipeline. The design is developed using Verilog-HDL. The main goal is to do the complete ASIC flow (RTL to GDS II), using Cadence tool. The module functionality and performance issues like area, power dissipation and propagation delay are analyzed using Cadence RTL complier using typical libraries of tsmc 0.18 um technology.**

***Keywords-* MIPS, 5-stage pipeline, ASIC flow.**

I. INTRODUCTION

Microprocessors and Microcontrollers have been designed around two philosophies: Complex Instruction Set Computer (CISC) and Reduced Instruction Set Computer (RISC). The CISC concept is an approach to the Instruction Set Architecture (ISA) design that emphasizes doing more with each instruction using a wide variety of addressing modes, variable number of operands in various locations in its Instruction Set. As a result, the instructions are of widely varying lengths and execution times thus demanding a very complex Control Unit, which occupies a large real estate on chip. On the other hand, the RISC Processor have reduced number of Instructions, fixed instruction length, more general-purpose registers, load-store architecture and simplified addressing modes which makes individual instructions execute faster, achieve a net gain in performance and an overall simpler design with less silicon consumption as compared to CISC. The above features make the RISC design ideally suited to participate in a powerful trend in the embedded Processor market – the "system-on-a-chip" [9]. The most common RISC microprocessors are ARM, SPARC, MIPS and IBM's PowerPC. There are number of semiconductor companies implementing RISC processor on "system on chip" such as Atmel AVR, Micro Blaze which are widely used for embedded & DSP applications.

MIPS processor design is based on the RISC design principle that emphasizes on load/store architecture [1]. Due to the difference in time taken to access a register as compared to a memory location, it is much faster to perform operations in on chip registers rather than in memory. The architecture remains the same for all MIPS based processors while the implementations may differ like single cycle, multi-cycle and pipelined implementation [2].

Nowadays power is most important performance parameter for embedded and portable application [8]. But in any integrated circuit, a tradeoff between power, area and delay is there. For certain applications, low power circuits will be needed and the design engineers have to compromise with more area and delay. In this paper, power reduction is achieved through by-passing pipeline stages that cause unnecessary switching activity. Dynamic power depends upon the switching activity or in general number of transitions and is given by equation [6],

$$P = \frac{1}{2} C_0 V_{in}^2 N f \qquad (1)$$

Thereby decreasing number of transitions (N) results in reduced dynamic power consumption [6].

The section II deals with the MIPS Instruction Set, while III with the MIPS architecture and the IV with the pipelined architecture with improved datapath. The section V gives simulation results with explanations.

II. MIPS INSTRUCTION SET

MIPS design consist three types of instruction set. Register type, immediate type and Jump type. The instruction format is shown in the below figure, respectively [3].

A. Register Type (R-Type):

	(31 - 26)	(25 - 21)	(20 - 16)	(15 - 11)	(10 - 6)	(5 - 0)
R-Format	Opcode	Rs	Rt	Rd	shift	Function

Figure 1. R-Type Instructions Format

Figure 1 shows R-Type instruction format. Here last 6 bits represents the opcode. Next 15 bits represents 3 registers Rs, Rt and Rd respectively, on which operations are performed. Rs, Rt are source registers and Rd is the destination registers respectively. Next 5 bits is shift amount which is point to the number of bits to be shifted. Last 6 bits is function field points to the function that needs to be performed on the registers.

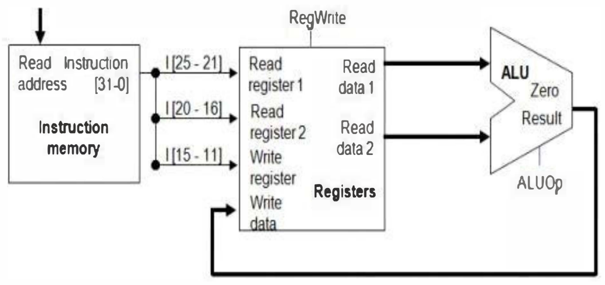


Figure 2. Data Path for R-Type Instructions

The data path for R-Type instruction can be depicted as follows in figure 2. It is used mainly for ADD, SUB and OR operation. E.g. add Rd, Rs, Rt.

Here signed addition contents of (Rs) + (Rt) is saved into Rd.

B. Immediate Type (I-Type):

	(31 - 26)	(25 - 21)	(20 - 16)	(15 – 0)
I- Format	Opcode	Rs	Rt	Immediate value

Figure 3. I-Type Instructions Format

Figure 3 shows I-Type instruction format. Similar to R-type, first 6 bits represents the opcode and next 10 bits represents Rs, Rt respectively. Here Rs is source register and also note that Rt is source register for store and destination register for load operation. Last 16 bits represents immediate value which is a part of instruction but not part of memory.

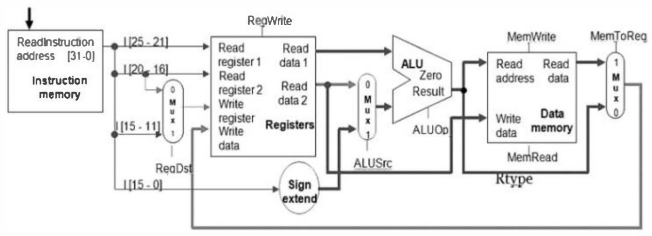


Figure 4. Data Path for I-Type Instructions

The data path for I-Type instruction can be depicted as follows in figure 4. It shows that Rt register can be used both as source and destination accordingly and last16 bits is the immediate value sent to sign extend and then to ALU, for performing the required operation. It is used for ADDI, ANDI, and ORI operation. E.g. addi Rt, Rs, 5

Here (Rs) + 5 is stored in the destination register Rt. 5 is immediate value.

C. Jump Type (J-Type):

	(31 - 26)	(25 - 0)
J-Format	Opcode	Branch target address

Figure 5. J-Type Instructions Format

Figure 5 shows J-Type instruction format. First 5 bits of this instruction format represent the type of branch operation to be performed. The remaining 26 bits represents the branch offset in 2's complement format. This number is added to the value of the PC to obtain the branch target address.

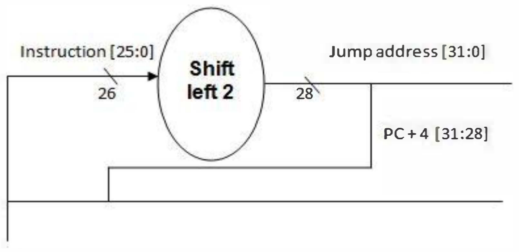


Figure 6. Data Path for J-Type Instructions

In figure 6 shows functionality of J-type instruction. It shows that the last four bits of PC+4 are appended to the shift left by 2 value of the 26-bit instruction, taken from Instruction Memory, to get the 32-bit jump address. E.g. j trgt.

Here, j is jump instruction word and trgt is target. It skips the other instructions and jumps to the target.

III. MIPS ARCHITECTURE

MIPS based RISC processor is basically pipelined architecture implementation. MIPS architecture carried 5 stages of pipeline. Pipelining is nothing but doing more than one operation, in a single data path. Pipelining is a technique which is used to improve overall performance of RISC processor [7]. A multicycle CPU consists of many processes. For example load might take up to 5 clock cycles, but beq takes only 3 clock cycles. So if one process is taking place, instead of waiting for the process to complete, simultaneously start a new process in the same data path, without disturbing the previous process. For this to happen, the each part of the process is divided into various pipelined stages. So after every clock, the process is stored into next pipelined stage, enabling another operation to start in that stage without disturbing the previous process. Hence all the stages in the path can be used simultaneously. This in turn can increase the throughput of MIPS design.

MIPS processor architecture has been implemented using 5 pipeline stages. These pipeline stages are Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory Access (MEM) and Write Back (WB). These stages are separated by special registers called pipeline registers. The purpose of these registers is to separate the stages of the instructions so that there is no conflicting data due to multiple instructions being executed simultaneously. They are named after the stages that they are placed in-between: IF/ID Register, ID/EX Register, EX/MEM Register, and MEM/WB Register. Figure 7 shows datapath for MIPS pipelined processor.

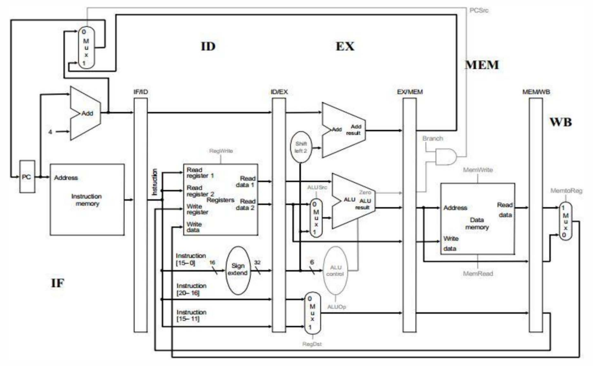


Figure 7. MIPS Pipelined Processor Datapath

A. Instruction Fetch:

Program Counter (PC) is used to fetch the instruction from the Instruction Memory and is stored in the Instruction Register (IF/ID) at the next positive clock. This stage has various modules like Instruction Memory, which holds the instructions needed. PC holds the address of the current instruction, which is used as address to the Instruction Memory. The instructions read out from the Instruction memory are stored in the Instruction Register, which is a part of IF/ID stage.

B. Instruction Decode:

In this stage, decodes the instructions sent from Instruction register. Based on the instructions, it reads the operands required for register file. Out of 32-bits, 16 go to sign extend, where those 16 bits are extended to 32-bits. The register file module gives out the value of 2 registers, which are sent to ALU through ID/EX stage.

C. Execution:

All the instructions are executed in this stage. All ALU operations like arithmetic and logical operations, take place in this stage. It performs operations on the data sent from ID/EX stage. This stage also has left shift by2 and an adder, for beq operation. The result from ALU is sent to ALUout register which is in EX/MEM stage.

D. Memory Access:

In this stage, memory access stage's purpose is to read from and write to the data memory. The control signals passed in the EX/MEM register to determine which of the operations to do. The output of the memory is written into the MEM/WB register along with the WB control that is passed from the EX/MEM register.

E. Writeback:

This stage basically writes back the result into register file. Also it is responsible for taking the writing the data that which have just computed or loaded from memory out of the EX/MEM register and writing it to a one of the registers in the register file.

IV. Pipelined Architecture With Improved Datapath

This pipelined architecture consist two extra modules Hazard detection unit and Data forwarding unit to improve datapath of architecture.

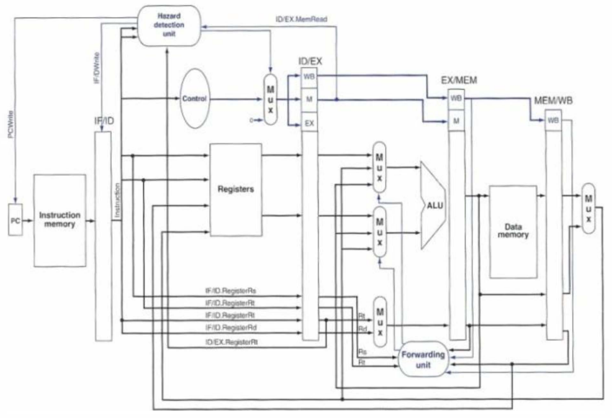


Figure 8. Pipelined Architecture with Improved Datapath

A. Hazard Detection Unit:

There are three types of pipeline hazards: Structural hazard, data hazard and control hazard. In this paper we are concentrating on data hazard only. Data hazards arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of the instructions in the pipeline, thus causing the pipeline to stall until the results are made available.

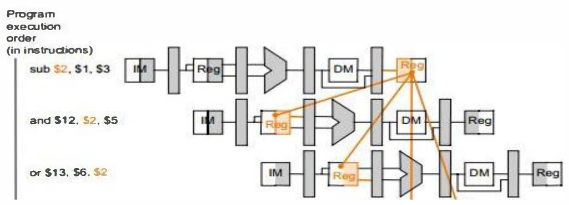


Figure 9. Pipelined Data Dependencies

Let us take an example of the group of successive instructions and understand the issue of data hazard, E.g. [4]:

sub $**2**, $1,$3	# Register $2 written by sub
and $12,$**2**,$5	# 1st operand($2) depends on sub
or $13,$6,$**2**	# 2nd operand($2) depends on sub

Above all instructions are dependent on sub instruction. \$2 stores resulting subtraction of \$1 and \$3. Figure 9 shows the dependency of these instructions. It is clearly shown that \$2 updates its value at clock cycle 5 and before that the written value is unavailable but all the successive instructions followed by sub instruction read the value from \$2. So basically they need updated value in very next clock cycle. This is called **data hazard**.

B. Data Forwarding Unit:

One solution of data hazard is called Data forwarding, which supplies the resulting operand to the dependant instruction as soon it has been computed. Figure 10 shows how dependencies are resolved using a Data forwarding unit. It shows that in sub instruction, result is available at EXECUTION stage (after 3rd clock cycle) and successive instructions reads \$2 at the end of execution stage or 4th or 5th clock cycle. This means instructions can be execute without stalls by just forwarding the data. So, here forwarding unit gives remedy for data hazard.

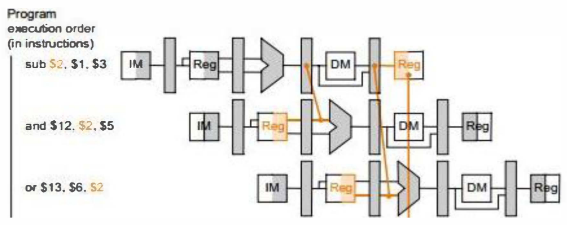


Figure 10. Pipelined Data Dependencies Resolved with Forwarding

V. SIMULATION RESULTS

The verilog code for 32-bit MIPS based RISC Processor compiled using Cadence NC launch and simulated using Cadence Simvision tool to check their outputs. Simulation waveform is shown in figure 11.

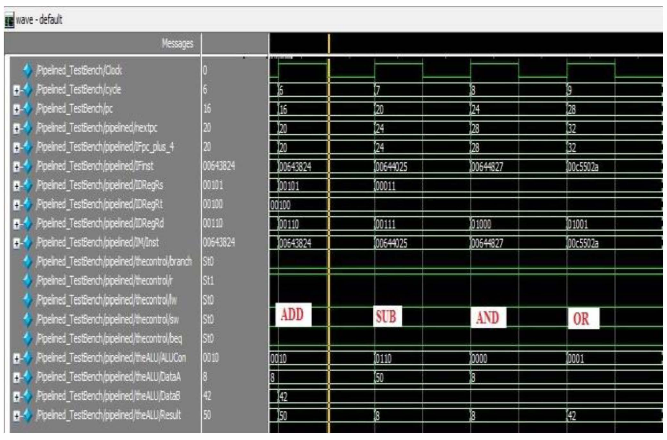
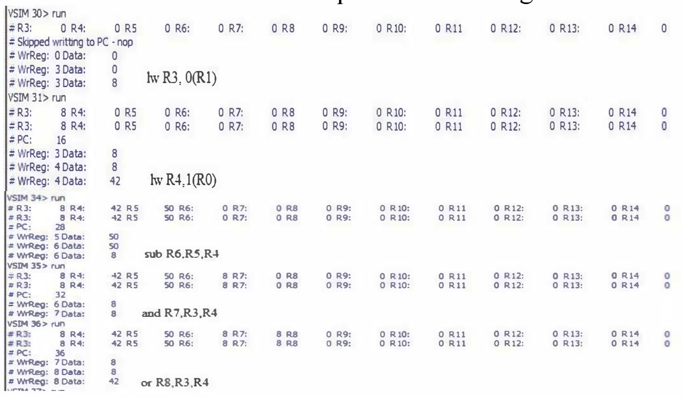

wave - default
Messages
/Pipelined_TestBench/Clock 0
/Pipelined_TestBench/cycle 6 | 6 7 8 9
/Pipelined_TestBench/pc 16 | 16 20 24 28
/Pipelined_TestBench/pipelined/nextpc 20 | 20 24 28 32
/Pipelined_TestBench/pipelined/IFpc_plus_4 20 | 20 24 28 32
/Pipelined_TestBench/pipelined/IFinst 00643824 | 00643824 00644025 00644827 00c5502a
/Pipelined_TestBench/pipelined/IDRegRs 00101 | 00101 00011
/Pipelined_TestBench/pipelined/IDRegRt 00100 | 00100
/Pipelined_TestBench/pipelined/IDRegRd 00110 | 00110 00111 01000 01001
/Pipelined_TestBench/pipelined/IM/Inst 00643824 | 00643824 00644025 00644827 00c5502a
/Pipelined_TestBench/pipelined/thecontrol/branch St0
/Pipelined_TestBench/pipelined/thecontrol/r St1
/Pipelined_TestBench/pipelined/thecontrol/lw St0
/Pipelined_TestBench/pipelined/thecontrol/sw St0
/Pipelined_TestBench/pipelined/thecontrol/beq St0
ADD
SUB
AND
OR
/Pipelined_TestBench/pipelined/theALU/ALUCon 0010 | 0010 0110 0000 0001
/Pipelined_TestBench/pipelined/theALU/DataA 8 | 8 50 8
/Pipelined_TestBench/pipelined/theALU/DataB 42 | 42
/Pipelined_TestBench/pipelined/theALU/Result 50 | 50 8 8 42

Figure. 11 Simulation Waveform of MIPS

The test bench is written to test the instructions of MIPS design. Some series of instructions have sent in test bench, which were read from a memory to instruction memory. The registers R0 and R1 are already stored with data 32’d42 and 32’d8 respectively. Test bench is simulated successfully in Cadence Simvision tool and output is shown in figure 12.



```
VSIM 30> run
# R3:      0 R4:      0 R5      0 R6:      0 R7:      0 R8      0 R9:      0 R10:      0 R11      0 R12:      0 R13:      0 R14      0
# Skipped writting to PC - nop
# WrReg:  0 Data:      0
# WrReg:  3 Data:      0
# WrReg:  3 Data:      8        lw R3, 0(R1)
VSIM 31> run
# R3:      8 R4:      0 R5      0 R6:      0 R7:      0 R8      0 R9:      0 R10:      0 R11      0 R12:      0 R13:      0 R14      0
# R3:      8 R4:      0 R5      0 R6:      0 R7:      0 R8      0 R9:      0 R10:      0 R11      0 R12:      0 R13:      0 R14      0
# PC:     16
# WrReg:  3 Data:      8
# WrReg:  4 Data:      8
# WrReg:  4 Data:     42        lw R4,1(R0)
VSIM 34> run
# R3:      8 R4:     42 R5     50 R6:      0 R7:      0 R8      0 R9:      0 R10:      0 R11      0 R12:      0 R13:      0 R14      0
# R3:      8 R4:     42 R5     50 R6:      0 R7:      0 R8      0 R9:      0 R10:      0 R11      0 R12:      0 R13:      0 R14      0
# PC:     28
# WrReg:  5 Data:     50
# WrReg:  6 Data:     50
# WrReg:  6 Data:      8        sub R6,R5,R4
VSIM 35> run
# R3:      8 R4:     42 R5     50 R6:      8 R7:      0 R8      0 R9:      0 R10:      0 R11      0 R12:      0 R13:      0 R14      0
# R3:      8 R4:     42 R5     50 R6:      8 R7:      0 R8      0 R9:      0 R10:      0 R11      0 R12:      0 R13:      0 R14      0
# PC:     32
# WrReg:  6 Data:      8
# WrReg:  7 Data:      8        and R7,R3,R4
VSIM 36> run
# R3:      8 R4:     42 R5     50 R6:      8 R7:      8 R8      0 R9:      0 R10:      0 R11      0 R12:      0 R13:      0 R14      0
# R3:      8 R4:     42 R5     50 R6:      8 R7:      8 R8      0 R9:      0 R10:      0 R11      0 R12:      0 R13:      0 R14      0
# PC:     36
# WrReg:  7 Data:      8
# WrReg:  8 Data:      8
# WrReg:  8 Data:     42        or R8,R3,R4
```

Figure 12. Snapshot of Test Bench Output

Then the MIPS design synthesized in Cadence RTL compiler using typical libraries of TSMC 0.18 um technology and area and power analysis was done. Frequencies varying from 100MHz to 1000MHz and results are shown in Table I.

TABLE I: FREQUENCY VS. POWER

	Frequency (MHz)	Leakage Power (mW)	Switching Power (mW)	Total Power (mW)
1	100	0.0315	79.250	79.282
2	250	0.0315	187.653	187.684
3	500	0.0315	356.736	356.768
4	750	0.0315	518.206	518.237
5	1000	0.0315	674.778	674.810

Figure 13 show the graph between dynamic power and frequency. This graph concludes that dynamic power linearly increases with frequency. Figure 14 shows the area of MIPS design which is generated by Cadence tool.

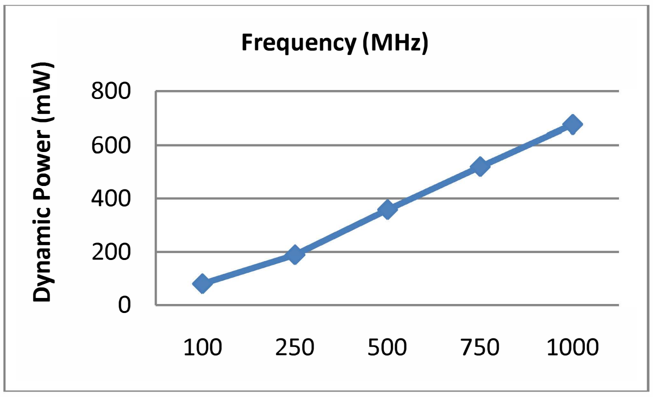
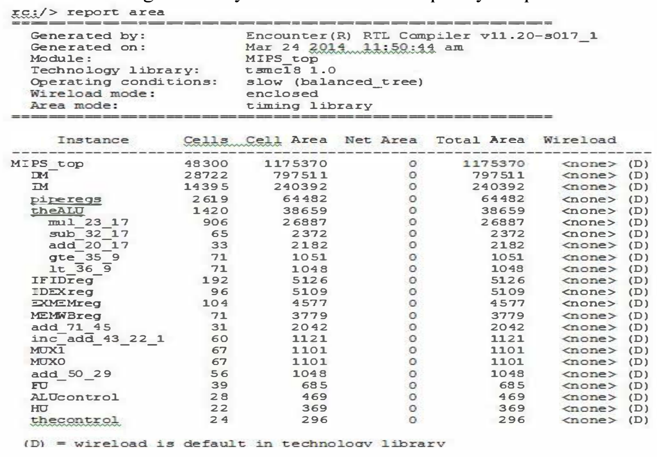


Figure 13. Dynamic Power Vs. Frequency Graph



```
rc:/> report area
============================================================
  Generated by:             Encounter(R) RTL Compiler v11.20-s017_1
  Generated on:             Mar 24 2014  11:50:44 am
  Module:                   MIPS_top
  Technology library:       tsmc18 1.0
  Operating conditions:     slow (balanced_tree)
  Wireload mode:            enclosed
  Area mode:                timing library
============================================================

     Instance        Cells  Cell Area  Net Area  Total Area  Wireload
---------------------------------------------------------------------
MIPS_top             48300    1175370         0     1175370  <none> (D)
  DM                 28722     797511         0      797511  <none> (D)
  IM                 14395     240392         0      240392  <none> (D)
  piperegs            2619      64482         0       64482  <none> (D)
  theALU              1420      38659         0       38659  <none> (D)
    mul_23_17          906      26887         0       26887  <none> (D)
    sub_32_17           65       2372         0        2372  <none> (D)
    add_20_17           33       2182         0        2182  <none> (D)
    gte_35_9            71       1051         0        1051  <none> (D)
    lt_36_9             71       1048         0        1048  <none> (D)
  IFIDreg              192       5126         0        5126  <none> (D)
  IDEXreg               96       5109         0        5109  <none> (D)
  EXMEMreg             104       4577         0        4577  <none> (D)
  MEMWBreg              71       3779         0        3779  <none> (D)
  add_71_45             31       2042         0        2042  <none> (D)
  inc_add_43_22_1       60       1121         0        1121  <none> (D)
  MUX1                  67       1101         0        1101  <none> (D)
  MUX0                  67       1101         0        1101  <none> (D)
  add_50_29             56       1048         0        1048  <none> (D)
  FU                    39        685         0         685  <none> (D)
  ALUcontrol            28        469         0         469  <none> (D)
  HU                    22        369         0         369  <none> (D)
  thecontrol            24        296         0         296  <none> (D)

 (D) = wireload is default in technology library
```

Figure 14. Area Report Generated by Cadence Tool

Netlist of design is generated in pre-layout steps. This synthesized Verilog netlist and respective design constraints file (.sdc file) are imported to Cadence SoC Encounter and are used to generate automated layout from standard cells, placement and routing. After performing nano route step, final layout of design is shown in figure 15.

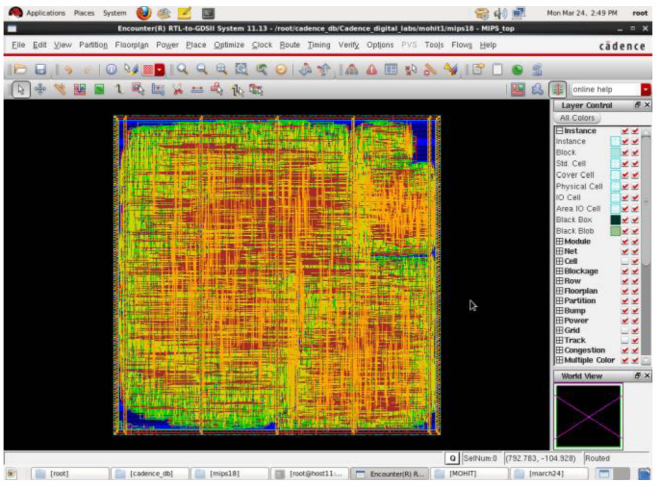

Encounter(R) RTL-to-GDSII System 11.13 - /root/cadence_db/Cadence_digital_labs/mohit1/mips18 - MIPS_top
File Edit View Partition Floorplan Power Place Optimize Clock Route Timing Verify Options PVS Tools Flows Help
cādence
Layer Control
World View

Figure. 15 Layout of MIPS Design using Cadence SoC Encounter

VI. Conclusion

In this paper, design of 32-bit MIPS based RISC processor is implemented successfully with pipeline functionalities. Every instruction is executed in one clock cycle with 5-stage pipelining. This design shows the implementation of MIPS based CPU capable of handling various R -type, J-type and I-type of instruction and each of these categories has a different format. These instructions are verified successfully through testbench. Designing Forwarding unit and hazard detection unit to overcome the data dependencies was critical task and it was implemented successfully. The design is implemented using Verilog-HDL and synthesized using Cadence RTL complier using typical libraries of TSMC 0.18 um technology. Design of MIPS processor is optimized both in timing and area. Also complete ASIC flow till RTL to GDS II have done using Cadence SoC Encounter, and analyzed the complete physical design flow.

Acknowledgement

The authors would like to acknowledge the facilities provided towards Cadence Encounter 11.13 by the Research Lab, Department of Electronics and Communication Engineering, SRM University, Chennai.

References

[1] Preetam Bhosle, Hari Krishna Moorthy, “FPGA Implementation of low power pipelined 32-bit RISC Processor”, International Journal of Innovative Technology and Exploring Engineering (IJITEE), August 2012.

[2] Gautham P, Parthasarathy R, Karthi, Balasubramanian, “Low Power Pipelined MIPS Processor Design,” in the proceedings of the 2009, 12th international symposium,2009 pp. 462-465.

[3] Neenu Joseph, Sabarinath.S, “FPGA based Implementation of High Performance Architectural level Low Power 32-bit RISC Core”, 2009 IEEE.

[4] “Computer Organization and Design- the hardware/software interface”, 3rd edition by David A. Patterson and John L. Hennessy, pp. 370-412.

[5] Mrs. Rupali Balpande, Mrs. Rashmi Keote, “Design of FPGA based Instruction Fetch & Decode Module of 32-bit RISC (MIPS) Processor”, 2011 IEEE.

[6] Harpreet Kaur, Nitika Gulati, “Pipelined MIPS With Improved Datapath”, IJERA, Vol. 3, Issue 1, January -February 2013, pp.762-765.

[7] Sharda P. Katke, G.P. Jain,“Design and Implementation of 5 Stages Pipelined Architecture in 32 Bit RISC Processor”, IJETAE, Volume 2, Issue 4, April 2012, pp. 340-346.

[8] Pejman Lotfi-Kamran, Ali-Asghar Salehpour, Amir-Mohammad Rahmani, Ali Afzali-Kusha, and Zainalabedin Navabi, “Dynamic Power Reduction of Stalls in Pipelined Architecture Processors”, International Journal Of Design, Analysis And Tools For Circuits And Systems, Vol. 1, No. 1, June 2011.

[9] Neeraj Jain, “VLSI Design and Optimized Implementation of a MIPS RISC Processor using XILINX Tool”, International Journal of Advanced Research in Computer Science and Electronics Engineering (IJARCSEE) Volume 1, Issue 10, December 2012.