EE 714 Digital IC Design Term Project

2020, Fall

Supervisor: Prof. Kim, Jinsang
ID: 2015104027 Name: Park, Jung Jin
ID: Name:

Design Proposal: Comparison between MIPS and RISC-V

I. Introduction:

The era of rapid transistor performance improvement by aggressive scaling down allows unprecedented development of computers on a single chip. For the sake of improving power efficiency, RISC, *Reduced Instruction Set Computer*, which was coined by David A. Patterson, UC Berkeley, came out.[1] Unlike CISC, *Complex Instruction Set Computer*, a conventional design in the early 1960s and onwards, RISC places emphasis on simplicity and efficiency. The below lists are distinct features of RISC.[2]

RISC Features:
1. Large register file
   -Variables and intermediate results can be stored in registers and do not require repeated loads and stores from/to memory
   - All local variables of procedures and the passed parameters can be stored in registers.
2. Emphasis on register-oriented operations (Register-to-register operations)
3. Instructions that primarily execute in a single cycle
4. Simple LOAD/STORE instructions for memory access
5. Limited addressing modes
   - Most RISC instructions use simple register addressing. Complex modes can be optimized and simplified in the compiler.
6. Simple and fixed-length instructions
   - Decoding is simplified since the opcode and address fields are located in the same index for all instructions. Hence, simple and small decode and execution hardware are required, and it takes up less area, which means runs faster.
7. Pipelined instruction cycle (typically uniform delay pipelines)

Under certain circumstances, such as mobile systems and portable devices, RISC has significant advantages over CISC. The below list is the potential advantages of RISC. [2]

RISC Potential advantages:
1. Fast instruction execution
2. Simple control unit
3. Fast decode
4. Faster processor design, development, and test (Reducing the time between designing and marketing)
5. Improved optimizing compiler support

Due to these advantages, RISC designs become the main trend in computer design. After the evolution of generation, which was RISC-I, II, III, IV in order, the RISC-V was presented in 2010. RISC-V is different from other CPUs in terms of proprietary. RISC-V is open-source and design without royalties. Therefore, the RTL code implementation of RISC-V is more comfortable than other architectures.

In the project, first, we will design MIPS, *Microprocessor without Interlocked Pipelined Stages*. MIPS is also RISC and is useful for academic purposes. The CSE203, Computer Architecture lecture at Kyung Hee University, also teaches MIPS and recommends implementing RTL code for a deep understanding of computer architecture. For MIPS design, we will follow steps described in a book, *Computer Organization and Design*, by David A. Patterson and John L. Hennessy. After accumulating fundamental knowledge about computer architecture, we will design the RISC-V CPU from scratch. Based on MIPS knowledge and comparison during implementation, we will try to find the main difference between MIPS and RISC-V and why RISC-V has become the main trend in computer architecture.

II. MIPS vs RISC-V

In this report, we will skip a MIPS description. It is content in the CSE203, Computer Architecture lecture at Kyung Hee University. Immediately, we will look at the differences between MIPS and RISC-V.

All RISC processors(MIPS, RISC-V, ARM...) are effectively the same except for one or two design decisions that "seemed like a good idea at the time." The concept of 'seems like a good idea' in MIPS is the branch delay slot. The branch delay slot is generated by the compiler. By the branch delay slot, MIPS always execute the instruction after a branch or jump whether or not the branch or jump is taken. It is a crucial point that decreases CPU performance. If the compiler gets a branch or jump instructions, the next and/or more instructions will be bubbles till a branch or jump instruction's target address is computed. Also, the branch delay slot complicates multicycle CPUs, superscalar CPUs, and long pipelines. On the contrary, RISC-V does not include a branch delay slot, a position after a branch instruction that can be filled with an instruction that is executed whether or not the branch is taken. Besides, dynamic branch predictors have succeeded well enough to reduce the need for delayed branches. On the first encounter with a branch, RISC-V CPU assumes that a negative relative branch (i.e. the sign bit of the offset is "1") will be taken. This assumes that a backward branch is a loop, and provides a default direction so that a simple pipelined CPU can fill their pipeline of instructions. Other than this, RISC-V does not require branch prediction, but core implementations are allowed to add it. RV32I, RISC-V 32 bits Integer, reserves a "HINT" instruction space that presently does not contain any hints on branches. The differences are not only branch or jump but also instructions set. First, the two architectures have different register naming and calling conventions. For example, register name $4 in MIPS is x4 in RISC-V. Second, there is a difference in instruction encodings and encoding formats. MIPS has three encodings whereas RISC-V has 6 encodings. NOOP instruction in MIPS is all-0 bits but all-0 bits mean "invalid" in RISC-V. All RISC-V immediate bits are addressed sign-extended number so that RISC-V does not support trap on overflow signed math. Third, instruction alignment is different. The RISC-V only requires 2-byte alignment for instructions when including an optional 16b instruction encoding. The RISC-V also supports some 48 bits, 64 bits, and more bits instructions in extensions. Fig 2.1 shows extend encoding in RISC-V. That is to say, it has extra address space for extensions. Lastly, RISC-V has a dedicated "compare 2 registers & branch" operation.

Fig 2.1 RISC-V instruction length encoding [3]

We looked at some differences between MIPS and RISC-V. Then, we will analyze the reason why RISC-V become popular in computer architecture. The simple reason is that RISC-V is designed to be simple and open so that it is intended for education and research, although there is commercial interest as well. The popular commercial ISAs, such as x86 and ARM architecture, are both very complex to implement in hardware to the level of supporting common operating systems. All complexity of these architectures does not truly improve efficiency, rather are bad or outdated. The main point of the RISC-V's popularity is extensibility or flexibility. The above complex architectures were not particularly designed for extensibility, and as a consequence have added considerable instruction encoding complexity as their instruction sets have grown. However, the RISC-V has flexibility so that it can include custom hardware simply by defining custom instruction in the ISA spec. For example, in this dominant machine learning era, accelerators are essential. If accelerators are operated by defined instructions architecture, the RISC-V can add accelerators simply whereas the popular CPUs need to add a considerable complicate encoder, controller, and so on. This is the main strength of the RISC-V. Beyond machine learning, the RISC-V can readily use for cryptology or something that requires high computational cost.

III. RISC-V Overview

In this project, we implemented the base RISC-V, specifically, RV32IM.

The implemented RISC-V only supports 4-bytes signed and unsigned integer values. There are no byte nor half-word values and no floating-point. The number of general-purpose registers, which hold integer values, is 31. Register x0 is hardwired to the constant zero. Each register is 32 bits wide. The below list shows the registers' names, addresses, and purposes.

+ x0 : zero the constant value 0
+ x1 : ra return address (caller saved)
+ x2 : sp stack pointer (called saved)
+ x3 : gp global pointer
+ x4 : tp thread pointer
+ x5 : t0 temporary registers (caller saved)
+ x6 : t1 "
+ x7 : t2 "
+ x8 : s0/fp saved register or frame pointer (callee saved)
+ x9 : s1 saved register (callee saved)
+ x10 : a0 function arguments and/or return values (caller saved)
+ x11 : a1 "
+ x12 : a2 function arguments (caller saved)
+ x13 : a3 "
+ x14 : a4 "
+ x15 : a5 "
+ x16 : a6 "
+ x17 : a7 "
+ x18 : s2 saved registers (callee saved)
+ x19 : s3 "
+ x20 : s4 "
+ x21 : s5 "
+ x22 : s6 "
+ x23 : s7 "
+ x24 : s8 "
+ x25 : s9 "
+ x26 : s10 "
+ x27 : s11 "
+ x28 : t3 temporary registers (caller saved)
+ x29 : t4 "
+ x30 : t5 "
+ x31 : t6 "

The memory replaced to registers since we do not know how to simulate the memory operation in the Samsung library yet.

The below figure, Fig 3.1 shows RISC-V Instruction set. RISC-V has an asymmetric immediate encoding which means that the immediates are formed by concatenating different bits in an asymmetric order based on the specific immediate formats. Again, all RISC-V immediate bits are addressed sign-extended number, and the sign-bit for the immediate is always in MSB.

RISC-V instructions have 6 types. First, R-type is an instruction using three registers inputs. The R-type is used for arithmetic and logical operation. (i.e. add, xor, mul) I-type is an instruction for arithmetic and logical operation with immediates as well as load. (i.e. addi, slli, lw, jalr) S-type is an instruction for storing data. (i.e. sw, sb) SB-type is branch instructions. (i.e. beq, bne, bge) U-type is an instruction with upper immediate. (i.e. lui, auipc) UJ-type is jump instructions. (i.e. jal, j)

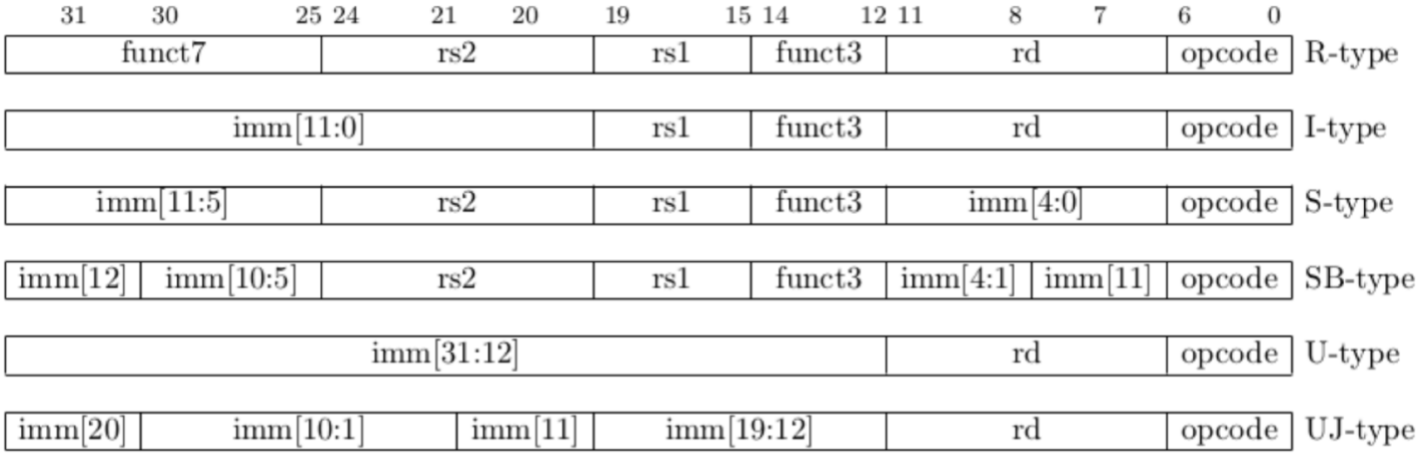
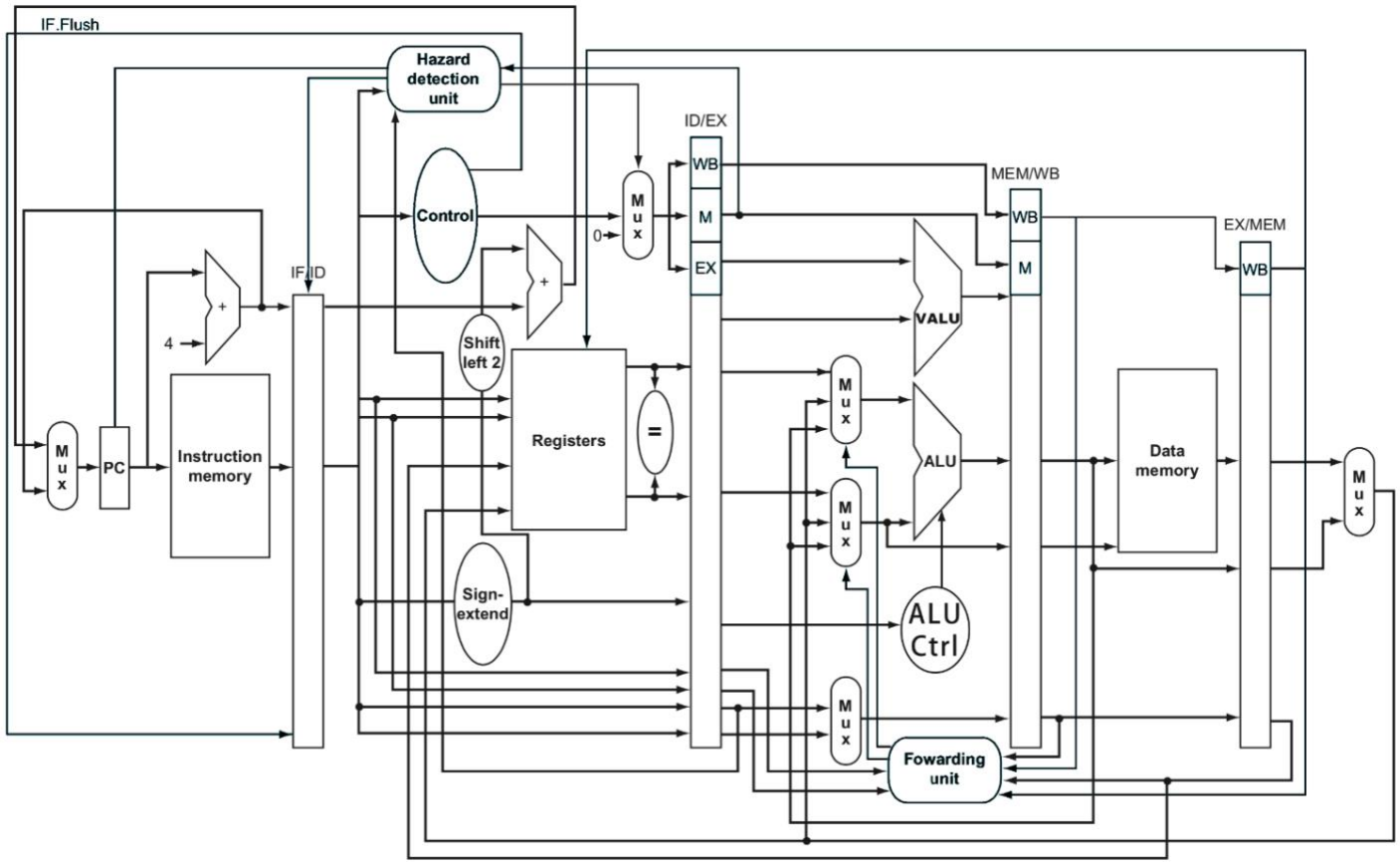


Fig 3.1 RISC-V instructions type

The Fig 3.2 is block diagram of RISC-V.



The VALU is a vector arithmetic logic unit for vector(parallel) computation. It is a concept of an accelerator. Except for VALU and instructions set, there are no differences with MIPS in terms of architecture.

IV. Logic Synthesis

With the RTL source code of the RISC-V, we compile the code using Design Compiler, Synopsys. The results are below pictures.

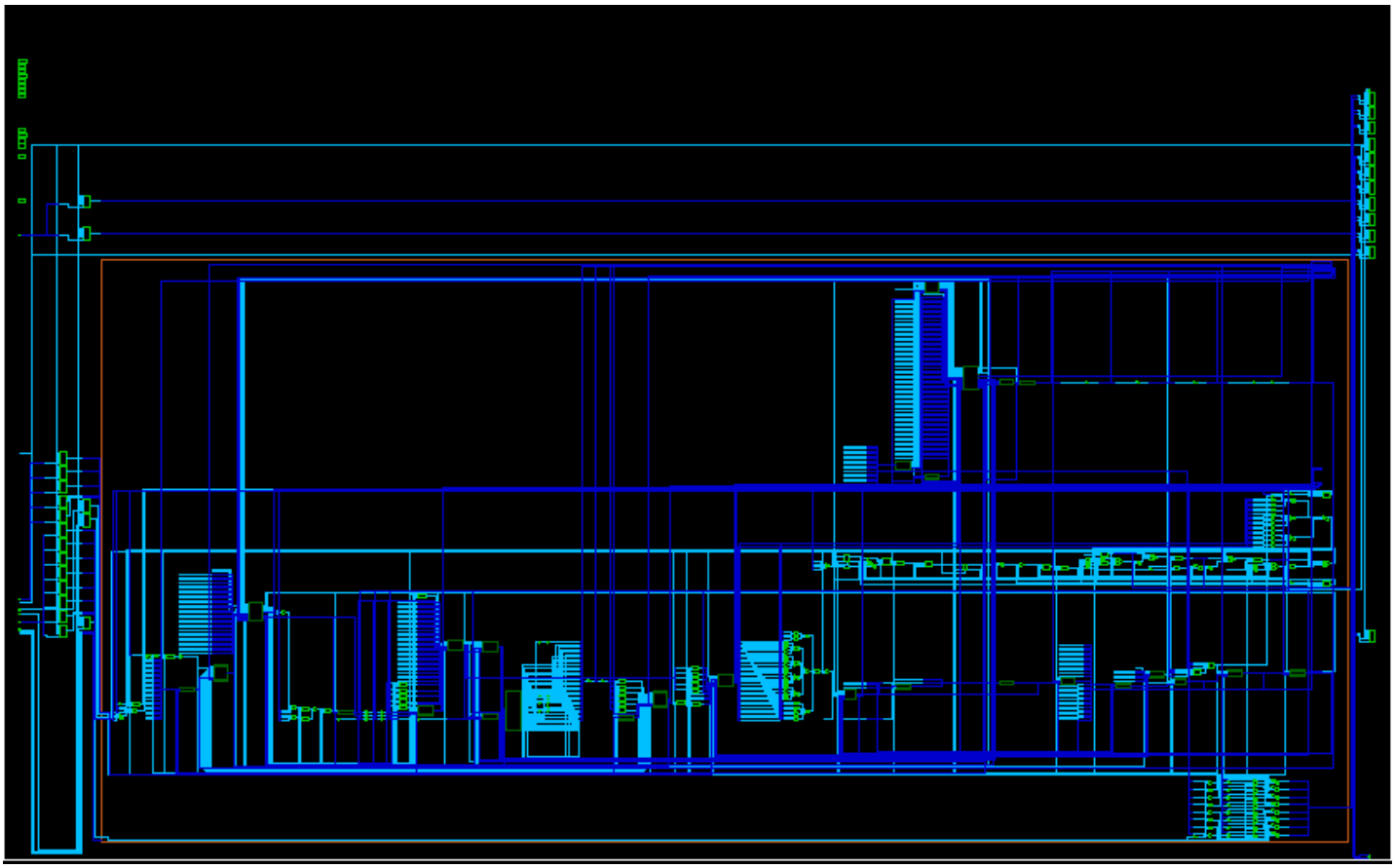


Fig 4.1 RISC-V Schematic

The square blocks at both ends of Fig 4.1 are I/O PAD. The compiling strategies are clock gating and Zero Wire Load Model(Zero-WLM). The reason why we adopted the strategies is the following.

First, clock gating is for a reduction in dynamic power. First, clock gating is for reduction in dynamic power. As Fig 4.2, the power report indicates that the (net) switching power takes up a great proportion of total power. The dynamic power is defined as the equation: $P = \alpha C_L V_{DD}^2 f_{sw}$. In the components of the dynamic power, as the scaling-down, the load capacitance C_L is getting lower, and the supply voltage V_{DD} remains the same or be a little smaller. However, the switching frequency f_{sw} is going to higher significantly. Hence, as scaling down the channel length, the dynamic power usually increases. To reduce the dynamic power, the design should reduce the activity factor α by clock gating. The clock has a high activity factor, $\alpha = 1$, so it considerably affects the dynamic power of the design. To reduce activity factor, clock gating is adopted. Clock gating cell includes enable signal(EN) and clock(CLK) so that only when enable signal is on, the enable clock(ENCLK) is switched. The clock gating cell is latch-based due to prevention of glitch. Fig 4.3 shows a latch-based clock gating cell. The important point when using clock gating is constraints. For P&R, to reduce clock skew as much as possible between ENCLK to clock-gated registers, we set max delay to 0. Also, timing analysis of clock gating is essential because if EN signal setup margin is too tight, the timing optimization is difficult or failed during P&R. Thus, we did the timing analysis of paths which is connected to EN.

Second, because we consider physical synthesis, the Zero-WLM is adopted. Wire load models are very subjective to variations due to P&R, design style, tools, number of metal layers, and other design specific parameters. The statistical WLM is practically useless with modern processes. The Zero-WLM keep synthesis from wasting time on pointless buffering and sizing. Using the Zero-WLM with tightened timing constraints encourages Design Compiler to produce a design that is more suitable for physical synthesis. The one thing to consider when using Zero-WLM is that we should set more timing margin considering wire delay during P&R. Thus, We set 40% of the clock cycle margin in the Front-End stage.

The cell count is 13k, which is over 8k. Table 4.1 is a specification of our RISC-V. We set the clock speed 10ns, and the defined clock speed in the Design compiler is 6ns due to Zero-WLM as mentioned above. Fig 4.4, Fig 4.5, Fig 4.6 are area and timing reports of RISC-V.

```
Operating Conditions: ss_1p08v_125c   Library: scmetropmk_cmos10lp_rvt_ss_1p08v_125c_sadhm
Wire Load Model Mode: top


Global Operating Voltage = 1.08
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000pf
    Time Units = 1ns
    Dynamic Power Units = 1mW    (derived from V,C,T units)
    Leakage Power Units = 1mW


  Cell Internal Power  =   8.5063 mW    (4%)
  Net Switching Power  = 194.8737 mW   (96%)
                         ---------
Total Dynamic Power    = 203.3799 mW  (100%)

Cell Leakage Power     = 170.1900 uW

                 Internal         Switching           Leakage            Total                          Cell
Power Group      Power            Power               Power              Power   (   %    )  Attrs   Count
--------------------------------------------------------------------------------------------------------------
io_pad             6.7276          194.7851        7.8151e-03          201.5205  (  99.00%)              44
memory             0.0000            0.0000            0.0000            0.0000  (   0.00%)              0
black_box          0.0000            0.0000            0.0000            0.0000  (   0.00%)              0
clock_network      1.0581        1.7892e-02        8.5820e-03            1.0846  (   0.53%)           1315
register           0.7126        2.5929e-04        4.8452e-02            0.7613  (   0.37%)          11745
sequential         0.0000            0.0000            0.0000            0.0000  (   0.00%)              0
combinational  8.0119e-03        7.0429e-02            0.1053            0.1838  (   0.09%)         108827
--------------------------------------------------------------------------------------------------------------
Total              8.5063 mW       194.8737 mW         0.1702 mW       203.5501 mW
1
```

Fig 4.2 The power report

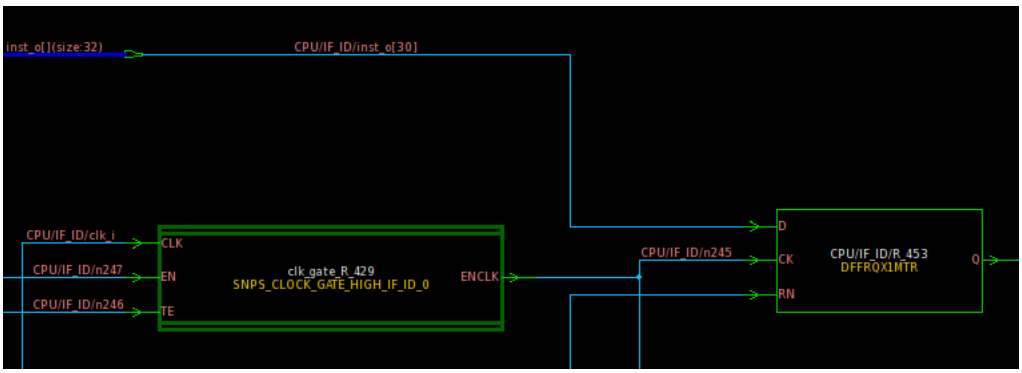


Fig 4.3 The clock gating cell

```
****************************************
Report : area
Design : CPU_PAD
Version: L-2016.03-SP5-5
Date   : Fri Dec 11 03:29:21 2020
****************************************

Library(s) Used:

    scmetro_cmos10lp_rvt_ss_1p08v_125c_sadhm (File: /Tools/Libra
ec100226_0042_SS65LP_Normal_RVT_Normal_FE_Common_N/synopsys/scme
db)
    ss65lp3p3v_wst_108_300_p125 (File: /Tools/Library/samsung65_
st_108_300_p125.db)

Number of ports:                         8958
Number of nets:                        140509
Number of cells:                       133009
Number of combinational cells:         108831
Number of sequential cells:             22781
Number of macros/black boxes:              41
Number of buf/inv:                       4962
Number of references:                       9

Combinational area:             218729.280058
Buf/Inv area:                     5032.959894
Noncombinational area:          111345.603824
Macro/Black Box area:                0.000000
Net Interconnect area:       undefined  (No wire load specified)

Total cell area:                330074.883882
Total area:                  undefined
1
```

Fig 4.4 The area report

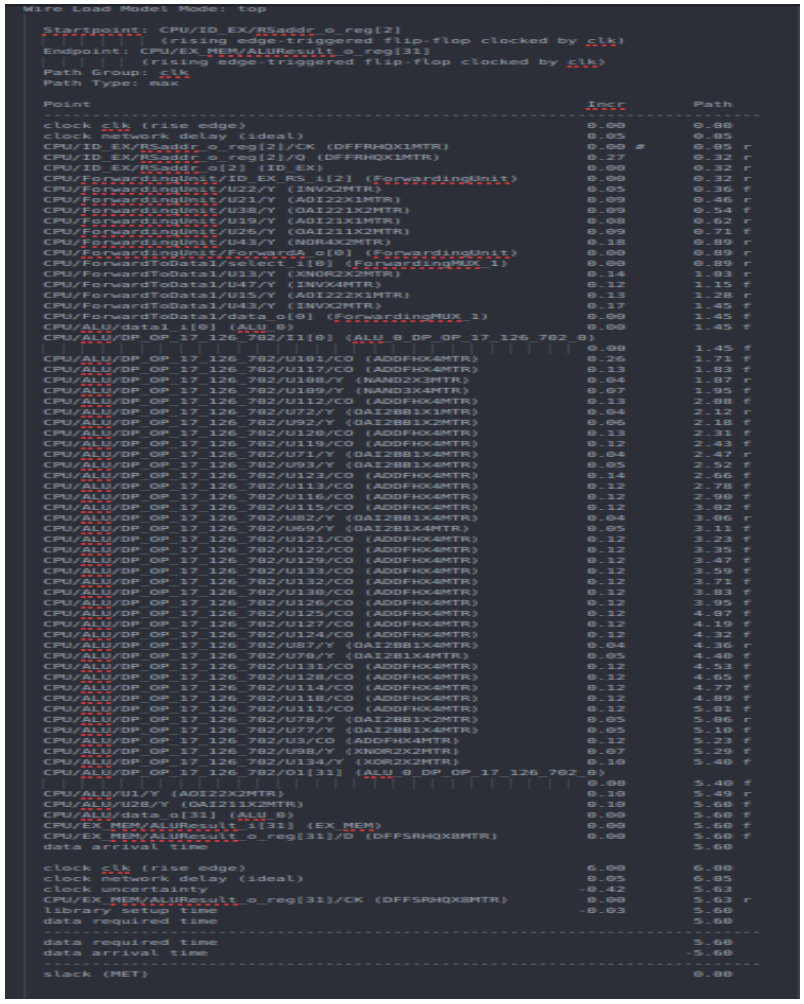


Fig 4.5 The max delay report

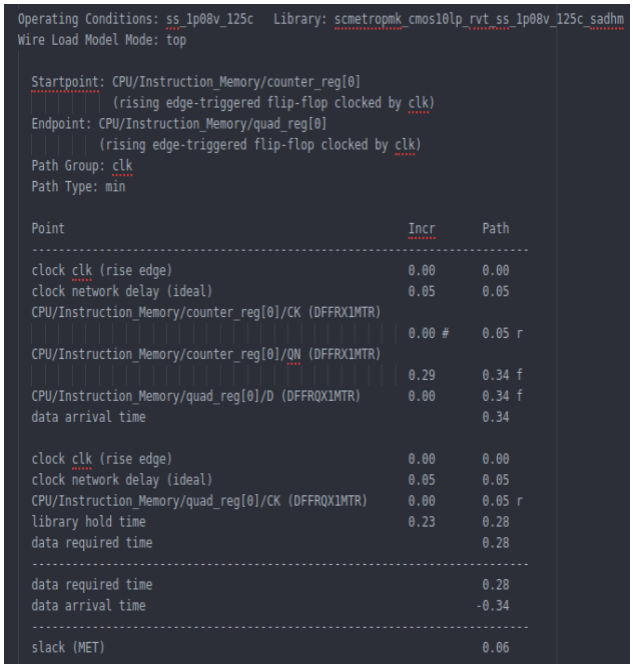
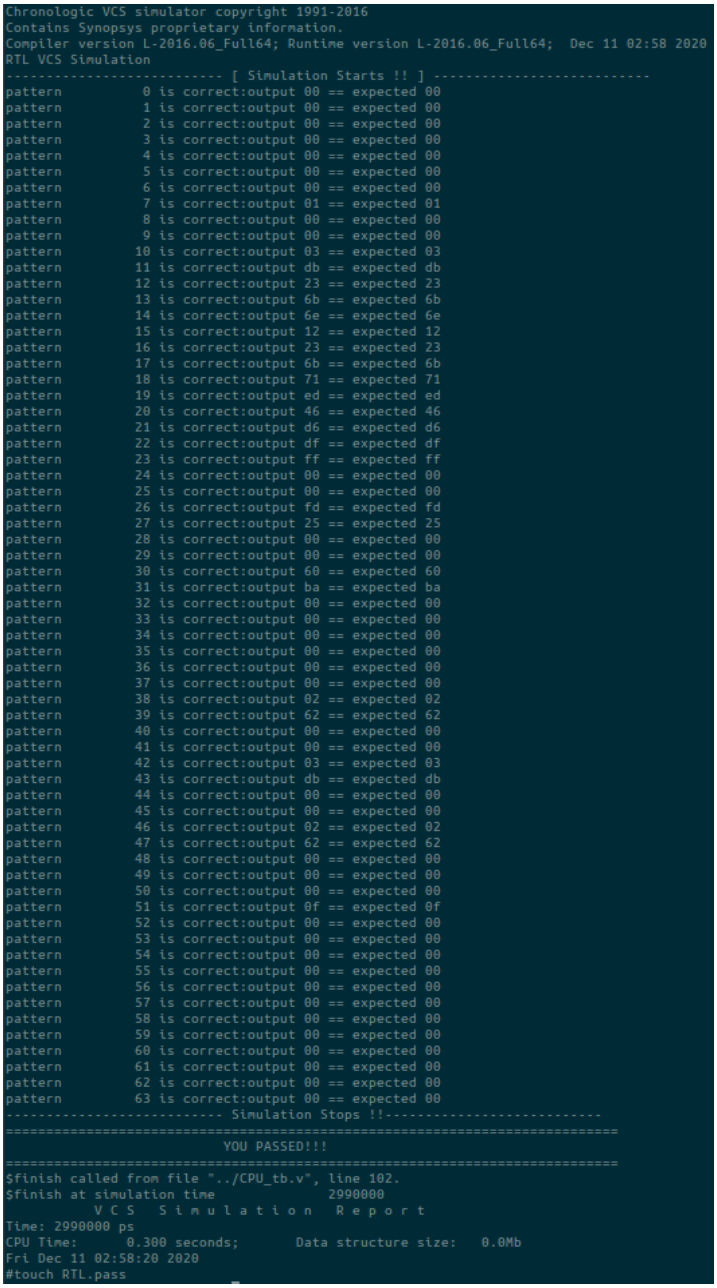


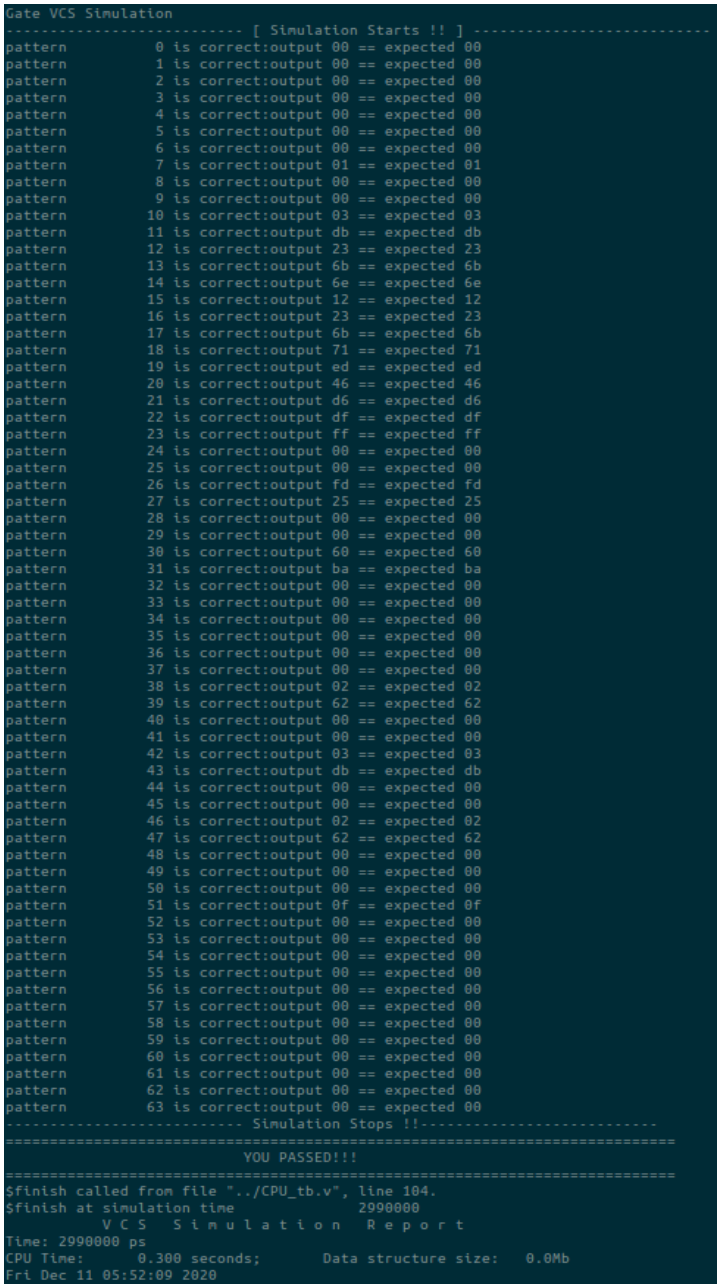
Fig 4.6 The min delay report

Table 4.1 The specification of RISC-V

Samsung 65nm	
Spec	Value
Main clock	100 MHz
Area	330074.88 μm^2
Power	203.55 mW

An RTL model and a gate-level netlist of RISC-V are verified by VCS. We implemented a testbench that checks register and data memory data when finishing the process. We called the correct data "golden data.". If stored data by VCS simulation is different from the golden data, VCS shows the error information. On the contrary, if stored data is the same as the golden data, VCS shows the sentence "You are passed!" It is the same as gate-level netlist simulation. However, one different thing is the sdf(synopsys delay format) file is annotated when simulating gate-level netlist. The below figures show the simulation results of our RISC-V.





Reference:

[1] Reilly, Edwin D., *Milestone in computer science and information technology*, Westport, CT, 2003
[2] A. D. George, "An overview of RISC vs. CISC," *Proceedings. The Twenty-Second Southeastern Symposium on System Theory*, Cookeville, TN, USA, 1990, pp. 436-438, doi: 10.1109/SSST.1990.138185.
[3] A. Waterman, K. Asanović, "The RISC-V Instruction Set Manual Volume I: Unprivileged ISA," CS Division, EECS, Department, University of California, Berkeley, Dec 13, 2019.