EE3043 Computer Architecture

Semester 242

Design of a Single Cycle RISC-V Processor

Student: Hoàng Sỹ Nhất – 2111915

1. Introduction

Trong bối cảnh kiến trúc mã nguồn mở ngày càng định hình tương lai của điện toán hiện đại, RISC-V nổi bật nhờ tính linh hoạt, modular và khả năng tùy biến không giới hạn. Dự án này tập trung vào việc thiết kế và triển khai một lõi xử lý 32-bit dựa trên tập lệnh cơ bản RV32I của RISC-V bằng ngôn ngữ Verilog, nhằm tạo nền tảng cho các hệ thống nhúng, IoT, hoặc nghiên cứu chuyên sâu về kiến trúc máy tính.

Bằng cách mô phỏng các module chính như ALU, bộ đệm thanh ghi, khối điều khiển và giao tiếp bộ nhớ, dự án không chỉ minh họa nguyên lý hoạt động của CPU RISC-V pipeline mà còn góp phần vào cộng đồng phần cứng mở. Thiết kế pipeline giúp tối ưu hóa hiệu suất xử lý bằng cách cho phép nhiều lệnh được thực hiện song song, từ đó nâng cao khả năng xử lý và giảm thời gian thực hiện.

Dự án cũng bao gồm các bước kiểm thử nghiêm ngặt thông qua testbench và công cụ mô phỏng, nhằm đảm bảo rằng thiết kế đạt được độ tin cậy cao. Với sự cân bằng giữa tính giáo dục và ứng dụng thực tiễn, dự án không chỉ phục vụ cho việc nghiên cứu mà còn mở ra cơ hội cho việc phát triển các tập lệnh tùy chỉnh và tối ưu hóa hiệu năng trong tương lai.

2. Design Strategy

2.1. Overview

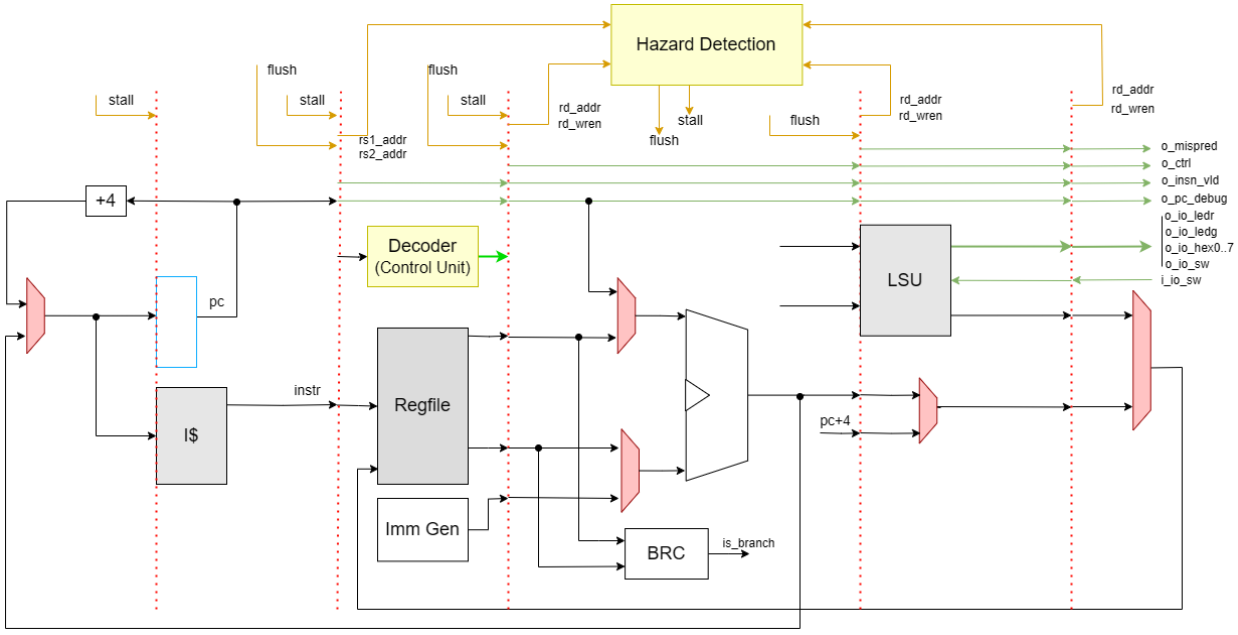
Ở Milestone 3 này ta sẽ tập chung nâng cấp thiết kế RiscV singlecycle đã thiết kế ở Milestone 2 thành pipeline để tối ưu hóa thời gian xử lí một câu lệnh. Ta sẽ tiến hành thiết kế 5 kiến trúc pipeline là non-forwarding, forwarding, always taken, 2-bit shceme prediction và 2-bit prediction Gshare để so sánh hiệu suất giữa các model.

2.1.1. Specification

Signal	**Width**	**Direction**	**Description**
i_clk	1	input	Global clock, active on the rising edge.
i_reset	1	input	Global low active reset.
o_pc_debug	32	output	Debug program counter.
o_insn_vld	1	output	Instruction valid.
o_ctrl	1	output	Control transfer instructions (branch or jump).
o_mispred	1	output	Mispredict.
o_io_ledr	32	output	Output for driving red LEDs.
o_io_ledg	32	output	Output for driving green LEDs.
o_io_hex0..7	7	output	Output for driving 7-segment LED displays.
o_io_lcd	32	output	Output for driving the LCD register.
i_io_sw	32	input	Input for switches.

2.2. Non-forwarding

Phân tách RiscV single cycle thành 5 tầng là IF, ID, EX, MEM và WB mỗi tầng bao gồm các khối chức năng như hình dưới. Ở giữa mỗi tầng là các thanh ghi cập nhật giá trị mới mỗi khi có xung clock cạnh lên và tín hiệu điều khiển *stall* LOW, reset giá trị đồng bộ clock khi *flush* HIGH hoặc bất đồng bộ khi có cạnh lên *reset.*

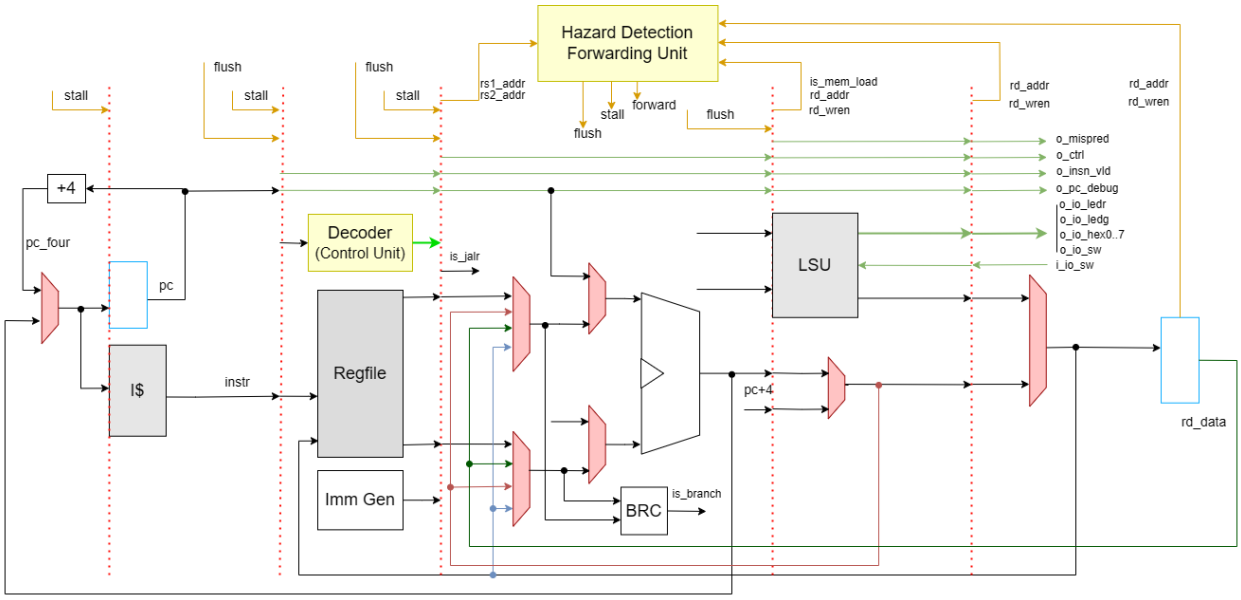


LSU thay đổi từ ghi đồng bộ thành đọc ghi đồng bộ clock. Ở tầng EX bộ BRC sẽ so sánh 2 ngõ vào và trả kết quả quyết định rẽ nhánh dựa theo opcode func3 của instruction và câu lệnh tại tầng EX có đang là câu lệnh rẽ nhánh hay không.

Harzard detection thiết kế để phát hiện các trường hợp Data harzard và Control harzard. Data harzard xảy ra khi thanh ghi đích các tầng EX, MEM và WB khác zero và bằng ít nhất 1 thanh ghi nguồn đồng thời cờ bật cho phép ghi Regfile tại 3 tầng này bằng 1, lúc này các thanh ghi PC, IF/ID sẽ stall, thanh ghi ID/EX sẽ flush đến khi khồn còn data harzard. Control harzard xảy ra khi tại tầng EX có thực hiện câu lệnh nhảy hoặc rẽ nhánh, lúc này sẽ flush thanh ghi IF/ID và ID/EX.

2.3. Forwarding

Với model forwarding sẽ giải quyết phần lớn tính trạng data harzard chỉ cần thêm 2 bộ mux và bộ forwarding unit.



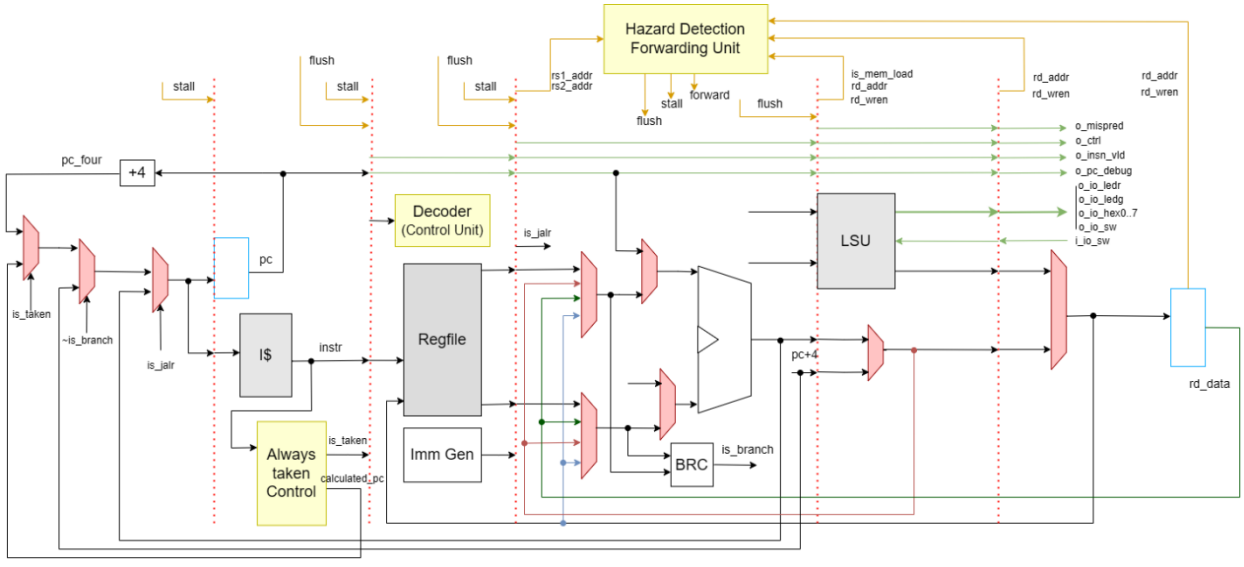
Forwarding unit sẽ kiểm tra nếu 2 thanh ghi nguồn ở tầng EX, nếu 2 thanh ghi này trùng với thanh ghi đích tại tầng MEM, tầng WB và thanh ghi đệm sau tầng WB thì bộ forwarding sẽ đưa giá trị sẽ ghi vào thanh ghi đích trong các chu kì tiếp theo vào thẳng ALU mà không cần chờ đến khi thanh đích ghi dữ liệu. Sở dĩ phải có thêm một thanh ghi đệm sau tầng WB vì Regfile ghi giá trị mới vào thanh ghi tại cạnh lên xung clock trong lúc đó dữ liệu đó chưa kịp cập nhật đã qua tầng EX nên ta cần thiết kế thêm thanh ghi đệm để giải quyết trường hợp này.

Harzard detection lúc này chỉ cần xử lí các trường hợp control harzard như model non-forwarding và trường hợp load use harzard nếu dữ liệu từ câu lệnh load được sử dụng ở ngay câu lệnh phía sau. Vì khi thực hiện câu lệnh load, dữ liệu load phải đợi 1 xung clock mới xuất hiện tại tầng WB vì thế cần phải stall các thanh ghi trước tầng MEM và flush thanh

ghi tầng EX/MEM, khi đã có dữ liệu load ở tầng WB bộ forwarding sẽ chuyển tiếp dữ liệu đó đến tầng EX. Tầng nào càng gần tầng EX sẽ được ưu tiên forward dữ liệu đến tầng EX để đảm bảo dữ liệu đưa vô ALU là chính xác.

2.4. Forwarding with always taken

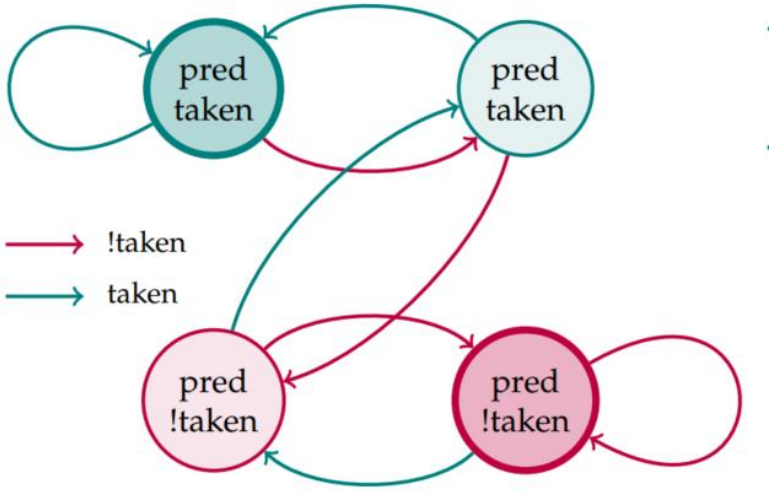
Ở model này thay vì thiết kế Branch Target Buffer (BTB) ta sẽ thiết kế bộ tính toán PC mà câu lệnh JAL hoặc câu lệnh Branch sẽ nhảy tới vì 2 câu lệnh này PC nhảy tới sẽ là giá trị PC hiện tại cộng với giá trị tức thời có thể tính được từ instruction, bỏ qua câu lệnh JALR vì có thể sẽ xảy ra data harzard. Vì thế ngay tại tầng Fetch, Instruction cash sẽ đọc không đồng bộ, instruction sẽ cập nhật tức thì ngay tầng IF và truyền tới bộ Always Taken Control. Bộ này sẽ kiểm tra câu lệnh hiện tại nếu là câu lệnh JAL hoặc câu lệnh Branch sẽ tính toán PC mà câu lệnh nhảy tới, câu lệnh tiếp theo sẽ nhảy tới PC vừa tính được. Model này phù hợp với với các chương trình ít sử dụng câu lệnh JALR hoặc vòng lặp hoặc với các chương trình sử dụng nhiều câu lệnh rẽ nhánh hoặc If/else.

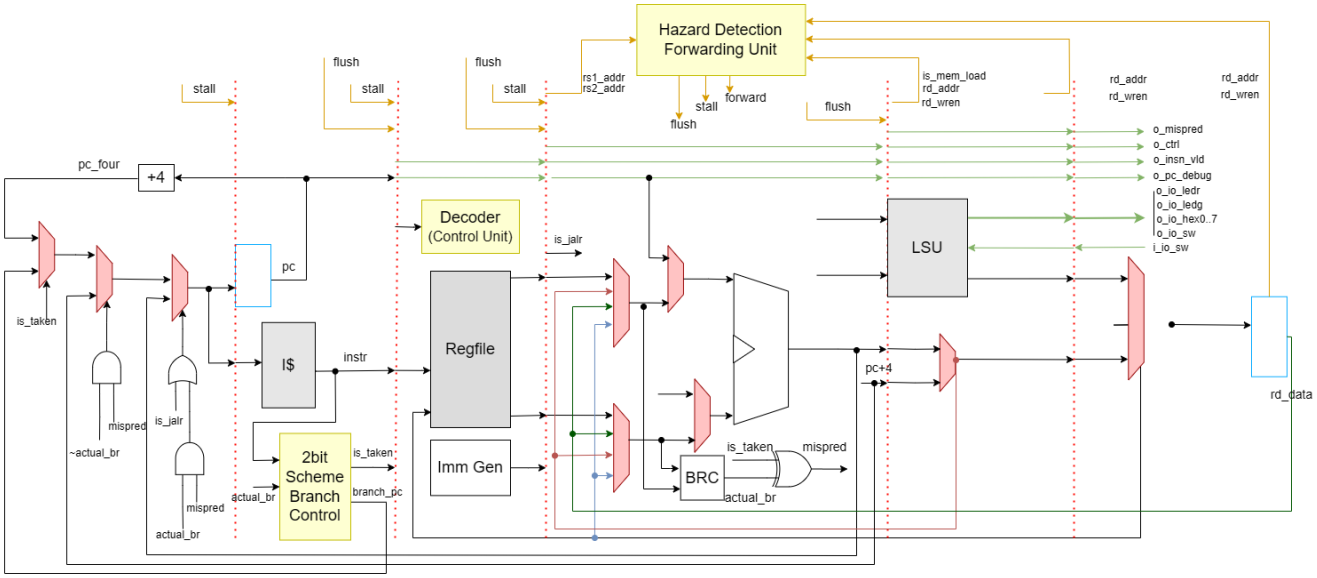


Nếu ở tầng EX câu lệnh rẽ nhánh là không nhảy, PC tiếp theo sẽ là giá trị PC tại câu lệnh EX +4. Nếu câu lệnh JALR ở tầng EX sẽ tiến hành nhảy như bình thường.

2.5. 2-bit Scheme prediction

Ở model này kiến trúc tương tự model trên, điểm khác biệt là thay vì lúc nào cũng nhảy tới PC đã tính toán thì quyết định nhảy hay không nhảy sẽ phụ thuộc vào máy trạng thái 2 bit.





Máy trạng thái sẽ đổi trạng thái phụ thuộc vào quyết định rẻ nhánh ở câu lệnh rẽ nhánh ở tầng EX. Nếu dự đoán sai sẽ quay lại PC cũ hoặc nhảy tới PC mới tùy thuộc vào kết quả quyết định rẽ nhánh tại tầng EX.

2.6. Gshare 2-bit Scheme prediction

Model này kiến trúc giống với model 2-bit scheme prediction tuy nhiên thay vì có sử dụng 1 máy trạng thái để dự đoán câu lệnh branch thì ta sử dụng N máy trạng thái với index phụ thuộc vào giá trị PC. Tại tầng IF sử dụng PC làm index cho mảng chứa giá trị trạng thái của 2-bit scheme và đưa ra quyết định nhảy/không nhảy. Tại tầng EX, sử dụng giá trị PC tại đó làm index và kết quả từ bộ BRC để cập nhật trạng thái của máy trạng thái.

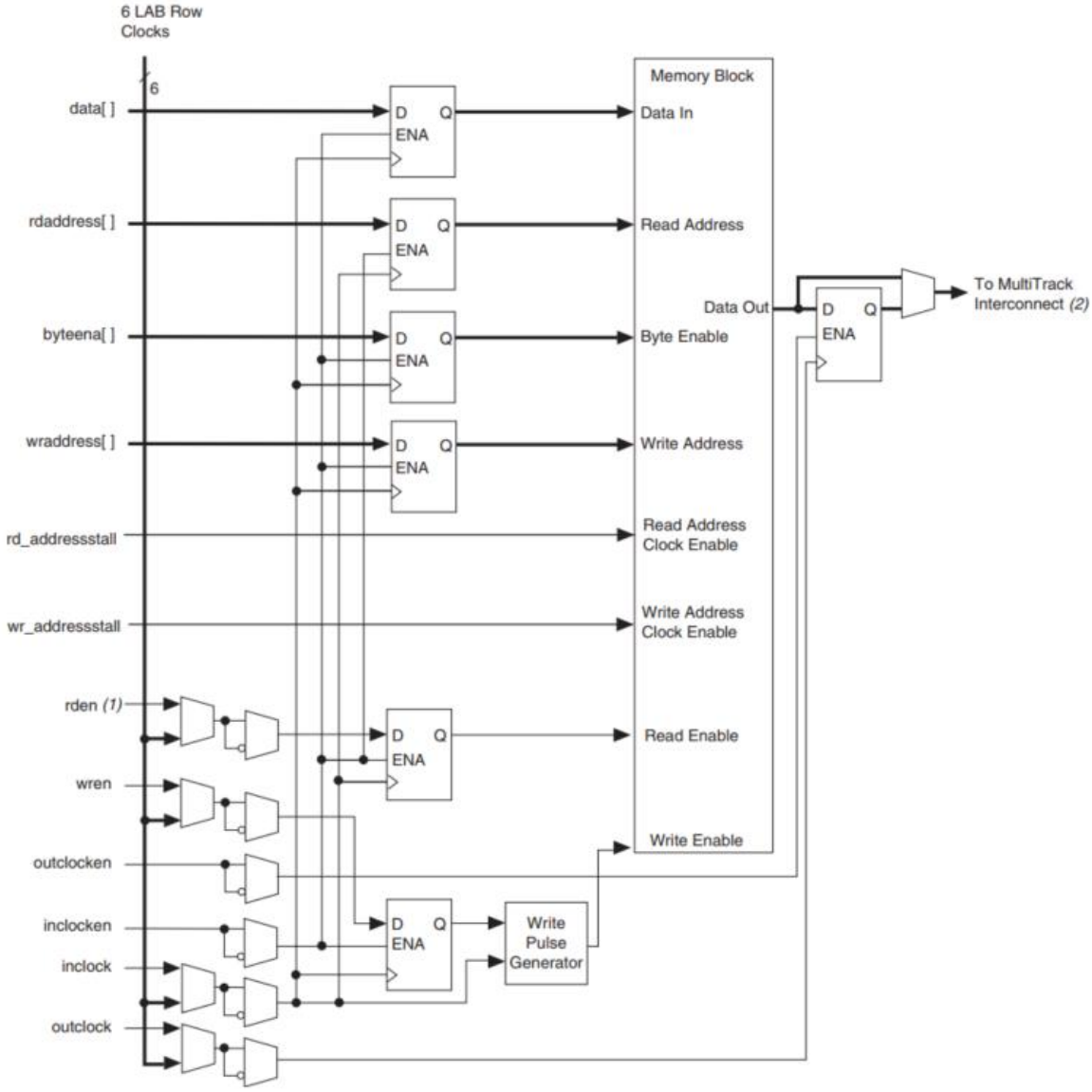
```
always @(*) begin
    predict_taken = counter[index][1];  // Predict taken if MSB is 1
end

// Update logic
always @(posedge clk or posedge reset) begin
    if (reset) begin
        // Initialize all counters to weakly taken (01)
        counter = '{default: 2'b01};
    end
    else if (update_en) begin
        // Update saturating counter
        case (counter[ex_index])
            2'b00: counter[ex_index] <= actual_taken ? 2'b01 : 2'b00;
            2'b01: counter[ex_index] <= actual_taken ? 2'b10 : 2'b00;
            2'b10: counter[ex_index] <= actual_taken ? 2'b11 : 2'b01;
            2'b11: counter[ex_index] <= actual_taken ? 2'b11 : 2'b10;
        endcase
    end
end
```

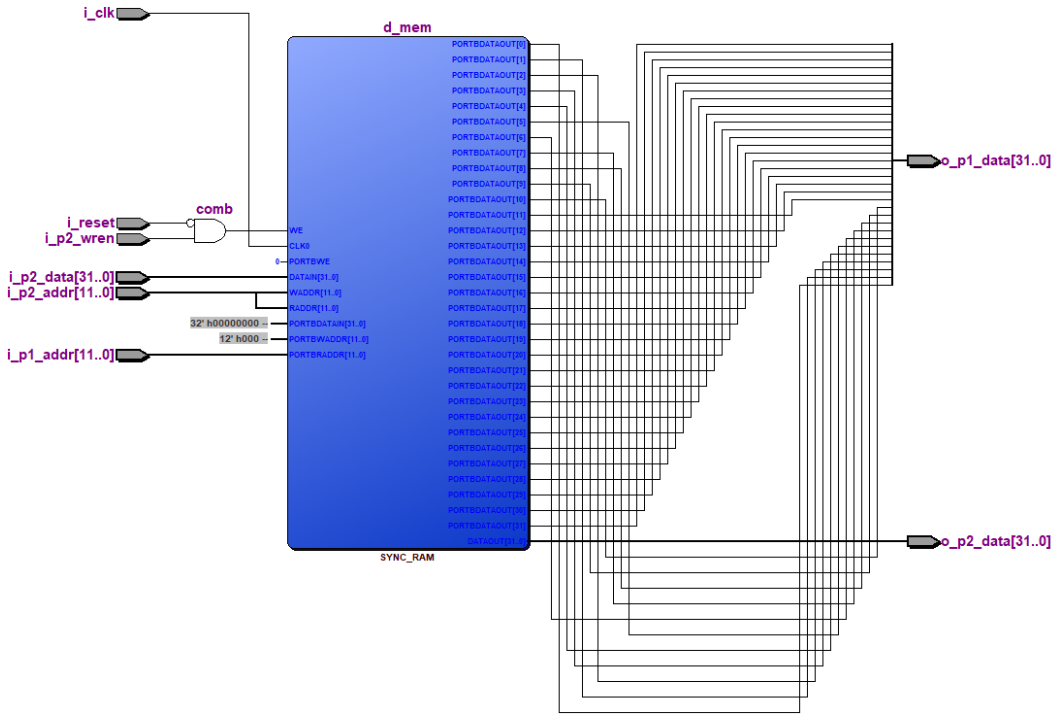
2.7. BRAM

Bộ nhớ cần phải chứa bộ nhớ Intruction cash và memory cash vì vậy cần tạo bộ nhớ có 2 port đọc ghi độc lập nhau vì vậy ta cần init model BRAM đáp ứng được điều kiện trên. Vì vậy ta sử dụng cấu hình BRAM Simple Dual-Port hoạt động ở Input/Output clock mode với clock độc lập nhau ở mỗi port và hỗ trợ chế độ đọc bất đồng bộ clock.

Figure 8–15. Cyclone II Input/Output Clock Mode in Simple Dual-Port Mode *Notes (1), (2)*



Hình dưới là kết quả Netlist Bram trên quartus. Vì trên chip cyclone II EP2C35F672C6 chỉ hỗ trợ 105 block memory M4K là khoảng 483840 bit nên khi tạo BRAM với kích thước 16Kbyte cùng với bộ nhớ ROM của control unit của CPU thiết kế đã chiếm 53% bộ nhớ, nếu tạo kích thước Bram 32Kbyte sẽ chiếm 105% vì thế tối đa bộ nhớ ram của CPU chỉ hỗ trợ 16Kbyte.



3. Verification Strategy

3.1. Kiểm tra hoạt động CPU với từng câu lệnh

Kiểm tra hoạt động của từng model RiscV với ISA test để xem có câu lệnh nào vi xử lý thiết kế bị lỗi hay không.

Nhóm Lệnh	Lệnh	Kết Quả	Ghi chú
Số học	add		Kiểm tra cộng 2 số nguyên.
	addi		Kiểm tra cộng với hằng số.
	sub		Kiểm tra trừ 2 số nguyên.
	and/andi		Kiểm tra phép AND.
	or/ori		Kiểm tra phép OR.

	xor/xori		Kiểm tra phép XOR.
So sánh	slt/slti		Kiểm tra so sánh nhỏ hơn (có dấu).
	sltu/sltiu		Kiểm tra so sánh nhỏ hơn (không dấu).
Dịch bit	sll/slli		Kiểm tra dịch trái logic.
	srl/srli		Kiểm tra dịch phải logic.
	sra/srai		Kiểm tra dịch phải số học.
Truy cập bộ nhớ	lw		Kiểm tra đọc từ bộ nhớ (word).
	lh/lb		Kiểm tra đọc half-word/byte.
	sw		Kiểm tra ghi vào bộ nhớ.
Nhảy/Branch	beq/bne		Kiểm tra nhảy có điều kiện.
	blt/bltu		Kiểm tra nhảy nếu nhỏ hơn.
	bge/bgeu		Kiểm tra nhảy nếu lớn hơn hoặc bằng.
	jal/jalr		Kiểm tra nhảy không điều kiện và liên kết.
Khác	auipc		Kiểm tra tính địa chỉ tương đối PC.
	lui		Kiểm tra tải hằng số vào thanh ghi.
	malgn		Lỗi căn chỉnh bộ nhớ (Optional)

3.2. Kiểm tra IPC và Misprediction của model

Ta viết một số chương trình để benchmark các model CPU đã thiết kế như bảng dưới đây.

Kiểm tra	Chương trình	Kết Quả	Ghi chú

IPC	ISA test		Kiểm tra IPC bằng code sử dụng nhiều các câu lệnh gây harzard như đọc ghi và tính toán, nhảy và rẽ nhánh.
IPC & Mispredict	Benchmark 1		Chương trình có 2 câu lệnh rẽ nhánh taken và 1 câu lệnh non-taken lặp lại nhiều lần.
IPC & Mispredict	Benchmark 2		Chương trình có 1 câu lệnh rẽ nhánh taken và 2 câu lệnh non-taken lặp lại nhiều lần.
IPC & Mispredict	Benchmark 3		Chương trình có 1 câu lệnh rẽ nhánh non-taken và 2 câu lệnh JAL lặp lại nhiều lần.
IPC & Mispredict	Benchmark 4		Chương trình có 1 câu lệnh rẽ nhánh non-taken và 2 câu lệnh JALR lặp lại nhiều lần.

3.3.1. Bechmark 1

```
lui x2, 0x00004 # 0x000F4
addi x2, x2, 1000 #0x240
loop: addi x2, x2, -1
beq x2, x0, cont
bne x2, x0, cont
nop
cont: bge x2, x0, loop
ebreak
```

3.3.1. Bechmark 2

```
lui x2, 0x00004 # 0x000F4
addi x2, x2, 1000 #0x240
loop: addi x2, x2, -1
beq x2, x0, cont
beq x2, x0, cont
nop
cont: bge x2, x0, loop
ebreak
```

3.3.1. Bechmark 3

```
lui x2, 0x00004 # 0x000F4
```

```
addi x2, x2, 1000 #0x240
loop: addi x2, x2, -1
beq x2, x0, end
jal x0, cont
nop
cont: jal x0, loop
end: ebreak
```

3.3.1. Bechmark 4

```
lui x2, 0x00004 # 0x000F4
addi x2, x2, 1000 #0x240
loop: addi x2, x2, -1
beq x2, x0, end
jalr x0, x0, cont
nop
cont: jalr x0, x0, loop
end: ebreak
```

Hình dưới đây là kết quả ISA test model Gshare:

File Edit View Terminal Tabs Help

```
or.......PASS
ori......PASS
xor......PASS
xori.....PASS
slt......PASS
slti.....PASS
sltu.....PASS
sltiu....PASS
sll......PASS
slli.....PASS
srl......PASS
srli.....PASS
sra......PASS
srai.....PASS
lw.......PASS
lh.......PASS
lhu......PASS
lb.......PASS
lbu......PASS
sw.......PASS
sh.......PASS
sb.......PASS
auipc....PASS
lui......PASS
beq......PASS
bne......PASS
blt......PASS
bltu.....PASS
bge......PASS
bgeu.....PASS
jal......PASS
jalr.....PASS
malgn....ERROR
iosw.....PASS

Result

IPC = 0.87
IPC          =  0.87
Mispred Rate = 44.33 %
```

4. Result

4.1. Quartus

Model 1: Non-forwarding		
Logic Utilization/Total logic elements	2446	7%
Fmax	67.93 MHz	
Model 2: Forwarding		
Logic Utilization/Total logic el ements	2764	8%
Fmax	60.24 MHz	
Model 3: Always taken		
Logic Utilization/Total logic elements	2934	9%
Fmax	56.2 MHz	
Model 4: 2-bit Prediction		
Logic Utilization/Total logic elements	2939	9%
Fmax	55.77 MHz	
Model 5: Gshare 2-bit Prediction		
Logic Utilization/Total logic elements	7140	21%
Fmax	55.26 MHz	

Dưới đây là hình ảnh kết quả Synthesize, Timing và Netlist của model 5 (Gshare) trên quartus.

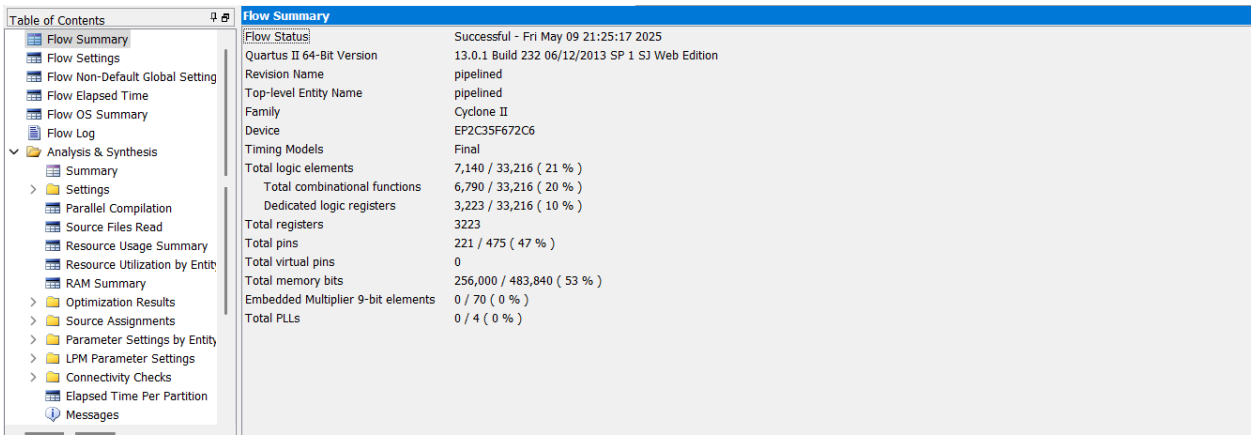
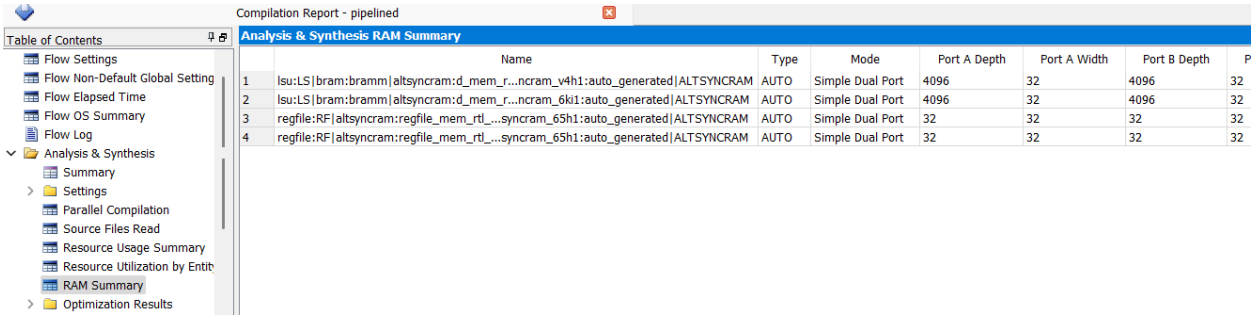
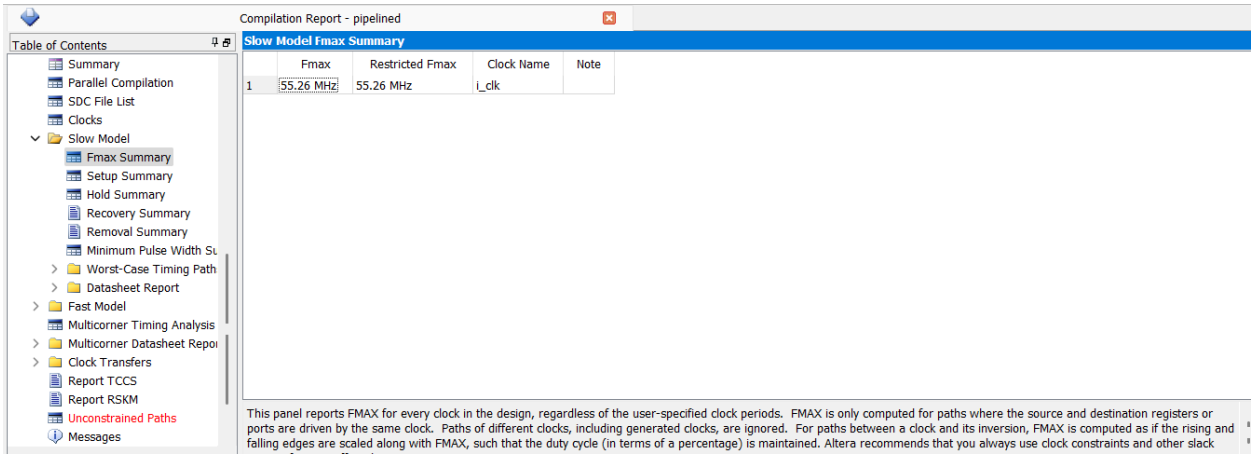


Table of Contents
Flow Summary
Flow Settings
Flow Non-Default Global Setting
Flow Elapsed Time
Flow OS Summary
Flow Log
Analysis & Synthesis
Summary
Settings
Parallel Compilation
Source Files Read
Resource Usage Summary
Resource Utilization by Entit
RAM Summary
Optimization Results
Source Assignments
Parameter Settings by Entity
LPM Parameter Settings
Connectivity Checks
Elapsed Time Per Partition
Messages
Flow Summary
Flow Status Successful - Fri May 09 21:25:17 2025
Quartus II 64-Bit Version 13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name pipelined
Top-level Entity Name pipelined
Family Cyclone II
Device EP2C35F672C6
Timing Models Final
Total logic elements 7,140 / 33,216 (21 %)
Total combinational functions 6,790 / 33,216 (20 %)
Dedicated logic registers 3,223 / 33,216 (10 %)
Total registers 3223
Total pins 221 / 475 (47 %)
Total virtual pins 0
Total memory bits 256,000 / 483,840 (53 %)
Embedded Multiplier 9-bit elements 0 / 70 (0 %)
Total PLLs 0 / 4 (0 %)


Compilation Report - pipelined
Table of Contents
Flow Settings
Flow Non-Default Global Setting
Flow Elapsed Time
Flow OS Summary
Flow Log
Analysis & Synthesis
Summary
Settings
Parallel Compilation
Source Files Read
Resource Usage Summary
Resource Utilization by Entit
RAM Summary
Optimization Results
Analysis & Synthesis RAM Summary
Name Type Mode Port A Depth Port A Width Port B Depth P
1 lsu:LS|bram:bramm|altsyncram:d_mem_r...ncram_v4h1:auto_generated|ALTSYNCRAM AUTO Simple Dual Port 4096 32 4096 32
2 lsu:LS|bram:bramm|altsyncram:d_mem_r...ncram_6ki1:auto_generated|ALTSYNCRAM AUTO Simple Dual Port 4096 32 4096 32
3 regfile:RF|altsyncram:regfile_mem_rtl_...syncram_65h1:auto_generated|ALTSYNCRAM AUTO Simple Dual Port 32 32 32 32
4 regfile:RF|altsyncram:regfile_mem_rtl_...syncram_65h1:auto_generated|ALTSYNCRAM AUTO Simple Dual Port 32 32 32 32


Compilation Report - pipelined
Table of Contents
Summary
Parallel Compilation
SDC File List
Clocks
Slow Model
Fmax Summary
Setup Summary
Hold Summary
Recovery Summary
Removal Summary
Minimum Pulse Width Su
Worst-Case Timing Path
Datasheet Report
Fast Model
Multicorner Timing Analysis
Multicorner Datasheet Repor
Clock Transfers
Report TCCS
Report RSKM
Unconstrained Paths
Messages
Slow Model Fmax Summary
Fmax Restricted Fmax Clock Name Note
1 55.26 MHz 55.26 MHz i_clk
This panel reports FMAX for every clock in the design, regardless of the user-specified clock periods. FMAX is only computed for paths where the source and destination registers or ports are driven by the same clock. Paths of different clocks, including generated clocks, are ignored. For paths between a clock and its inversion, FMAX is computed as if the rising and falling edges are scaled along with FMAX, such that the duty cycle (in terms of a percentage) is maintained. Altera recommends that you always use clock constraints and other slack

4.2. Performance

Benchmark		Non-forwarding	Forwarding	Always taken	Two bit	Gshare
ISA Test	IPC	0.49	0.8	0.86	0.87	0.87
	Mis%	90.39	90.52	50.87	42.64	44.33
Benchmark 1	IPC	0.55	0.75	0.83	0.83	1
	Mis%	66.67	66.67	33.33	33.34	0.01
Benchmark 2	IPC	0.6	0.86	0.78	0.86	1
	Mis%	33.33	33.33	66.67	33.34	0.01
Benchmark 3	IPC	0.55	0.75	0.83	1	1
	Mis%	66.67	66.67	33.33	0	0
Benchmark 4	IPC	0.55	0.75	0.7	0.75	0.75
	Mis%	66.67	66.67	100	66.67	66.67

Nhận xét về IPC: So với model 1 là non-forwarding thì các model còn lại đều là forwarding ta có thể thấy IPC cải thiện rất nhiều (Gần như gấp đôi) so với non-forwarding vì đã giải quyết phần lớn các trường hợp Data harzard .

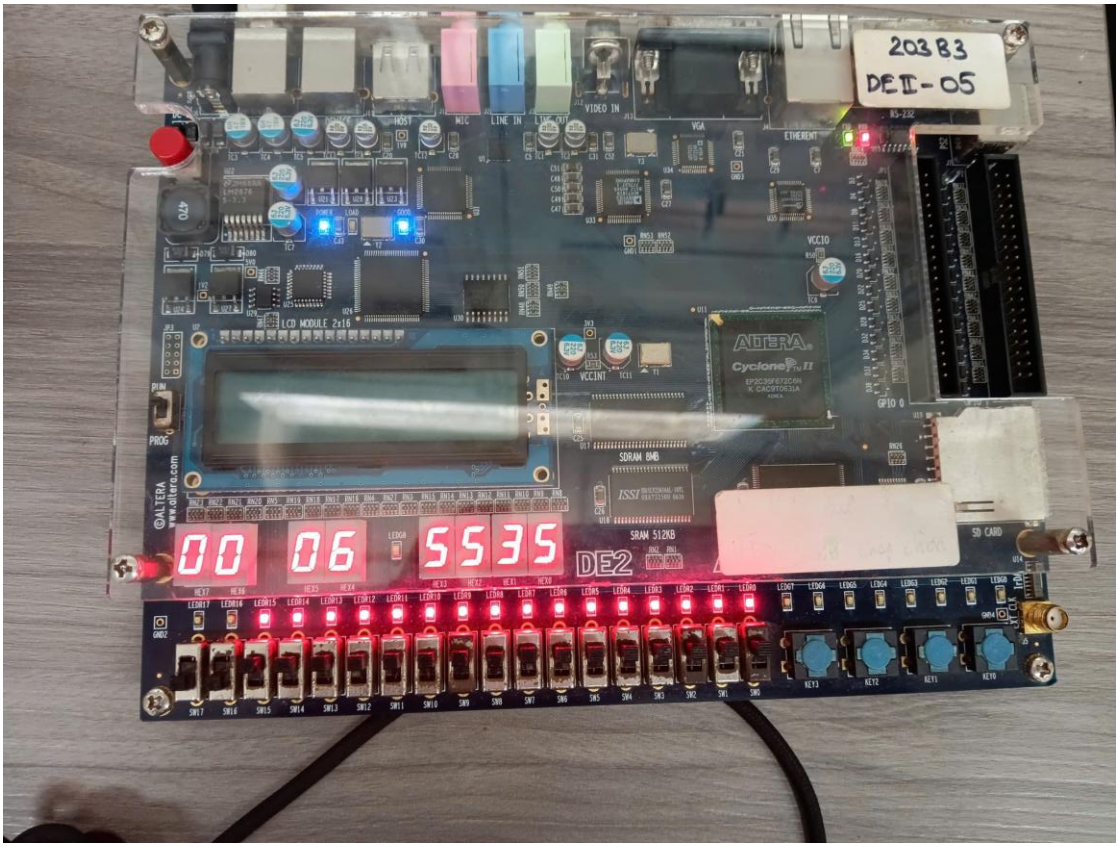
Nhận xét về Misprediction: Model 1 và 2 hoạt đầu tốt với các câu lệnh rẽ nhánh non-taken vì thế với benchmark 2 có 1 câu lệnh rẽ nhánh taken và 2 non-taken nện tỉ lệ dự đoán sai là 1/3=33.3%, các benchmark 1,2,4 có 2 câu lệnh taken nên tỉ lệ là 2/3=66.67%.

Model 3 là always taken các câu lệnh rẽ nhánh và JAL nên với benchmark 1 và 3 có 1 câu lệnh non-taken có tỉ lệ dữ đoán sai 1/3=33.33%, benchmark 2 có 2 câu lệnh non-taken nên tỉ lệ là 2/3=66.67% và benchmark 4 1 câu lệnh non-taken và 2 câu lệnh JALR nên mispredict là 100%.

Model 4 2-bit scheme nên với benchmark 1, 2 lúc dự đoán sẽ có ít nhất một câu lệnh sai ví dụ có 2 câu lệnh taken thì máy trạng thái có khuynh hương đưa ra dự đoán là taken nên tỉ lệ miss là 1/3=33%. Với benchmark 3 và 4 chỉ có 1 câu lệnh rẽ nhánh làm thay đổi máy trạng thái nên cpu sẽ luôn dự đoán đúng câu lệnh rẽ nhánh đó nên với banchmark 3 tỉ lệ miss là 0% và với benchmark 4 tỉ lệ miss là 2/3=66.67% do không dự đoán PC cho câu lệnh JALR.

Model 4 là 2 bit-scheme có Gshare, kết quả benchmark 3, 4 sẽ giống model trên do chỉ có 1 câu lệnh rẽ nhánh, ta chỉ xét benchmark 1 và 2. Với benchmark 1, 2 có 3 câu lệnh rẽ nhánh thì ta thấy tỉ lệ miss là 0.01% vì tại mỗi câu lệnh rẽ nhánh trạng thái quyết định rẽ nhánh là độc lập với nhau nên tỉ lệ dự đoán đúng tăng lên.

4.3. Đổ kit thực tế



Cho CPU chạy với clock 50Mhz, sử dụng chương trình chuyển số nhị phân base 2 thành số tự nhiên base 10 và in kết quả lên led 7 đoạn. Ta cho input từ các switch là 0xFFFF kết quả in ra led 7 đoạn là 65535 là chính xác.

References

Patterson, D. A., & Hennessy, J. L. (2017). *Computer Organization and Design: The Hardware/Software Interface, RISC-V Edition*. Morgan Kaufmann.

Waterman, A., & Asanović, K. (Eds.). (2019). *The RISC-V Reader: An Open Architecture Atlas*. Strawberry Canyon LLC.

Harris, S. L., & Harris, D. M. (2021). *Digital Design and Computer Architecture: RISC-V Edition*. Morgan Kaufmann.

RISC-V Foundation. (2023). *RISC-V Specifications: Volume I – User-Level ISA*. Truy cập từ https://riscv.org/technical/specifications/

Hennessy, J. L., & Patterson, D. A. (2019). *A New Golden Age for Computer Architecture: Domain-Specific Hardware/Software Co-Design*. Communications of the ACM, 62(2), 48–60.

MIT OpenCourseWare. (2016). *6.004: Computation Structures – Lecture Notes*. Truy cập từ https://ocw.mit.edu/courses/6-004-computation-structures-spring-2017/

Weste, N. H. E., & Harris, D. M. (2015). *CMOS VLSI Design: A Circuits and Systems Perspective*. Pearson.