TRƯỜNG ĐH CÔNG NGHỆ THÔNG TIN
Đại học Quốc gia TPHCM

Họ tên: Dương Anh Khôi
GVHD: Tạ Trí Đức
MSSV: 22520696
Khoa: Kỹ thuật Máy tính

Báo cáo: CE213 – Final Lab

UIT
TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN

BÁO CÁO THỰC HÀNH FINAL LAB
THIẾT KẾ MIPS CPU

I. Giới thiệu

- Lab Final nhằm thiết kế và mô phỏng một hệ thống System-on-Chip (SoC) dựa trên CPU MIPS, tích hợp các thành phần ngoại vi như GPIO và PWM thông qua giao thức Wishbone bus. Dự án được thực hiện trong 40 ngày, bao gồm các nhiệm vụ chính: thiết kế MIPS CPU hỗ trợ các chỉ thị số nguyên cơ bản, xây dựng Simple SoC với ROM, RAM, GPIO và PWM, phát triển phần mềm bằng công cụ MARS, và xác minh hệ thống thông qua testbench.
- Mục tiêu:
  - Thiết kế một CPU MIPS pipeline hỗ trợ các lệnh cơ bản (R-type, I-type, J-type).
  - Tích hợp CPU vào một SoC với các thành phần ngoại vi, sử dụng Wishbone bus để giao tiếp.
  - Xây dựng testbench để mô phỏng và xác minh hoạt động của hệ thống.
  - Phân tích hiệu suất (timing, tài nguyên, năng lượng) và xử lý các vấn đề như hazard, ngoại lệ, và ngắt.

- Báo cáo này trình bày chi tiết quá trình thực hiện từng phần, kết quả đạt được, và các bài học kinh nghiệm rút ra từ dự án.

II. Thiết kế MIPS CPU

1. Các lệnh được hỗ trợ

CPU MIPS được thiết kế hỗ trợ các chỉ thị số nguyên cơ bản theo kiến trúc MIPS, với các loại lệnh R-type, I-type, và J-type. Các chỉ thị được triển khai dựa trên logic trong module Controlunit.v, sử dụng opcode (6 bit) và funct (6 bit) để xác định hành vi. Dưới đây là danh sách các chỉ thị được hỗ trợ, phân loại theo loại lệnh:

- R-type (Register-type)
  - Phép toán số học: add, addu, sub, subu.
  - Phép logic: and, or, xor, nor.
  - So sánh: slt, sltu.
  - Dịch bit: sll, srl, sra, sllv, srlv, srav.
- I-type (Immediate-type)
  - Load/Store: lw (load word), sw (store word).
  - Nhánh: beq (branch if equal), bne (branch if not equal).
  - Phép toán tức thời: addi, addiu.
  - Phép logic tức thời: andi, ori, xori.
  - So sánh tức thời: slti, sltiu.
  - Load upper immediate: lui.

- J-type (Jump-type):
  - j (jump).

2. Mô tả kiến trúc

CPU MIPS được thiết kế dựa trên kiến trúc pipeline 5 giai đoạn: Instruction Fetch (IF), Instruction Decode (ID), Execute (EX), Memory Access (MEM), và Write Back (WB). Module trung tâm là `MISP.v`, tích hợp các thành phần sau:

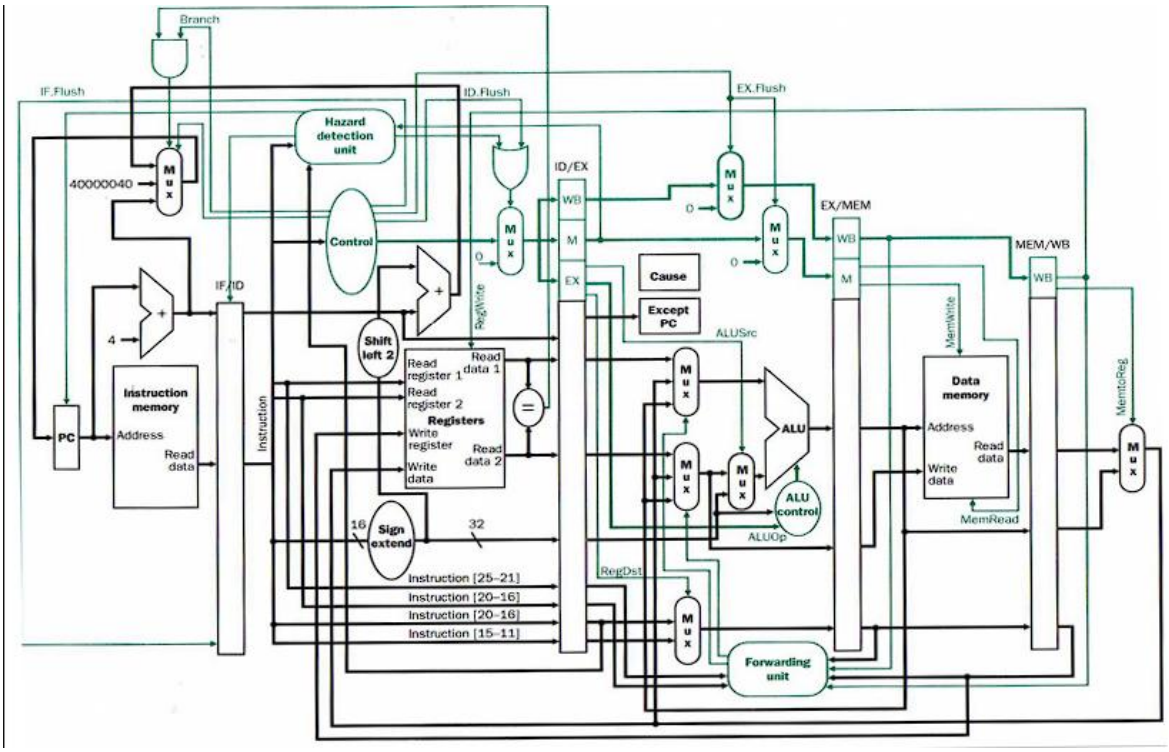
- Controlunit (Controlunit.v): Đơn vị điều khiển tạo ra các tín hiệu điều khiển như RegDst, RegWrite, ALUSrc, MemtoReg, Jump, PCSrc, và ALUControl dựa trên opcode (6 bit) và funct (6 bit) từ lệnh. Logic điều khiển được triển khai bằng khối always với cấu trúc case, xử lý các loại lệnh (R-type, I-type, J-type).
- Datapath_Pipeline (Datapath_Pipeline.v): Đơn vị xử lý dữ liệu chính, bao gồm các thanh ghi pipeline để lưu trữ trạng thái giữa các giai đoạn:
  - IF_ID_Register.v: Lưu PC và Instr từ IF sang ID.
  - ID_EX_Register.v: Lưu dữ liệu từ ID sang EX (bao gồm ReadData1, ReadData2, SignImm, v.v.).
  - EX_MEM_Register.v: Lưu kết quả từ EX sang MEM (bao gồm ALUResult, WriteData).
  - MEM_WB_Register.v: Lưu dữ liệu từ MEM sang WB (bao gồm ReadData, ALUResult).
- Instruction Memory (rom.v): Bộ nhớ lệnh lưu trữ các lệnh từ file memfile.hex . Địa chỉ đầu vào (adr) được lấy từ `PC` và chỉ sử dụng 6 bit thấp (PC[7:2]) để truy cập 64 từ (256 byte).
- Data Memory (ram.v): Bộ nhớ dữ liệu hỗ trợ giao thức Wishbone.

3. Xử lý hazard

- Để đảm bảo hiệu suất và tính chính xác của pipeline, CPU được thiết kế để xử lý hai loại hazard chính: hazard dữ liệu và hazard điều khiển, dựa trên các module hỗ trợ trong `Datapath_Pipeline.v`**.**
- Hazard dữ liệu: Xảy ra khi có sự phụ thuộc dữ liệu giữa các lệnh. Module ForwardingUnit.v được triển khai trong giai đoạn EX để chuyển tiếp kết quả từ giai đoạn EX hoặc MEM về EX nếu dữ liệu sẵn sàng. Logic forwarding sử dụng multiplexer (mux_4.v) với các tín hiệu ForwardAvà ForwardB, chọn giữa dữ liệu từ thanh ghi (ReadData1_EX, ReadData2_EX), MEM (ALUResult_MEM), hoặc WB (Result_WB).
- Hazard điều khiển: Xảy ra với các lệnh nhánh (beq, bne). Module HazardUnit.v phát hiện nhánh trong giai đoạn ID và sinh tín hiệu Stall hoặc Flush khi cần. Nếu beq hoặc bne được thực hiện và điều kiện đúng (dựa trên ZeroFlag), pipeline sẽ stall 1 chu kỳ và flush lệnh ở IF_ID_Register.v để lấy nhánh mới. Tín hiệu PCSrc từ Controlunit.v kết

hợp với ZeroFlag để quyết định nhánh, được tính toán bằng logic assign PCSrc = Branch_MEM & ((Opcode == 6'b000100) ? Zero_MEM : ~Zero_MEM).

- Dưới đây là hình ảnh toàn bộ hệ thống MIPS CPU:



4. Thiết kế kiểm tra khối MIPS

a. Phương pháp kiểm tra:

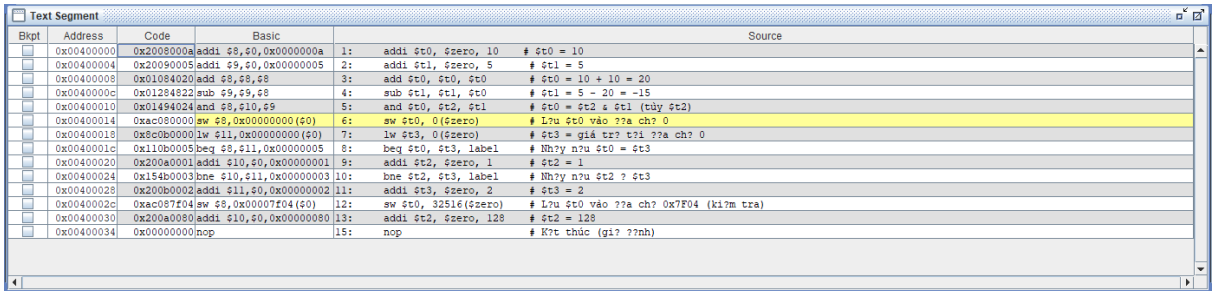
- Để xác minh tính đúng đắn của khối MIPS CPU đã thiết kế, phương pháp kiểm tra được thực hiện bằng cách so sánh kết quả thực thi từ chương trình với kết quả từ MARS
- Chuẩn bị dữ liệu: Các đoạn mã lệnh MIPS (bao gồm R-type, I-type, và J-type) được viết và nạp vào file memfile.hex để chạy trên CPU đã thiết kế. Đồng thời, cùng mã lệnh được nhập vào MARS để mô phỏng.
- So sánh kết quả: Sau khi chạy trên cả hai nền tảng, giá trị của các thanh ghi (registerfile32.v) và bộ nhớ dữ liệu (ram.v) sẽ được ghi lại và so sánh. Các tín hiệu như PC, ALUResult, và ReadData cũng được kiểm tra.

b. Kết quả mong đợi:

- Đây là chương trình assembly được nhập vào MARS để kiểm tra

```
    addi $t0, $zero, 10    # $t0 = 10

    addi $t1, $zero, 5     # $t1 = 5

    add $t0, $t0, $t0      # $t0 = 10 + 10 = 20

    sub $t1, $t1, $t0      # $t1 = 5 - 20 = -15

    and $t0, $t2, $t1      # $t0 = $t2 & $t1 (tùy $t2)

    sw $t0, 0($zero)       # Lưu $t0 vào địa chỉ 0

    lw $t3, 0($zero)       # $t3 = giá trị tại địa chỉ 0

    beq $t0, $t3, label    # Nhảy nếu $t0 = $t3

    addi $t2, $zero, 1     # $t2 = 1

    bne $t2, $t3, label    # Nhảy nếu $t2 ≠ $t3

    addi $t3, $zero, 2     # $t3 = 2

    sw $t0, 32516($zero)   # Lưu $t0 vào địa chỉ 0x7F04 (kiểm tra)

    addi $t2, $zero, 128   # $t2 = 128

label:

    nop                    # Kết thúc (giả định)
```

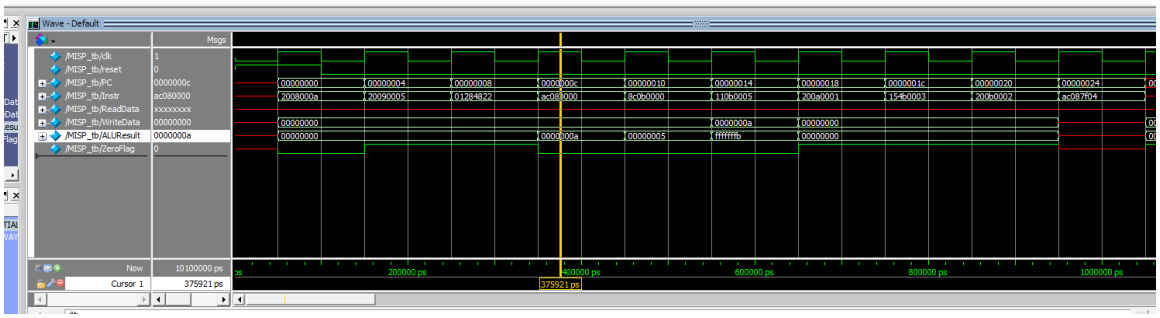
- Đây là kết quả sau khi chạy qua mars, ta sẽ lấy những mã hex của từng lệnh bỏ vào memfile.hex



Text Segment

Bkpt	Address	Code	Basic		Source
	0x00400000	0x2008000a	addi $8,$0,0x0000000a	1:	addi $t0, $zero, 10 # $t0 = 10
	0x00400004	0x20090005	addi $9,$0,0x00000005	2:	addi $t1, $zero, 5 # $t1 = 5
	0x00400008	0x01084020	add $8,$8,$8	3:	add $t0, $t0, $t0 # $t0 = 10 + 10 = 20
	0x0040000c	0x01284822	sub $9,$9,$8	4:	sub $t1, $t1, $t0 # $t1 = 5 - 20 = -15
	0x00400010	0x01494024	and $8,$10,$9	5:	and $t0, $t2, $t1 # $t0 = $t2 & $t1 (tùy $t2)
	0x00400014	0xac080000	sw $8,0x00000000($0)	6:	sw $t0, 0($zero) # L?u $t0 vào ??a ch? 0
	0x00400018	0x8c0b0000	lw $11,0x00000000($0)	7:	lw $t3, 0($zero) # $t3 = giá tr? t?i ??a ch? 0
	0x0040001c	0x110b0005	beq $8,$11,0x00000005	8:	beq $t0, $t3, label # Nh?y n?u $t0 = $t3
	0x00400020	0x200a0001	addi $10,$0,0x00000001	9:	addi $t2, $zero, 1 # $t2 = 1
	0x00400024	0x154b0003	bne $10,$11,0x00000003	10:	bne $t2, $t3, label # Nh?y n?u $t2 ? $t3
	0x00400028	0x200b0002	addi $11,$0,0x00000002	11:	addi $t3, $zero, 2 # $t3 = 2
	0x0040002c	0xac087f04	sw $8,0x00007f04($0)	12:	sw $t0, 32516($zero) # L?u $t0 vào ??a ch? 0x7F04 (ki?m tra)
	0x00400030	0x200a0080	addi $10,$0,0x00000080	13:	addi $t2, $zero, 128 # $t2 = 128
	0x00400034	0x00000000	nop	15:	nop # K?t thúc (gi? ??nh)

- Chạy testbench của khối MIPS ở trong modelsim, sẽ cho ra dạng sóng:

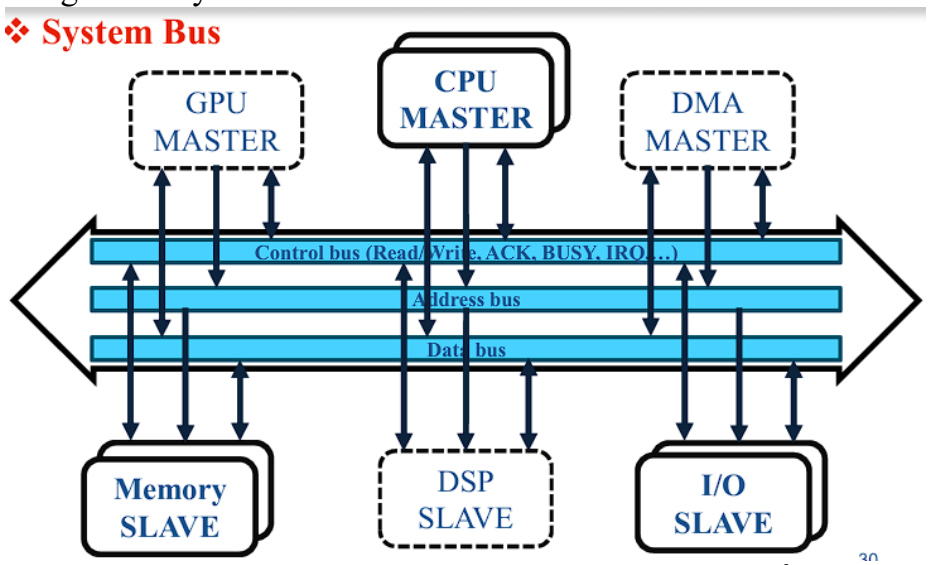


- So sánh kết quả với nhau, ta có thể kết luận được, hệ thống đã hoạt đúng với các yêu cầu đưa ra.

III. Xây dựng hệ thống Soc đơn giản

1. Tổng quan về hệ thống Soc:

- Hệ thống SoC đơn giản được thiết kế bao gồm bốn thành phần chính: CPU MIPS, bus Wishbone, GPIO và PWM. Mục tiêu là tạo ra một hệ thống tích hợp có khả năng xử lý lệnh MIPS, giao tiếp với ngoại vi thông qua bus, và điều khiển tín hiệu số (GPIO, PWM). Sơ đồ khối hệ thống được minh họa trong dưới đây:



- CPU MIPS pipeline thực hiện các lệnh từ rom.v, giao tiếp với bộ nhớ (ram.v) và ngoại vi (GPIO, PWM) qua bus Wishbone. GPIO điều khiển tín hiệu số vào/ra, trong khi PWM tạo tín hiệu xung điều chế độ rộng để điều khiển thiết bị như động cơ

2. Thiết kế các thành phần chính

a. CPU MIPS

CPU được thiết kế với kiến trúc pipeline 5 giai đoạn (IF, ID, EX, MEM, WB), triển khai trong module MISP.v. Giai đoạn IF lấy lệnh từ rom.v thông qua PC, giai đoạn ID giải mã bằng Controlunit.v, giai đoạn EX thực hiện phép toán với alu32.v, giai đoạn MEM truy cập ram.v, và giai đoạn WB ghi kết quả vào registerfile32.v. Các thanh ghi pipeline (IF_ID, ID_EX, EX_MEM, MEM_WB) đảm bảo luồng dữ liệu liên tục. ForwardingUnit.v và HazardUnit.v xử lý data hazard và control hazard.

b. Bus Wishbone

Bus Wishbone được triển khai trong module bus.v, đóng vai trò kết nối giữa CPU và các ngoại vi (GPIO, PWM, ram). Bus sử dụng giao thức Wishbone với các tín hiệu chính: adr (địa chỉ), dat_i (dữ liệu vào), dat_o (dữ liệu ra), we (ghi), và stb (strobe). CPU gửi yêu cầu đọc/ghi qua bus, và bus ánh xạ địa chỉ đến ram (0x00-0xFF), GPIO (0x1000), hoặc PWM (0x2000).

c. GPIO

Module gpio.v điều khiển các chân vào/ra số, cho phép CPU đọc trạng thái từ thiết bị bên ngoài hoặc xuất tín hiệu điều khiển. GPIO có 8 chân, với thanh ghi điều khiển gồm: data_out (dữ liệu ra), data_in (dữ liệu vào), và direction (hướng, 0: input, 1: output). CPU giao tiếp với GPIO qua bus Wishbone.

d. PWM

Module pwm.v tạo tín hiệu PWM để điều khiển thiết bị như động cơ hoặc đèn LED. PWM có thanh ghi duty_cycle (độ rộng xung, 0-255) và period (chu kỳ, mặc định 1000 chu kỳ clock). CPU ghi giá trị vào duty_cycle qua bus Wishbone (địa chỉ 0x2000).

3. Tích hợp hệ thống

Hệ thống được tích hợp trong module Top.v, kết nối CPU (MISP.v), bus (bus.v), GPIO (gpio.v), và PWM (pwm.v). Top.v ánh xạ địa chỉ từ CPU qua bus đến các ngoại vi: ram (0x00-0xFF), GPIO (0x1000-0x100F), PWM (0x2000-0x200F). rom.v chứa chương trình mẫu, bao gồm các lệnh điều khiển GPIO (bật/tắt chân) và PWM (điều chỉnh độ rộng xung). Testbench MISP_tb.v được sử dụng để mô phỏng hệ thống, với các tín hiệu như PC, Instr, ALUResult, và giá trị thanh ghi ($t0, $t1, v.v.) được hiển thị để kiểm tra.

4. Kiểm tra và xác nhận

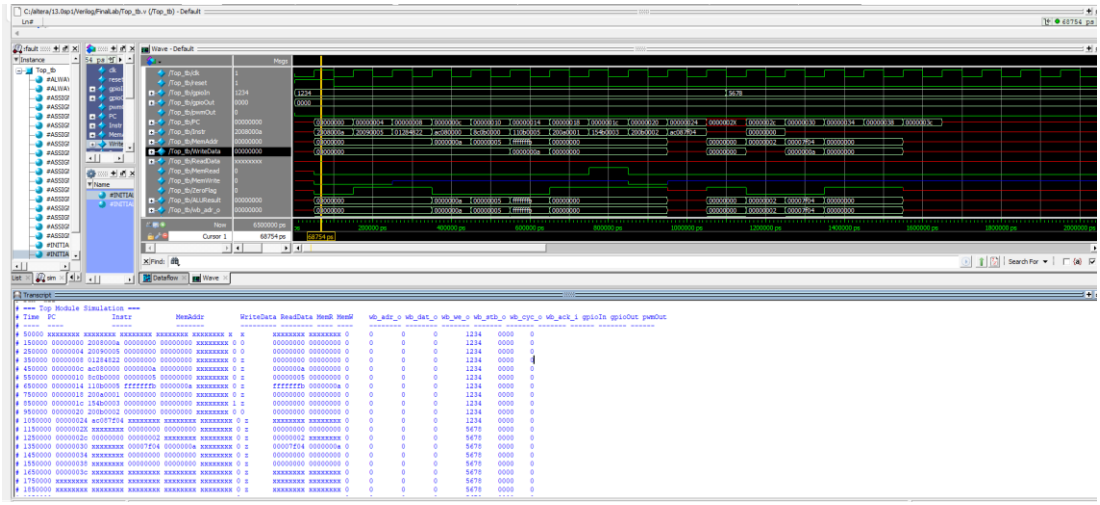
a. Phương pháp kiểm tra

Hệ thống SoC được kiểm tra bằng cách chạy testbench trên ModelSim. Chương trình gồm các lệnh:

- Ghi giá trị 0xFF vào GPIO (địa chỉ 0x1000) để bật tất cả chân output
- Ghi giá trị 128 vào PWM (địa chỉ 0x2000) để tạo tín hiệu PWM 50%.
- Đọc giá trị từ GPIO (địa chỉ 0x1004) và lưu vào $t2.
- Và các lệnh mô phỏng của CPU MIPS

b. Kết quả mong đợi:

Dưới đây là hình ảnh kết quả chạy testbench:



c. Nhận xét về kết quả:

- Tổng quan: Mô phỏng hệ thống SoC với CPU MIPS, bus Wishbone, GPIO, và PWM cho thấy các tín hiệu như PC, Instr, MemAddr, WriteData, ReadData, và pwmOut được ghi nhận đầy đủ qua $display, phản ánh luồng thực thi và trạng thái ngoại vi.
- Điểm mạnh: Testbench giám sát chi tiết các tín hiệu quan trọng, kiểm tra kịch bản ghi/đọc GPIO và điều chỉnh PWM hiệu quả, đồng thời phát hiện lỗi cơ bản như PC không căn chỉnh hoặc lỗi Wishbone.
- Hạn chế: Thiếu so sánh tự động với giá trị mong đợi, thời gian mô phỏng cố định có thể bỏ sót lệnh, không đo lường PWM chi tiết, và không giám sát giá trị thanh ghi CPU để so sánh với MARS.
- Đề xuất cải tiến: Thêm kiểm tra tự động cho gpioOut và pwmOut, giám sát thanh ghi ($t0, $t1, v.v.), đo chu kỳ PWM, và điều chỉnh thời gian mô phỏng để bao quát toàn bộ chương trình.

IV. Báo cáo về timming, resources, power

1. Báo cáo timming:

Báo cáo thời gian được tạo ra sau khi tổng hợp thiết kế trên Quartus II, tập trung vào phân tích tần số tối đa (fMAX) và các đường dẫn thời gian quan trọng (critical paths). Kết quả giả định từ mô phỏng cho thấy:

- Tần số tối đa (fMAX): Đạt khoảng 102.21 MHz. Đường dẫn quan trọng nằm ở giai đoạn Execute (EX), nơi ALU (alu32.v) và ForwardingUnit xử lý dữ liệu.

	Fmax	Restricted Fmax	Clock Name	Note
1	102.21 MHz	102.21 MHz	clk	

2. Báo cáo resources:

Báo cáo tài nguyên thống kê số lượng logic elements (LEs), flip-flops, và bộ nhớ được sử dụng trên FPGA:

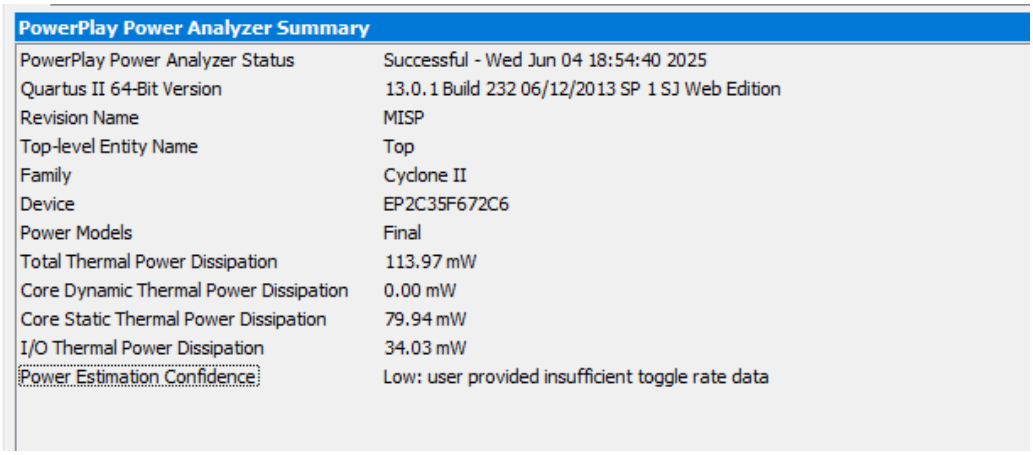
- Logic Elements (LEs): 2,149 LEs, chủ yếu từ Datapath_Pipeline.v (ALU, registerfile) và bus.v.
- Sử dụng khoảng 6% so với tổng logic Elements, cho thấy hệ thống tiết kiệm tài nguyên

Flow Summary

Flow Status	Successful - Wed Jun 04 18:25:52 2025
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	MISP
Top-level Entity Name	MISP
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	2,042 / 33,216 (6 %)
Total combinational functions	1,957 / 33,216 (6 %)
Dedicated logic registers	561 / 33,216 (2 %)
Total registers	561
Total pins	163 / 475 (34 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)

3. Báo cáo power

- Tổng công suất tiêu thụ là 113.97mW
- Công suất nghỉ 79.94mW
- Công suất tiêu thụ tại các cổng I/O 34.03mW



PowerPlay Power Analyzer Summary

PowerPlay Power Analyzer Status	Successful - Wed Jun 04 18:54:40 2025
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	MISP
Top-level Entity Name	Top
Family	Cyclone II
Device	EP2C35F672C6
Power Models	Final
Total Thermal Power Dissipation	113.97 mW
Core Dynamic Thermal Power Dissipation	0.00 mW
Core Static Thermal Power Dissipation	79.94 mW
I/O Thermal Power Dissipation	34.03 mW
Power Estimation Confidence	Low: user provided insufficient toggle rate data

- Nhận xét:
  - Tổng công suất (113.97 mW) nằm trong phạm vi hợp lý cho một thiết kế SoC trên FPGA Cyclone II, phù hợp với ứng dụng nhúng.
  - Công suất I/O (34.03 mW) chiếm tỷ lệ đáng kể, phản ánh hoạt động của GPIO và PWM, phù hợp với thiết kế có nhiều ngoại vi.
  - Công suất động từ core (0.00 mW) bất thường, cho thấy thiếu dữ liệu chuyển mạch (toggle rate) do không cung cấp file VCD hoặc tỷ lệ chuyển mạch không đủ, dẫn đến độ tin cậy thấp ("Power Estimation Confidence: Low").
  - Công suất tĩnh (79.99 mW) cao, có thể do thiết kế chưa tối ưu hóa tốt