ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA KĨ THUẬT MÁY TÍNH

BÀI BÁO CÁO THỰC HÀNH LAB04 MỞ RỘNG THIẾT KẾ LUẬN LÝ SỐ

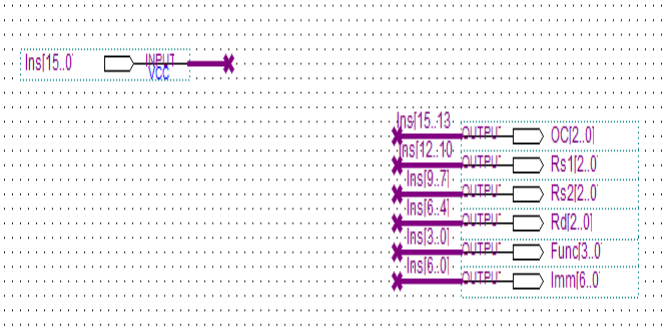
Sinh Viên: Trương Thiên Quý

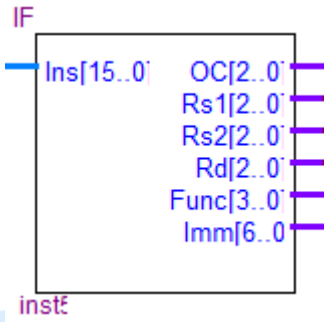
MSSV: 23521321

Lớp: CE118.P11.1

Giảng viên hướng dẫn: Tạ Trí Đức

1/ Instr. Fetch and Decode





Khi một lệnh ở dạng **mã nhị phân (binary)** được nhập vào, một **khối phân tích lệnh** sẽ chia mã nhị phân đó thành các **trường (fields)** tương ứng với cấu trúc của lệnh. Các trường này bao gồm:

1. **Opcode[2..0]**: Được lấy từ bit thứ 15 đến 13 của Instructions. Trường này chứa mã lệnh để xác định loại thao tác cần thực hiện (ví dụ: cộng, trừ, dịch bit, v.v.).
2. **Rs1[2..0]**: Được lấy từ bit thứ 12 đến 10 của Instructions. Trường chứa địa chỉ của thanh ghi nguồn đầu tiên (Source Register 1).
3. **Rs2[2..0]**: Được lấy từ bit thứ 9 đến 7 của Instructions . Trường chứa địa chỉ của thanh ghi nguồn thứ hai (Source Register 2, nếu cần).
4. **Rd**: Được lấy từ bit thứ 6 đến 4 của Instructions. Trường chứa địa chỉ của thanh ghi đích (Destination Register), nơi kết quả sẽ được lưu.
5. **Funct3**: Được lấy từ bit thứ 3 đến 0 của Instructions. Trường mã hóa phụ để xác định biến thể cụ thể của lệnh.
6. **Immediate**: Được lấy từ bit thứ 6 đến 0 của Instructions. Trường chứa giá trị số tức thời (Immediate Value), dùng trong lệnh địa chỉ.

Quá trình hoạt động:

- **Phân tích lệnh**: Khối sẽ tách các trường này từ chuỗi binary dựa trên định dạng cố định.
- **Phân phối đầu vào**: Các trường được tách ra sau đó được truyền tới đầu vào của các khối chức năng khác trong CPU, như:
  - Khối **ALU (Arithmetic Logic Unit)** để thực hiện các phép toán.
  - Khối **Đọc/ghi bộ nhớ** để truy cập bộ nhớ.
  - Khối **Điều khiển** để xác định đường dữ liệu và hoạt động tương ứng.

2/ Phát triển tập lệnh:

2.1/ Tập lệnh RRR:

Func[3]	Func[2]	Func[1]	Func[0]	S2	S1	S0	Shift2	Shift1	Chức năng
0	0	0	0	0	0	0	0	0	Add
0	0	0	1	0	1	0	0	0	Sub
0	0	1	0	0	0	1	0	0	Inc
0	0	1	1	0	1	1	0	0	Dec
0	1	0	0	1	0	0	0	0	And
0	1	0	1	1	1	0	0	0	Or
0	1	1	0	1	1	1	0	0	Xor
0	1	1	1	1	0	1	0	0	Nand
1	0	0	0	1	0	0	1	1	Shl1
1	0	0	1	1	0	0	1	0	Shl2
1	0	1	0	1	0	0	0	1	Sh3
1	0	1	1	X	X	X	0	0	None
1	1	0	0	X	X	X	0	0	None
1	1	0	1	X	X	X	0	0	None
1	1	1	0	X	X	X	0	0	None
1	1	1	1	X	X	X	0	0	None

Q3Q2	Q1Q0			
	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	X	X	X	X
10	1	1	X	1

➔ S2 = Q2 + Q3

Q3Q2	Q1Q0			
	00	01	11	10
00	0	1	1	0
01	0	1	0	1
11	X	X	X	X
10	0	0	X	0

➔ S1 = Q2Q1Q0’ + Q3’Q1’Q0 + Q3’Q2’Q0

Q3Q2	Q1Q0			
	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	X	X	X	X
10	0	0	X	0

➔ S0 = Q3'Q1

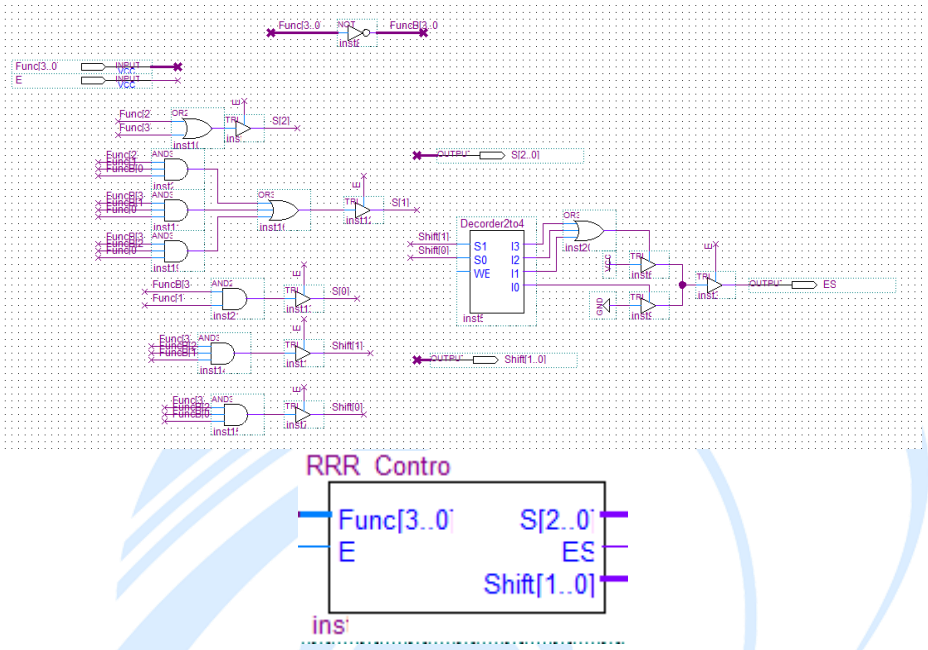
Q3Q2	Q1Q0			
	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	1	0	0

➔ Shift1 = Q3Q2'Q1'

Q3Q2	Q1Q0			
	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	0	0	1

➔ Shift0 = Q3Q2'Q0'

*****RRR Control Hoàn Chỉnh:**



Khi tín hiệu **Enable** được kích hoạt (active), trường **Func[3..0]** sẽ đảm nhận vai trò điều khiển logic để truyền các tín hiệu **S[2..0]** và **Shift[1..0]** đến các đầu vào của các khối chức năng khác trong CPU.

Các khối chức năng này có thể bao gồm:

- **ALU**: Đảm nhận thực hiện các phép toán số học và logic.
- **Shift Unit**: Khối thực hiện các phép dịch bit (shift) như dịch trái, dịch phải số học, hoặc dịch vòng.

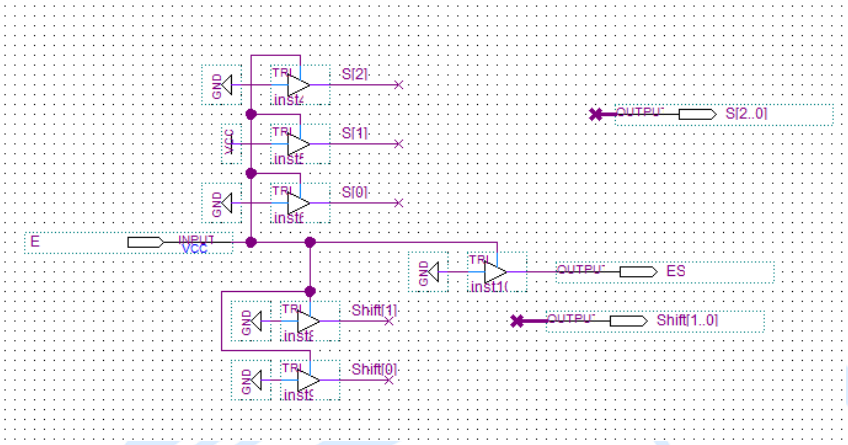
Sau đó, **Shift[1..0]** được đưa qua một bộ **giải mã (Decoder)** loại 2-to-4.

- Bộ **Decoder** này chuyển đổi tín hiệu nhị phân 2 bit (**Shift[1..0]**) thành tín hiệu điều khiển 4 trạng thái, mỗi trạng thái tương ứng với một đầu ra.

Các trạng thái cụ thể:

- Khi **Shift[1..0] = 00**:
  - Tín hiệu **ES** sẽ được thiết lập là **0**.
  - Trạng thái này có thể đại diện cho một chế độ hoạt động mặc định (không cần dịch bit hoặc một thao tác khác mà không liên quan đến Shift).
- Khi **Shift[1..0] = 01**, **10**, hoặc **11**:
  - Tín hiệu **ES** sẽ được thiết lập là **1**.
  - **ES = 1** sẽ kích hoạt một cơ chế đặc biệt, trong đó đầu vào của **Thanh ghi 2** được chọn là **Rs1** thay vì **Rs2**.

2.2/ Tập Lệnh RRI Control:



Khi lệnh **BEQ** được giải mã:

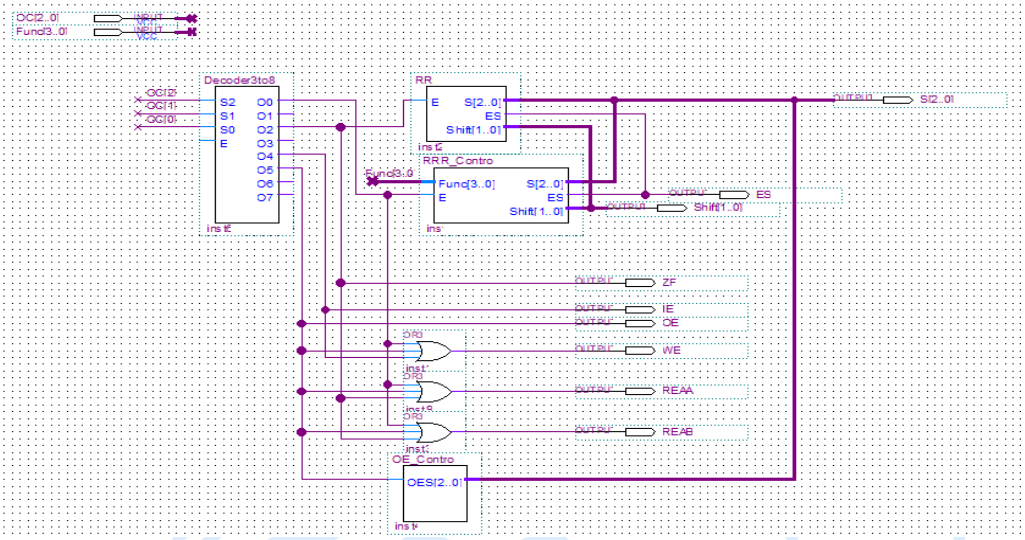
2.2.1/Trạng thái của S[2..0]:

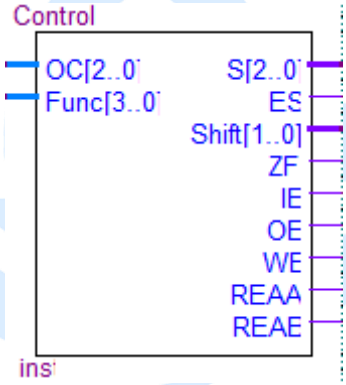
- **S[2..0] = 010**, đây là mã xác định trạng thái thực hiện chức năng **trừ (subtraction)** trong ALU.
- Trong lệnh **BEQ (Branch if Equal)**, chức năng trừ được sử dụng để so sánh hai thanh ghi:
    - ALU thực hiện phép **trừ giữa thanh ghi 1 (Rs1)** và thanh ghi 2 (Rs2)**.
    - Nếu kết quả của phép trừ bằng **0**, điều này nghĩa là giá trị của hai thanh ghi bằng nhau. Khi đó, **PC (Program Counter)** sẽ nhảy đến địa chỉ được tính bằng giá trị **Immediate**.

2.2.2/Trạng thái của ES:

- Giá trị **ES = 0** trong trường hợp này, bởi vì lệnh **BEQ** không yêu cầu thực hiện bất kỳ thao tác **dịch bit (shift)** nào.
- Điều này tương ứng với giá trị **Shift[1..0] = 00**, nghĩa là khối dịch bit không hoạt động.

2.3/ Bộ Control Hoàn Chỉnh:





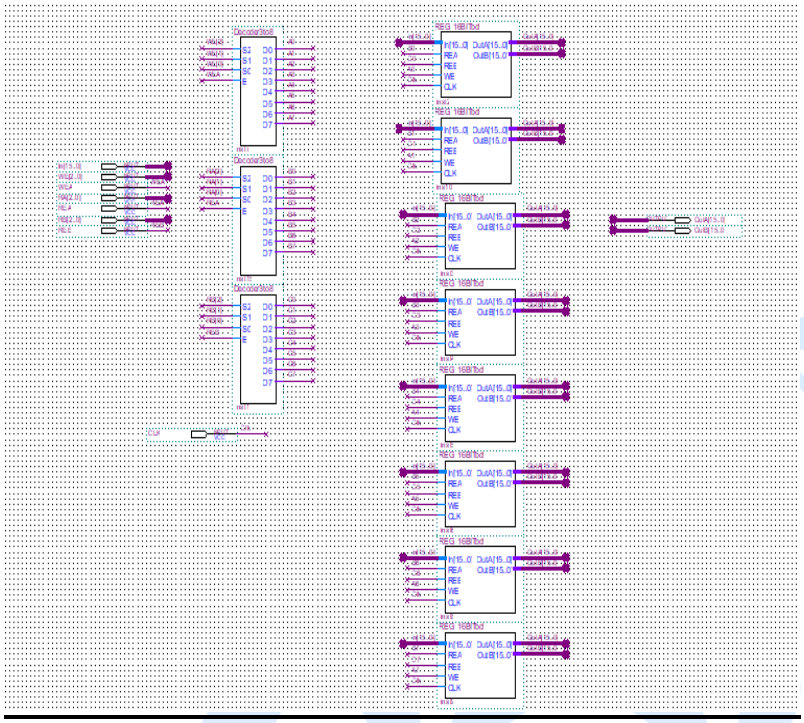
Chức năng

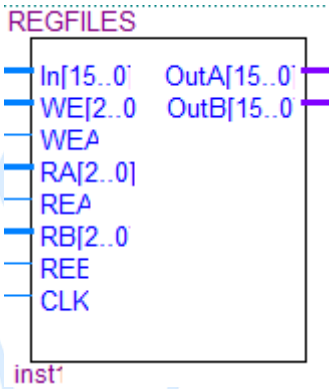
1. **Giải mã Opcode (OC[2..0]):**
   - **Decoder3to8** nhận tín hiệu OC[2..0] (3 bit) để tạo ra 8 tín hiệu đầu ra (O0 đến O7).
   - Mỗi đầu ra của bộ giải mã tương ứng với một mã lệnh duy nhất, và chỉ một đầu ra sẽ ở mức cao tại một thời điểm
   - Có 3 loại lệnh cần dùng đó là OC[2..0] = 000, 010, 100 và 101 theo đề bài.
2. **Tạo tín hiệu điều khiển từ các khối logic phức tạp:**
   - Các khối **RR**, **RRR_Contro**, **Func[3..0]**, và **OE_Contro** phối hợp tín hiệu từ **Decoder3to8**, **Fund[3..0]** (mã chức năng 4 bit), và tín hiệu **Enable (E)** để tạo ra các tín hiệu điều khiển:
     - **S[2..0]:** Điều khiển hoạt động của **ALU**, xác định phép toán cần thực hiện (như cộng, trừ, AND, OR, XOR,...).
     - **ES (Enable Shift):** Điều khiển bộ chọn thanh ghi thứ 2.
     - **Shift[1..0]:** Chọn loại dịch bit hoặc xoay bit (dịch trái, dịch phải, xoay trái, xoay phải...).

- **ZF (Zero Flag):** Đặt cờ báo hiệu kết quả của phép toán là 0.
- **OE (Output Enable):** Kích hoạt đầu ra, thường dùng để điều khiển bus hoặc bộ nhớ.
- **WE (Write Enable):** Kích hoạt ghi, cho phép ghi dữ liệu vào bộ nhớ hoặc thanh ghi.
- **REAA, REAB:** Kích hoạt đọc dữ liệu từ các địa chỉ thanh ghi A và B.

3. **Phân tích tương tác giữa các tín hiệu:**
   - **Kết hợp OC[2..0] và Fund[3..0]:**
     - Mã lệnh **OC[2..0]** xác định **loại hoạt động chính** (ví dụ: phép toán số học, logic, hoặc điều khiển).
     - Mã chức năng **Fund[3..0]** xác định **biến thể của hoạt động** (ví dụ: cộng, trừ, AND,...).
     - Sự kết hợp này cho phép hệ thống xử lý đa dạng các lệnh mà không cần tăng số bit của Opcode.
   - **Vai trò của tín hiệu Enable (E):**
     - Tín hiệu **E** đóng vai trò **kích hoạt hệ thống**, đảm bảo rằng các khối chức năng chỉ hoạt động khi cần thiết, giảm tiêu thụ năng lượng và tránh nhiễu tín hiệu.
   - **Chức năng:**
     - Khi OC[2..0] = 010, E của RRI tích cực sẽ kéo theo ZF, REAA, REAB tích cực và S[2..0] = 010 để thực hiện lệnh trừ. Khi đó Shift sẽ mặc định là 00, không cho phép dịch.
     - Khi OC[2..0] = 000, E của RRR tích cực sẽ kéo theo WE để ghi giá trị vào thanh ghi đích, REAA và REAB tích cực để đọc dữ liệu hai thanh ghi nguồn. Đồng thời xuất ra giá trị điều khiển ALU và Shift tương ứng.
     - Khi OC[2..0] = 100, khối điều khiển sẽ thực thi lệnh ghi đầu vào. Khi đó, IE và WE sẽ được tích cực để ghi giá trị vào thanh ghi đích.
     - Khi OC[2..0] = 101, khối điều khiển sẽ thực hiện chức năng đọc ra output. Khi đó OE sẽ được tích cực để cho phép đọc, WE để ghi lại giá trị, REAA và REAB để đọc giá trị thanh ghi để ALU thực hiện chức năng AND (100) với chính thanh ghi đó.

3/ Reg. Access:





Khối này bao gồm 8 thanh ghi bao gồm:

+ Input[15..0]:

- Là giá trị dữ liệu đầu vào (16 bit) được ghi vào thanh ghi đích.

+ WE[2..0] (Write Enable Address):

- Là địa chỉ 3 bit xác định thanh ghi đích trong tập thanh ghi.
- Kết hợp với **WEA** (Write Enable Active) để kiểm soát việc ghi dữ liệu.

+ WEA (Write Enable Active):

- Tín hiệu cho phép ghi. Khi **WEA** được kích hoạt (mức cao hoặc thấp, tùy thiết kế), dữ liệu từ **Input[15..0]** sẽ được ghi vào thanh ghi đích được chỉ định bởi **WE[2..0]**.

+ RA[2..0] (Read Address 1):

- Là địa chỉ 3 bit xác định thanh ghi đầu tiên cần đọc dữ liệu.

+ REA (Read Enable Active 1):

- Tín hiệu cho phép đọc địa chỉ thứ nhất. Khi **REA** được kích hoạt, dữ liệu từ thanh ghi được chỉ định bởi **RA[2..0]** sẽ xuất hiện ở đầu ra.

+ RB[2..0] (Read Address 2):

- Là địa chỉ 3 bit xác định thanh ghi thứ hai cần đọc dữ liệu.

+ REB (Read Enable Active 2):

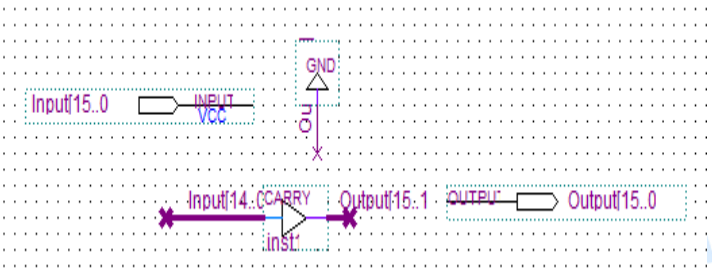
- Tín hiệu cho phép đọc địa chỉ thứ hai. Khi **REB** được kích hoạt, dữ liệu từ thanh ghi được chỉ định bởi **RB[2..0]** sẽ xuất hiện ở đầu ra.

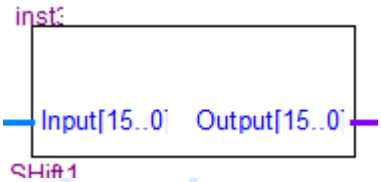
+ CLK (Clock):

- Tín hiệu xung clock, có thể tích cực mức cao hoặc mức thấp (tùy thuộc vào thiết kế cụ thể).
- Dùng để đồng bộ hóa hoạt động đọc/ghi của thanh ghi.

4/ Execution (Bao gồm Shift và ALU)

4.1/ Bộ dịch trái 1 bit:

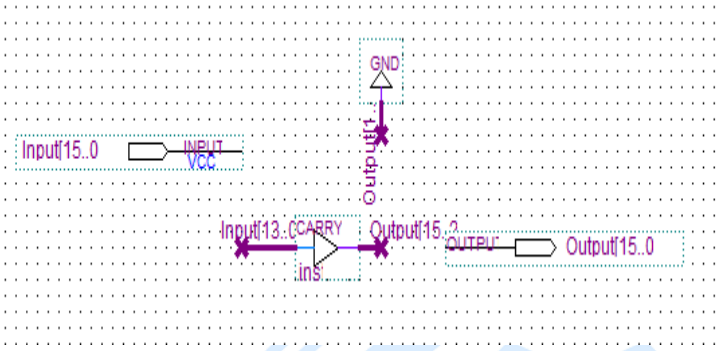


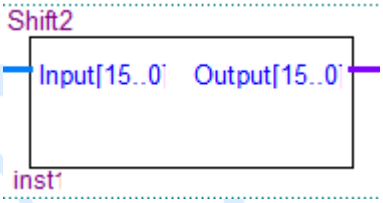


1. **Tín hiệu đầu vào:**
   - **Input[15..0]** là tín hiệu đầu vào chính của mạch. Đây có thể là dữ liệu hoặc kết quả trung gian từ một khối chức năng khác.
   - Tín hiệu **Input[14..CARRY]** được đưa qua một **bộ đệm (inst)**. Bộ đệm này không thay đổi giá trị logic của tín hiệu nhưng có thể được sử dụng để:
     - **Khuếch đại tín hiệu:** Tăng khả năng điều khiển tải của tín hiệu.
     - **Cách ly tín hiệu:** Đảm bảo rằng các mạch khác không ảnh hưởng đến tín hiệu đang xử lý.
2. **Tín hiệu CARRY:**
   - Tín hiệu **CARRY** có thể là **bit nhớ** từ một phép toán trước đó, thường xuất hiện trong các phép toán số học (cộng/trừ) hoặc trong các mạch xử lý dữ liệu phức tạp hơn.
   - Kết hợp **Input[14] với CARRY** cho thấy rằng bit thứ 14 của **Input** có thể bị ảnh hưởng bởi **CARRY**, ví dụ như trong:
     - **Phép toán số học:** CARRY ảnh hưởng đến kết quả ở bit 14.
     - **Mạch dịch bit hoặc xoay bit:** CARRY có thể được đưa vào bit đã dịch ra.
3. **Đầu ra qua bộ đệm:**
   - Đầu ra của bộ đệm được đưa đến **Output[15..1]**, nghĩa là:
     - Các bit từ 14 trở xuống (sau khi được xử lý bởi bộ đệm) được ánh xạ trực tiếp đến các bit từ 15 đến 1 của đầu ra.

- **Output[15..0]** là kết quả cuối cùng của mạch, bao gồm:
  - Các bit từ **Output[15..1]** (đã được xử lý).
  - Bit 0, có thể đến từ một nguồn khác (không được mô tả trong sơ đồ).

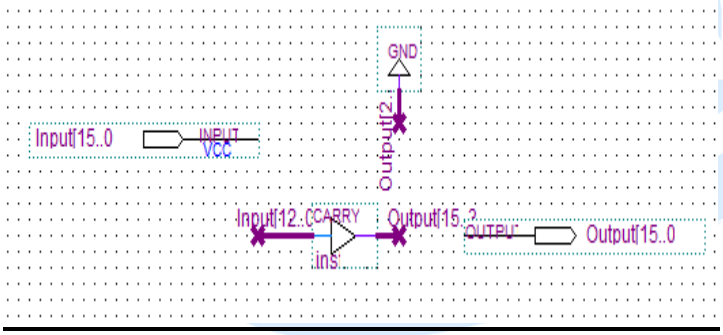
4.2/ Bộ dịch trái 2 bit:

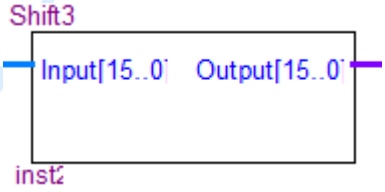




***Tương tự bộ dịch 1 bit.

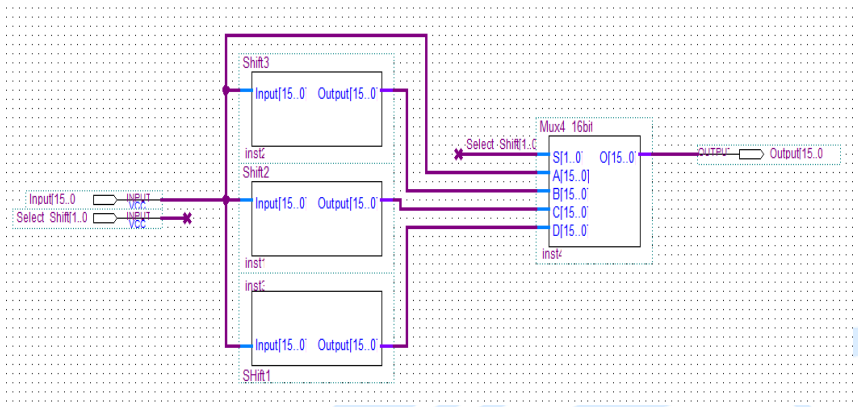
4.3/ Bộ dịch trái 3 bit:

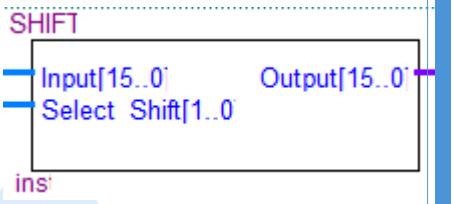




***Tương tự bộ dịch 1 bit.

4.4/ Bộ dịch hoàn chỉnh:



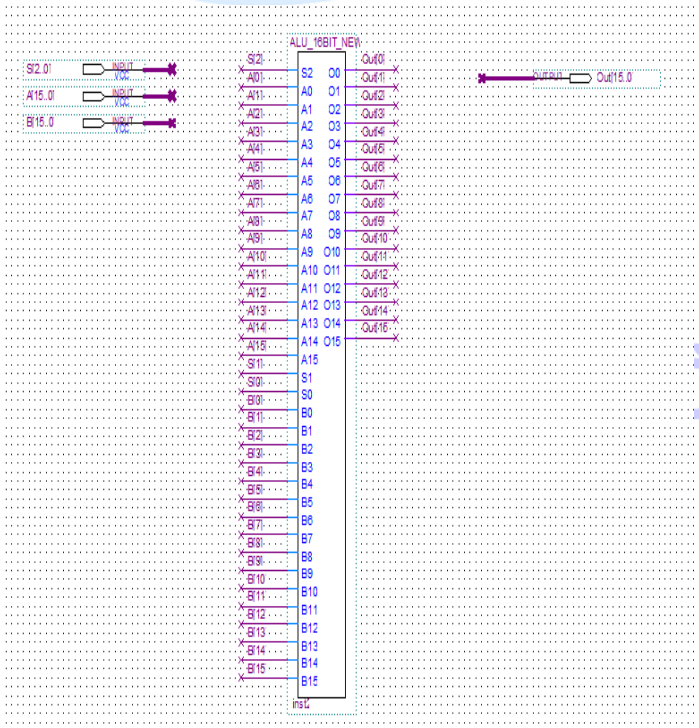


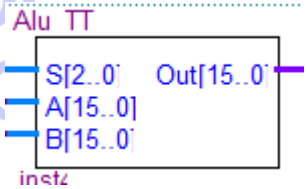
Select_Shift	Dịch trái
00	No shift
01	<< 3
10	<< 2
11	<< 1

Chức năng hoạt động:

1. **Các tín hiệu Input[15..0] từ Shift1, Shift2 và Shift3 được đưa vào bộ ghép kênh.**
2. **Tín hiệu Select Shift[1..0] quyết định đầu vào nào sẽ được chọn.** Dựa trên bảng chân lý ở trên, giá trị của Select Shift[1..0] sẽ chọn một trong bốn đầu vào.
3. **Đầu vào được chọn sẽ được chuyển đến đầu ra Output[15..0].**

4.5/ ALU:





S2	S1	S0	Chức năng
0	0	0	Add
0	0	1	Inc
0	1	0	Sub
0	1	1	Dec
1	0	0	And
1	0	1	Nand
1	1	0	Or
1	1	1	Xor

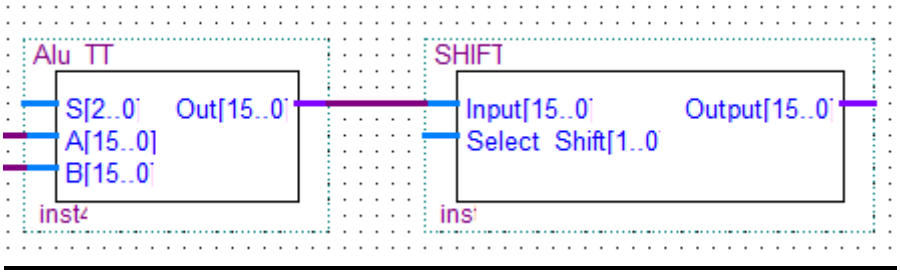
Chức năng hoạt động:

ALU là một khối mạch tổ hợp thực hiện các phép toán số học và logic trên các toán hạng đầu vào. Trong trường hợp này, ALU nhận hai toán hạng 16 bit (A và B) và một tín hiệu điều khiển 3 bit (S[2..0]). Tín hiệu điều khiển này sẽ chọn phép toán cụ thể mà ALU sẽ thực hiện. Kết quả của phép toán được xuất ra trên bus Out[15..0].

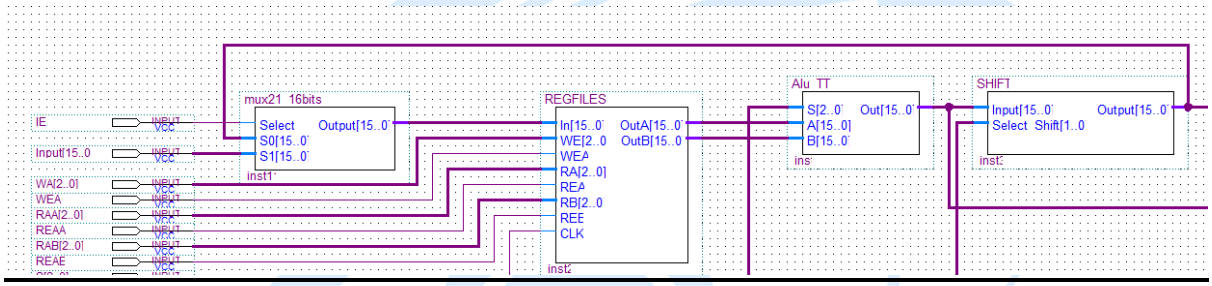
Chi tiết các phép toán:

1. **Phép toán số học:**
    - **A + B (000):** Cộng hai giá trị đầu vào, có thể được sử dụng trong các phép tính thông thường.
    - **A + 1 (001):** Tăng giá trị **A** lên 1, thường dùng để tạo các bộ đếm.
    - **A - B (010):** Trừ giá trị **B** khỏi **A**, thường dùng trong các phép so sánh hoặc kiểm tra điều kiện.
    - **A - 1 (011):** Giảm giá trị **A** đi 1, cũng thường dùng trong các bộ đếm hoặc vòng lặp.
2. **Phép toán logic:**
    - **A and B (100):** Kết hợp hai giá trị bằng phép **AND** logic, giữ lại các bit mà cả **A** và **B** đều bằng 1.
    - **A nand B (101):** Là phủ định của **A and B**, các bit bằng 1 khi bất kỳ bit nào của **A** hoặc **B** khác 1.
    - **A or B (110):** Kết hợp hai giá trị bằng phép **OR** logic, giữ lại các bit mà ít nhất một trong **A** hoặc **B** bằng 1.
    - **A xor B (111):** Thực hiện phép **XOR**, giữ lại các bit mà **A** và **B** khác nhau.

4.6/ Bộ Exc hoàn chỉnh:



5/Write back to Reg:



Trong quá trình **Wrb to Reg**, việc sử dụng **MUX2to1 16 bits** là một cơ chế hiệu quả để lựa chọn giữa giá trị **output** (kết quả sau dịch) và một **input** khác (giá trị từ nguồn bên ngoài hoặc mặc định).

Chức năng của khối MUX2to1 16 bits:

- **Input của MUX:**
  - **Vị trí 0 (Output):** Đây là kết quả từ quá trình dịch bit (hoặc kết quả tính toán trước đó) trong hệ thống.
  - **Vị trí 1 (Input):** Đây là giá trị từ một nguồn khác, có thể là giá trị đầu vào mới được cung cấp từ bên ngoài hoặc giá trị cần được gán đặc biệt để đảm bảo tính ổn định.
- **Tín hiệu điều khiển IE:**
  - **IE tích cực cao (1):** MUX chọn **input** ở vị trí 1 làm đầu ra. Điều này cho phép hệ thống sử dụng giá trị đầu vào mới thay vì tiếp tục sử dụng kết quả đã được xử lý. Điều này thường dùng để **khởi tạo** hoặc **reset** hệ thống, hoặc để **tránh lỗi** khi giá trị output trước đó không hợp lệ.
  - **IE tích cực thấp (0):** MUX chọn **output** ở vị trí 0 làm đầu ra. Giá trị đã qua xử lý được tiếp tục sử dụng trong các lệnh tiếp theo.

Lý do thiết kế như vậy:

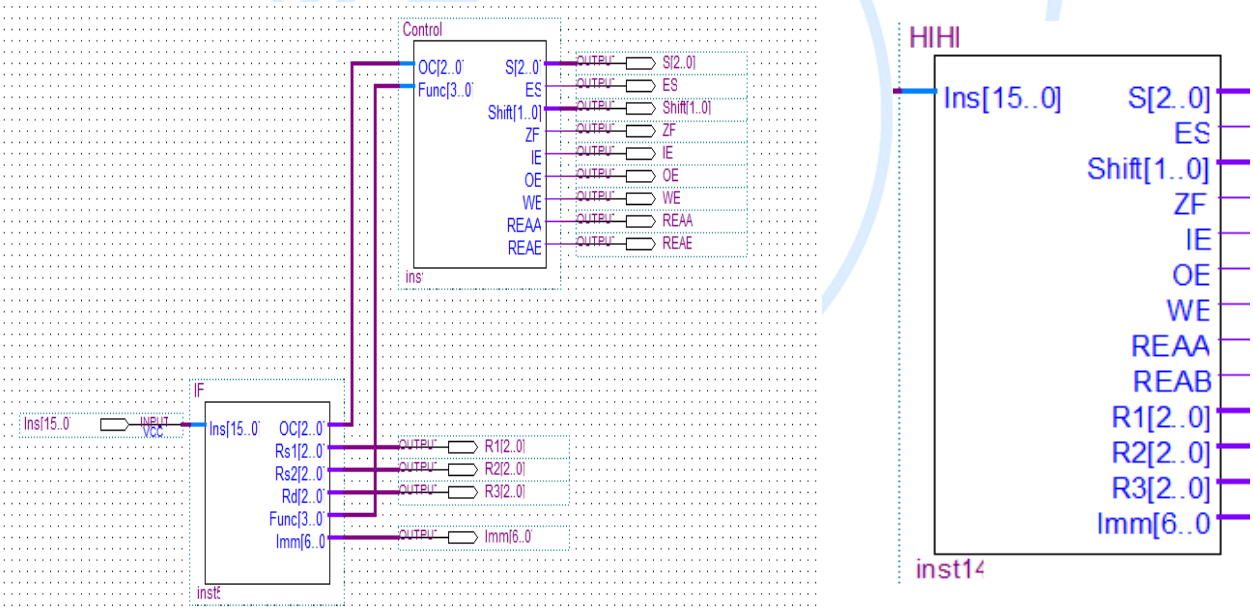
1. **Tránh lỗi:**
   - Nếu giá trị output trước đó không còn hợp lệ (ví dụ, sau một chu kỳ xử lý hoặc khi dữ liệu không được cập nhật kịp thời), việc cho phép chọn input mới sẽ đảm bảo tính chính xác và ổn định cho hệ thống.

2. **Khởi tạo hoặc điều kiện đặc biệt:**
    - Khi cần khởi tạo giá trị trong thanh ghi hoặc thực hiện các thao tác đặc biệt, tín hiệu điều khiển **IE** cho phép hệ thống ghi đè bằng giá trị input mới thay vì sử dụng output trước đó.
3. **Tái sử dụng dữ liệu:**
    - Khi **IE** ở mức thấp, việc tái sử dụng **output** làm đầu vào cho các bước tiếp theo giúp tiết kiệm tài nguyên và tăng hiệu quả xử lý.

6/ Bộ Control và Datapath hoàn chỉnh:

6.1/ Khối Control

Kết hợp các khối mạch đã thiết kế trước đó, cho ra bộ Control hoàn chỉnh như sau:



Sơ đồ mạch trên thể hiện hai khối chính: khối Instruction Fetch (IF) và khối Control. Chúng phối hợp với nhau để giải mã lệnh và tạo ra các tín hiệu điều khiển cho các phần khác của hệ thống, ví dụ như ALU, bộ nhớ, thanh ghi.

1. Khối Instruction Fetch (IF):

- **Ins[15..0]:** Bus đầu vào 16 bit chứa lệnh (Instruction).
- **OC[2..0]:** Đầu ra 3 bit chứa mã lệnh (Opcode).
- **Rs1[2..0]:** Đầu ra 3 bit chứa địa chỉ thanh ghi nguồn thứ nhất.
- **Rs2[2..0]:** Đầu ra 3 bit chứa địa chỉ thanh ghi nguồn thứ hai.
- **Rd[2..0]:** Đầu ra 3 bit chứa địa chỉ thanh ghi đích.
- **Func[3..0]:** Đầu ra 4 bit chứa mã chức năng (Function code).
- **Imm[6..0]:** Đầu ra 7 bit chứa giá trị tức thời (Immediate value).

- **inst (Instance):** Khối này đại diện cho logic bên trong của khối IF, có nhiệm vụ phân tích lệnh và tách ra các thành phần (opcode, địa chỉ thanh ghi, giá trị tức thời).

Chức năng của khối IF:

Khối IF nhận lệnh 16 bit (Ins[15..0]) và phân tích nó để trích xuất các thông tin sau:

- **Mã lệnh (Opcode):** Xác định loại lệnh (ví dụ: cộng, trừ,..) theo yêu cầu đề.
- **Địa chỉ thanh ghi nguồn:** Xác định các thanh ghi được sử dụng làm toán hạng.
- **Địa chỉ thanh ghi đích:** Xác định thanh ghi sẽ chứa kết quả.
- **Mã chức năng:** Cung cấp thông tin chi tiết hơn về hoạt động của lệnh (ví dụ: loại phép cộng, loại dịch bit).
- **Giá trị tức thời:** Một hằng số được nhúng trực tiếp trong lệnh.

2. Khối Control:

- **OC[2..0]:** Đầu vào 3 bit nhận mã lệnh từ khối IF.
- **Func[3..0]:** Đầu vào 4 bit nhận mã chức năng từ khối IF.
- **S[2..0]:** Đầu ra 3 bit, có thể là tín hiệu chọn cho ALU, xác định phép toán mà ALU sẽ thực hiện.
- **ES:** Tín hiệu Enable.
- **Shift[1..0]:** Tín hiệu điều khiển dịch bit.
- **ZF (Zero Flag):** Cờ báo kết quả bằng không.
- **IE (Interrupt Enable):** Cho phép ngắt.
- **OE (Output Enable):** Cho phép đầu ra.
- **WE (Write Enable):** Cho phép ghi (thường dùng cho bộ nhớ).
- **REAA (Read Enable Address A):** Cho phép đọc địa chỉ A (thường dùng cho bộ nhớ).
- **REAE (Read Enable Address E):** Cho phép đọc địa chỉ E (chức năng cụ thể cần thêm thông tin).
- **inst (Instance):** Khối này chứa logic điều khiển, có nhiệm vụ tạo ra các tín hiệu điều khiển dựa trên mã lệnh và mã chức năng.

Chức năng của khối Control:

Khối Control nhận mã lệnh (OC[2..0]) và mã chức năng (Func[3..0]) từ khối IF, sau đó tạo ra các tín hiệu điều khiển để điều khiển các thành phần khác của hệ thống. Ví dụ:

- Dựa trên mã lệnh, khối Control có thể tạo ra tín hiệu S[2..0] và Shift[1..0] để chọn phép toán phù hợp trên ALU.
- Các tín hiệu WE, ZF, REAA, REAE, ES được sử dụng để điều khiển việc đọc/ghi dữ liệu từ bộ nhớ hoặc các thanh ghi và các lệnh tương ứng nếu cần.

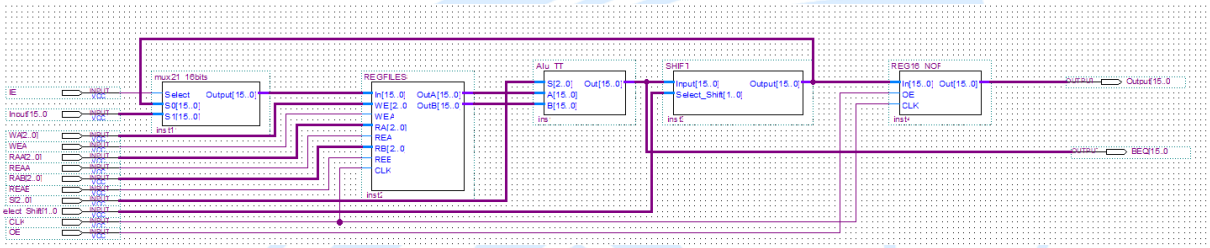
- Tín hiệu IE, OE được sử dụng để quản lý vào/ra.

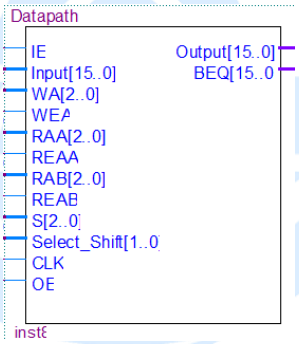
Mối quan hệ giữa hai khối:

Khối IF lấy lệnh từ bộ nhớ (không được hiển thị trong sơ đồ) và phân tích nó. Sau đó, nó chuyển mã lệnh và mã chức năng đến khối Control. Khối Control sử dụng thông tin này để tạo ra các tín hiệu điều khiển cần thiết để thực hiện lệnh.

6.2/ Khối Datapath:

Ta được khối Datapath hoàn chỉnh như sau:





Các khối chính:

- **mux21_16bit:** Bộ ghép kênh (Multiplexer) 2 chọn 1, 16 bit. Nó chọn một trong hai đầu vào 16 bit dựa trên tín hiệu điều khiển Select.
- **REGFILE8:** Khối điều khiển đọc/ghi bộ nhớ (Memory Read/Write Control). Nhận các tín hiệu điều khiển và tạo ra các tín hiệu cho phép đọc/ghi.
- **ALU_TT:** Khối Arithmetic Logic Unit (ALU), thực hiện các phép toán số học và logic trên hai đầu vào 16 bit (A[15..0] và B[15..0]) dựa trên tín hiệu điều khiển S[2..0].
- **SHIFT:** Một bộ dịch bit (Shifter), dịch đầu vào 16 bit dựa trên tín hiệu điều khiển Select_Shift[1..0] tương ứng.
- **REGFILE_NOR:** Một thanh ghi (Register) có chức năng cho phép ghi (Write Enable - WE). Lưu trữ giá trị đầu vào khi tín hiệu OE (Output Enable) và CLK (Clock) hoạt động.

Các tín hiệu điều khiển:

- **WA[2..0], WEA, REAA, REAB, RA[2..0], RB2[2..0]:** Các tín hiệu này liên quan đến việc đọc/ghi bộ nhớ hoặc các thanh ghi.
- **IE:** Tín hiệu điều khiển bộ ghép kênh mux21.

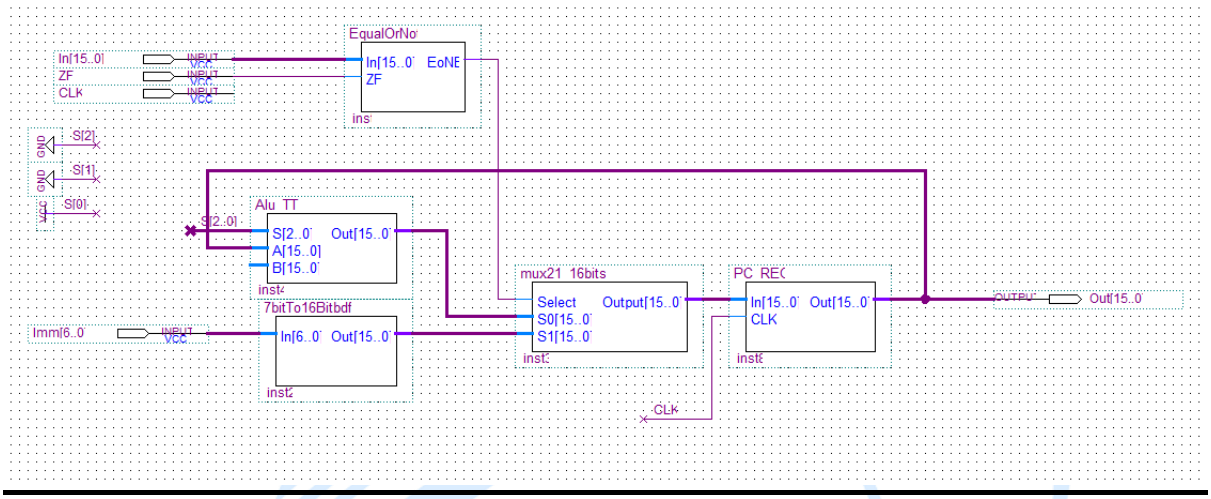
- **S[2..0]:** Tín hiệu điều khiển ALU, chọn phép toán được thực hiện.
- **Select_Shift[1..0]:** Tín hiệu điều khiển bộ dịch bit SHF1.
- **CLK:** Tín hiệu xung nhịp, đồng bộ hoạt động của các khối, đặc biệt là thanh ghi REGIE NOF.
- **OE:** Output Enable cho thanh ghi REGFILE_NOR, cho phép dữ liệu được xuất ra từ thanh ghi.

Luồng dữ liệu và hoạt động:

1. **Đọc dữ liệu:** Dựa vào các tín hiệu RAA[2..0], REAA, RAB[2..0], REAB, WA[2..0], WE và khối REOFLES, dữ liệu được đọc từ bộ nhớ hoặc các thanh ghi.
2. **Chọn dữ liệu:** Bộ ghép kênh mux21 chọn một trong hai nguồn dữ liệu 16 bit dựa trên tín hiệu Select.
3. **Xử lý ALU:** Dữ liệu được chọn được đưa vào ALU (ALU_TT), nơi nó được xử lý dựa trên tín hiệu S[2..0].
4. **Dịch bit (tùy chọn):** Kết quả từ ALU có thể được đưa qua bộ dịch bit SHF1 để dịch trái hoặc dịch phải dựa trên tín hiệu Select_Shift[1..0].
5. **Lưu trữ kết quả:** Kết quả cuối cùng được lưu trữ trong thanh ghi REGFILE_NOR khi tín hiệu CLK và OE được kích hoạt.

Ngoài ra, còn 1 output **BEQ[15..0]** được sử dụng dành riêng cho lệnh **BEQ** để lưu kết quả khi trừ đi 2 thanh ghi**. Output** này sau đó sẽ tiếp tục được sử dụng ở **khối PC.**

7. Khối PC



Các khối chính:

- **EqualOrNo (EONE):** Bộ so sánh (Comparator), kiểm tra xem hai đầu vào 16 bit có bằng nhau hay không. Đầu ra EoNE sẽ ở mức cao nếu hai đầu vào bằng nhau.
- **Alu TT:** Khối ALU (Arithmetic Logic Unit), thực hiện các phép toán số học và logic trên hai đầu vào 16 bit (A[15..0] và B[15..0]) dựa trên tín hiệu điều khiển S[2..0].
- **mux21 16bits:** Bộ ghép kênh (Multiplexer) 2 chọn 1, 16 bit. Nó chọn một trong hai đầu vào 16 bit dựa trên tín hiệu điều khiển Select.
- **7bit To 16Bitbdf:** Khối này chuyển đổi một đầu vào 7 bit (Imm[6..0]) thành một đầu ra 16 bit. Có thể nó thực hiện việc mở rộng dấu (sign extension) hoặc thêm các bit 0 vào đầu.
- **PC REC:** Khối này có vẻ là một thanh ghi (Register) Program Counter hoặc một thanh ghi khác. Nó lưu trữ giá trị đầu vào 16 bit khi tín hiệu CLK (Clock) hoạt động.

2.0 Flash Experimental. Might not work as expected.

Đây là sơ đồ mạch thể hiện một phần của hệ thống xử lý, tập trung vào khối ALU và các thành phần liên quan. Dưới đây là phân tích chi tiết:

Các khối chính:

- **EqualOrNo (EONE):** Khối này có vẻ như là bộ so sánh (Comparator), kiểm tra xem hai đầu vào 16 bit có bằng nhau hay không. Đầu ra ZF (Zero Flag) sẽ ở mức cao nếu hai đầu vào bằng nhau.
- **Alu_TT (Thay cho khối Adder):** Khối ALU (Arithmetic Logic Unit), thực hiện các phép toán số học và logic trên hai đầu vào 16 bit (A[15..0] và B[15..0]) dựa trên tín hiệu điều khiển S[2..0].
- **mux21 16bits:** Bộ ghép kênh (Multiplexer) 2 chọn 1, 16 bit. Nó chọn một trong hai đầu vào 16 bit dựa trên tín hiệu điều khiển Select.

- **7bit To 16Bitbdf:** Khối này chuyển đổi một đầu vào 7 bit (Imm[6..0]) thành một đầu ra 16 bit. Có thể nó thực hiện việc mở rộng dấu (sign extension) hoặc thêm các bit 0 vào đầu.
- **PC REC:** Khối này có vẻ là một thanh ghi (Register) Program Counter hoặc một thanh ghi khác. Nó lưu trữ giá trị đầu vào 16 bit khi tín hiệu CLK (Clock) hoạt động.

Các tín hiệu điều khiển và dữ liệu:

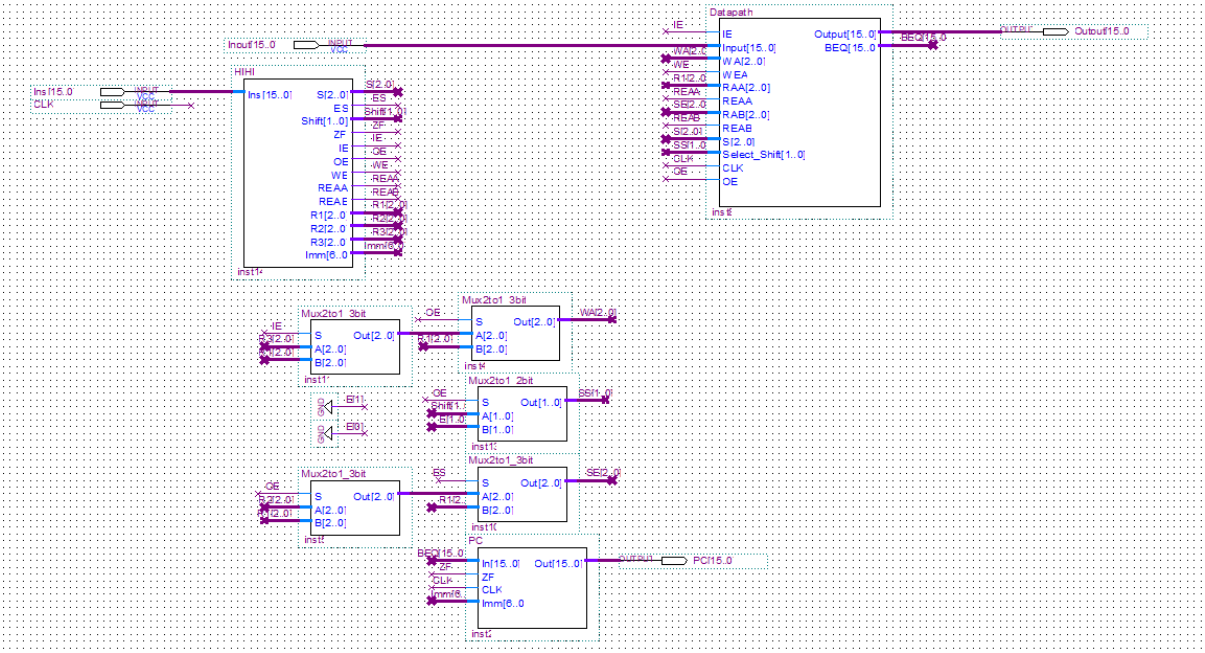
- **In[15..0] (cho EqualOrNo):** Hai đầu vào 16 bit được so sánh bởi khối EqualOrNo.
- **ZF:** Zero Flag, tín hiệu điều khiển xem coi mạch có đang thực hiên lệnh BEQ hay không.
- **CLK:** Tín hiệu xung nhịp, đồng bộ hoạt động của các khối, đặc biệt là thanh ghi PC REC.
- **S[2..0]:** Tín hiệu điều khiển ALU, chọn phép toán được thực hiện.
- **A[15..0], B[15..0]:** Hai đầu vào 16 bit của ALU.
- **Imm[6..0]:** Đầu vào 7 bit cho khối chuyển đổi 7bit To 16Bitbdf.
- **Select:** Tín hiệu điều khiển bộ ghép kênh mux21.
- **Out[15..0] (từ Alu TT và mux21):** Đầu ra 16 bit của ALU và bộ ghép kênh.

Luồng dữ liệu và hoạt động:

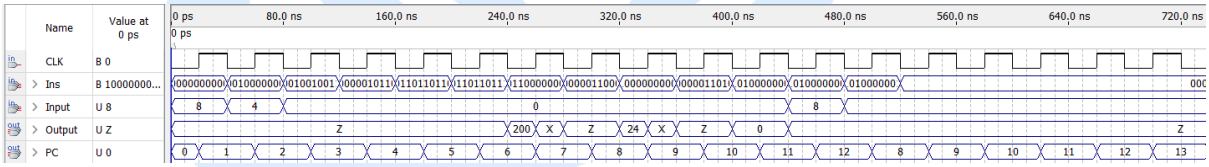
1. **So sánh (EqualOrNo):** Nếu **ZF** tích cực mức cao, hai đầu vào 16 bit được so sánh bởi khối EqualOrNo. Nếu chúng bằng nhau, EoNE được đặt lên mức cao.
2. **Chuyển đổi 7 bit sang 16 bit:** Đầu vào 7 bit Imm[6..0] được chuyển đổi thành 16 bit bởi khối 7bit To 16Bitbdf.
3. **Xử lý ALU:**
   - Một trong hai nguồn dữ liệu được chọn làm đầu vào B cho ALU: hoặc là kết quả chuyển đổi từ Imm[6..0] (đã được mở rộng thành 16 bit), hoặc một giá trị PC trước đó đã được cộng thêm 1. Việc lựa chọn này được thực hiện bởi bộ ghép kênh mux21 dựa trên tín hiệu Select.
   - ALU thực hiện phép toán trên đầu vào A và B dựa trên tín hiệu S[2..0] = 001 (Thực hiện phép toán cộng 1 cho giá trị hiện tại của PC).
4. **Lưu trữ kết quả:** Kết quả từ PC được lưu trữ trong thanh ghi PC REC khi có xung nhịp CLK.

8/ Bộ vi xử lý hoàn chỉnh:

Kết hợp tất cả các khối trên ta được CPU hoàn chỉnh:



Mô phỏng chức năng của mạch:



Với instruction lần lượt là:

+) **1000000000000000 (Input = 8)**:

- Ghi giá trị **8** vào thanh ghi số **0**.

+) **1000010000000000 (Input = 4)**:

- Ghi giá trị **4** vào thanh ghi số **1**.

+) **0000010010011001**:

- Dịch trái thanh ghi số **1** (001) đi 2 bit.
- Giá trị sau khi dịch: 4 × 4 = 16.

+) **0000000010110000**:

- Cộng thanh ghi số **0** (giá trị 8) và thanh ghi số **1** (giá trị 16).
- Lưu kết quả vào thanh ghi số **3** (011).
- Giá trị sau cộng: 8 + 16 = 24.

+) **0000110110110010**:

- Cộng thêm 1 vào giá trị thanh ghi số **3** (giá trị hiện tại 24).
- Giá trị sau khi cộng: 24 + 1 = 25.

+) **0000110110111010**:

- Dịch trái thanh ghi số **3** đi 3 bit.
- Giá trị sau dịch: 25×8=200

+) **1010110000000000**:

- Xuất giá trị thanh ghi số **3** ra ngoài output.
- Output: **200**.

+) **0000000011000110**:

- Thực hiện XOR hai thanh ghi số **0** (8) và số **1** (16).
- Lưu kết quả vào thanh ghi số **4** (100).
- Kết quả: 8 ⊕ 16 = 24.

+) **1011000000000000**:

- Xuất giá trị thanh ghi số **4** ra ngoài output.
- Output: **24**.

+) **0000000011010100**:

- Thực hiện AND hai thanh ghi số **0** (8) và số **1** (16).
- Lưu kết quả vào thanh ghi số **5** (101).
- Kết quả: 8 & 16 = 0

+) **1011010000000000**:

- Xuất giá trị thanh ghi số **5** ra ngoài output.
- Output: **0**.

+) **1001010000000000**:

- Ghi giá trị **8** vào thanh ghi số **5**.

+) **0101010000001000**:

- Thực hiện lệnh BEQ để kiểm tra giá trị của thanh ghi số **5** và số **0**.
- Vì cả hai đều bằng **8**, PC nhận giá trị **8** từ trường Immediate.

9/Kiểm tra tài nguyên:

	Resource	Usage
1	Estimated Total logic elements	492
2		
3	Total combinational functions	380
4	Logic element usage by number of LUT inputs	
1	-- 4 input functions	311
2	-- 3 input functions	53
3	-- <=2 input functions	16
5		
6	Logic elements by mode	
1	-- normal mode	380
2	-- arithmetic mode	0
7		
8	Total registers	151
1	-- Dedicated logic registers	151
2	-- I/O registers	0
9		
10	I/O pins	65
11	Embedded Multiplier 9-bit elements	0
12	Maximum fan-out node	CLK
13	Maximum fan-out	151
14	Total fan-out	1904
15	Average fan-out	3.19

9.1. Estimated Total Logic Elements:

- **492** logic elements được dự báo sử dụng trong thiết kế. Đây là tổng số logic elements cần thiết để thực hiện toàn bộ thiết kế.

9.2. Total Combinational Functions:

- **380** chức năng tổ hợp (combinational logic) được sử dụng. Đây là phần không lưu trữ trạng thái trong thiết kế.

9.3. Logic Element Usage by Number of LUT Inputs:

- **4 input functions**: Có **311** logic sử dụng LUT (Look-Up Table) với 4 đầu vào.
- **3 input functions**: Có **53** logic sử dụng LUT với 3 đầu vào.
- **≤2 input functions**: Có **16** logic sử dụng LUT với 2 hoặc ít hơn đầu vào.

9.4. Logic Elements by Mode:

- **Normal Mode**: **380** logic elements được sử dụng ở chế độ bình thường (tức không xử lý số học).
- **Arithmetic Mode**: Không có logic element nào ở chế độ xử lý số học (**0**).

9.5. Total Registers:

- **151** thanh ghi được sử dụng trong thiết kế.
    - **Dedicated Logic Registers**: Tất cả **151** thanh ghi thuộc về logic cụ thể.
    - **I/O Registers**: Không có thanh ghi I/O nào được sử dụng (**0**).

9.6. I/O Pins:

- **65** chân I/O được sử dụng, phù hợp với thiết kế và gán chân.

9.7. Embedded Multiplier 9-bit Elements:

- Không có bộ nhân nhúng nào được sử dụng (**0**). Điều này cho thấy thiết kế không có các phép nhân phức tạp cần xử lý.

9.8. Maximum Fan-Out Node:

- **CLK** là node có độ fan-out cao nhất (số tín hiệu mà nó điều khiển).

9.9. Maximum Fan-Out:

- Số lượng tối đa tín hiệu được điều khiển bởi một node là **151**.

9.10. Total Fan-Out:

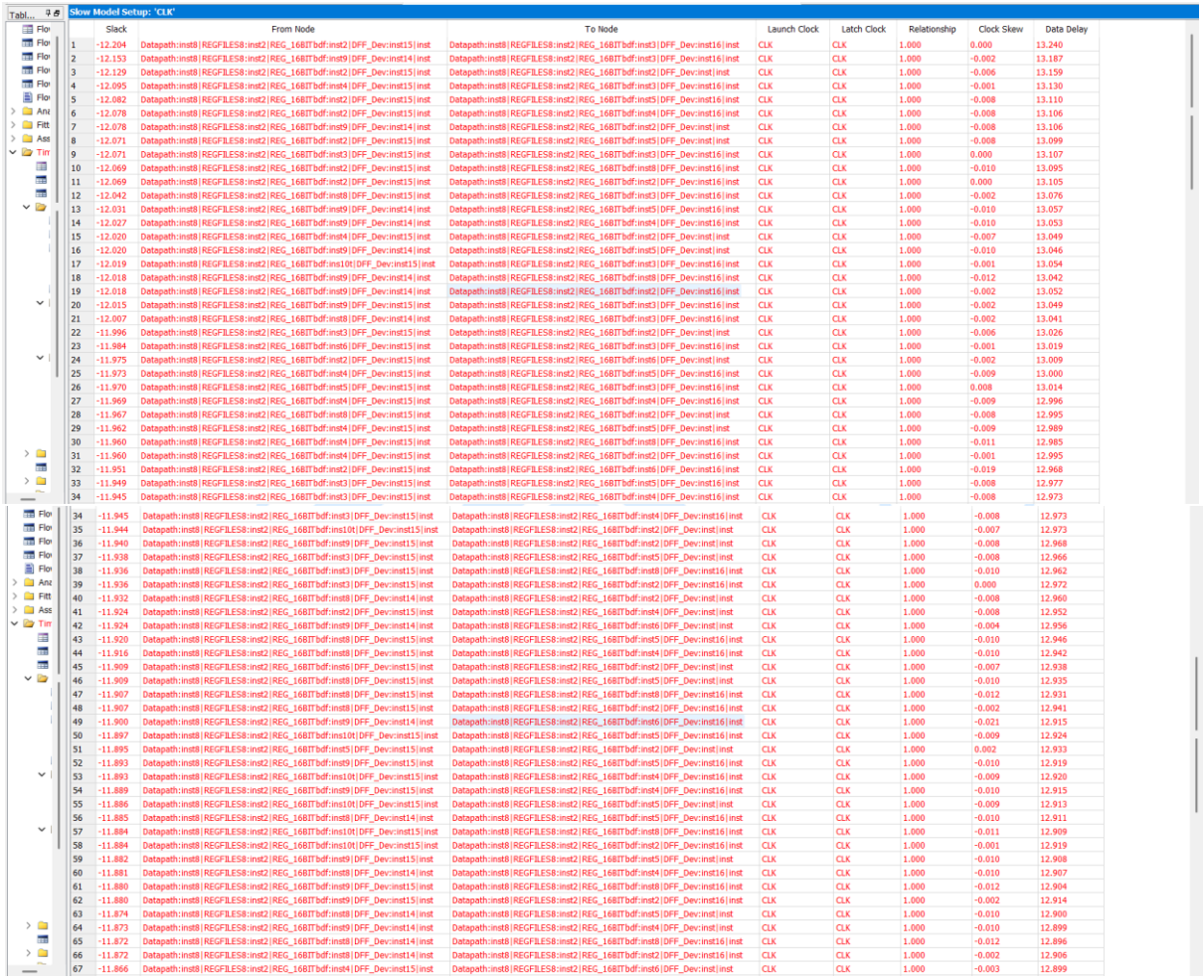
- Tổng số tín hiệu điều khiển (fan-out) trong toàn bộ thiết kế là **1904**.

9.11. Average Fan-Out:

- Trung bình mỗi tín hiệu điều khiển **3.19** tín hiệu khác.

10/Kiểm tra định thời:

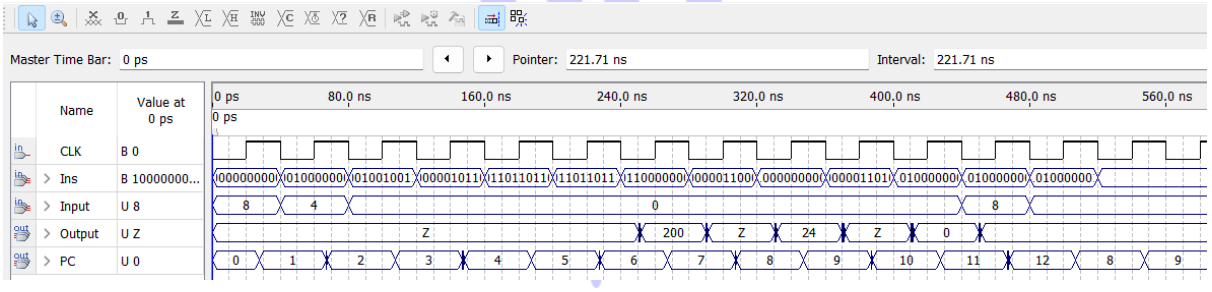
10.1/ Report định thời



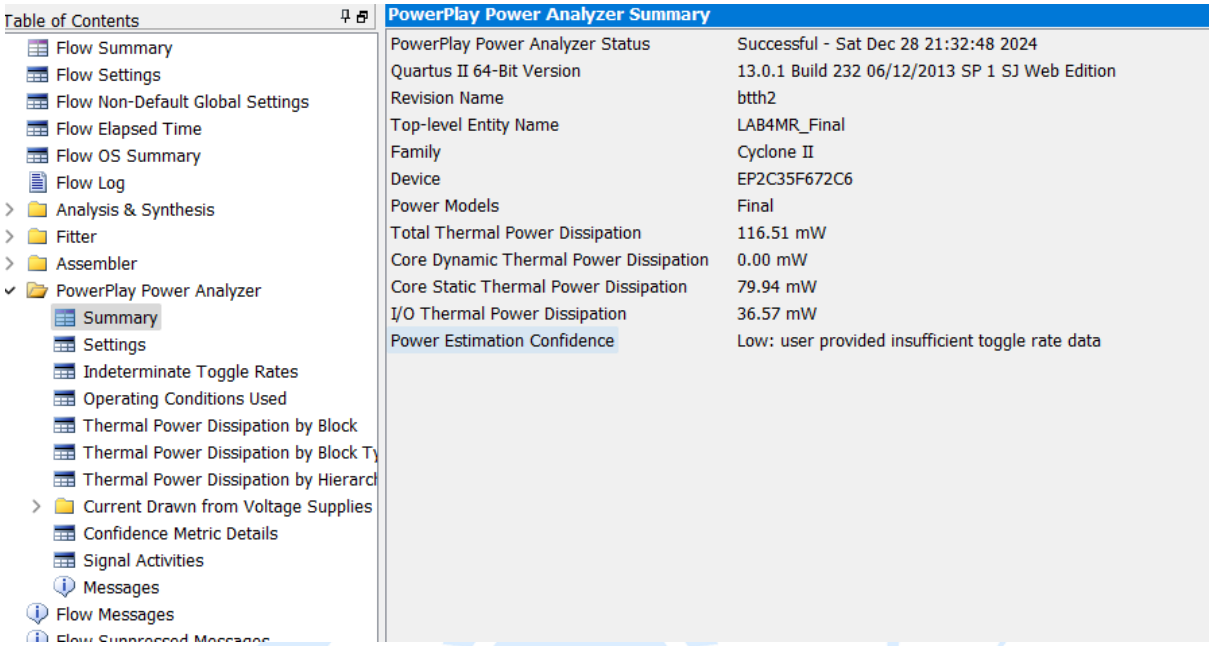
Slow Model Setup: 'CLK'

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	-12.204	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst16|inst	CLK	CLK	1.000	0.000	13.240
2	-12.153	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.002	13.187
3	-12.129	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.006	13.159
4	-12.095	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.001	13.130
5	-12.082	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.008	13.110
6	-12.078	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.008	13.106
7	-12.078	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.008	13.106
8	-12.071	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.008	13.099
9	-12.071	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst16|inst	CLK	CLK	1.000	0.000	13.107
10	-12.069	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.010	13.095
11	-12.069	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst16|inst	CLK	CLK	1.000	0.000	13.105
12	-12.042	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.002	13.076
13	-12.031	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.010	13.057
14	-12.027	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.010	13.053
15	-12.020	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.007	13.049
16	-12.020	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.010	13.046
17	-12.019	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:ins10t|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.001	13.054
18	-12.018	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.012	13.042
19	-12.018	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.002	13.052
20	-12.015	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.002	13.049
21	-12.007	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.002	13.041
22	-11.996	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.006	13.026
23	-11.984	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst6|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.001	13.019
24	-11.975	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst6|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.002	13.009
25	-11.973	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.009	13.000
26	-11.970	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst16|inst	CLK	CLK	1.000	0.008	13.014
27	-11.969	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.009	12.996
28	-11.967	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.008	12.995
29	-11.962	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.009	12.989
30	-11.960	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.011	12.985
31	-11.960	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.001	12.995
32	-11.951	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst6|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.019	12.968
33	-11.949	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.008	12.977
34	-11.945	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.008	12.973
34	-11.945	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.008	12.973
35	-11.944	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:ins10t|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.007	12.973
36	-11.940	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.008	12.968
37	-11.938	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.008	12.966
38	-11.936	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.010	12.962
39	-11.936	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst3|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst16|inst	CLK	CLK	1.000	0.000	12.972
40	-11.932	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.008	12.960
41	-11.924	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.008	12.952
42	-11.924	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst6|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.004	12.956
43	-11.920	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.010	12.946
44	-11.916	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.010	12.942
45	-11.909	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst6|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.007	12.938
46	-11.909	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.010	12.935
47	-11.907	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.012	12.931
48	-11.907	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.002	12.941
49	-11.900	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst6|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.021	12.915
50	-11.897	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:ins10t|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.009	12.924
51	-11.895	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst|inst	CLK	CLK	1.000	0.002	12.933
52	-11.893	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.010	12.919
53	-11.893	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:ins10t|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.009	12.920
54	-11.889	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.010	12.915
55	-11.886	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:ins10t|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.009	12.913
56	-11.885	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.010	12.911
57	-11.884	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:ins10t|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.011	12.909
58	-11.884	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:ins10t|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.001	12.919
59	-11.882	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.010	12.908
60	-11.881	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.010	12.907
61	-11.880	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.012	12.904
62	-11.880	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.002	12.914
63	-11.874	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst5|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.010	12.900
64	-11.873	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst9|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.010	12.899
65	-11.872	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.012	12.896
66	-11.872	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst8|DFF_Dev:inst14|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst2|DFF_Dev:inst16|inst	CLK	CLK	1.000	-0.002	12.906
67	-11.866	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst4|DFF_Dev:inst15|inst	Datapath:inst8|REGFILES8:inst2|REG_16BITbdf:inst6|DFF_Dev:inst|inst	CLK	CLK	1.000	-0.003	12.899

10.2/ Mô phỏng waveform định thời mức cổng



11/ Kiểm tra công suất



11.1. PowerPlay Power Analyzer Status

- **Status**: **Successful** - Quá trình phân tích công suất đã hoàn thành thành công.
- **Time**: Thời gian thực hiện phân tích được hiển thị.
- **Quartus II Version**: Phiên bản sử dụng là **13.0.1 Build 232**.
- **Device**: FPGA được sử dụng là **Cyclone II (EP2C35F672C6)**.

11.2. Total Thermal Power Dissipation

- **116.51 mW**: Tổng công suất nhiệt tiêu thụ bởi thiết kế.

11.3. Core Dynamic Thermal Power Dissipation

- **0.00 mW**: Không có công suất động tiêu thụ trong lõi (core dynamic power). Điều này có thể do thiếu dữ liệu về toggle rate hoặc thiết kế không có phần động đáng kể.

11.4. Core Static Thermal Power Dissipation

- **79.94 mW**: Công suất tĩnh tiêu thụ bởi lõi (core static power). Đây là công suất cố định khi FPGA hoạt động mà không liên quan đến chuyển đổi trạng thái tín hiệu.

11.5. I/O Thermal Power Dissipation

- **36.57 mW**: Công suất tiêu thụ liên quan đến các hoạt động ở đầu vào/đầu ra (I/O pins).

11.6. Power Estimation Confidence

- **Low: User provided insufficient toggle rate data**:

➔ Độ tin cậy của ước tính công suất là **thấp**, do thiếu dữ liệu về **toggle rate** (tần suất chuyển đổi tín hiệu trong thiết kế).

UIT

TRƯỜNG ĐẠI HỌC
CÔNG NGHỆ THÔNG TIN