

**ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ
KHOA CÔNG NGHỆ THÔNG TIN**

**HƯỚNG DẪN THỰC HÀNH
KIẾN TRÚC MÁY TÍNH
BẰNG MÔ PHỎNG**

TÁC GIẢ

PGS. TS. Nguyễn Đình Việt (chủ biên)

ThS. Lương Việt Nguyên

KS. Nguyễn Hoài Nam

KS. Nguyễn Đức Thiện

HÀ NỘI - 2010

LỜI NÓI ĐẦU

Máy tính điện tử ra đời vào đầu thập kỷ thứ tư của thế kỷ 20 và đã trở nên hết sức phổ biến trong mọi lĩnh vực hoạt động của con người. Cấu tạo của MTĐT ngày càng tinh vi và phức tạp, bao gồm nhiều thành phần khác nhau. Việc nghiên cứu thiết kế máy tính điện tử là một bài toán khổng lồ và phức tạp, cần không ngừng được cải tiến. Ngày nay người ta thường sử dụng các phần mềm chuyên dụng để thiết kế máy tính, đặc biệt là để thiết kế phần cứng và các thành phần phần mềm ở các mức thấp có chức năng chủ yếu là điều khiển phần cứng.

Giáo trình “Hướng dẫn thực hành Kiến trúc máy tính bằng mô phỏng” này được biên soạn nhằm phục vụ cho môn học “Kiến trúc máy tính” ở khoa Công nghệ Thông tin, trường Đại học Công nghệ, Đại học Quốc gia Hà Nội. Các bài thực hành theo giáo trình này bám sát nội dung cuốn sách “Kiến trúc máy tính” của tác giả Nguyễn Đình Việt. Giáo trình gồm 10 bài thực hành, mỗi bài được thực hiện trong thời gian 2 tiết học tại phòng máy tính, có giáo viên hướng dẫn. Tuy nhiên, sinh viên cũng có thể tự thực hành theo giáo trình.

Sau khi hoàn thành các bài thực hành theo giáo trình này, sinh viên có thể tự thiết kế các mạch logic số từ đơn giản đến tương đối phức tạp, có thể tự xây dựng một máy tính mô phỏng với phần cứng đơn giản và một vi chương trình với chức năng của bộ thông dịch, làm cho chiếc máy tính với phần cứng đơn giản này có khả năng thực hiện các chương trình ở mức “Máy thông thường” tương tự như tất cả các máy tính hiện đại khác.

Giáo trình này mới được biên soạn và sử dụng trong năm học 2010-2011, chắc chắn không tránh khỏi việc vẫn còn thiếu sót. Nhóm tác giả chân thành mong muốn nhận được các ý kiến đóng góp của các thầy, các bạn đồng nghiệp, các bạn sinh viên và bạn đọc về mọi mặt của giáo trình.

Các ý kiến đóng góp xin gửi về địa chỉ:

Nguyễn Đình Việt

Khoa Công nghệ Thông tin,

Trường Đại học Công nghệ, ĐHQGHN

144 Đường Xuân Thủy, Quận Cầu Giấy, Hà Nội

MỤC LỤC

BÀI 1 Học sử dụng Logisim theo trợ giúp – Help	7
1.1. Các yêu cầu và nội dung chính	7
1.2. Cài đặt bộ mô phỏng Logisim	7
1.3. Các bước tạo một mạch logic số đơn giản	7
1.4. Bài tập (theo giáo trình Kiến trúc máy tính)	7
BÀI 2 Học sử dụng Logisim theo trợ giúp – Help (tiếp)	9
2.1. Các yêu cầu và nội dung chính	9
2.2. Sử dụng thư viện của Logisim và thay đổi thuộc tính của các phần tử logic	9
2.3. Tạo và sử dụng các mạch con	9
2.4. Tra cứu theo các menu	9
2.5. Bài tập (theo giáo trình Kiến trúc máy tính)	9
BÀI 3 Học sử dụng Logisim theo trợ giúp – Help (tiếp)	11
3.1. Các yêu cầu và nội dung chính	11
3.2. Tạo và sử dụng các bó dây	11
3.3. Phân tích các mạch tổ hợp (Combinational circuits)	11
3.4. Bài tập (theo giáo trình Kiến trúc máy tính)	11
BÀI 4 Học sử dụng Logisim theo trợ giúp – Help (tiếp)	13
4.1. Các yêu cầu và nội dung chính	13
4.2. Các thành phần của bộ nhớ	13
4.3. Bài tập (theo giáo trình Kiến trúc máy tính)	13
BÀI 5: Xây dựng ALU-16bits	15
5.1. Các yêu cầu và nội dung chính	15
5.2. Phân tích bài toán	15
5.3. Xây dựng ALU 1 bit	16
5.4. Xây dựng ALU 8 bit	17
5.5. Xây dựng ALU 16 bit hoàn chỉnh	17
5.6. Bài tập (không có trong giáo trình Kiến trúc máy tính)	18

BÀI 6: Xây dựng khối 16 thanh ghi.....19

6.1. Các yêu cầu và nội dung chính.....	19
6.2. Phân tích bài toán.....	19
6.3. Xây dựng và mô phỏng khối 16-Register.....	22
6.3.1 Xây dựng khối 16-Register.....	22
6.3.2 Mô phỏng hoạt động của khối 16-Register.....	22
6.4. Bài tập (chủ yếu theo giáo trình Kiến trúc máy tính).....	23

BÀI 7: Xây dựng các thành phần còn lại của đường dữ liệu.....24

7.1. Các yêu cầu và nội dung chính.....	24
7.2. Xây dựng và mô phỏng sự hoạt động của shifter.....	25
7.2.1 Phân tích.....	25
7.2.2 Xây dựng mạch con shifter.....	25
7.2.3 Mô phỏng hoạt động của shifter.....	27
7.3. Xây dựng và mô phỏng sự hoạt động của các thanh ghi chốt A Latch và B Latch...27	
7.3.1 Phân tích.....	27
7.3.2 Xây dựng thanh ghi chốt.....	27
7.3.3 Mô phỏng hoạt động của thanh ghi chốt (Latch Register).....	28
7.4. Xây dựng và mô phỏng sự hoạt động của AMUX.....	28
7.5 Xây dựng và mô phỏng sự hoạt động của MAR.....	28
7.5.1 Phân tích.....	28
7.5.2 Xây dựng thanh ghi MAR.....	28
7.5.3 Mô phỏng hoạt động của MAR.....	29
7.6 Xây dựng và mô phỏng sự hoạt động của MBR.....	29
7.6.1 Phân tích.....	29
7.6.2 Xây dựng thanh ghi MBR.....	30
7.6.3 Mô phỏng hoạt động của MBR.....	31
7.7. Bài tập.....	32

BÀI 8: Xây dựng Control Store, MIR và Micro Seq của đơn vị điều khiển.....33

8.1. Các yêu cầu và nội dung chính.....	33
8.2. Xây dựng và mô phỏng sự hoạt động của Control Store.....	34
8.2.1 Phân tích.....	34
8.2.2. Xây dựng và nạp nội dung cho Control Store.....	34

8.2.3. Mô phỏng sự hoạt động của Control Store.....	40
8.2. Xây dựng và mô phỏng sự hoạt động của MIR.....	40
8.2.1 Phân tích.....	40
8.2.2 Xây dựng thanh ghi MIR.....	40
8.2.3 Mô phỏng hoạt động của thanh ghi MIR.....	41
8.3. Xây dựng và mô phỏng sự hoạt động của Micro Seq.....	41
8.3.1 Phân tích.....	41
8.3.2 Xây dựng mạch con Micro Seq.....	41
8.3.3 Mô phỏng hoạt động của Micro Seq.....	42
8.4. Bài tập (thầy Việt mới bổ sung 3/11/2010).....	43
 BÀI 9: Xây dựng các thành phần còn lại của đường điều khiển.....	 45
9.1. Các yêu cầu và nội dung chính.....	45
9.2. Xây dựng và mô phỏng hoạt động của thanh ghi MPC.....	45
9.2.1 Phân tích.....	45
9.2.2 Xây dựng thanh ghi MPC.....	45
9.2.3 Mô phỏng hoạt động của thanh ghi MPC.....	46
9.3. Xây dựng và mô phỏng hoạt động của đơn vị Increment.....	46
9.3.1 Phân tích.....	46
9.3.2 Xây dựng mạch con Increment.....	46
9.3.3 Mô phỏng hoạt động của Increment.....	47
9.4. Xây dựng và mô phỏng hoạt động của đơn vị Mmux.....	47
9.5. Xây dựng và mô phỏng hoạt động của decoder.....	47
9.5.1 Phân tích.....	47
9.5.2 Xây dựng mạch con làm A, B decoder và mô phỏng sự hoạt động của nó.....	47
9.5.3 Xây dựng mạch con làm C decoder và mô phỏng sự hoạt động của nó.....	48
9.6. Xây dựng và mô phỏng hoạt động của Clock.....	49
9.6.1 Phân tích.....	49
9.6.2 Xây dựng mạch con làm đơn vị Clock.....	50
9.6.3 Mô phỏng sự hoạt động của Clock.....	51
9.7. Bài tập (thầy Việt mới bổ sung 5/11/2010).....	52
 BÀI 10: Xây dựng vi kiến trúc đầy đủ.....	 53
10.1. Các yêu cầu và nội dung chính.....	53
10.2. Lắp ghép vi kiến trúc đầy đủ.....	53

10.3. Bổ sung bộ nhớ chính và lắp ghép với vi kiến trúc.....	54
10.4. Kiểm tra hệ thống máy tính (New !!!).....	55
10.4.1 Chương trình kiểm tra thứ nhất - test-01-addition.txt.....	55
10.4.2 Chương trình kiểm tra thứ hai: under construction.....	59
PHỤ LỤC A: HƯỚNG DẪN SỬ DỤNG LOGISIM.....	60
1. CHỈ DẪN CHO NGƯỜI MỚI SỬ DỤNG.....	61
1.1 Bước 0: Tự định hướng chính bản thân bạn.....	62
1.2 Bước 1: Thêm các cổng.....	63
1.3 Bước 2: Thêm dây nối.....	65
1.4 Bước 3: Chèn text.....	66
1.5 Bước 4: Kiểm tra mạch.....	66
2. THƯ VIỆN VÀ CÁC THUỘC TÍNH.....	68
2.1 Explorer pane.....	68
2.2 Bảng thuộc tính.....	70
2.3 Thuộc tính của công cụ.....	71
3. CÁC MẠCH CON.....	73
3.1 Tạo mới các mạch.....	73
3.2 Sử dụng mạch con.....	74
3.3 Sửa lỗi các mạch con.....	76
3.4 Thư viện Logisim.....	77
4. DÂY.....	77
4.1 Tạo các bó.....	77
4.2 Dụng cụ rẽ nhánh.....	79
4.3 Màu của dây.....	80
5. PHÂN TÍCH TỔNG HỢP.....	81
5.1 Cách mở bảng phân tích tổng hợp.....	81
5.2 Sửa đổi bảng chân lí.....	82
5.3 Tạo các biểu thức.....	84
5.4 Tạo mạch.....	85
6. GIỚI THIỆU CÁC MENU.....	86
6.1 Menu File.....	86
6.2 Menu Edit.....	87
6.3 Menu Project.....	87
6.4 Menu Simulate.....	88
6.5 Menu Window.....	89

6.6 Menu Help.....	90
7. CÁC THÀNH PHẦN CỦA BỘ NHỚ.....	90
7.1 Truy cập bộ nhớ.....	90
7.2 Pop-up menus and files.....	91
7.3 Hex editor.....	92
8. LOGGING.....	92
8.1 Thẻ lựa chọn (the selection tab).....	93
8.2 Thẻ table.....	95
8.3 Thẻ tệp.....	95
9. APPLICATION PREFERENCES.....	97
9.1 Thẻ Template.....	97
9.2 Thẻ International.....	98
9.3 Thẻ Experimental.....	99
9.4 Tùy chọn Command-line.....	100
10. PROJECT OPTION.....	101
10.1 Thẻ Canvas.....	102
10.2 Thẻ Simulation.....	104
10.3 Thẻ Toolbar.....	105
10.4 Thẻ Mouse.....	106
11. SỬ TRUYỀN GIÁ TRỊ.....	107
11.1 Sự trễ của cổng.....	107
11.2 Lỗi dao động.....	108
11.3 Hạn chế.....	109
PHỤ LỤC B: ĐÁP ÁN MỘT SỐ BÀI TẬP.....	110
Bài số 5.....	110

BÀI 1 Học sử dụng Logisim theo trợ giúp – Help

1.1. Các yêu cầu và nội dung chính

- Biết cách cài đặt và cho chạy chương trình mô phỏng
- Biết cách sử dụng trợ giúp (Help) của logisim
- Biết các bước tạo một mạch logic số đơn giản
- Biết cách lưu giữ (save) các mạch điện đã tạo ra vào file và mở (open)

1.2. Cài đặt bộ mô phỏng Logisim

Có thể download các bản logisim cho Windows, Linux, MacOS tại địa chỉ sau:

<http://sourceforge.net/projects/circuit/>

Bản cho Windows chỉ gồm có 1 file: logisim-win-2.3.5.exe (3.5 MB).

Bản cho MacOS là file: logisim-macosx-2.3.5.tar.gz.

Bản cho Linux (và nhiều OS khác) là: logisim-2.3.5.jar.

Chạy logisim trong môi trường Windows: Nếu máy tính đã cài đặt java, thì có thể chạy logisim ngay, như chạy các file chương trình khả thi bình thường khác. Nếu chưa, thì phải cài java rồi mới chạy được logisim.

Chạy logisim trong môi trường linux (ubuntu): Cần phải cài đặt java trước mới có thể chạy được logisim. Chúng tôi đã cài sun-java6-jre trong Ubuntu 10.04 và Ubuntu Remix và không gặp trở ngại nào. Có thể dùng Synaptic Package Manager để tìm và cài bản java này.

Chạy logisim như sau: `java -jar logisim-2.3.5.jar [Enter]`

1.3. Các bước tạo một mạch logic số đơn giản

Thực hiện các bước theo “Help” của logisim (Help \ Tutorial \ Beginner’s Tutorial). Trong “Bài thực hành số 1”, sinh viên cần đọc và làm theo 5 bước (step 0 – step 4) của mục “Beginner’s Tutorial”.

1.4. Bài tập (theo giáo trình Kiến trúc máy tính)

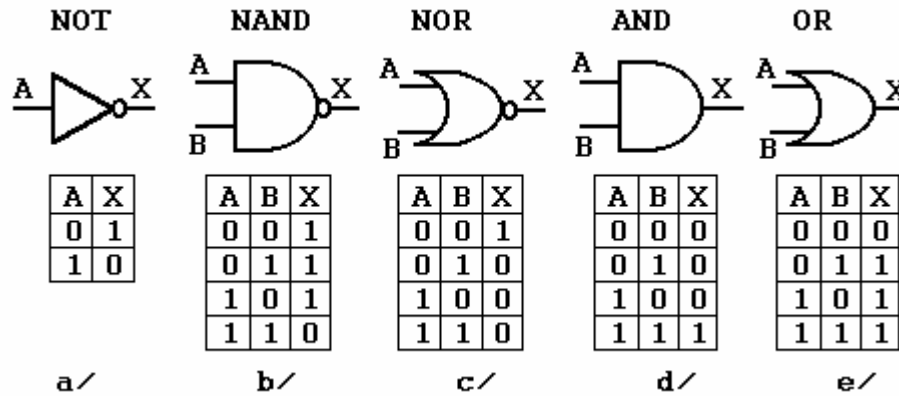
Chú ý khi làm các bài tập của chương này:

- Đầu ra của 1 cổng có thể nối với đầu vào của 1 cổng khác. Đối với các đầu vào/ra không nối với các đầu vào/ra của các cổng khác, mỗi đầu vào cần nối với một phần tử “input” và mỗi đầu ra cần nối với phần tử “output”.
- Để thử lại chức năng, sử dụng công cụ “poke tool” trỏ vào các phần tử “input” và kích nút chuột bên trái để đảo giá trị của input, đồng thời quan sát giá trị của phần tử “output”.
- Sau khi đã tạo và thử các mạch logic, cần “save” vào bộ nhớ ngoài (ổ đĩa cứng, ổ đĩa USB...), sau đó thử “open” để xem lại. Tên file lưu mạch điện theo các hình vẽ trong giáo

trình nên đặt theo tên hình vẽ cho dễ nhớ. Thí dụ, với mạch điện trên hình 3-02, tên file nên đặt là “hinh3-02.circ”, hoặc “figure3-02.circ”.

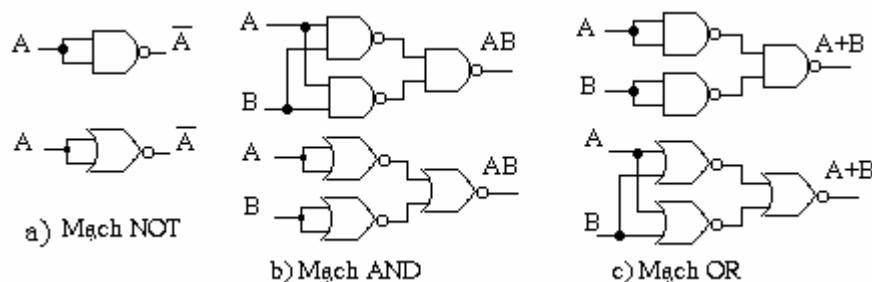
Bài tập:

1. Mô phỏng sự hoạt động của các cổng logic đơn giản nhất trên hình 3-02.



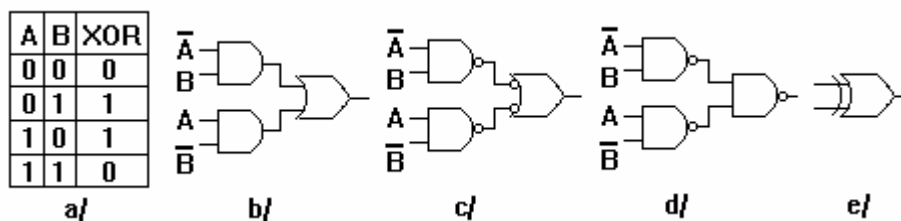
Hình 3-02 Ký hiệu và hành trạng chức năng của 5 cổng cơ bản

2. Tạo và mô phỏng hoạt động của các mạch logic số như trên Hình 3-03, kiểm tra sự tương đương của chúng.



Hình 3-03 Các mạch logic có chức năng tương đương

3. Tạo và mô phỏng hoạt động của các mạch logic số như trên Hình 3-06.



Hình 3-06 Bảng chân lý của hàm XOR và các mạch điện thực hiện hàm này

BÀI 2 Học sử dụng Logisim theo trợ giúp – Help (tiếp)

2.1. Các yêu cầu và nội dung chính

- Biết sử dụng thư viện của logisim
- Biết bổ sung các mạch con (subcircuit) mà mình tạo ra vào thư viện
- Tạo và mô phỏng một số mạch điện có trong giáo trình Kiến trúc máy tính.

2.2. Sử dụng thư viện của Logisim và thay đổi thuộc tính của các phần tử logic

Thực hành trên máy theo “Help” của logisim, xem mục **Libraries and Attributes**.

2.3. Tạo và sử dụng các mạch con

Thực hành trên máy theo “Help” của logisim, xem mục **Subcircuits**.

2.4. Tra cứu theo các menu

Thực hành trên máy theo “Help” của logisim, xem mục **Menu Reference**.

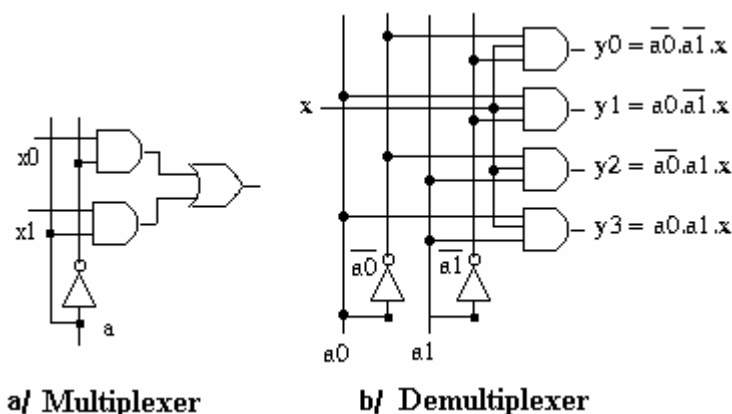
2.5. Bài tập (theo giáo trình Kiến trúc máy tính)

Chú ý:

- Các mạch logic số sau đây đã có trong thư viện của logisim, nhưng sinh viên vẫn cần tạo ra chúng bằng các phần tử logic cơ bản để hiểu sâu về sự hoạt động của chúng.
- Sinh viên nên lấy từ thư viện một phần tử có chức năng tương tự chức năng của mạch logic số mà mình vừa tạo ra để so sánh chức năng của chúng.
- Sau này, khi xây dựng “Micro Architecture” (hình 4-07, hình 4-09), chúng ta sẽ sử dụng các phần tử đã có trong thư viện.

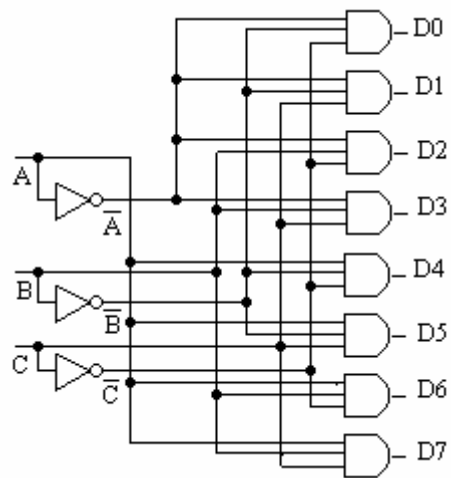
Bài tập:

1. Tạo và mô phỏng hoạt động của multiplexer và demultiplexer trên Hình 3-07.



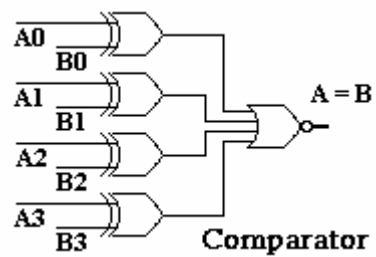
Hình 3-07 Bộ dồn kênh và bộ phân kênh

2. Tạo và mô phỏng hoạt động của bộ decoder “3 to 8” trên Hình 3-08.



Hình 3-08 Mạch giải mã “3 to 8”

3. Tạo và mô phỏng hoạt động của bộ so sánh 4 bit trên Hình 3-09.



Hình 3-09 Bộ so sánh 4 bit

BÀI 3 Học sử dụng Logisim theo trợ giúp – Help (tiếp)

3.1. Các yêu cầu và nội dung chính

- Biết tạo và sử dụng các bó dây (wire bundles)
- Biết phân tích, tạo và mô phỏng các mạch tổ hợp (combinational circuit)
- Tạo và mô phỏng được các mạch tổ hợp có trong giáo trình Kiến trúc máy tính.

3.2. Tạo và sử dụng các bó dây

Thực hành trên máy theo “Help” của logisim, xem mục **Wire bundles**.

3.3. Phân tích các mạch tổ hợp (Combinational circuits)

Thực hành trên máy theo “Help” của logisim, xem mục **Combinational analysis**.

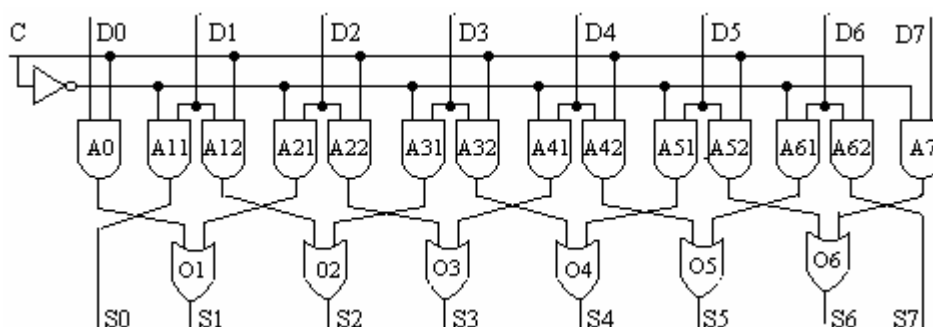
3.4. Bài tập (theo giáo trình Kiến trúc máy tính)

Chú ý:

- Một số mạch logic số sau đây đã có trong thư viện của logisim, nhưng sinh viên vẫn cần tạo ra chúng bằng các phần tử logic cơ bản để hiểu sâu về sự hoạt động của chúng.
- Sinh viên nên lấy từ thư viện một phần tử có chức năng tương tự chức năng của mạch logic số mà mình vừa tạo ra để so sánh chức năng của chúng.
- Sau này, khi xây dựng “Micro Architecture” (hình 4-07, hình 4-09), chúng ta sẽ sử dụng các phần tử đã có trong thư viện.

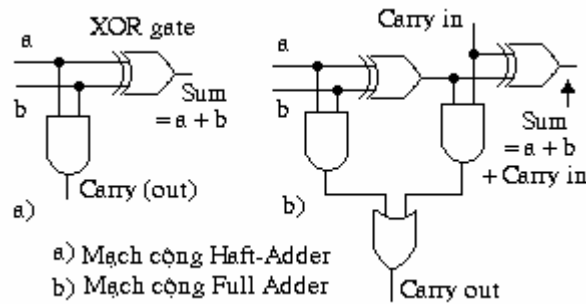
Bài tập:

1. Tạo và mô phỏng hoạt động của bộ dịch (shifter) trên Hình 3-11. Thực tập sử dụng các phần tử “Shifter” và “Shift Register” có trong thư viện của logisim (trong các nhóm “Arithmetic” và “Memory”).



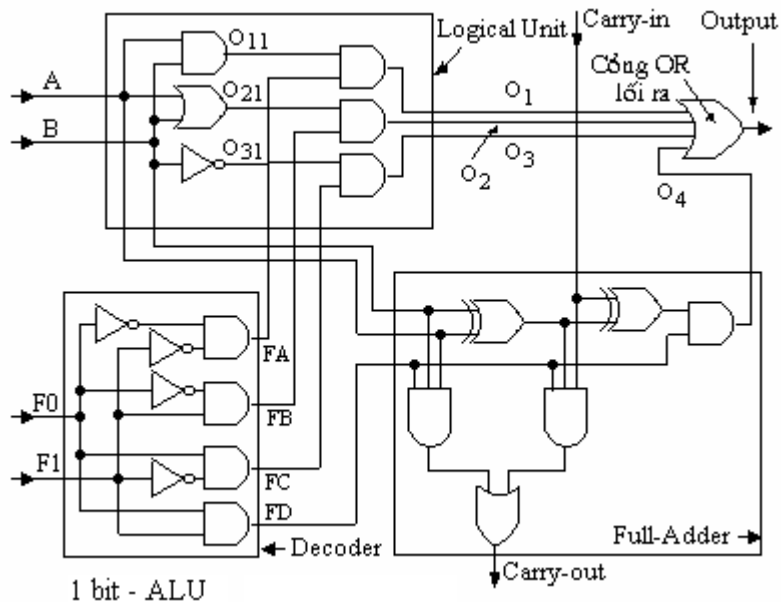
Hình 3-11 Bộ dịch 1 bit sang trái/phải

2. Tạo và mô phỏng hoạt động của các bộ cộng (Haft-Adder và Full-Adder) trên Hình 3-12. Thực tập sử dụng phần tử “Adder” có trong thư viện của logisim (trong nhóm “Arithmetic”).



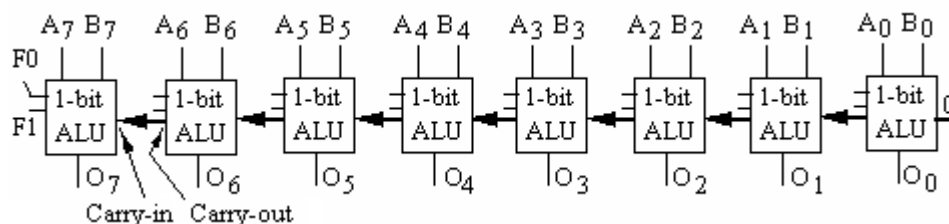
Hình 3-12 Bộ cộng

3. Tạo và mô phỏng hoạt động của bộ số học là logic trên Hình 3-13 (1-bit ALU). Lưu (save) vào file “ALU-1bit.circ” để sử dụng cho bài tập sau.



Hình 3-13 Bộ Số học và Logic (ALU) 1 bit

4. Tạo và mô phỏng hoạt động của một ALU 8 bit từ các mạch con là ALU 1 bit như trên hình 3-14 (sử dụng file ALU-1bit.circ).



Hình 3-14 ALU 8-bit được tạo bởi “8 mảnh” ALU 1-bit

BÀI 4 Học sử dụng Logisim theo trợ giúp – Help (tiếp)

4.1. Các yêu cầu và nội dung chính

- Biết được khả năng mô phỏng bộ nhớ của logisim.
- Biết tạo và mô phỏng sự hoạt động của các phần tử nhớ cơ bản.
- Biết sử dụng các phần tử nhớ có trong thư viện của logisim.

4.2. Các thành phần của bộ nhớ

Thực hành trên máy theo “Help” của logisim, xem mục **Memory**.

Trong giờ thực hành có giáo viên hướng dẫn, nên tập trung thực hành sử dụng các thành phần bộ nhớ sau:

- D Flip-Flop
- S-R Flip-Flop
- Register

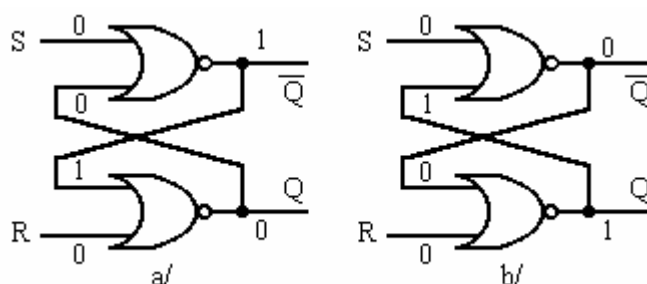
4.3. Bài tập (theo giáo trình Kiến trúc máy tính)

Chú ý:

- Một số mạch logic số sau đây đã có trong thư viện của logisim, nhưng sinh viên vẫn cần tạo ra chúng bằng các phần tử logic cơ bản để hiểu sâu về sự hoạt động của chúng.
- Sinh viên nên lấy từ thư viện một phần tử có chức năng tương tự chức năng của mạch logic số mà mình vừa tạo ra để so sánh chức năng của chúng.
- Sau này, khi xây dựng “Micro Architecture” (hình 4-07, hình 4-09), chúng ta sẽ sử dụng các phần tử đã có trong thư viện.

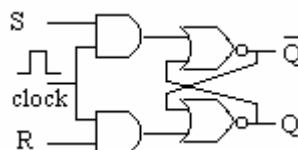
Bài tập:

1. Tạo và mô phỏng sự hoạt động của Thanh ghi chốt đơn giản nhất - thanh ghi chốt RS, theo Hình 3-16; sau đó lặp lại việc này nhưng sử dụng các phần tử NAND thay cho các phần tử NOR.



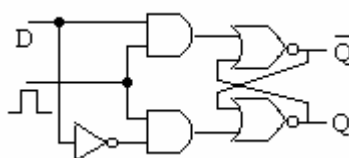
Hình 3-16 a/ Thanh ghi chốt dùng cổng NOR ở trạng thái 0; b/ Thanh ghi chốt dùng cổng NOR ở trạng thái 1

2. Tạo và mô phỏng sự hoạt động của Thanh ghi chốt RS hoạt động theo nhịp xung đồng hồ theo Hình 3-17. Chú ý thử tín hiệu đồng hồ (clock) theo 2 cách: 1/ sử dụng phần tử “input” và dùng phần tử “poke tool” để thay đổi giá trị của “input”; 2/ sử dụng đơn vị tạo tín hiệu clock thuộc nhóm “Base” trong thư viện của logisim.



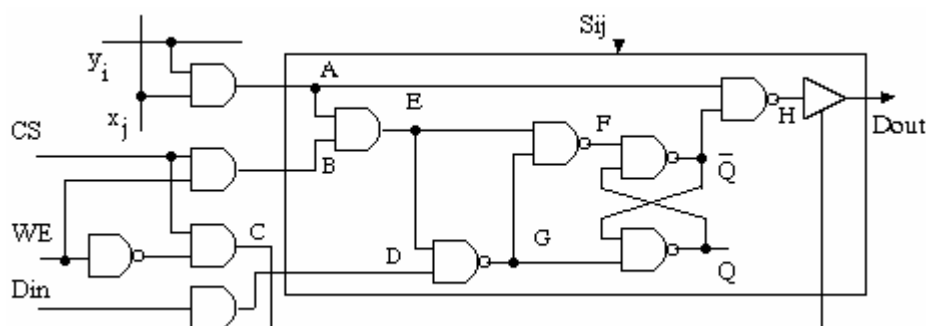
Hình 3-17 Thanh ghi chốt RS hoạt động theo nhịp xung đồng hồ

3. Tạo và mô phỏng sự hoạt động của Thanh ghi chốt D hoạt động theo nhịp xung đồng hồ (Hình 3-18).



Hình 3-18 Thanh ghi chốt D hoạt động theo nhịp xung đồng hồ

4. Tạo và mô phỏng sự hoạt động của Phần tử nhớ 1 bit của bộ nhớ RAM tĩnh (SRAM) như trên Hình 3-19.



Hình 3-19 Mạch điện của một phần tử nhớ RAM tĩnh 1 bit Sij

Chú ý:

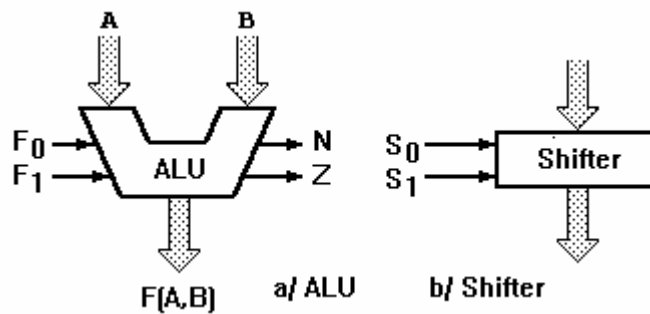
- Sử dụng phần tử “Controlled Buffer” thuộc nhóm “Gates” trong thư viện làm bộ đệm 3 trạng thái có đầu ra nối với “Dout”.
- Sử dụng phần tử “Buffer” thuộc nhóm “Gates” trong thư viện làm bộ đệm có đầu vào nối với “Din”.

5. Tự đọc các mục Logging; Application Referenes và Projection trong Help của logisim.

BÀI 5: Xây dựng ALU-16bits

5.1. Các yêu cầu và nội dung chính

Xây dựng và mô phỏng sự hoạt động của đơn vị ALU của vi kiến trúc được mô tả trên hình 4-09. ALU này được mô tả kỹ tại mục “4.1.4 Đơn vị Số học-Logic và Bộ dịch” và được minh hoạ trên hình 4-04.



Hình 4-04 a/ Ký hiệu một ALU; b/ Ký hiệu một Shifter

ALU này thực hiện một trong 4 phép tính trên toán hạng 16 bit, theo mã phép toán chứa trong trường ALU (gồm 2 bit F0 và F1) của thanh ghi MIR chứa vi chỉ thị đang thi hành:

- ALU (= F0 F1) = 00: Cộng 2 toán hạng 16 bits ở 2 đầu vào, số nhớ đưa vào (carry-in) bằng 0, số nhớ đưa ra được sử dụng để tạo thành các tín hiệu trạng thái N và Z.
- ALU (= F0 F1) = 01: AND 2 toán hạng 16 bits ở 2 đầu vào.
- ALU (= F0 F1) = 10: Cho toán hạng ở đầu vào bên trái (đầu vào A) đi ra đầu ra, không thay đổi; các bit trạng thái N và Z được thiết lập phụ thuộc vào kết quả ở đầu ra.
- ALU (= F0 F1) = 11: Đảo toán hạng ở đầu vào bên trái (đầu vào A) và đưa ra đầu ra.

5.2. Phân tích bài toán

- ALU này thực hiện các chức năng không hoàn toàn giống với ALU đã học và được mô tả trên hình 3-13. Tuy nhiên, sự khác nhau là không nhiều. Chúng ta chỉ phải tạo ra một mạch điện thực sự mới để sinh ra tín hiệu trạng thái N và Z, theo nguyên tắc sau:
 - N - chỉ báo rằng kết quả ra của ALU là âm (Negative). Thực chất, bit N chỉ là copy của bit có bậc cao nhất của kết quả đưa ra.
 - Z - chỉ báo rằng kết quả ra của ALU bằng không (Zero). Thực chất, bit Z là NOR của tất cả các bit của kết quả đưa ra.
 - Khi xây dựng ALU-16bits:
 - Carry-in cho bộ cộng 2 bit bậc 0 được cho bằng 0.
 - Carry-out của bộ cộng 2 bit bậc cao nhất (15) không cần quan tâm (để mạch điện và việc phân tích nó không quá phức tạp).

- Để việc xây dựng mạch điện theo kiểu “mô-đun” hóa, chúng ta sẽ xây dựng mạch ALU 1 bit trước, lưu vào file ALU-1bit.circ; sau đó sử dụng nó để xây dựng ALU-8bits; cuối cùng xây dựng ALU-16bits theo yêu cầu từ 2 ALU 8 bit và một số phần tử mạch điện khác, thí dụ mạch tạo các tín hiệu trạng thái N và Z.

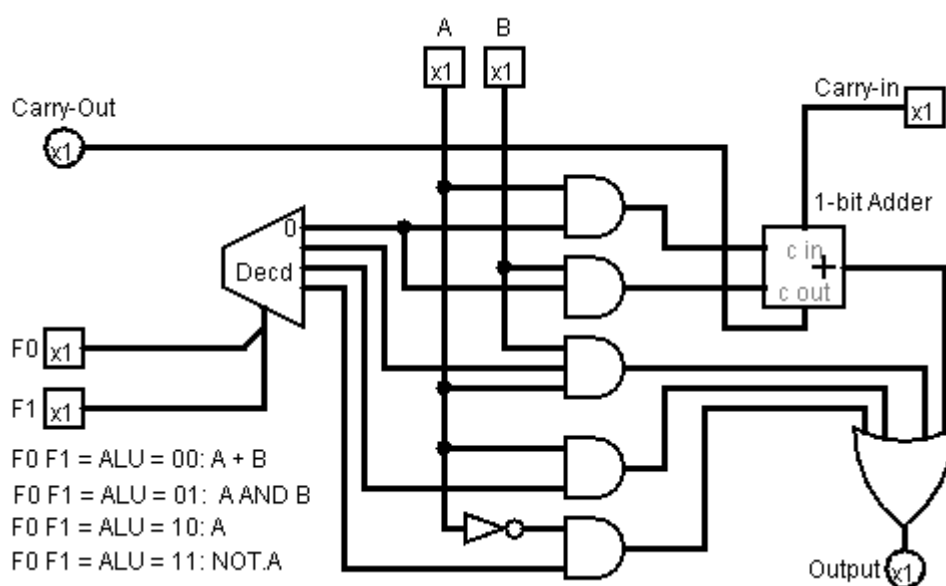
Chú ý: Chúng ta có thể xây dựng 1 ALU-16bits từ 16 ALU-1bit, thậm chí từ các phần tử logic cơ bản. Tuy nhiên, nếu làm như vậy sẽ không thể nhìn thấy toàn bộ mạch điện trên màn hình.

5.3. Xây dựng ALU 1 bit

- Chạy logisim; Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là ALU-1bit, không cần đánh vào tên mở rộng luôn được đặt mặc định là “.circ”. Trong cửa sổ con ở góc trên bên trái, ngoài tên main, logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là ALU-1bit.
- Lưu (save) mạch điện mà chúng ta tạo ra: vào menu File \ Save | Save As. Khi logisim hỏi tên mà chúng ta muốn đặt, hãy đặt tên là: Micro-Architecture-SVxx.circ. Trong đó, SVxx là mã số sinh viên và “.circ” là tên mở rộng (mặc định). Trong thời gian thực hành, hoặc khi kết thúc thực hành, nên thường xuyên “save”.
- Xây dựng ALU 1 bit như ở hình dưới đây.

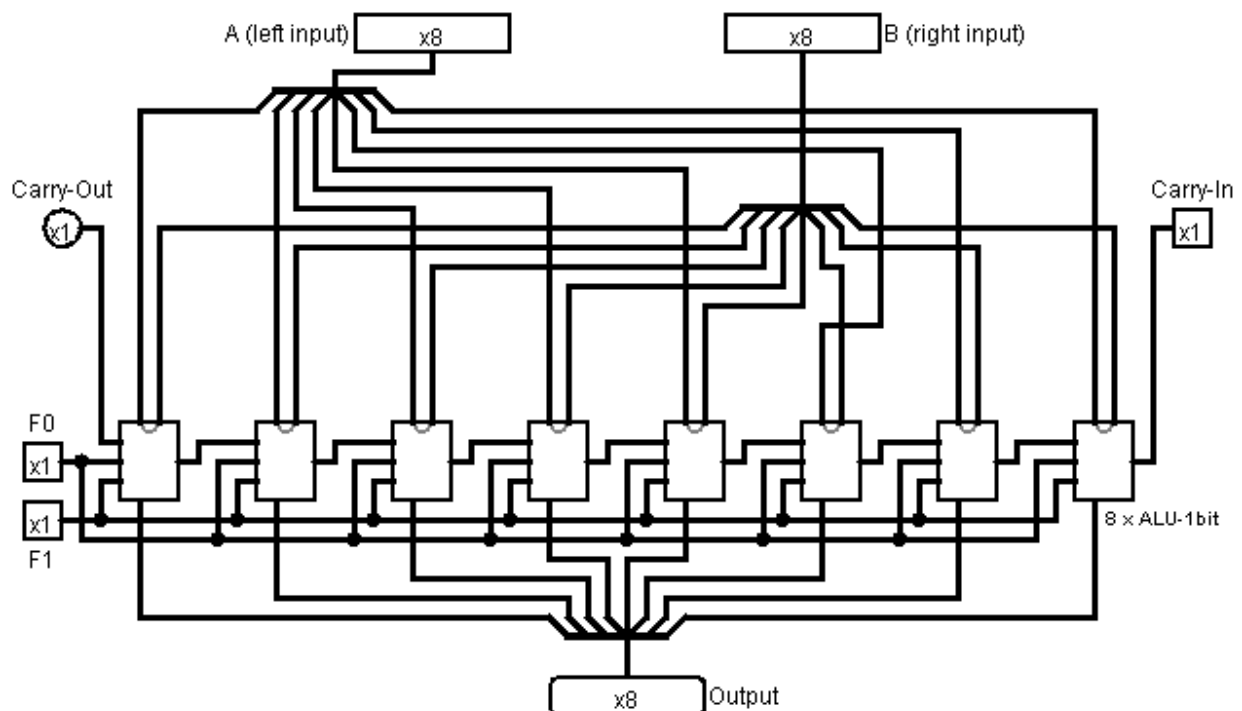
Chú ý:

1. Mạch con này sẽ được chúng ta sử dụng để xây dựng ALU-8bit.
 2. Các phần tử input/output của mạch con nằm ở hướng nào (North, South, East, West) của mạch điện này, thì khi chúng ta sử dụng mạch con này như một phần tử của thư viện, các đầu (“chân”) input/output của phần tử sẽ có cùng hướng (North, South, East, West).
- Kiểm tra sự thực hiện 4 chức năng của ALU bằng cách dùng phần tử “poke tool” để thiết lập các giá trị khác nhau của các đầu vào: F0, F1, A, B và Carry-in; đồng thời quan sát giá trị trên các đầu ra: Output và Carry-Out.



5.4. Xây dựng ALU 8 bit

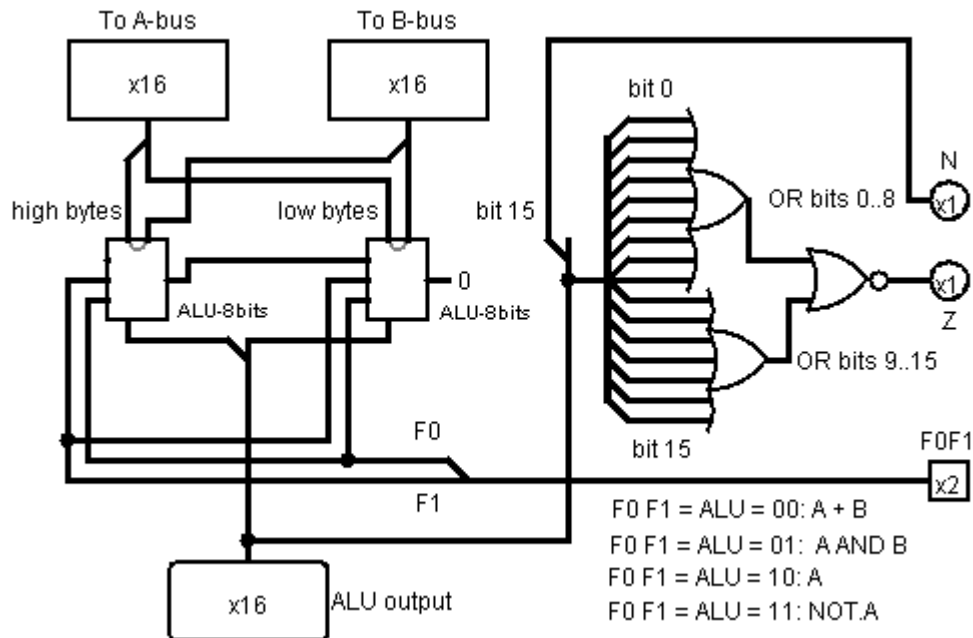
- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là ALU-8bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và ALU-1bit, logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là ALU-8bits.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con ALU-8bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Xây dựng ALU-8bits như ở hình dưới đây. Chú ý: hãy sử dụng các phần tử ALU-1bit do chúng ta tạo ra. Việc này không khác gì so với việc sử dụng các phần tử có sẵn trong thư viện của logisim.
- Kiểm tra sự thực hiện 4 chức năng của ALU bằng cách dùng phần tử “poke tool” để thiết lập các giá trị khác nhau của các đầu vào: F0, F1, A, B và Carry-in; đồng thời quan sát giá trị trên các đầu ra: Output và Carry-Out.



5.5. Xây dựng ALU 16 bit hoàn chỉnh

- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là ALU-16bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main, ALU-1bit, ALU-8bits, logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là ALU-16bits.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con ALU-16bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.

- Xây dựng ALU-16bits như ở hình dưới đây. Chú ý: hãy sử dụng các phần tử ALU-8bits do chúng ta tạo ra.
- Kiểm tra sự thực hiện 4 chức năng của ALU bằng cách dùng phần tử “poke tool” để thiết lập các giá trị khác nhau của các đầu vào: F0, F1, A, B và Carry-in; đồng thời quan sát giá trị trên các đầu ra: Output và Carry-Out.



5.6. Bài tập (không có trong giáo trình Kiến trúc máy tính)

Hãy sử dụng các các phần tử “Hex Digit Display” và “Splitter” để đọc được dưới dạng số Hexa các giá trị đưa vào ALU và kết quả nhận được ở đầu ra ALU.

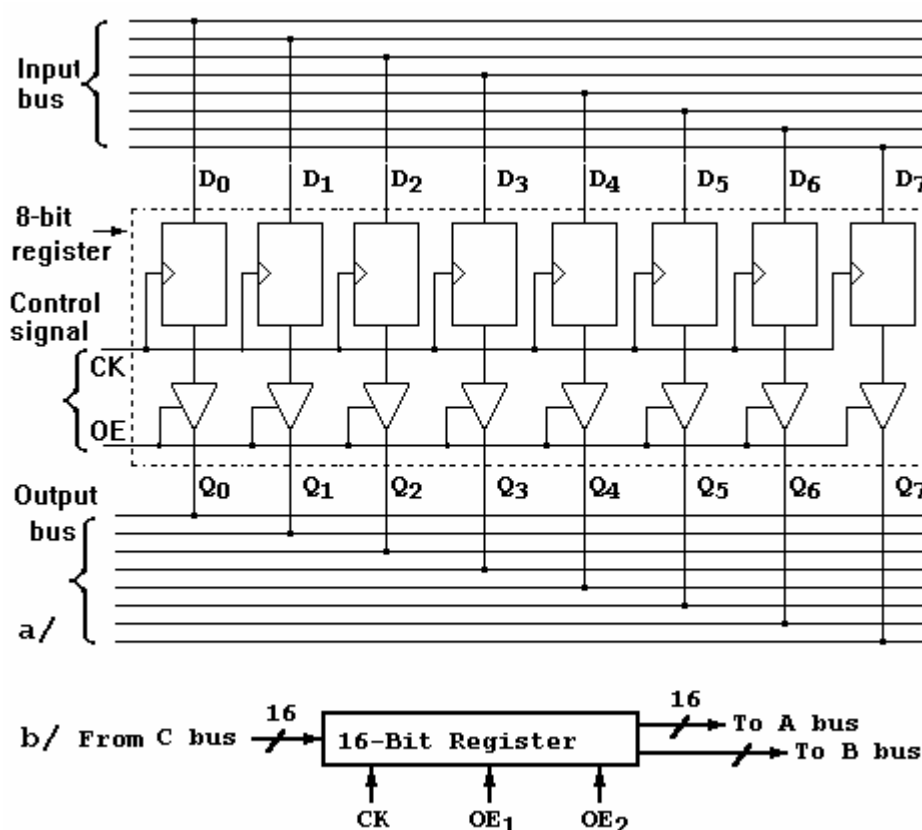
BÀI 6: Xây dựng khối 16 thanh ghi

6.1. Các yêu cầu và nội dung chính

- Xây dựng và mô phỏng sự hoạt động của khối các thanh ghi “16 Registers” của vi kiến trúc được mô tả trên hình 4-09. Khối này gồm 16 thanh ghi, mỗi thanh ghi có kích thước 16 bits, thứ tự các thanh ghi từ 0..15 là: PC, AC, SP, IR, TIR, 0, +1, -1, AMASK, SMASK, A, B, C, D, E, F.
- Mỗi một trong 16 thanh ghi nói trên có thể được điều khiển để nhận dữ liệu từ Bus-C bằng tín hiệu từ trường ENC trong thanh ghi MIR, việc nhận dữ liệu từ Bus-C chỉ có thể xảy ra trong chu kỳ con thứ 4 của tín hiệu đồng hồ. Việc chọn 1 trong 16 thanh ghi thực hiện bằng “C decoder”, còn đầu vào 4 bit của “C decoder” nối với trường C của thanh ghi MIR.
- Mỗi một trong 16 thanh ghi nói trên có thể được điều khiển để đổ nội dung của nó ra Bus-A, hoặc Bus-B, hoặc đồng thời ra cả Bus-A và Bus-B.

6.2. Phân tích bài toán

Cách tạo khối các thanh ghi này được mô tả tại mục “4.1.1 Thanh ghi” và mục “4.1.2 Bus” và được minh họa trên hình 4-02. Chú ý rằng hình 4-02a này mô tả các tạo 1 thanh ghi kích thước 8 bits từ 8 thanh ghi 1 bit; hình 4-02b mô tả các bus và các tín hiệu nối với thanh ghi 16 bits.



Hình 4-02 a/ Thanh ghi 8 bit nối với một bus vào (input bus) và một bus ra (output bus).
b/ Ký hiệu của một thanh ghi 16 bit với một bus vào và hai bus ra

Trong bài thực hành này chúng ta sẽ sử dụng phần tử “register” thuộc nhóm “Memory” trong thư viện của logisim. Một số thuộc tính của phần tử này có thể thay đổi được thông qua cửa sổ “Attribute Table” ở góc dưới bên trái cửa sổ chính của logisim. Ý nghĩa của các thuộc tính như sau:

Data bits: từ 1 đến 32, chúng ta chọn bằng 16.

Trigger: tín hiệu đồng hồ kích hoạt phần tử nhớ (nhận input hoặc đưa nội dung ra output) (đầu vào có ghi đầu mũi tên), có thể chọn 1 trong 4 tín hiệu:

- Rising Edge: phần tử nhớ bị kích hoạt bằng sườn lên (dương) của xung kích
- Falling Edge: phần tử nhớ bị kích hoạt bằng sườn xuống (âm) của xung kích
- High Level: phần tử nhớ bị kích hoạt bằng mức cao của xung kích
- Low Level: phần tử nhớ bị kích hoạt bằng mức thấp của xung kích.

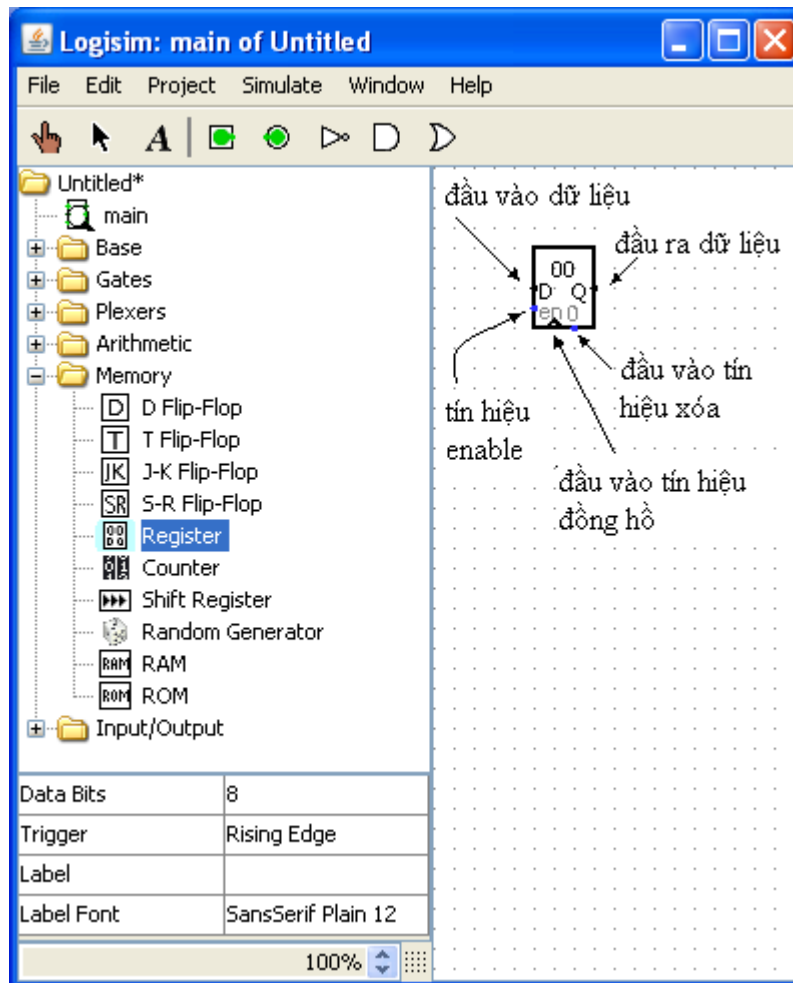
Chúng ta chọn “Rising Edge”, vì chúng ta muốn các thanh ghi được kích hoạt ngay ở sườn lên (đầu) của xung đồng hồ.

Label: nhãn mà chúng ta muốn đặt cho phần tử nhớ. Chúng ta có thể đặt nhãn khác nhau cho các thanh ghi khác nhau, thí dụ: thanh ghi có đầu “en” nối với đầu ra 0 của decoder nên gán nhãn là PC, tương tự: 1: AC, 2: SP, ...

Label Font: font chữ và cỡ chữ của nhãn; Chúng ta cần chọn sao cho đẹp, chữ không đè lên các nét của hình vẽ.

Các đầu vào/ra của phần tử nhớ:

- D (Data bits): đầu vào dữ liệu (1..32 bits).
- Q (đầu ra dữ liệu): có độ rộng (số bit) bằng đầu vào D.
- “^”: đầu vào (1 bit) của tín hiệu đồng hồ.
- en (Enable): tín hiệu mở chip chứa phần tử nhớ, tương tự tín hiệu CS (Chip Select) hoặc CE (Chip Enable) mà chúng ta đã gặp khi phân tích một số mạch điện trong “Chương 3 Mức logic số”.
- “0”: đầu vào tín hiệu xóa (đặt giá trị 0) nội dung phần tử nhớ.



Sau khi biết được các thuộc tính và các đầu vào/ra của phần tử nhớ “Register“, chúng ta sẽ thực hiện xây dựng khối 16 thanh ghi theo cách như sau (Việc phân tích cần đối chiếu với một số hình vẽ trong giáo trình Kiến trúc máy tính):

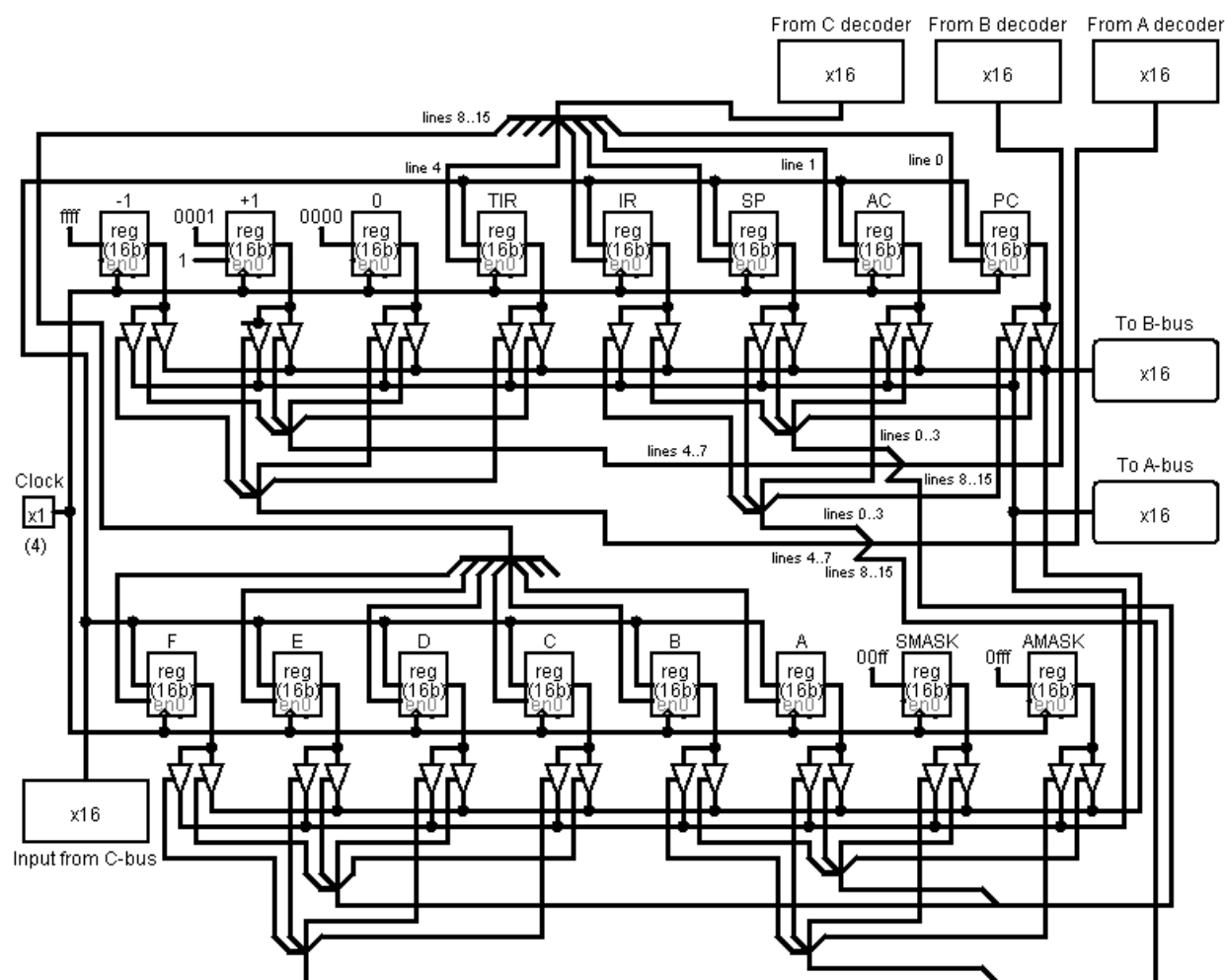
- Đầu vào D (16 dây) của cả 16 thanh ghi nối chung lại với nhau (các đường truyền bit 0 nối với nhau, các đường truyền bit 1 nối với nhau,...) và sẽ nối với Bus-C thông qua phần tử “input“ 16 bit (Xem Hình 4-02 và Hình 4-07).
- Đầu ra Q (16 dây) của mỗi một trong số 16 phần tử “Register“ được nối với 2 đầu vào của 2 bộ đệm 3 trạng thái (phần tử “Controlled buffer“ thuộc nhóm “Gates“), để có thể được điều khiển để đổ dữ liệu ra hoặc Bus-A, hoặc Bus-B hoặc đồng thời ra cả 2 Bus-A và Bus-B. 16 đầu ra của 16 bộ đệm 3 trạng thái thứ nhất nối với nhau và nối với một phần tử “output“ để đổ dữ liệu ra Bus-A; 16 đầu ra của 16 bộ đệm 3 trạng thái thứ hai nối với nhau và nối với một phần tử “output“ để đổ dữ liệu ra Bus-B. Phần tử “Controlled buffer“ có đầu ra nối với Bus-A được điều khiển bởi tín hiệu từ một trong các đầu ra của A-decoder. Tương tự như vậy, phần tử “Controlled buffer“ có đầu ra nối với Bus-B được điều khiển bởi tín hiệu từ một trong các đầu ra của B-decoder (Xem Hình 4-02 và Hình 4-07).
- Đầu vào tín hiệu đồng hồ “^” của tất cả 16 thanh ghi được nối với nhau (Xem Hình 4-02).
- Mỗi đầu vào en (Enable) của một trong 16 phần tử “register“ nối với một đầu ra của C-decoder. Như vậy C-decoder sẽ chọn lấy 1 trong 16 thanh ghi để nhận dữ liệu từ Bus-C vào.

- Đầu vào tín hiệu xóa “0”: trong các bài thực hành này, chúng ta chưa sử dụng đến nên để hở (không nối đi đâu cả).

6.3. Xây dựng và mô phỏng khối 16-Register

6.3.1 Xây dựng khối 16-Register

- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là 16registers-block. Trong cửa sổ con ở góc trên bên trái, ngoài tên main, ALU-1bit, ALU-8bits, ALU-16bits logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là 16register-block.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con 16register-block sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con “Canvas Window” của logisim, chúng ta tạo một tập 16 thanh ghi như ở hình vẽ dưới đây.



6.3.2 Mô phỏng hoạt động của khối 16-Register

Chúng ta có thể mô phỏng để xem đơn vị 16-Registers mà chúng ta vừa tạo ra có thực hiện đúng các chức năng như ý đồ thiết kế không.

Chọn 1 trong 16 thanh ghi để nhận dữ liệu vào từ Bus-C:

1. Đặt giá trị cho phần tử input “clock” bằng 1 (sử dụng “poke tool”).
2. Đặt giá trị cho phần tử input “Input from C-bus” bằng một giá trị nhị phân nào đó, chẳng hạn bằng “1111.0000.1100.1100”.
3. Đặt giá trị cho phần tử input “From C-decoder” bằng “0000.0000.0000.0001” để chọn thanh ghi “PC”, cho nó nhận dữ liệu từ phần tử input “Input from C-bus”. Nếu thay đổi giá trị này (chú ý: luôn chỉ có 1 bit bằng 1, còn các bit khác bằng 0) thì một trong 15 thanh ghi khác được chọn.

Chú ý: Với mạch điện và các giá trị được thiết lập như trên, chúng ta thường không đọc được giá trị nhị phân chứa vào thanh ghi được chọn. Để có thể nhìn thấy, chúng ta thực hiện thêm 2 bước như sau:

4. Đặt giá trị cho phần tử input “From B-decoder” giá trị nhị phân giống giá trị đặt cho phần tử input “From C-decoder”.
5. Dùng phần tử “poke tool” đảo giá trị của phần tử input “clock” 2 lần; chúng ta sẽ nhìn thấy giá trị mà thanh ghi được chọn đã nhận từ Bus-C, nay truyền ra phần tử output “To B-bus”.

6.4. Bài tập (chủ yếu theo giáo trình Kiến trúc máy tính)

Hãy sử dụng các các phần tử “Hex Digit Display” và “Splitter” để đọc được dưới dạng số Hexa các giá trị đưa vào các phần tử output “To A-Bus” và “To B-Bus”.

BÀI 7: Xây dựng các thành phần còn lại của đường dữ liệu

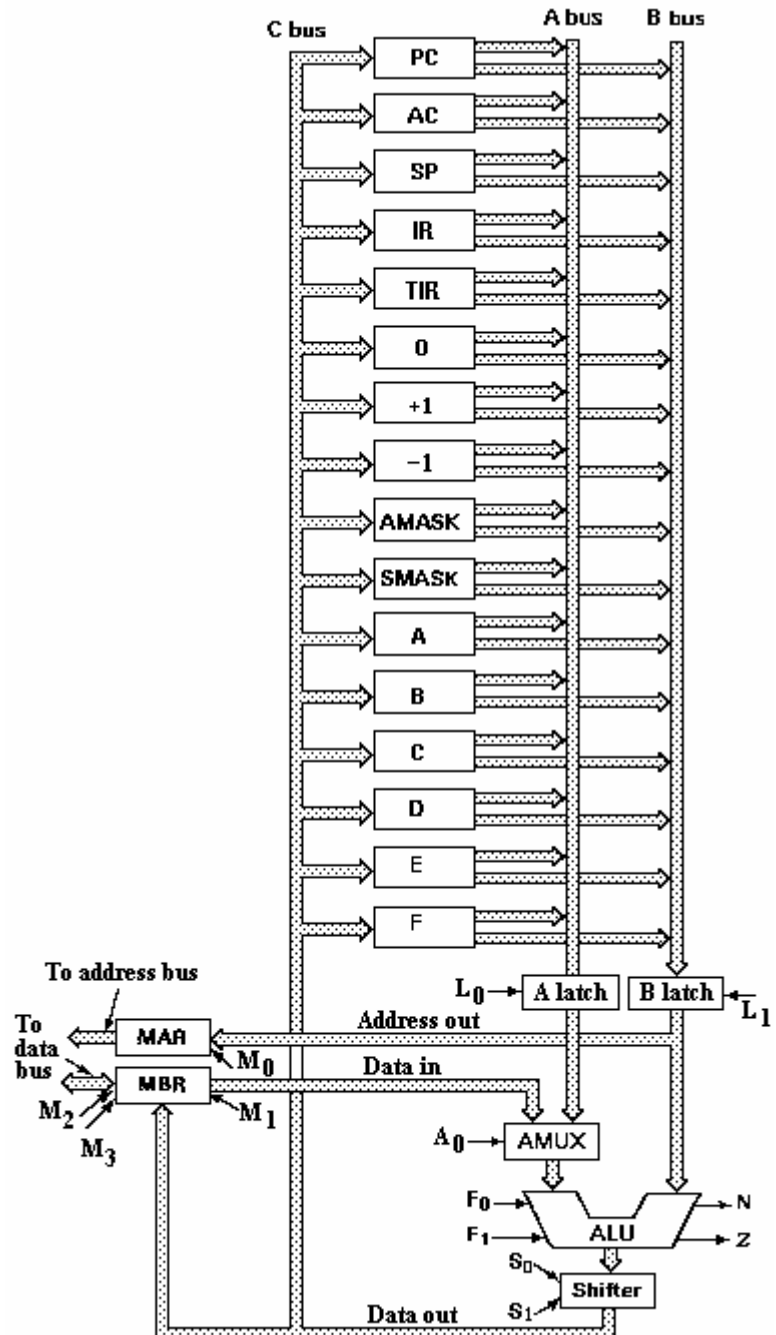
7.1. Các yêu cầu và nội dung chính

Đường dữ liệu (Data Path) được trình bày trong “Chương 4 Mức Vi chương trình” và được minh họa trên hình vẽ bên (Hình 4-07).

Trong 2 bài thực hành trước, chúng ta đã xây dựng và mô phỏng sự làm việc của 2 thành phần là ALU-16bits khối 16 thanh ghi.

Trong bài thực hành này, chúng ta sẽ xây dựng và mô phỏng sự hoạt động của các thành phần còn lại của đường dữ liệu, đó là:

- Shifter
- A Latch, B Latch
- AMUX
- MAR
- MBR



7.2. Xây dựng và mô phỏng sự hoạt động của shifter

7.2.1 Phân tích

- Thanh ghi dịch (Shifter) mà chúng ta cần xây dựng thực hiện 1 trong 3 thao tác (phép tính) theo tín hiệu điều khiển 2 bit được ký hiệu là S1S0.

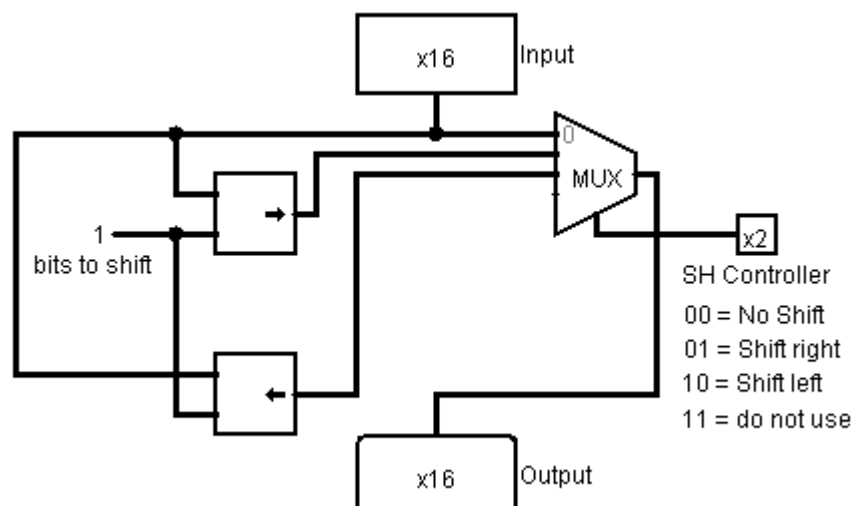
S1S0	Thao tác
00	No Shift (không dịch bit)
01	Shift Left (dịch trái 1 vị trí bit)
10	Shift Right (dịch phải 1 vị trí bit)
11	Không sử dụng giá trị này

Khi dịch trái/phải, giá trị 0 được đưa vào vị trí bit 0/15. Bit 15/0 bị “bật ra” khi dịch trái/phải nhưng (trong môn học này) chúng ta không cần quan tâm.

- Đầu vào (16 bits) của shifter nối với đầu ra của ALU-16bits.
- Đầu ra (16 bits) của shifter nối với đầu vào của MBR và C-Bus. Điều này có nghĩa là kết quả có thể đưa ra bộ nhớ chính (Main memory) thông qua MBR, hoặc đưa vào 1 trong 16 thanh ghi qua C-Bus để nó tham gia vào phép tính khác.

7.2.2 Xây dựng mạch con shifter

- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Shifter-16bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main, ALU-1bit, ALU-8bits, ALU-16bits, 16registers-block logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Shifter-16bits.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Shifter-16bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con “Canvas Window” của logisim, chúng ta xây dựng Shifter-16bits như ở hình vẽ dưới đây.



- Ngoài các phần tử input và output đã biết, chúng ta sử dụng các phần tử shifter trong nhóm Arithmetic và phần tử MUX (Multiplexer) trong nhóm Plexers của thư viện trong logisim.
- Phần tử shifter:
 - Số thuộc tính: 2, chúng ta cần thiết lập giá trị của chúng cho phù hợp với yêu cầu của mình:
 - Data Bits: 16 (có thể nhận các giá trị trong miền 1..32)
 - Shift Type: Logical Left (các giá trị khác có thể nhận là: Logical Right, Arithmetic Right, Roll Left, Roll Right)
 - Số chân: 3 (2 chân cho tín hiệu vào, 1 chân cho tín hiệu ra):
 - 1 chân vào cho data (16 bits): phía trên bên trái.
 - 1 chân ra cho data (16 bits): phía bên phải, có đánh dấu mũi tên, hướng mũi tên tùy theo chúng ta chọn thuộc tính “Shift Type” là left hay right.
 - 1 chân vào để điều khiển số vị trí bit sẽ dịch, trên hình có ghi nhãn “bits to shift”. Nếu chúng ta đã thiết lập “Data Bits” = 16 (2^4) thì đầu vào này nhất thiết phải là 4 bit. Nếu chúng ta chỉ muốn dịch 1 bit, thì thiết lập giá trị cho phần tử input nối với đầu vào này bằng 0001.
- Phần tử MUX (Multiplexer):
 - Số thuộc tính: 3, chúng ta cần thiết lập giá trị của chúng cho phù hợp với yêu cầu của mình:
 - Facing: East - hướng đầu ra về phía đông (các giá trị khác: West, North, South)
 - Select Bits: 2 (vì chúng ta cần chọn 1 trong 3 kết quả để đưa ra)
 - Data Bits: 16
 - Số chân: 6 (5 chân cho tín hiệu vào, 1 chân cho tín hiệu ra):
 - Chân vào số 0 (có ký hiệu 0): 16 bits cho giá trị “no shift”.
 - Chân vào số 1 (bên dưới chân 0): 16 bits cho giá trị “shift left”.
 - Chân vào số 2: 16 bits cho giá trị “shift right”.
 - Chân vào số 3: không sử dụng đến.
 - Chân vào cho 2 bits điều khiển: ở trên cạnh bên của hình thang cân (ký hiệu của Multiplexer).
 - Chân ra: 16 bits, nằm ở cạnh đáy nhỏ của hình thang cân, có giá trị bằng 1 trong 3 đầu vào, được chọn bằng 2 bits điều khiển.

7.2.3 Mô phỏng hoạt động của shifter

Chúng ta có thể mô phỏng để xem đơn vị shifter mà chúng ta vừa tạo ra có thực hiện đúng các chức năng như ý đồ thiết kế không. Có thể thực hiện như sau:

1. Đặt giá trị cho phần tử “Input” bằng một giá trị nhị phân nào đó, chẳng hạn bằng “1111.0000.1100.1100”.
2. Sử dụng phần tử “poke tool” để thiết lập giá trị cho 2 bits điều khiển (SH Controller) lần lượt bằng 00, 01 và 10. Nếu chúng ta xây dựng đúng, tại đầu ra “Output” chúng ta sẽ thấy các giá trị đưa ra lần lượt là: “1111.0000.1100.1100”, “1110.0001.1001.1000” và “0111.1000.0110.0110”. Đó chính là kết quả “no shift”, “shift left” và “shift right” đối với dữ liệu đưa vào.

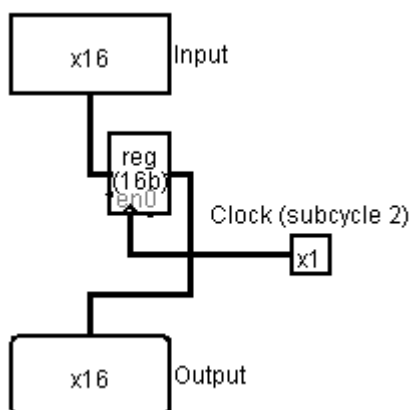
7.3. Xây dựng và mô phỏng sự hoạt động của các thanh ghi chốt A Latch và B Latch

7.3.1 Phân tích

Thanh ghi chốt mà chúng ta sẽ xây dựng để làm A Latch và B Latch có chức năng rất đơn giản. Nó có đầu vào và đầu ra 16 bits, đầu vào điều khiển 1 bit (L0 hoặc L1). Khi tín hiệu trên đầu vào điều khiển bằng 1, giá trị 16 bits ở đầu sẽ được nhớ và đưa ra đầu ra. Khi tín hiệu điều khiển từ 1 trở về 0, giá trị trong thanh ghi (và đầu ra) không thay đổi được nữa, dù giá trị ở đầu vào có thay đổi.

7.3.2 Xây dựng thanh ghi chốt

- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Latch-16bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main, ALU-1bit, ALU-8bits, ALU-16bits, 16registers-block, Shifter-16bits logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Latch-16bits.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Latch-16bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con “Canvas Window” của logisim, chúng ta xây dựng Latch-16bits như ở hình vẽ dưới đây. Tất cả các phần tử cần sử dụng chúng ta đều đã biết.



7.3.3 Mô phỏng hoạt động của thanh ghi chốt (Latch Register)

Chúng ta có thể mô phỏng để xem đơn vị latch mà chúng ta vừa tạo ra có thực hiện đúng các chức năng như ý đồ thiết kế không. Có thể thực hiện như sau:

1. Ban đầu, bit “Clock” bằng 0 và đặt giá trị cho phần tử “Input” bằng một giá trị nhị phân nào đó, chẳng hạn bằng “1111.0000.1100.1100”.
2. Thiết lập giá trị cho bit “Clock” bằng 1, chúng ta sẽ thấy đầu ra “Output” sẽ có giá trị đúng bằng “Input” là: “1111.0000.1100.1100”.
3. Đảo bit “Clock” bằng 0.
4. Thay đổi giá trị của “Input”, chúng ta sẽ thấy “Output” không bị thay đổi.

7.4. Xây dựng và mô phỏng sự hoạt động của AMUX

AMUX là một bộ dồn kênh 2-to-1, chúng ta sẽ sử dụng phần tử Multiplexer trong nhóm Plexers có trong thư viện của logisim. Ngoài phần tử này, chúng ta không cần thêm một phần tử chức năng nào khác để tạo nên AMUX.

7.5 Xây dựng và mô phỏng sự hoạt động của MAR

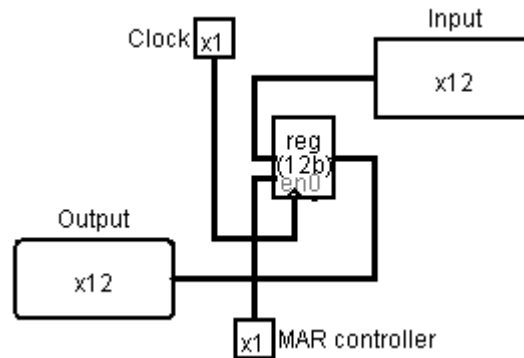
7.5.1 Phân tích

- Chức năng của thanh ghi chốt địa chỉ bộ nhớ MAR được mô tả tại mục “4.1.6 Bộ nhớ chính” của giáo trình kiến trúc máy tính. Trên hình vẽ minh họa đường dữ liệu ở đầu Bài thực hành số 7 này, tín hiệu điều khiển M_0 tích cực (=1) làm cho MAR chốt dữ liệu ở đầu vào nối với đầu ra B Latch của nó, khi M_0 đảo giá trị (từ 1 về 0), nội dung MAR không thay đổi theo tín hiệu vào và luôn được truyền ra đầu ra (chính là Address Bus, thuộc bus hệ thống của máy tính). Điều đáng lưu ý khi thực hành mô phỏng là, đầu ra và đầu vào của MAR đều truyền giá trị 12 bits chứ không phải 16 bits. 12 bits đưa vào đầu vào của MAR lấy từ 12 bits thấp trong số 16 bits truyền từ đầu ra của B Latch. Việc tách 12 bits này chúng ta sẽ thực hiện bằng phần tử splitter thuộc nhóm Base có trong thư viện của logisim, khi chúng ta lắp ghép toàn bộ vi kiến trúc.
- Ngoài tín hiệu điều khiển M_0 nói trên phải có mức tích cực, chúng ta đã biết rằng, MAR chỉ được chốt dữ liệu trong chu kỳ con thứ 3 của đơn vị đồng hồ (vấn đề này được phân tích trong mục “4.2.3 Việc định thời vi chỉ thị”, giáo trình Kiến trúc máy tính). Tóm lại, có 2 tín hiệu điều khiển MAR chốt dữ liệu, một là tín hiệu lấy từ bit MAR trong thanh ghi MIR (chính là M_0 trong hình vẽ ở đầu bài thực hành này); hai là tín hiệu lấy từ đường chu kỳ con thứ 3 của đơn vị đồng hồ.

7.5.2 Xây dựng thanh ghi MAR

- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là MAR-16bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là MAR-16bits.

- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con MAR-16bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con “Canvas Window” của logisim, chúng ta xây dựng MAR-16bits như ở hình vẽ dưới đây. Tất cả các phần tử cần sử dụng chúng ta đều đã biết.



7.5.3 Mô phỏng hoạt động của MAR

Chúng ta có thể mô phỏng để xem thanh ghi chốt MAR mà chúng ta vừa tạo ra có thực hiện đúng các chức năng như ý đồ thiết kế không. Việc thực hiện tương tự như đối với thanh ghi chốt đã trình bày tại mục “7.3.3 Mô phỏng hoạt động của thanh ghi chốt“, chỉ có một sự khác nhau nhỏ, đó là có 2 tín hiệu điều khiển MAR. Các bước thực hiện như sau:

1. Ban đầu, bit “Clock” và “Mo” bằng 0, chúng ta đặt giá trị cho phần tử “Input” bằng một giá trị nhị phân nào đó, chẳng hạn bằng “1111.0000.1100.1100”.
2. Thiết lập giá trị cho bit “Clock” và “Mo” đồng thời bằng 1, chúng ta sẽ thấy đầu ra “Output” sẽ có giá trị đúng bằng “Input” là: “1111.0000.1100.1100”. Hãy thử chỉ cho một trong 2 tín hiệu “Clock” và “Mo” bằng 1, giá trị còn lại bằng 0, chúng ta sẽ thấy thanh ghi không thể nhận giá trị “Input”.
3. Đảo bit “Clock” bằng 0.
4. Thay đổi giá trị của “Input”, chúng ta sẽ thấy “Output” không bị thay đổi.

7.6 Xây dựng và mô phỏng sự hoạt động của MBR

7.6.1 Phân tích

- Chức năng của thanh ghi đệm dữ liệu bộ nhớ MBR được mô tả tại mục “4.1.6 Bộ nhớ chính” và mục “4.2.1 Đường dữ liệu (Data path)” của giáo trình kiến trúc máy tính.
- Trên hình vẽ minh họa đường dữ liệu ở đầu Bài thực hành số 7 này, chúng ta thấy có 3 tín hiệu điều khiển MBR:

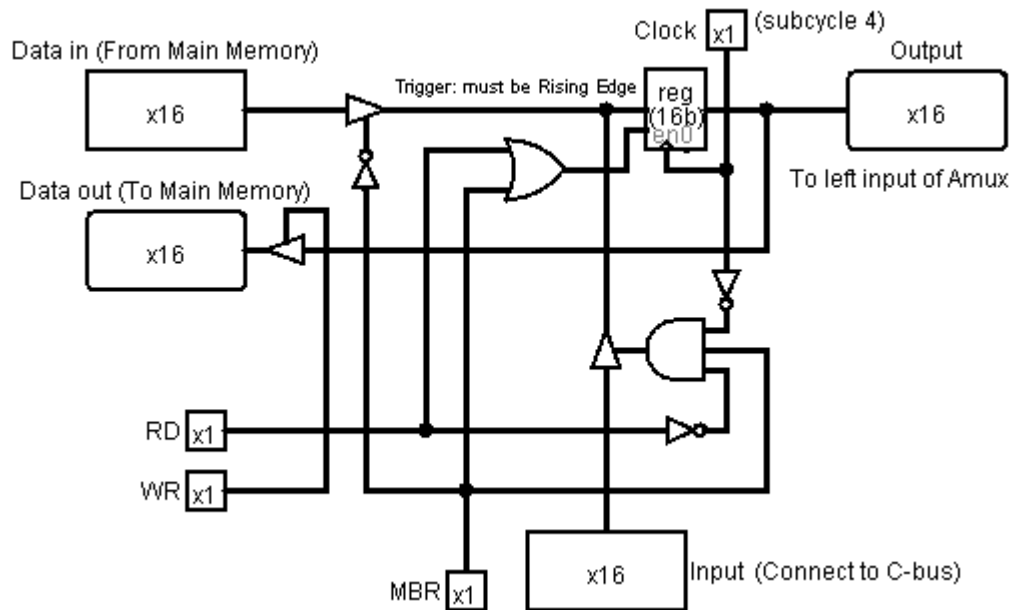
- M_1 : điều khiển việc nạp của MBR từ đầu ra của Shifter ($M_1 = 1$: nạp). Trong “Hình 4-09 Sơ đồ đầy đủ của một vi kiến trúc”, M_1 được nối với **bit MBR** của MIR.
 - M_2 : điều khiển việc đọc bộ nhớ, M_2 được nối với **bit RD** của MIR. Khi đọc, dữ liệu từ bộ nhớ được chứa vào MBR và truyền ra một trong 2 đầu vào của AMUX. Dữ liệu từ đầu vào này có thể được điều khiển để đi ra đầu ra và nạp vào đầu vào bên trái của ALU.
 - M_3 : điều khiển việc ghi bộ nhớ, M_3 được nối với **bit WR** của MIR. Khi ghi, dữ liệu đang chứa trong MBR được đưa ra bus dữ liệu nối với Main memory của MBR (bus bên trái MBR).
- Ngoài 3 tín hiệu điều khiển nói trên, chúng ta đã biết rằng, MBR chỉ được chột dữ liệu vào hoặc đưa dữ liệu ra trong chu kỳ con thứ 4 của đơn vị đồng hồ (vấn đề này được phân tích trong mục “4.2.3 Việc định thời vi xử lý”, giáo trình Kiến trúc máy tính).

7.6.2 Xây dựng thanh ghi MBR

- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là MBR-16bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là MBR-16bits.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con MBR-16bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con “Canvas Window” của logisim, chúng ta xây dựng MBR-16bits như ở hình vẽ dưới đây. Tất cả các phần tử cần sử dụng chúng ta đều đã biết.

Chú ý:

Bus dữ liệu nối MBR với Main memory là bus 2 chiều; tuy nhiên vì logisim không có phần tử nào vừa có thể là input vừa có thể là output, cho nên chúng ta đành phải tạo ra 2 bus một chiều để thay thế.



7.6.3 Mô phỏng hoạt động của MBR

Chúng ta có thể mô phỏng để xem thanh ghi MBR mà chúng ta vừa tạo ra có thực hiện đúng các chức năng như ý đồ thiết kế không. Việc thực hiện có nhiều bước tương tự như đối với thanh ghi chốt (latch) và thanh ghi MAR.

Mô phỏng việc chứa dữ liệu từ Bus-C vào MBR theo các bước như sau:

1. Ban đầu, cần thiết lập giá trị cho:
 - MBR, RD, WR và Clock: 0 (có thể chọn từ menu: simulate \ Reset Simulation).
 - “Input (Connect to C-bus)”: một giá trị nhị phân nào đó, chẳng hạn bằng “1111.0000.1100.1100”.
2. Thiết lập giá trị cho bit MBR bằng 1; sau đó đảo giá trị bit Clock từ 0 thành 1, chúng ta sẽ thấy đầu ra “Output” sẽ có giá trị đúng bằng “Input” là: “1111.0000.1100.1100”.

Mô phỏng việc chứa dữ liệu từ bộ nhớ vào MBR (đọc bộ nhớ) theo các bước như sau:

1. Ban đầu, để tránh nhầm lẫn, nên reset lại mô phỏng (menu: simulate \ Reset Simulation).
2. Thiết lập giá trị cho phần tử input “From Main Memory”: một giá trị nhị phân nào đó, chẳng hạn bằng “1111.0000.1100.1100”.
3. **Thiết lập giá trị cho phần tử input RD bằng 1**; sau đó đảo giá trị bit Clock từ 0 thành 1, chúng ta sẽ thấy đầu ra “Output” sẽ có giá trị đúng bằng “Input” là: “1111.0000.1100.1100”.

Mô phỏng việc đưa dữ liệu từ MBR ra bộ nhớ (ghi bộ nhớ):

Sau việc mô phỏng trên (chứa dữ liệu từ bộ nhớ vào MBR), phần tử “Output” đang chứa giá trị “1111.0000.1100.1100”, còn phần tử output “To Main Memory” chưa từng được gán giá trị nên được logisim coi là chứa giá trị không xác định và hiển thị nội dung phần tử này bằng 16 ký tự “x”.

Chúng ta thực hiện mô phỏng việc ghi bộ nhớ theo như sau:

Thiết lập giá trị cho bit MBR bằng 0 và bit WR bằng 1, chúng ta sẽ thấy đầu ra “To Main Memory” sẽ có giá trị đúng bằng “Output” là: “1111.0000.1100.1100”.

7.7. Bài tập

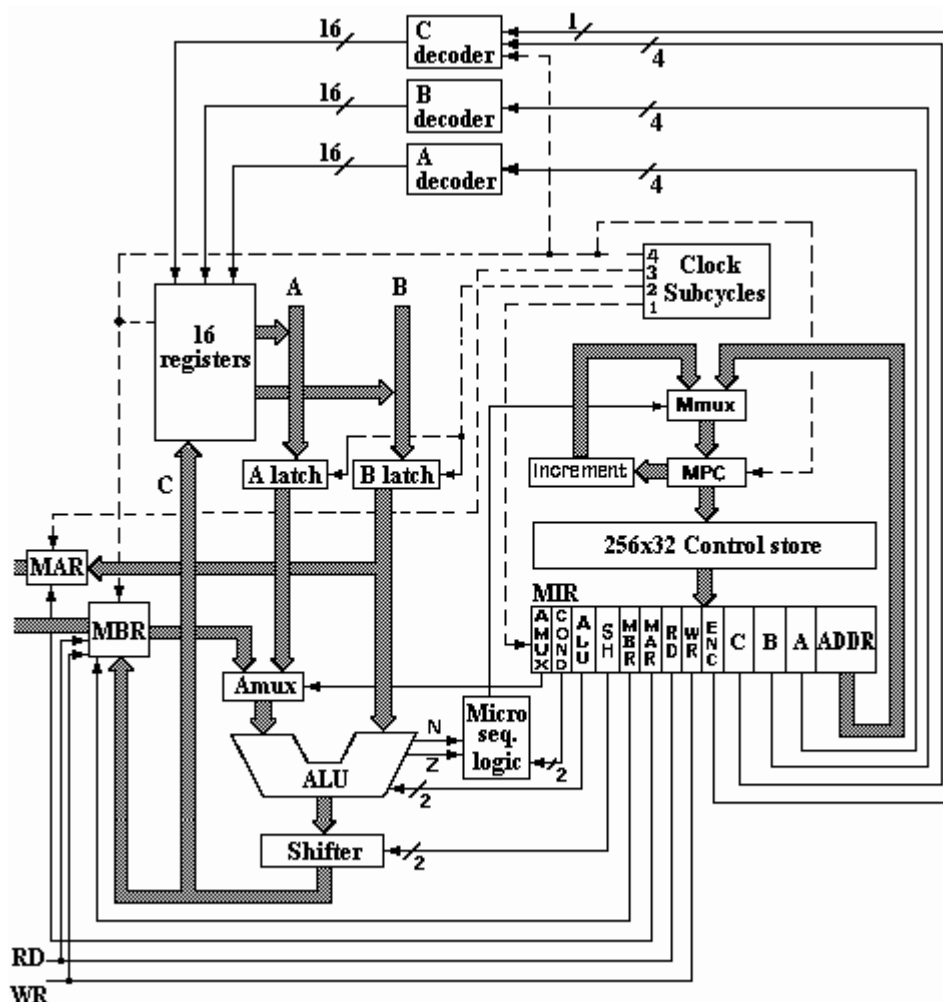
Với mạch con MBR-16bits như chúng ta đã xây dựng, việc ghi bộ nhớ chỉ đòi hỏi bit điều khiển WR bằng 1, chứ không cần “Clock (subcycle 4)” bằng 1. Hãy sửa lại mạch con đó sao cho dữ liệu từ MBR chỉ đưa ra được bus dữ liệu nối với Main Memory khi bit điều khiển “Clock (subcycle 4)” bằng 1.

BÀI 8: Xây dựng Control Store, MIR và Micro Seq của đơn vị điều khiển

8.1. Các yêu cầu và nội dung chính

Đơn vị điều khiển – CU (Control Unit) được giới thiệu khái quát trong mục “2.1 Kiến trúc chung của máy tính điện tử”; Tại chương 2, chúng ta đã biết rằng CU thực hiện việc điều khiển sự hoạt động của tất cả các đơn vị tạo nên hệ thống máy tính theo chương trình trong bộ nhớ chính.

CU được nghiên cứu sâu hơn trong chương 4, tại các mục “4.2.2 Vi chỉ thị - microinstruction”, “4.2.3 Việc định thời vi chỉ thị”, “4.2.4 Sự định trình tự các vi chỉ thị”,... Trong sơ đồ khối đầy đủ của một vi kiến trúc dưới đây, chúng ta thấy CU bao gồm các đơn vị sau: Control Store, MIR, Micro Seq, MPC, Increment, Mmux, A Decoder, B decoder, C decoder và đơn vị tạo tín hiệu đồng hồ - Clock.



Hình 4-09 Sơ đồ khối đầy đủ của một vi kiến trúc

Trong Bài thực hành số 8, chúng ta sẽ xây dựng và mô phỏng sự hoạt động của 3 trong số 9 đơn vị cấu thành CU, đó là: Control Store, MIR và Micro Seq.

- Control Store: phải chứa được toàn bộ vi chương trình thực hiện thông dịch các lệnh lấy về từ bộ nhớ chính (Main memory).
 - + Đầu vào: Có 1 đầu vào địa chỉ 8 bits nối với đầu ra của MPC. Địa chỉ này dùng để chọn 1 trong số 2^8 vi chỉ thị chứa trong Control Store.
 - + Đầu ra: Có 1 đầu ra 32 bits nối với đầu vào của MIR.
- MIR: nhận chỉ thị đưa ra từ Control Store và duy trì nội dung trong thời gian thi hành.
- Micro Seq: sinh ra tín hiệu Mmux quyết định chọn vi chỉ thị tiếp theo, căn cứ vào các tín hiệu trạng thái N, Z và tín hiệu điều khiển COND.

8.2. Xây dựng và mô phỏng sự hoạt động của Control Store

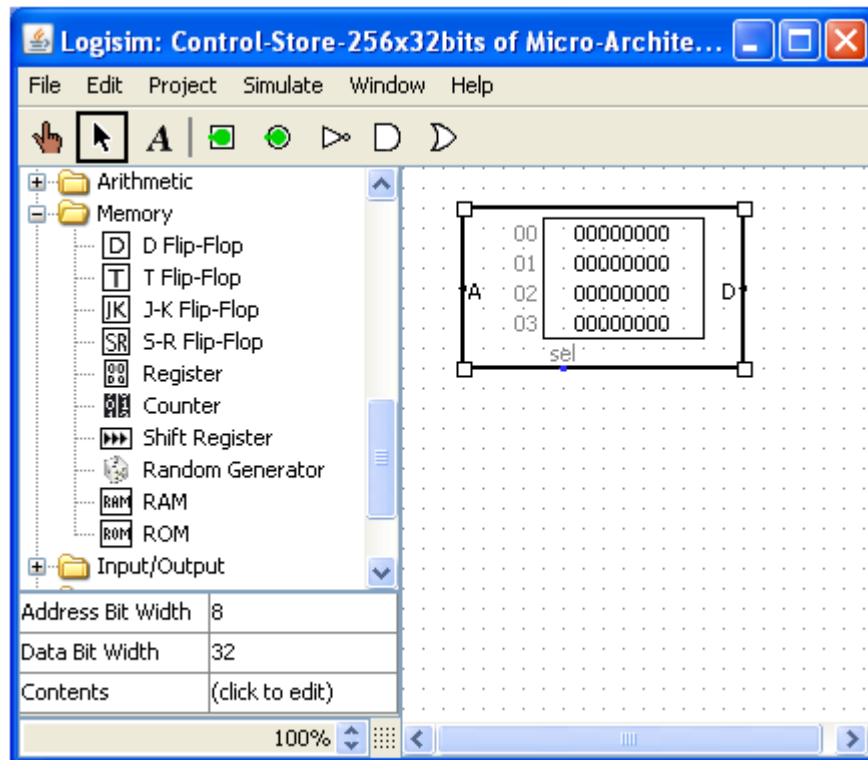
8.2.1 Phân tích

- Về nguyên tắc, Control Store có thể là ROM hoặc RAM. Trong bài thực hành này, chúng ta sẽ **sử dụng phần tử ROM** có trong thư viện của logisim làm Control Store. Lý do: Trong bài thực hành số 10, chúng ta sẽ tạo ra và mô phỏng hoạt động của bộ nhớ chính (Main memory), chỉ có thể sử dụng phần tử RAM. Như vậy chúng ta sử dụng cả 2 loại phần tử nhớ trong thực hành.
- Kích thước: Trong mục “4.4 Thí dụ về một vi chương trình”, chúng ta đã phân tích kỹ lưỡng một vi chương trình gồm 79 vi chỉ thị, mỗi vi chỉ thị dài 32 bit. Như vậy kích thước của Control Store tối thiểu là 79x32 bit. Tuy nhiên, để có thể mở rộng được, yêu cầu sinh viên thiết kế Control Store có kích thước **256x32 bits**.
- Từ kích thước 256x32 dẫn đến việc một số trong các thuộc tính của phần tử ROM làm Control Store bắt buộc phải có giá trị như sau:
 - + Address Bit Width: 8 (để có thể chọn 1 trong 256)
 - + Data Bit Width: 32 (kích thước 1 vi chỉ thị)
- Nội dung (Contents): Chính là vi chương trình được trình bày tại mục “4.4.2 Thí dụ về một vi chương trình”, sinh viên cần nạp vào phần tử ROM, sau khi đã chuyển từ dạng MAL sang dạng nhị phân.

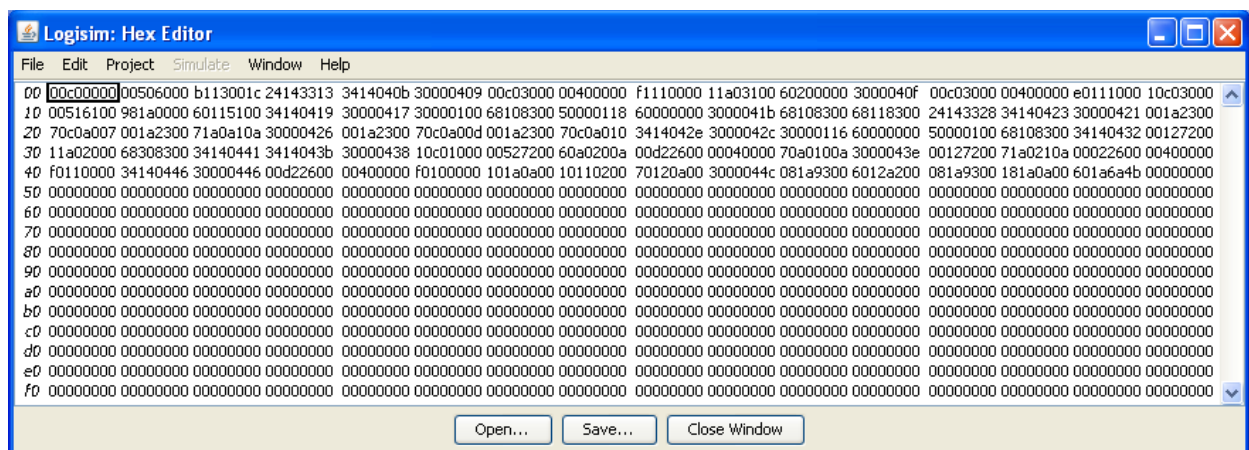
8.2.2. Xây dựng và nạp nội dung cho Control Store

- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Control-Store-256x32bits. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Control-Store-256x32bits.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Control-Store-256x32bits sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con “Canvas Window” của logisim, việc xây dựng Control-Store-256x32bits hết sức đơn giản:

- Lấy một phần tử ROM trong nhóm Memory đặt vào “Canvas Window”.
- Thiết lập giá trị cho các thuộc tính của nó, như đã phân tích ở trên (Address Bit Width: 8; Data Bit Width: 32).



- Để nạp nội dung cho phần tử nhớ Control-Store, hãy kích nút chuột bên phải, logisim hiện lên một danh sách các lựa chọn. Hãy chọn “Edit Contents...”, logisim sẽ hiện lên một cửa sổ “Logisim: Hex Editor” để người sử dụng nạp nội dung cho phần tử nhớ Control-Store.



- Việc nạp nội dung cho Control-Store tuy đơn giản, nhưng mất nhiều thời gian và đòi hỏi tuyệt đối chính xác. Hình vẽ trên là nội dung Control-Store đã được nạp 79 (4FH = 4*16 + 15) words dưới dạng số hexa, mỗi word 32 bits là một vi chỉ thị của vi chương trình làm nhiệm vụ thông dịch.

Ví chương trình được nạp vào Control Store

Địa chỉ	Statement (Chỉ thị ngôn ngữ MAL)	A M U X	C O N D	A L U	S H	M B R	M A R	R D	W R	E N C	C	B	A	A D D R	Chỉ thị (Hexa)
0 (00h)	mar := pc; rd	0	00	00	00	0	1	1	0	0	0000	0000	0000	0000.0000	00.C0.00.00
1 (01h)	pc := pc+1; rd;	0	00	00	00	0	0	1	0	1	0000	0110	0000	0000.0000	00.50.60.00
2 (02h)	ir := mbr; if n then goto 28;	1	01	10	00	0	0	0	0	1	0011	0000	0000	0001.1100	B0.13.00.1C
3 (03h)	tir := lshift(ir+ir); if n then goto 19;	0	01	00	10	0	0	0	0	1	0100	0011	0011	0001.0011	24.14.33.13
4 (04h)	tir := lshift(tir); if n then goto 11;	0	01	10	10	0	0	0	0	1	0100	0000	0100	0000.1011	34.14.04.0B
5 (05h)	alu := tir; if n then goto 9;	0	01	10	00	0	0	0	0	0	0000	0000	0100	0000.1001	30.00.04.09
6 (06h)	mar := ir; rd;	0	00	00	00	0	1	1	0	0	0000	0011	0000	0000.0000	00.C0.30.00
7 (07h)	rd;	0	00	00	00	0	0	1	0	0	0000	0000	0000	0000.0000	00.40.00.00
8 (08h)	ac := mbr; goto 0	1	11	10	00	1	0	0	0	1	0001	0000	0000	0000.0000	F1.11.00.00
9 (09h)	mar := ir; mbr := ac; wr;	0	00	10	00	1	1	0	1	0	0000	0011	0001	0000.0000	11.A0.31.00
10 (0Ah)	wr; goto 0;	0	11	00	00	0	0	0	1	0	0000	0000	0000	0000.0000	60.20.00.00
11 (0Bh)	alu := tir; if n then goto 15;	0	01	10	00	0	0	0	0	0	0000	0000	0100	0000.1111	30.00.04.0F
12 (0Ch)	mar := ir; rd;	0	00	00	00	0	1	1	0	0	0000	0011	0000	0000.0000	00.C0.30.00
13 (0Dh)	rd;	0	00	00	00	0	0	1	0	0	0000	0000	0000	0000.0000	00.40.00.00
14 (0Eh)	ac := mbr + ac; goto 0;	1	11	00	00	1	0	0	0	1	0001	0001	0000	0000.0000	E1.11.10.00
19 (13h)	tir := lshift(tir); if n then goto 25;	0	01	10	10	0	0	0	0	1	0100	0000	0100	0001.1001	34.14.04.19
25 (19h)	alu := tir; if n then goto 27;	0	01	10	00	0	0	0	0	0	0000	0000	0100	0001.1011	30.00.04.1B
26 (1Ah)	pc := band(ir, amask); goto 0;	0	11	01	00	0	0	0	0	1	0000	1000	0011	0000.0000	68.10.83.00

Địa chỉ (Hexa)	Statement (Chỉ thị ngôn ngữ MAL)	A M U X	C O N D	A L U	S H	M B R	M A R	R D	W R	E N C		B	A	A D D R	Chỉ thị (Hexa)
-------------------	-------------------------------------	------------------	------------------	-------------	--------	-------------	-------------	--------	--------	-------------	--	---	---	------------------	----------------

(Các vi chỉ thị dưới đây cần được kiểm tra lại bằng việc cho thực hiện)

Địa chỉ (Hexa)	Chỉ thị MAL	Chỉ thị (nhị phân)	Chỉ thị (Hexa)
0E	ac := mbr + ac; goto 0;	1110.0000.0001.0001.0001.0000.0000.0000	e0.11.10.00
0F	mar := ir; rd;	0001.0000.1100.0000.0011.0000.0000.0000	10.c0.30.00
10	ac := ac + 1; rd;	0000.0000.0101.0001.0110.0001.0000.0000	00.51.61.00
11	a := inv(mbr);	1001.1000.0001.1010.0000.0000.0000.0000	98.1a.00.00
12	ac := ac + a; goto 0;	0110.0000.0001.0001.0101.0001.0000.0000	60.11.51.00
13	tir := lshift(tir); if n then goto 25;	0011.0100.0001.0100.0000.0100.0001.1001	34.14.04.19
14	alu := tir; if n then goto 23;	0011.0000.0000.0000.0000.0100.0001.0111	30.00.04.17
15	alu := ac; if n then goto 0;	0011.0000.0000.0000.0000.0001.0000.0000	30.00.01.00
16	pc := band(ir, amask); goto 0;	0110.1000.0001.0000.1000.0011.0000.0000	68.10.83.00
17	alu := ac; if z then goto 22;	0101.0000.0000.0000.0000.0001.0001.1000	50.00.01.18
18	goto 0;	0110.0000.0000.0000.0000.0000.0000.0000	60.00.00.00
1B	ac := band(ir, amask); goto 0;	0110.1000.0001.0001.1000.0011.0000.0000	68.11.83.00
1C	tir := lshift(ir+ir); if n then goto 40;	0010.0100.0001.0100.0011.0011.0010.1000	24.14.33.28
1D	tir := lshift(tir); if n then goto 35;	0011.0100.0001.0100.0000.0100.0010.0011	34.14.04.23
1E	alu := tir; if n then goto 33;	0011.0000.0000.0000.0000.0100.0010.0001	30.00.04.21
1F	a := ir + sp;	0000.0000.0001.1010.0010.0011.0000.0000	00.1a.23.00
20	mar := a; rd; goto 7;	0111.0000.1100.0000.1010.0000.0000.0111	70.c0.a0.07

21	a := ir + sp;	0000.0000.0001.1010.0010.0011.0000.0000	00.1a.23.00
22	mar := a; mbr := ac; wr; goto 10;	0111.0001.1010.0000.1010.0001.0000.1010	71.a0.a1.0a
23	alu := tir; if n then goto 38;	0011.0000.0000.0000.0000.0100.0010.0110	30.00.04.26
24	a := ir + sp;	0000.0000.0001.1010.0010.0011.0000.0000	00.1a.23.00
25	mar := a; rd; goto 13;	0111.0000.1100.0000.1010.0000.0000.1101	70.c0.a0.0d
26	a := ir + sp;	0000.0000.0001.1010.0010.0011.0000.0000	00.1a.23.00
27	mar := a; rd; goto 16;	0111.0000.1100.0000.1010.0000.0001.0000	70.c0.a0.10
28	tir := lshift(tir); if n then goto 46;	0011,0100.0001.0100.0000.0100.0010.1110	34.14.04.2e
29	alu := tir; if n then goto 44;	0011.0000.0000.0000.0000.0100.0010.1100	30.00.04.2c
2A	alu := ac; if n then goto 22;	0011.0000.0000.0000.0000.0001.0001.0110	30.00.01.16
2B	goto 0;	0110.0000.0000.0000.0000.0000.0000.0000	60.00.00.00
2C	alu := ac; if z then goto 0;	0101.0000.0000.0000.0000.0001.0000.0000	50.00.01.00
2D	pc := band(ir, amask); goto 0;	0110.1000.0001.0000.1000.0011.0000.0000	68.10.83.00
2E	tir := lshift(tir); if n then goto 50;	0011.0100.0001.0100.0000.0100.0011.0010	34.14.04.32
2F	sp := sp + (-1);	0000.0000.0001.0010.0111.0010.0000.0000	00.12.72.00
30	mar := sp; mbr := pc; wr;	0001.0001.1010.0000.0010.0000.0000.0000	11.a0.20.00
31	pc := band(ir, amask); wr; goto 0;	0110.1000.0011.0000.1000.0011.0000.0000	68.30.83.00
32	tir := lshift(tir); if n then goto 65;	0011.0100.0001.0100.0000.0100.0100.0001	34.14.04.41
33	tir := lshift(tir); if n then goto 59;	0011.0100.0001.0100.0000.0100.0011.1011	34.14.04.3b
34	alu := tir; if n then goto 56;	0011.0000.0000.0000.0000.0100.0011.1000	30.00.04.38
35	mar := ac; rd;	0001.0000.1100.0000.0001.0000.0000.0000	10.c0.10.00
36	sp := sp + (-1); rd;	0000.0000.0101.0010.0111.0010.0000.0000	00.52.72.00
37	mar := sp; wr; goto 10;	0110.0000.1010.0000.0010.0000.0000.1010	60.a0.20.0a
38	mar := sp; sp := sp + 1; rd;	0000.0000.1101.0010.0010.0110.0000.0000	00.d2.26.00
39	rd;	0000.0000.0100.0000.0000.0000.0000.0000	00.04.00.00
3A	mar := ac; wr; goto 10;	0111.0000.1010.0000.0001.0000.0000.1010	70.a0.10.0a
3B	alu := tir; if n then goto 62;	0011.0000.0000.0000.0000.0100.0011.1110	30.00.04.3e
3C	sp := sp + (-1);	0000.0000.0001.0010.0111.0010.0000.0000	00.12.72.00

3D	mar := sp; mbr := ac; wr; goto 10;	0111.0001.1010.0000.0010.0001.0000.1010	71.a0.21.0a
3E	mar := sp; sp := sp + 1; rd;	0000.0000.1101.0010.0010.0110.0000.0000	00.02.26.00
3F	rd;	0000.0000.0100.0000.0000.0000.0000.0000	00.40.00.00
40	ac := mbr; goto 0;	1111.0000.0001.0001.0000.0000.0000.0000	f0.11.00.00
41	tir := lshift(tir); if n then goto 73;	0011.0100.0001.0100.0000.0100.0100.0110	34.14.04.46
42	alu := tir; if n then goto 70;	0011.0000.0000.0000.0000.0100.0100.0110	30.00.04.46
43	mar := sp; sp := sp + 1; rd;	0000.0000.1101.0010.0010.0110.0000.0000	00.d2.26.00
44	rd;	0000.0000.0100.0000.0000.0000.0000.0000	00.40.00.00
45	pc := mbr; goto 0;	1111.0000.0001.0000.0000.0000.0000.0000	f0.10.00.00
46	a := ac;	0001.0000.0001.1010.0000.1010.0000.0000	10.1a.0a.00
47	ac := sp;	0001.0000.0001.0001.0000.0010.0000.0000	10.11.02.00
48	sp := a; goto 0;	0111.0000.0001.0010.0000.1010.0000.0000	70.12.0a.00
49	alu := tir; if n then goto 76;	0011.0000.0000.0000.0000.0100.0100.1100	30.00.04.4c
4A	a := band(ir, smask);	0000.1000.0001.1010.1001.0011.0000.0000	08.1a.93.00
4B	sp := sp + a; goto 0;	0110.0000.0001.0010.1010.0010.0000.0000	60.12.a2.00
4C	a := band(ir, smask);	0000.1000.0001.1010.1001.0011.0000.0000	08.1a.93.00
4D	a := inv(a);	0001.1000.0001.1010.0000.1010.0000.0000	18.1a.0a.00
4F	a := a + 1; goto 75;	0110.0000.0001.1010.0110.1010.0100.1011	60.1a.6a.4b

(Chú ý: dấu ‘.’ đã được “viết” thêm vào giữa các nhóm 4 bit nhị phân và các nhóm 2 số hexa để cho dễ đọc; không “nạp” chúng vào Control-Store)

8.2.3. Mô phỏng sự hoạt động của Control Store

Với cách xây dựng Control Store chỉ gồm 1 phần tử ROM lấy trong thư viện của logisim, chúng ta chỉ có thể mô phỏng sự hoạt động của nó khi lắp ghép đầy đủ các thành phần của vi kiến trúc.

Việc thực hành mô phỏng Control Store sẽ được thực hiện trong Bài thực hành số 10.

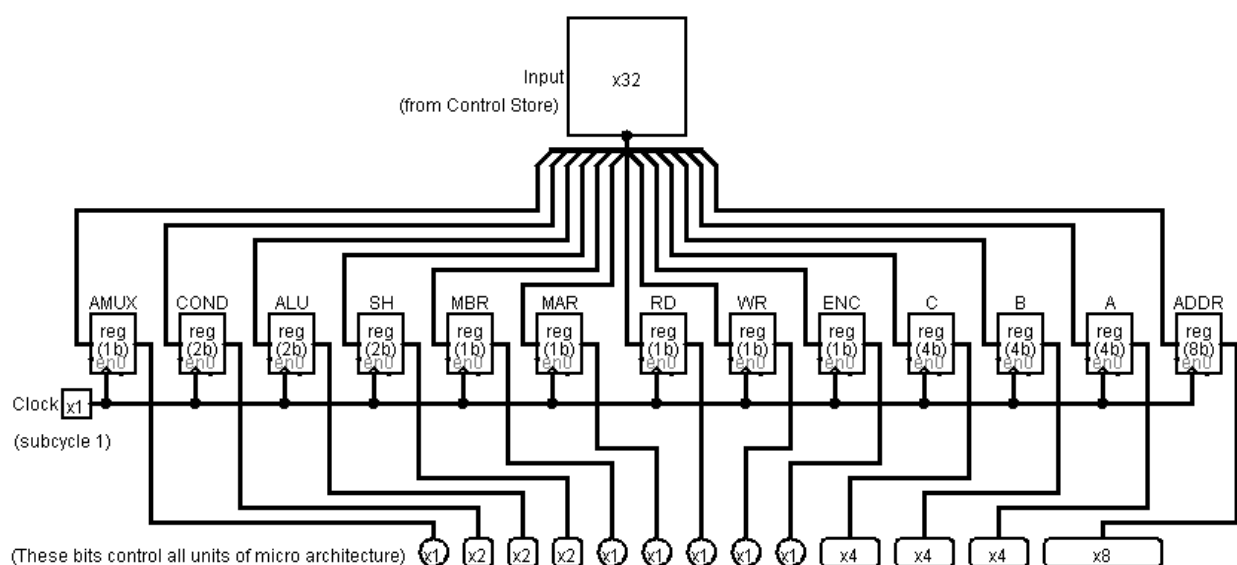
8.2. Xây dựng và mô phỏng sự hoạt động của MIR

8.2.1 Phân tích

- Kích thước: 32 bits (chứa được 1 vi chỉ thị).
- Đầu vào: Có 1 đầu vào 32 bits nối với đầu ra của Control Store.
- Đầu ra: 32 bit được nhóm thành 13 nhóm tín hiệu, mỗi nhóm tín hiệu hoặc điều khiển một đơn vị thành phần định trước nào đó của đường dữ liệu (thí dụ: ALU, SH,...), hoặc đưa vào một đơn vị thành phần định trước nào đó thuộc CU để điều khiển một đơn vị thành phần nhất định thuộc CU (thí dụ: Micro Seq, Mmux,) hoặc đường dữ liệu (thí dụ: các decoders).

8.2.2 Xây dựng thanh ghi MIR

- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là MIR. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là MIR.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con MIR sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con “Canvas Window” của logisim, xây dựng MIR như hình vẽ sau đây:



Chú ý: Các phần tử “reg” làm 16 thanh ghi trên có đầu vào “en” được logisim đặt giá trị mặc định bằng 1 (active), vì vậy chúng ta không cần nối 16 đầu “en” của 16 thanh ghi lại với nhau và truyền vào đó giá trị “1”.

8.2.3 Mô phỏng hoạt động của thanh ghi MIR

Mô phỏng việc đưa dữ liệu từ “Input” (nối với Control-Store) vào MIR:

- Thiết lập giá trị cho phần tử “Input”, thí dụ bằng:
“1111.0000.1111.0000.1111.0000.1111.0000”.
- Sử dụng phần tử “poke tool” đảo giá trị của phần tử input “Clock” từ 0 lên 1, chúng ta sẽ thấy phần tử “Output” nhận giá trị bằng giá trị của phần tử “Input”.

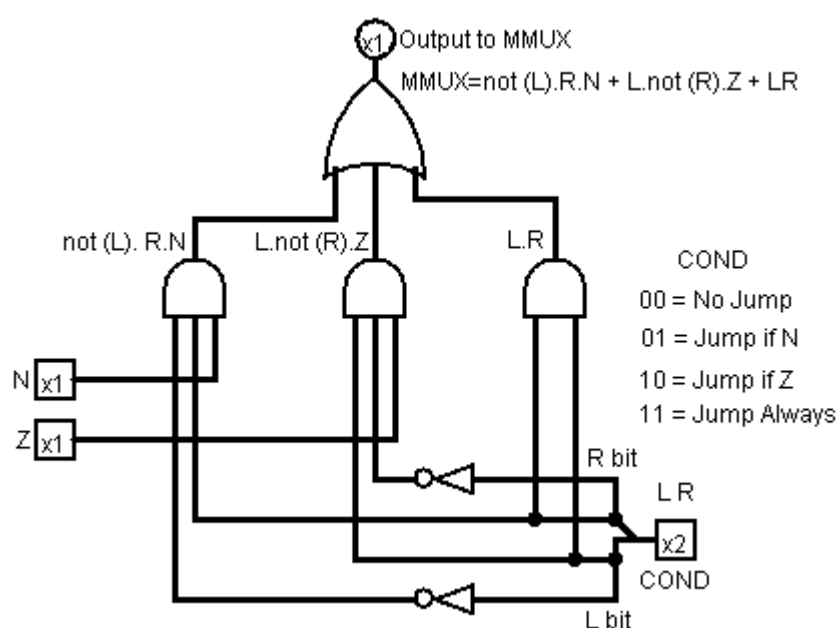
8.3. Xây dựng và mô phỏng sự hoạt động của Micro Seq

8.3.1 Phân tích

- $Mmux = \overline{L}RN + L\overline{R}Z + LR = RN + LZ + LR$
- Chúng ta sẽ xây dựng mạch sinh tín hiệu $Mmux = \overline{L}RN + L\overline{R}Z + LR$ cho dễ hiểu hơn.

8.3.2 Xây dựng mạch con Micro Seq

- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Micro-Seq. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Micro-Seq.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Micro-Seq sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con “Canvas Window” của logisim, xây dựng Micro-Seq như hình vẽ sau đây:



8.3.3 Mô phỏng hoạt động của Micro Seq

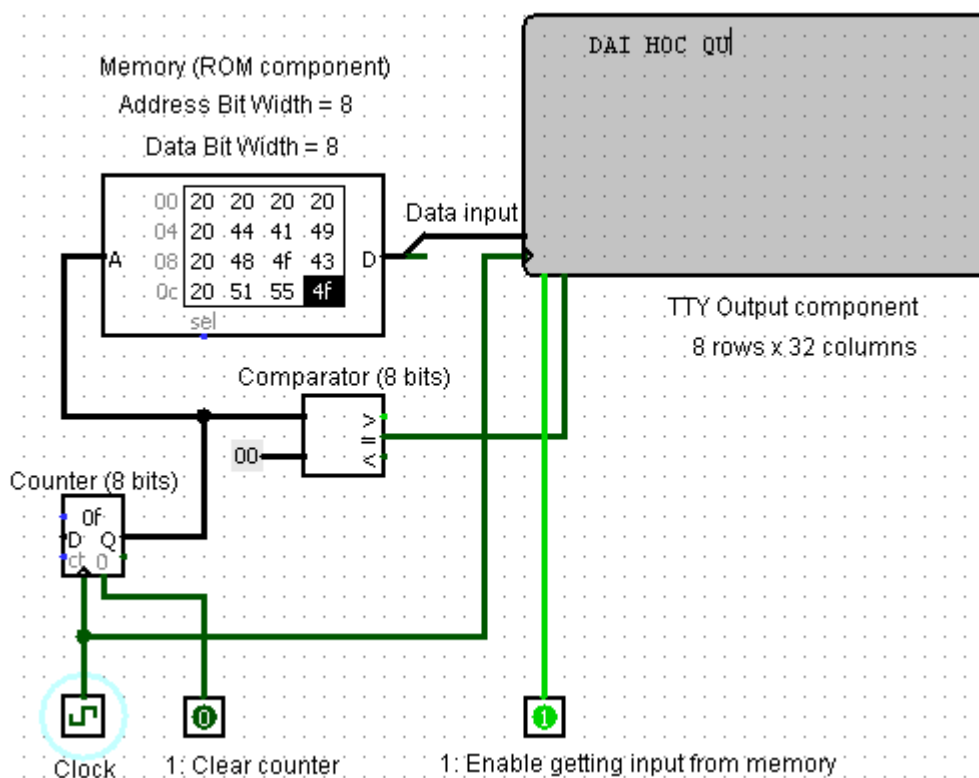
Mô phỏng hoạt động của mạch con Micro-Seq: Thiết lập giá trị cho phần tử input “COND” lần lượt bằng: 00, 01, 10 và 11, kiểm tra giá trị tại đầu ra MMUX phụ thuộc vào N và Z như thế nào:

- COND = 00: MMUX luôn bằng 0, không phụ thuộc vào N và Z.
- COND = 01: MMUX bằng 1 nếu N = 1 (N và Z không bao giờ đồng thời bằng 1).
- COND = 10: MMUX bằng 1 nếu Z = 1.
- COND = 11: MMUX luôn bằng 1, không phụ thuộc vào N và Z.

8.4. Bài tập (thầy Việt mới bổ sung 3/11/2010)

Hãy thiết kế bằng mô phỏng bảng hiển thị điện tử kích thước 8 hàng x 32 cột và hiện dòng chữ “DAI HOC QUOC GIA HA NOI” ở chính giữa dòng trên cùng. Yêu cầu từng ký tự của dòng chữ trên được hiện lên lần lượt, sau khi hiển thị đến ký tự cuối cùng thì xóa toàn bộ và lặp lại quá trình trên.

Gợi ý: Có thể thiết kế “bảng hiển thị điện tử” theo sơ đồ sau đây (sinh viên tự phân tích).



Trong đó các phần tử (components) có thuộc tính như sau:

- Phần tử TTY (TeleTYpe): Rows = 8, Columns = 32. Như vậy “màn hình” có thể hiển thị được tối đa là $8 \times 32 = 256$ ký tự. Hai giá trị này có thể thay đổi, thí dụ để “màn hình” hiển thị này giống màn hình máy tính ở chế độ Text, có thể chọn Rows = 25, Columns = 80. Cần chú ý: Đầu vào mã ký tự (Data) là 7 bit.
- Phần tử ROM memory: Address Bit Width = 8 (dung lượng “con” ROM này sẽ là $2^8 = 256$ Words); Data Bit Width = 8, do đó 1 word gồm 8 bit. Chú ý: Để nạp nội dung cho ROM (tức là nội dung hiển thị), hãy trở vào phần tử này rồi kích nút chuột phải, chọn “Edit contents...” và nhập vào các ô nhớ (byte) bắt đầu từ địa chỉ 00 các giá trị hexa sau: 20, 20, 20, 20, 20, 44, 41, 49, 20, 48, 4f, 43, 20, 51, 55, 4f, 43, 20, 47, 49, 41, 20, 48, 41, 20, 4e, 4f, 49
- Phần tử Comparator: Dùng để so sánh giá trị 8 bits đầu ra của bộ đếm (Counter) với hằng số 00h, khi 2 giá trị 8 bits này bằng nhau, tức là Counter đã đếm xong 1 vòng, từ 00 đến FFh và trở về 00, thì Comparator sẽ sinh ra giá trị 1 (“true”) tại đầu ra “=”, chúng ta đưa vào “Clear” của phần tử TTY để xóa “màn hình” đồng thời chuyển “con trỏ màn hình” về góc trên bên trái.

4. Phần tử Counter: Giá trị tại đầu ra Q sẽ tăng thêm 1 mỗi khi đầu vào “Clock” có 1 xung (chuyển mức từ 0 lên 1). Logisim đặt giá trị ban đầu của Counter bằng 0. Đầu ra của Counter được chúng ta nối với đầu vào địa chỉ A của phần tử ROM, để lần lượt đánh địa chỉ các ô nhớ liên tiếp từ 00 đến FFh (256 ô nhớ). Ô nhớ được đánh địa chỉ sẽ truyền nội dung (1 byte) ra đầu ra dữ liệu D để truyền sang đầu vào dữ liệu của phần tử TTY. Chú ý: chúng ta sử dụng phần tử Splitter để nối 7 bit thấp của đầu ra D với đầu vào 7 bits của phần tử TTY.
5. Phần tử Clock: chúng ta có thể kích bằng “tay” hoặc bằng cách cho chạy tự động: (Simulate \ Ticks Enable), có thể thay đổi tốc độ hiển thị bằng cách thực hiện: Simulate \ Tick Frequency rồi chọn1 giá trị cụ thể. Nếu chúng ta chọn giá trị quá lớn, logisim có thể không thực hiện được, khi đó chúng ta cần chọn lại một giá trị nhỏ hơn.
6. Phần tử Input (x 2): Chúng ta đã từng sử dụng nhiều lần ở các bài thực hành trước.
7. Phần tử Constant: Chúng ta đã từng sử dụng nhiều lần ở các bài thực hành trước.
8. Phần tử Splitter: Nói ở đoạn “4.” bên trên.

BÀI 9: Xây dựng các thành phần còn lại của đường điều khiển

9.1. Các yêu cầu và nội dung chính

Trong bài thực hành này, chúng ta sẽ tạo ra và mô phỏng sự hoạt động của các đơn vị còn lại của CU, bao gồm các đơn vị sau:

- MPC: con trỏ vi chỉ thị.
- Increment: nhận input từ MPC rồi cộng thêm 1 và đưa ra đầu ra để truyền tới Mmux.
- Mmux: Bộ dồn kênh 2-to-1, độ rộng các đường vào/ra là 8 bits.
- A Decoder, B decoder và C decoder
 - + A và B decoder là các bộ giải mã địa chỉ 4-to-16 thông thường.
 - + C decoder: ngoài chức năng giải mã thông thường như A và B decoder, C decoder còn phải có 1 đặc điểm khác nữa: 1 trong 16 đầu ra được chọn chỉ được truyền mức logic 1 ra ngoài khi và chỉ khi ENC=1 đồng thời tín hiệu đồng hồ ở đầu ra subcycle 4 có mức 1.
- Clock: đơn vị tạo tín hiệu đồng hồ có 4 đầu ra cùng tần số do chúng ta chọn, nhưng lệch pha nhau 90 độ.

9.2. Xây dựng và mô phỏng hoạt động của thanh ghi MPC

9.2.1 Phân tích

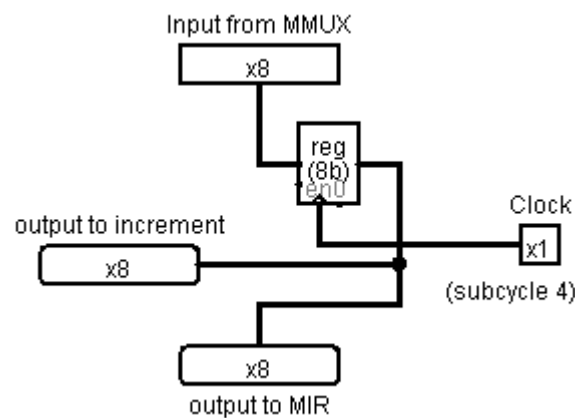
Như có thể thấy trên “Hình 4-09 Sơ đồ khối đầy đủ của một vi kiến trúc” trong Bài thực hành số 8, MPC có:

- 1 đầu vào dữ liệu 8 bits nối với đầu ra của đơn vị Mmux, đây chính là địa chỉ của vi chỉ thị trong Control Store cần đưa ra MIR để thực hiện.
- 1 đầu ra dữ liệu 8 bits đưa vào Control Store.
- 1 đầu ra dữ liệu 8 bits đưa vào đơn vị Increment để làm tăng lên 1.
- 1 đầu vào điều khiển nối với đầu ra tín hiệu chu kỳ con thứ 4 (subcycle 4) của đơn vị Clock, chỉ khi tín hiệu này bằng 1 thì MPC mới chốt giá trị ở đầu vào (nghĩa là giá trị ở đầu ra nhận giá trị của đầu vào), còn khi tín hiệu này bằng 0 thì giá trị chứa trong MPC (và giá trị ở đầu ra) không thay đổi.

9.2.2 Xây dựng thanh ghi MPC

- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là MPC. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là MPC.

- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con MPC sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con “Canvas Window” của logisim, xây dựng MPC như hình vẽ sau đây:



9.2.3 Mô phỏng hoạt động của thanh ghi MPC

Mô phỏng việc đưa dữ liệu từ “Input from MMUX” vào MPC:

- Thiết lập giá trị nhị phân 8 bits cho phần tử “Input from MMUX”, thí dụ bằng: 1111.0000.
- Sử dụng phần tử “poke tool” đảo giá trị của phần tử input “subcycle 4” từ 0 lên 1, chúng ta sẽ thấy phần tử “output to MIR” nhận giá trị bằng giá trị của phần tử “Input from MMUX”.

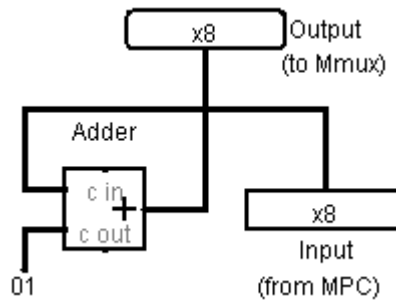
9.3. Xây dựng và mô phỏng hoạt động của đơn vị Increment

9.3.1 Phân tích

Với yêu cầu đã nêu trong mục 9.1 thì đây là bộ cộng, toán hạng vào thứ nhất là dữ liệu ra từ MPC (chính là địa chỉ của chỉ thị đưa ra thi hành); toán hạng vào thứ 2 là hằng +1. Kết quả của phép cộng sẽ đưa trở lại 1 trong 2 đầu vào của Mmux.

9.3.2 Xây dựng mạch con Increment

- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Increment. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Increment.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Increment sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con “Canvas Window” của logisim, xây dựng Increment như hình vẽ sau đây:



9.3.3 Mô phỏng hoạt động của Increment

Mô phỏng việc thực hiện chức năng cộng 1 của Increment:

- Thiết lập giá trị nhị phân 8 bits cho phần tử “Input (from MPC)”, thí dụ bằng: 1111.0000.
- Quan sát phần tử “Output (to Mmux)” chúng ta sẽ thấy giá trị vừa đưa vào tăng thêm 1, bằng 1111.0001.

9.4. Xây dựng và mô phỏng hoạt động của đơn vị Mmux

Đây là một bộ dồn kênh 2-to-1 thông thường, tương tự Amux mà chúng ta đã tìm hiểu, xây dựng và mô phỏng trong bài thực hành số 7 (mục 7.4). Điều khác nhau duy nhất là với Mmux, các đường dữ liệu vào/ra là 8 bits chứ không phải 16 bits như với Amux.

Tại bài thực hành số 9 này chúng ta chưa cần xây dựng ngay Mmux, đến bài thực hành số 10 chúng ta sẽ lấy một phần tử Multiplexer (thuộc nhóm Plexers trong thư viện của logisim) làm Mmux.

9.5. Xây dựng và mô phỏng hoạt động của decoder

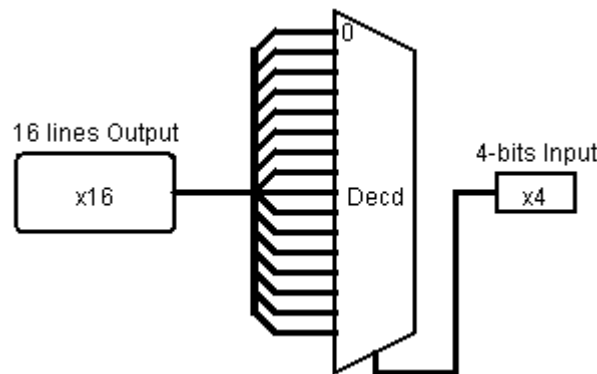
9.5.1 Phân tích

- A và B decoders: đây là các decoder 4-to-16 thông thường.
- C decoders: theo các yêu cầu đã trình bày ở mục 9.1, mỗi một trong số 16 đầu ra của C decoder phải đi qua một bộ đếm 3 trạng thái (trong logisim đó là phần tử controlled buffer) được điều khiển bởi tín hiệu là kết quả and logic của ENC và (Clock subcycle 4).

9.5.2 Xây dựng mạch con làm A, B decoder và mô phỏng sự hoạt động của nó

- Chúng ta có thể sử dụng phần tử Decoder trong nhóm Plexers trong thư viện của logisim, hoặc chúng ta sử dụng các phần tử đó có bổ sung thêm một số phần tử khác để mạch con mà chúng ta xây dựng trông “gọn gàng” hơn, như sẽ thực hiện dưới đây.
- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Decoder-A&B. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Decoder-A&B.

- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Decoder-A&B sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con “Canvas Window” của logisim, xây dựng Decoder-A&B như hình vẽ sau đây:



Chú ý:

Việc chúng ta tạo ra phần tử Decoder-A&B chỉ nhằm mục đích thực hành. Nếu chúng ta chỉ dùng 1 phần tử Decoder trong thư viện của logisim, thì chúng ta không cần phải dùng thêm đến 3 phần tử, đó là:

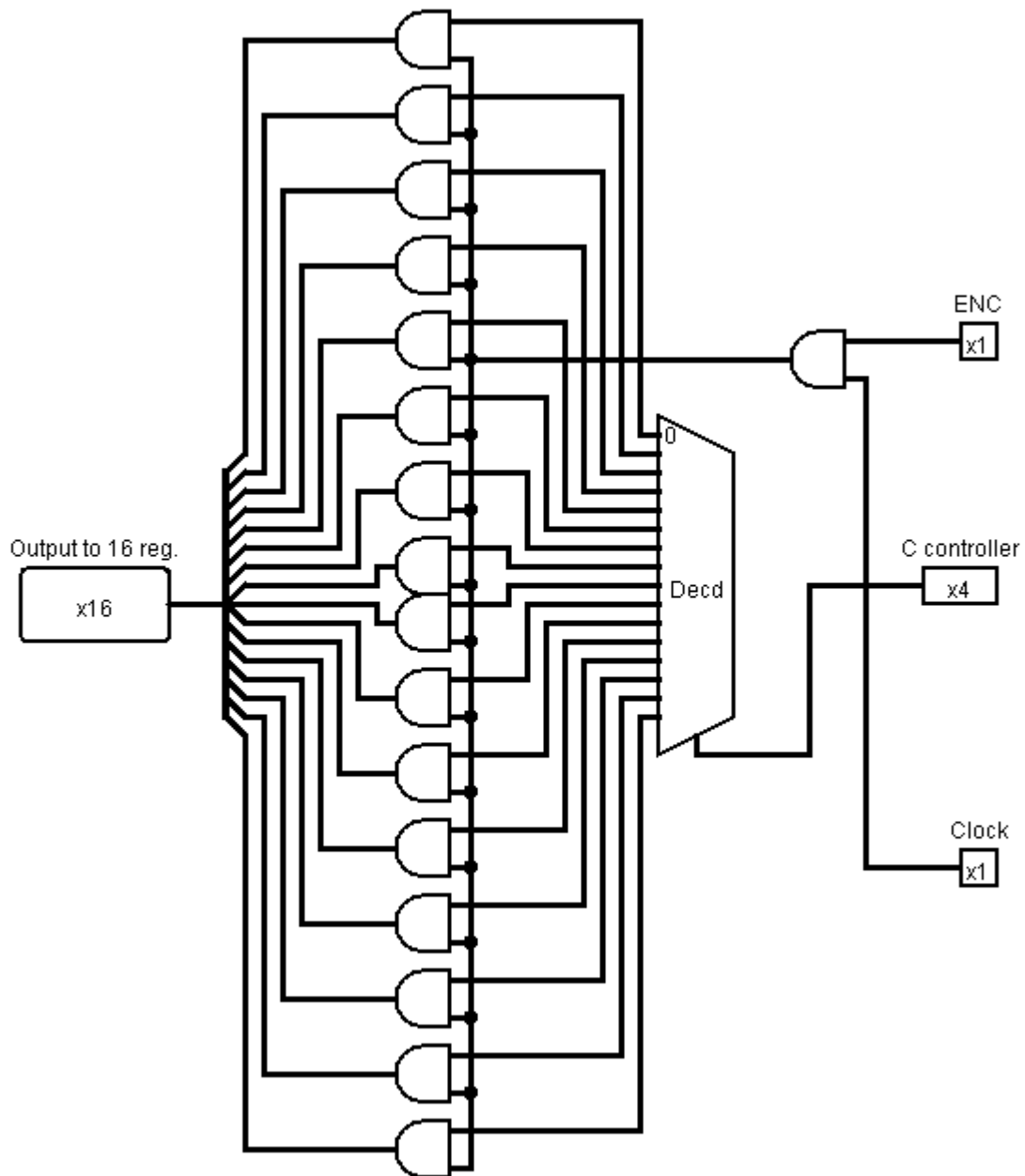
- Phần tử input “4-bit input”
- Phần tử output “16 lines output”
- Phần tử Splitter để chập 16 dây ra từ decoder thành 1 bó dây cho gọn.

Mô phỏng việc thực hiện chức năng của Decoder-A&B:

- Thay đổi tăng dần giá trị của phần tử “4-bit input” từ 0000 đến 1111.
- Quan sát phần tử “16 lines output” chúng ta sẽ thấy giá trị nhị phân 16 bit thay đổi, bit có giá trị 1 chuyển từ vị trí 0 đến vị trí 15, trong khi đó các bit còn lại bằng 0.

9.5.3 Xây dựng mạch con làm C decoder và mô phỏng sự hoạt động của nó

- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Decoder-C. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Decoder-C.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Decoder-C sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con “Canvas Window” của logisim, xây dựng Decoder-C như hình vẽ sau đây:



Mô phỏng việc thực hiện chức năng của Decoder-C: Thực hiện tương tự như khi mô phỏng Decoder-A&B, nhưng thực hiện 2 lần:

- Lần thứ nhất thiết lập giá trị cho ENC và CLOCK sao cho ENC.CLOCK bằng 1.
- Lần thứ hai thiết lập giá trị cho ENC và CLOCK sao cho ENC.CLOCK bằng 0.

9.6. Xây dựng và mô phỏng hoạt động của Clock

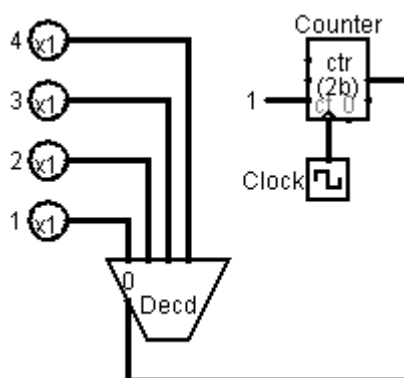
9.6.1 Phân tích

Đây là đơn vị không bị điều khiển bởi bất cứ đơn vị nào khác trong vi kiến trúc. Nhiệm vụ của nó là sinh ra 4 tín hiệu điện lệch pha nhau để điều khiển sự phối hợp công việc nhịp nhàng giữa các thành phần thuộc đường dữ liệu và thuộc đơn vị điều khiển CU.

Với vi kiến trúc được nghiên cứu trong môn học Kiến trúc máy tính, trừ các chỉ thị Read và Write, các chỉ thị được thi hành trong thời gian như nhau và được gọi là 1 chu kỳ đường dữ liệu. Như vậy, các tín hiệu đồng hồ được coi là lệch pha nhau 90° .

9.6.2 Xây dựng mạch con làm đơn vị Clock

- Vào menu Project \ Add Circuit, logisim sẽ hiện lên một cửa sổ nhỏ, hỏi tên mà ta muốn đặt cho mạch điện. Hãy đặt tên là Clock-4-subcycles. Trong cửa sổ con ở góc trên bên trái, ngoài tên main và tên các mạch con mà chúng ta đã tạo ra từ trước, logisim hiện tên mạch con (subcircuit) mà chúng ta vừa tạo ra, đó là Clock-4-subcycles.
- Lưu (save) mạch điện mà chúng ta tạo ra vào file (Micro-Architecture-SVxx.circ): vào menu File \ Save. Mạch con Clock-4-subcycles sẽ được lưu cùng với Micro-Architecture-SVxx.circ.
- Trong cửa sổ con “Canvas Window” của logisim, xây dựng Clock-4-subcycles như hình vẽ sau đây. Vì trong thư viện của logisim chỉ có 1 loại phần tử clock 1 pha, cho nên chúng ta bắt buộc phải sử dụng thêm một số phần tử khác nữa để tạo 4 tín hiệu đồng hồ có cùng tần số nhưng lệch pha nhau 90° . Các phần tử được sử dụng để xây dựng mạch con này bao gồm:
 - Clock: lấy trong thư viện của logisim. Số thuộc tính của nó có thể thay đổi được là 5, ý nghĩa các thuộc tính rất dễ hiểu. Ngoài ra khi mô phỏng hoạt động của clock, chúng ta có thể thay đổi được:
 - Tần số: từ 0.25 Hz đến 4096 Hz (vào Menu: Simulate \ Tick Frequency).
 - Cách chạy đồng hồ: 1/ Tick One (để debug); 2/ Tick Enable (để chạy bình thường).



- Counter: phần tử đếm có trong thư viện. Counter có các thuộc tính sau:
 - Data bits (1..32): độ lớn của số mà Counter sẽ đếm. Chúng ta cần đặt bằng 2 để Counter đếm từ 0 đến 3.
 - Maximum value: giới hạn số cao nhất mà Counter đếm, phải đặt không lớn hơn giá trị chứa được trong “Data bits”.
 - Action on Overflow: 1/ Wrap around (đếm vòng lại từ số bé nhất); 2/ Stay at value (dừng đếm) ...

- Trigger: 1/ Rising Edge (kích hoạt việc đếm bởi sườn dương); 2/ Falling Edge (kích hoạt việc đếm bằng sườn âm).
- Label: nhãn do chúng ta đặt.
- Label Font: kiểu và cỡ font chữ của Label.

Trên hình vẽ, Counter có tất cả 7 chân vào/ra. Trong “Canvas Window”, khi chúng ta di con trỏ đến các chân này, logisim sẽ hiện một dòng giải thích:

- “^”: Clock: value may update on trigger. Khi tín hiệu (xung đồng hồ) đặt vào đây thay đổi từ mức thấp lên mức cao, sẽ làm thay đổi giá trị chứa trong counter 1 đơn vị. Sự thay đổi có thể là tăng hoặc giảm phụ thuộc vào một số tín hiệu điều khiển khác, sẽ được mô tả dưới đây.
 - “0”: Clear: clear, when 1 resets to 0 asynchronously. Giá trị mặc định ở đầu vào này là 1, chúng ta có thể để trống chân này.
 - “ct”: Count, when 1, counter increments (or decrement if load =1). Nếu đưa giá trị 1 vào đây, counter sẽ đếm tiến.
 - “D”: value to load into counter. Đây là đầu vào cho giá trị ban đầu (nhiều bit) mà chúng ta muốn nạp cho counter.
 - “Q”: Output: current value of counter. Đây là đầu ra cho giá trị hiện thời của counter.
 - Đầu vào trên cùng bên trái (không có ký hiệu): Load: when 1, loads from data input (if Count =0) or decrements.
- Decd (Decoder): đây là phần tử giải mã thông thường, có sẵn trong thư viện.
 - 4 phân tử output được ghi nhãn 1, 2, 3, 4 tương ứng với 4 đầu ra truyền tín hiệu của 4 chu kỳ con.

9.6.3 Mô phỏng sự hoạt động của Clock

Hãy chọn “Tick Enable” hoặc chọn “Tick Once” nhiều lần, chúng ta sẽ nhìn thấy nội dung phần tử counter thay đổi tăng dần từ 0 đến 3 và vòng lại giá trị 0.

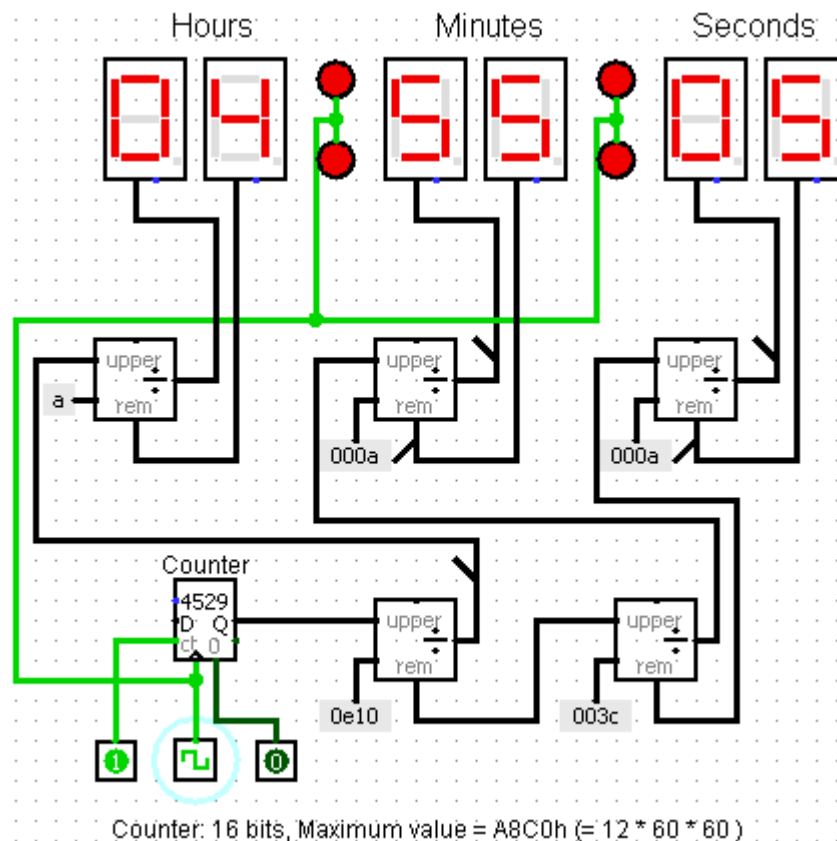
Có thể thay đổi tần số đồng hồ để thay đổi tốc độ đếm (Simulate \ Tick Frequency).

9.7. Bài tập (thầy Việt mới bổ sung 5/11/2010)

Hãy thiết kế một đồng hồ điện tử hiện số, với các yêu cầu sau:

- Thời gian được hiển thị dưới dạng: **hh : mm : ss**, trong đó:
 - hh** là số chỉ giờ (hour), tăng từ 0 đến 11 rồi lại trở về đếm từ 0.
 - mm** là số chỉ phút (minute), tăng từ 0 đến 59 rồi lại trở về đếm từ 0.
 - ss** là số chỉ giây (second), tăng từ 0 đến 59 rồi lại trở về đếm từ 0.
 - Giữa các số chỉ thời gian nói trên có dấu “:”, nhấp nháy 1 giây 1 lần.
- Mỗi chữ số được hiển thị bằng 1 phần tử “Hex Digit Display” trong thuộc nhóm Input/Output trong thư viện của logisim.

Gợi ý: Có thể thiết kế đồng hồ theo sơ đồ dưới đây. Các phần tử được sử dụng đều có trong thư viện của logisim, mô tả về chức năng hoạt động của chúng cũng có thể tra cứu trong “Help”.



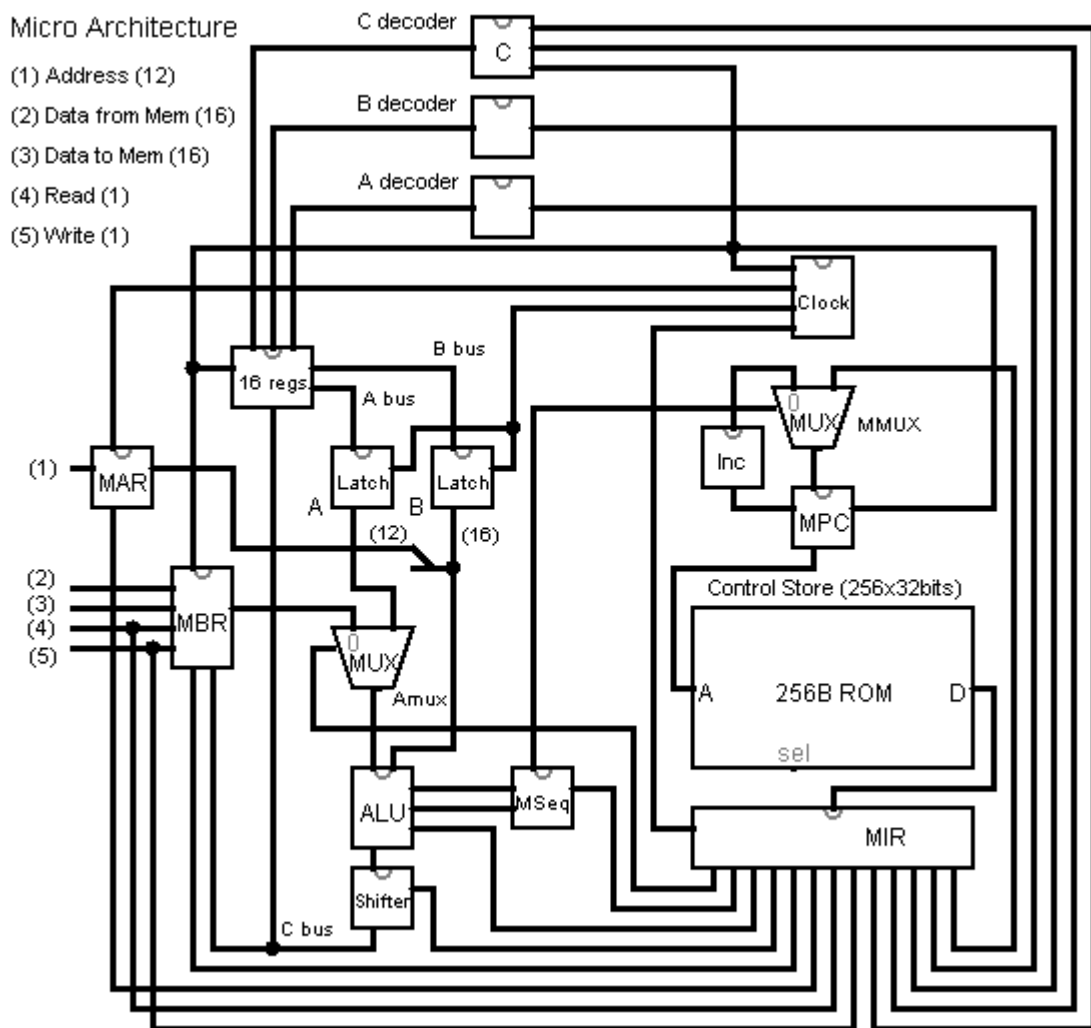
BÀI 10: Xây dựng vi kiến trúc đầy đủ

10.1. Các yêu cầu và nội dung chính

Trong bài thực hành này, chúng ta cần thực hiện được 3 việc chính sau:

1. Lắp ghép các đơn vị chức năng để tạo thành vi kiến trúc đầy đủ, như Hình 4-09 trong giáo trình Kiến trúc máy tính (có trong Bài thực hành số 8).
2. Bổ sung thêm bộ nhớ chính (RAM) và ghép nối với vi kiến trúc để tạo nên một “máy tính” chạy được chương trình.
3. Kiểm tra “máy tính” xem nó có được lắp ghép đúng đắn hay không bằng cách cho thực hiện một chương trình viết bằng tập chỉ thị của máy MAC-1 (có 23 chỉ thị, tương tự các chỉ thị của ngôn ngữ Assembly).

10.2. Lắp ghép vi kiến trúc đầy đủ



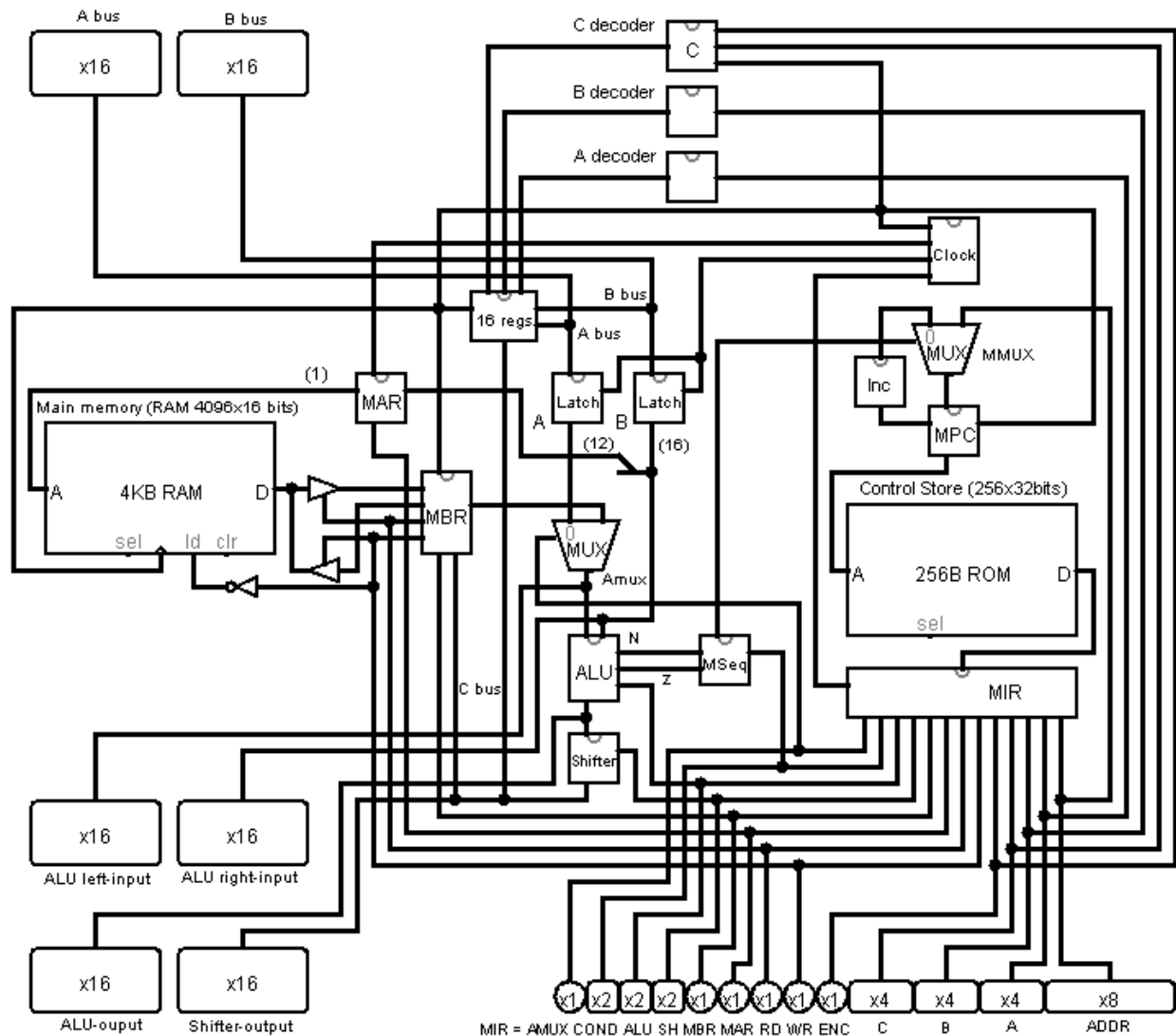
Vi kiến trúc đầy đủ (như Hình 4-09 trong giáo trình Kiến trúc máy tính)

- Khi bắt đầu chạy logisim, chúng ta vào thẳng ngay cửa sổ con “Canvas Window” của mạch chính là Main. Nếu chúng ta đang làm việc với một mạch con khác, hãy kích vào tên Main. Trong cửa sổ con “Canvas Window” của logisim, xây dựng vi kiến trúc đầy đủ như hình vẽ trên.
- Kiểm tra vi kiến trúc:

Sau khi đã xây dựng một công trình “đồ sộ” và “tinh vi” như thế này, việc kiểm tra xem các đơn vị cấu thành có thực hiện đúng chức năng theo thiết kế hay không, việc lắp ghép chúng có sai sót gì không thực sự là một việc quan trọng và không dễ dàng.

Để việc kiểm tra được thuận lợi, trước khi kiểm tra chúng ta sẽ bổ sung thêm bộ nhớ chính (RAM) cùng với các phần tử để ghép nối RAM với vi kiến trúc và một số phần tử Output để chúng ta có thể quan sát sự hoạt động của vi kiến trúc được thuận lợi. (Sinh viên khi thực hành hãy dựa vào hình vẽ dưới đây để bổ sung các phần tử nói trên vào vi kiến trúc của mình).

10.3. Bổ sung bộ nhớ chính và lắp ghép với vi kiến trúc



10.4. Kiểm tra hệ thống máy tính (New !!!)

Để kiểm tra hệ thống máy tính, cách tốt nhất là cho chạy thử một hoặc một số chương trình, sao cho tất cả 23 vĩ chỉ thị (xem mục “4.3.3 Tập vĩ chỉ thị”, trong giáo trình “Kiến trúc máy tính”) đều được lấy về (Fetch), giải mã (Decode) và thực hiện (Execute). Điều đó cũng có nghĩa là toàn bộ các vĩ chỉ thị của vi chương trình với chức năng thông dịch mà chúng ta đã viết (xem mục “4.4.2 Thí dụ về một vi chương trình”) đều được thực hiện.

Nếu tất cả các chương trình đều thực hiện đúng các yêu cầu của thuật toán mà nó thể hiện, thì có thể kết luận rằng hệ thống máy tính của chúng ta đã được thiết kế và thi công đúng đắn. Hệ thống máy tính nói ở đây bao gồm 2 thành phần: 1/ Phần cứng: đó là vi kiến trúc như ở hình vẽ trên; 2/ Phần mềm hệ thống: đó là vi chương trình thực hiện chức năng thông dịch mà chúng ta đã viết và nạp vào Control Store.

Để kiểm tra hệ thống máy tính, chúng ta sẽ viết và cho chạy một số chương trình, mỗi chương trình sẽ sử dụng một số chỉ thị trong tập 23 vĩ chỉ thị. Sau khi viết xong, chúng ta sẽ tự tay dịch nó sang dạng số nhị phân và dạng số hexa, sau đó nạp (load) nó vào trong bộ nhớ chính (main memory) và cho thi hành. Trong quá trình chương trình đang được thi hành, chúng ta có thể theo dõi tất cả các thành phần của hệ thống máy tính kể cả các thành phần mới bổ sung. Khi chương trình được thi hành xong, chúng ta sẽ kiểm tra kết quả (nằm trong bộ nhớ chính) xem có đúng không.

10.4.1 Chương trình kiểm tra thứ nhất - test-01-addition.txt

- Mô tả chương trình:
 - Input: Cho 2 số 1234h và 1010h
 - Output: Tổng của chúng
- Phân tích việc viết chương trình bằng vĩ chỉ thị:

Với một bài toán cộng đơn giản như thế này, không có gì đáng để nói về thuật toán giải quyết, chúng ta có thể nhẩm ra ngay kết quả: $1234h + 1010h = 2244h$.

Tuy nhiên để lập trình cho hệ thống máy tính đơn giản của chúng ta, có lẽ chỉ có một kiểu chương trình thích hợp nhất, đó là kiểu chương trình dạng “.com” theo cách gọi của những người lập trình Hợp ngữ (Assembly language). Các chương trình dạng “.com” có các đặc điểm chính sau:

1. Các chỉ thị của chương trình và dữ liệu của chương trình chiếm một vùng các ô nhớ liên tiếp.
2. Ô nhớ đầu tiên luôn là một lệnh nhảy vô điều kiện (JMP) qua vùng các ô nhớ chứa dữ liệu nằm ngay sau nó. Trong vùng chứa dữ liệu, một số ô nhớ sẽ chứa các toán hạng cho các chỉ thị của chương trình (giống như các ô nhớ tương ứng với các biến và các hằng của các chương trình khả thi được dịch từ chương trình viết bằng C++ hay Pascal); một số ô nhớ khác có thể chứa kết quả.

Ngoài vấn đề cần viết chương trình dạng “.com”, còn có một vấn đề khác, đó là chương trình của chúng ta sẽ kết thúc như thế nào? Hay nói cách khác sau khi thực hiện xong, máy tính

của chúng ta có dừng được không và dừng như thế nào. Người lập trình trên các máy tính có hệ điều hành nói chung không cần quan tâm đến vấn đề này.

Trong phạm vi môn học này, để cho đơn giản, chúng ta sẽ “dừng” chương trình của mình bằng cách viết một lệnh nhảy vô điều kiện đến chính nó, nghĩa là nhảy đến ô nhớ chứa chính lệnh nhảy này.

- Viết chương trình:
 - Ô nhớ có địa chỉ 000 chứa lệnh (vĩ chỉ thị) nhảy qua vùng dữ liệu.
 - Ô nhớ có địa chỉ 001 chứa dữ liệu là con số 1234h.
 - Ô nhớ có địa chỉ 002 chứa dữ liệu là con số 1010h.
 - Ô nhớ có địa chỉ 003 chứa dữ liệu là kết quả của phép cộng 2 số nói trên. Ban đầu chúng ta thiết lập cho nó giá trị 0, giống như với nhiều ngôn ngữ lập trình bậc cao, các biến chưa được khởi tạo thường được cho bằng 0.
 - Ô nhớ có địa chỉ 004 chứa chỉ thị LODD với địa chỉ toán hạng bộ nhớ là 001.
 - Ô nhớ có địa chỉ 005 chứa chỉ thị ADDD với địa chỉ toán hạng bộ nhớ là 002.
 - Ô nhớ có địa chỉ 006 chứa chỉ thị STOD với địa chỉ toán hạng bộ nhớ là 003.
 - Ô nhớ có địa chỉ 007 chứa chỉ thị JMP với địa chỉ nhảy đến là 007.

Chương trình của chúng ta được trình bày trong bảng dưới đây; ý nghĩa của các cột như sau:

- Cột đầu tiên (Memory address) là địa chỉ bộ nhớ cần nạp chương trình vào.
- Cột thứ 2 (Content) là chỉ thị viết dưới dạng số hexa.
- Cột thứ 3 (MAC-1 instruction) là ký hiệu vĩ chỉ thị tương ứng và địa chỉ toán hạng X.
- Cột ngoài cùng bên phải (Equivalent Micro Instruction) là nhóm các vĩ chỉ thị tương ứng với vĩ chỉ thị ghi ở cột 2.

Memory address	Content (Hex)	MAC-1 instruction	Equivalent Micro Instruction
000	6004	JMP ;X=004	19: tir:=lshift(tir); if n then goto 25; 25: alu := tir; if n then goto 27; {0110 or 0111?} 26: pc := band(ir, amask); goto 0; {pc := x}
001	1234		
002	1010		
003	0000		
004	0001	LODD ;X=001	6: mar := ir; rd; {mar := operand add, and read} 7: rd; {keep on reading} 8: ac := mbr; goto 0; {ac := Operand}
005	2002	ADDD ;X=002	11: alu := tir; if n then goto 15; {0010 or 0011?} 12: mar := ir; rd; {mar := operand add, and read} 13: rd; {keep on reading} 14: ac := mbr + ac; goto 0; {going to fetch a next instruction}
006	1003	STOD ;X=003	9: mar := ir; mbr := ac; wr; {mar := operand add, and write} 10: wr; goto 0; {keep on writing, going to fetch a next instruction}
007	6007	JMP ;X=007	

- Xác định tập vi chỉ thị sẽ thực hiện chương trình bằng vĩ chỉ thị:

Qua việc phân tích ở trên, có thể thấy rằng để thực hiện chương trình của chúng ta, các vi chỉ thị sau đây trong Control Store sẽ được thi hành. Trong số đó các chỉ thị ở dòng số 0, 1, 2, 3, 4 và 5 sẽ được thi hành nhiều lần vì chúng thuộc các bước “Fetch” và “Decode” của việc thực hiện mỗi dòng lệnh trong chương trình của chúng ta.

```

0: mar := pc; rd; {main loop. Fetch an Instruction}
1: pc := pc+1; rd; {increment pc, and keep on reading}
2: ir := mbr; if n then goto 28; {save, if opcode = 1xxx.12x then goto 28}
3: tir := lshift(ir+ir); if n then goto 19; {if opcode = 01xx.12x then goto 19 else next line}
4: tir := lshift(tir); if n then goto 11; {000x or 001x?}
5: alu := tir; if n then goto 9; {0000 or 0001?}

```

```

19: tir:=lshift(tir); if n then goto 25;
25: alu := tir; if n then goto 27; {0110 or 0111?}
26: pc := band(ir, amask); goto 0; {pc := x}

```

```

6: mar := ir; rd; {mar := operand add, and read}
7: rd; {keep on reading}
8: ac := mbr; goto 0; {ac := Operand}

```

```

11: alu := tir; if n then goto 15; {0010 or 0011?}
12: mar := ir; rd; {mar := operand add, and read}
13: rd; {keep on reading}
14: ac := mbr + ac; goto 0; {going to fetch a next instruction}

```

```

9: mar := ir; mbr := ac; wr; {mar := operand add, and write}
10: wr; goto 0; {keep on writing, going to fetch a next instruction}

```

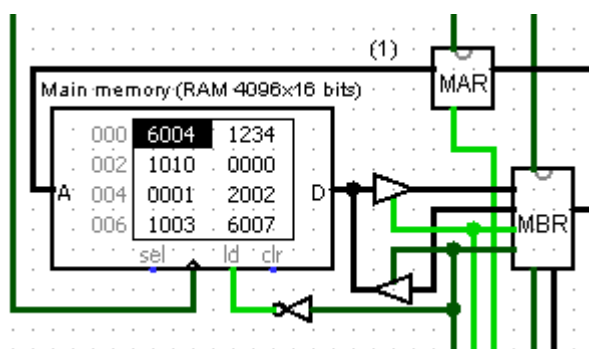
- Nạp chương trình vào bộ nhớ chính:

Có 2 cách:

1. Trong cửa sổ Main của logisim, trở vào phần tử Main Memory (RAM) rồi kích nút chuột phải, chọn “Edit contents...”, Logisim sẽ hiện lên cửa sổ “Hex Editor” để cho ta soạn thảo nội dung từng ô nhớ bằng số Hexa. Để đóng cửa sổ, kích vào nút “Close Windows”. Trong khi đang soạn thảo hoặc trước khi kết thúc soạn thảo nên thực hiện thao tác “Save” để ghi nội dung Main Memory vào một file. Trong chương trình kiểm tra thứ nhất này, chúng ta đặt tên file là test-01-addition.txt.
2. Dùng một chương trình soạn thảo văn bản tạo ra một file kiểu Plain Text, đặt tên file là: test-01-addition.txt. Nội dung file gồm các word ở cột 2 – “Content (Hex)” của bảng trên, các word cách nhau bởi ký tự trống (ký tự cách). Trong cửa sổ Main của logisim,

trở vào phần tử Main Memory (RAM) rồi kích nút chuột phải, chọn “Load Image...”, Logisim sẽ hiện lên cửa sổ để ta tìm file cần load.

Hình ảnh của bộ nhớ Main Memory sau khi nạp chương trình như sau:



- Cách cho chạy chương trình:

Sau khi đã nạp chương trình vào bộ nhớ, chúng ta cần chọn: Simulate \ Simulation Enabled để có thể chạy mô phỏng chương trình. Chúng ta có thể cho chạy chương trình theo một trong hai cách:

1. Chạy liên một mạch: Simulate \ Tick Enabled. Chỉ nên chạy chương trình theo cách này khi đã từng chạy chương trình thành công.
2. Chạy từng bước: Simulate \ Tick Once. Theo cách này toàn bộ hệ thống chỉ thực hiện công việc trong 1 chu kỳ con của đơn vị đồng hồ rồi dừng lại. Cách chạy này giúp chúng ta dễ dàng gỡ rối (debug) chương trình cũng như có thể phát hiện các sai sót về thiết kế của các đơn vị tạo nên hệ thống cũng như việc ghép nối các đơn vị.

- Giám sát việc chạy chương trình:

Cần cho chạy chương trình theo từng bước, mỗi khi chương trình dừng lại, chúng ta cần xem nội dung các thanh ghi, các ô nhớ... mà chúng ta biết chắc rằng vì chỉ thị làm thay đổi, hoặc không làm thay đổi so với ở chu kỳ con trước đó, để xem xem có đúng như vậy hay không.

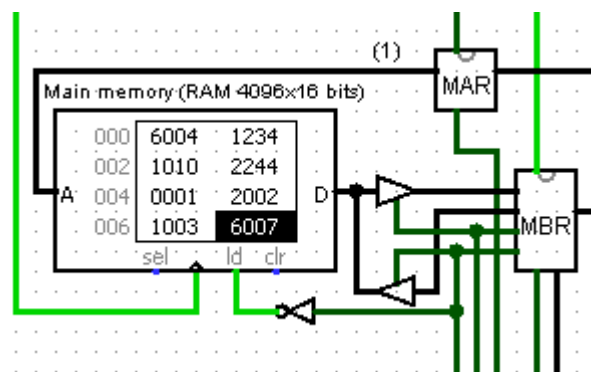
Thí dụ:

- Ở chu kỳ con đầu tiên (sau khi nạp chương trình vào Main memory): mar = 1, rd = 1, z = 1 và đầu ra chu kỳ con thứ nhất (của đơn vị đồng hồ) = 1.
- Ở chu kỳ con thứ 2: mar = 1, rd = 1, z = 1 và đầu ra chu kỳ con thứ hai = 1.
- Ở chu kỳ con thứ 3: mar = 1, rd = 1, z = 1, nội dung thanh ghi MAR=000 (địa chỉ chỉ thị đầu tiên trong Main Memory) và đầu ra chu kỳ con thứ ba = 1.
- Ở chu kỳ con thứ 4: mar = 1, rd = 1, z = 1, nội dung thanh ghi MAR=000, MBR=000 và đầu ra chu kỳ con thứ tư = 1.
- Ở chu kỳ con thứ 1 của vi chỉ thị kế tiếp: mar = 0, rd = 1, z = 1, ALU = 00 (phép cộng), ENC = 1, A = 0000 (chọn thanh ghi số 0 là pc để đưa nội dung ra bus A), B = 0110 (chọn thanh ghi số 6 là +1 để đưa nội dung ra bus B), C = 0000 (chọn thanh ghi số 0 là pc để chứa kết quả trên bus C)
- nội dung thanh ghi MAR=000, MBR=000 và đầu ra chu kỳ con thứ tư = 1.
- V.v.

Đây là một công việc khó, đòi hỏi sự nắm vững kiến thức về toàn bộ hệ thống, tập trung tư tưởng và rèn luyện để tích lũy kinh nghiệm.

Các bạn sinh viên cần dành đủ thời gian để giám sát việc thực hiện chương trình theo tất cả các bước, từ bước đầu tiên cho đến khi nhìn thấy tại ô nhớ 003 có giá trị bằng 2244 (kết quả của phép cộng). Sau khi thấy kết quả đúng và tiếp tục chạy các bước tiếp theo, nếu thấy hình vẽ bộ nhớ chính như dưới đây và không thay đổi nữa, thì có thể kết luận rằng:

1. Chương trình mà chúng ta xây dựng để cộng 2 số là đúng và đã “dừng”.
2. Các vi chỉ thị đã được lấy từ Control Store ra để thực hiện chương trình của chúng ta viết đã được thiết kế đúng.
3. Các thành phần của hệ thống thuộc đường dữ liệu và đơn vị điều khiển đã tham gia trực tiếp vào việc thi hành chương trình đã được xây dựng và ghép nối với nhau một cách đúng đắn.



10.4.2 Chương trình kiểm tra thứ hai: **under construction**

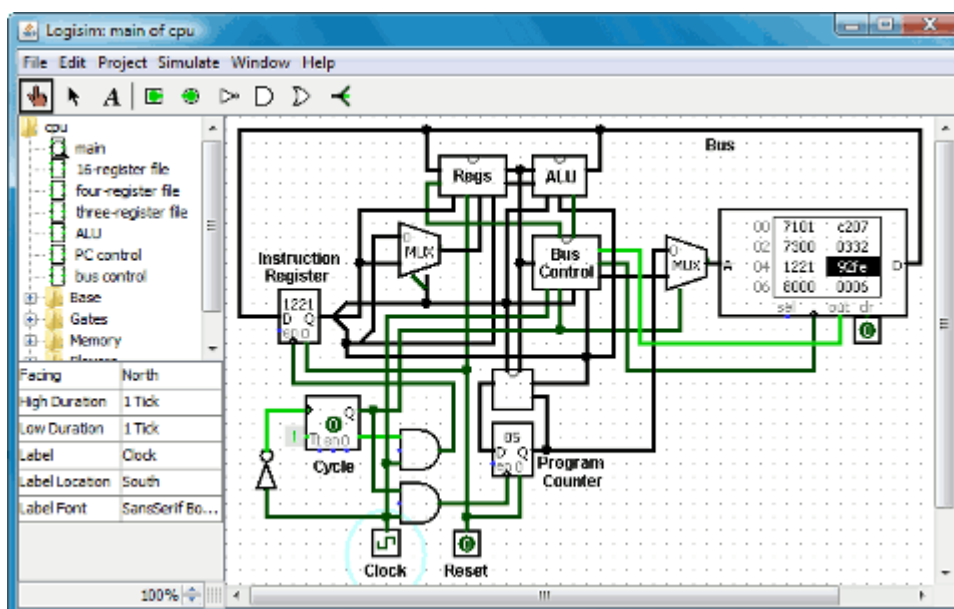
Nhóm tác giả của giáo trình hướng dẫn thực hành này sẽ bổ sung thêm các chương trình kiểm tra trong các năm học tiếp theo.

PHỤ LỤC A: HƯỚNG DẪN SỬ DỤNG LOGISIM

Logisim có các hướng dẫn rất tốt; khi đang chạy logisim, có thể mở “Help” để xem các tài liệu hướng dẫn sau:

- The Guide to Being a Logisim User
- Beginner’s tutorial
- Library Reference

Sau đây là bản dịch tài liệu “The Guide to Being a Logisim User” của một nhóm sinh viên K53CA, để chuẩn bị cho báo cáo seminar của nhóm thuộc môn học Kiến trúc máy tính. Bản dịch này còn một số lỗi và thiếu sót, nhưng có thể dùng tạm được nếu người đọc “ngại” đọc bản tiếng Anh.



Logisim là một công cụ giáo dục dùng cho việc thiết kế và mô phỏng các mạch logic số. Với giao diện đơn giản dạng thanh công cụ và việc mô phỏng các mạch giống như mạch được xây dựng, sẽ rất dễ dàng để đơn giản hóa việc học những tư tưởng cơ bản nhất liên quan đến mạch logic. Với chức năng xây dựng các mạch lớn từ các mạch con nhỏ hơn và vẽ các bó dây chỉ với một cú rê chuột, Logisim có thể được sử dụng(và đang được sử dụng) để thiết kế và mô phỏng toàn bộ các CPU cho mục đích giáo dục.

Sinh viên ở các trường đại học và cao đẳng trên toàn thế giới sử dụng Logisim cho rất nhiều mục đích, trong đó có:

- Một mô đun trong tổng quan khoa học máy tính đại cương

- Một đơn vị học trình trong khóa học tổ chức hệ thống máy tính của sinh viên năm thứ hai
- Trong suốt học kỳ của khóa học kiến trúc máy tính ở mức độ chuyên sâu.

Hướng dẫn sử dụng Logisim mà bạn đang đọc là tài liệu tham khảo chính thức cho các tính năng của Logisim. Phần thứ nhất của tài liệu là một chuỗi các đề mục giới thiệu những phần chính của Logisim. Các đề mục này được viết sao cho chúng được đọc kỹ càng để học tất cả những tính năng quan trọng nhất của Logisim.

Những đề mục còn lại là một tập hợp của các tư liệu tham khảo và chú thích của một vài khía cạnh nhỏ hơn của Logisim.

1. CHỈ DẪN CHO NGƯỜI MỚI SỬ DỤNG

Chào mừng bạn đến với Logisim

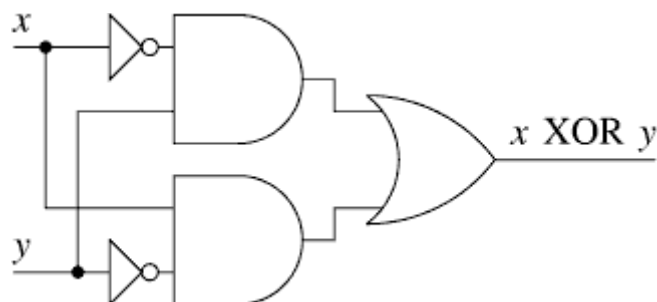
Logisim cho phép bạn thiết kế và mô phỏng mạch các kỹ thuật số. Nó được định hướng như một công cụ giáo dục, sử dụng để giúp đỡ cho việc học các mạch hoạt động như thế nào.

To practice using Logisim, let's build a XOR circuit - that is, a circuit that takes two inputs (which we'll call x and y) and outputs 0 if the inputs are the same and 1 if they are different. The following truth table illustrates.

Đề tập sử dụng Logisim, chúng ta sẽ xây dựng một mạch XOR- một mạch lấy 2 giá trị đầu vào (x và y) và cho giá trị đầu ra là 0 nếu x và y giống nhau, 1 nếu x và y khác nhau. Bảng chân lý sau đây thể hiện.

x	y	$x \text{ XOR } y$
0	0	0
0	1	1
1	0	1
1	1	0

Chúng ta nên thiết kế mạch như vậy trên giấy.

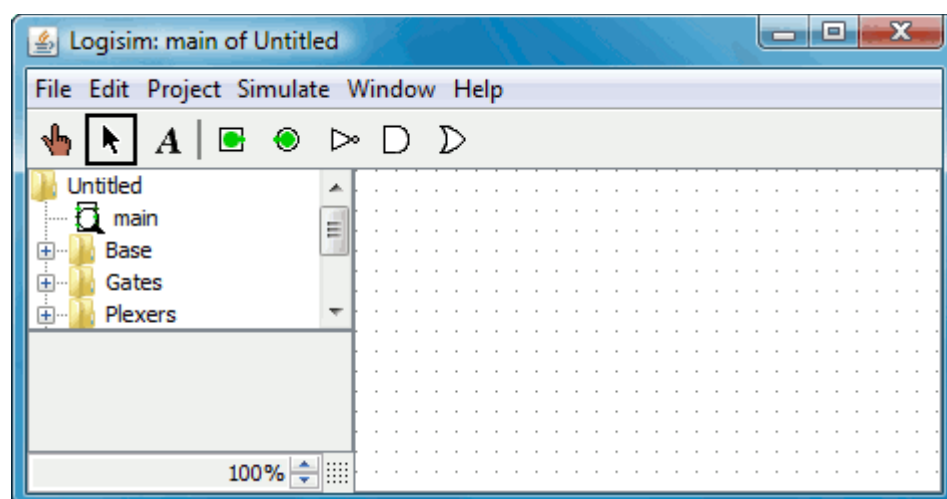


Tuy nhiên bởi vì mạch được vẽ trên giấy không có nghĩa mạch đó là chính xác. Để xác minh mạch đó, chúng ta sẽ cõ nó trên Logisim và kiểm tra. Như một lợi thế thêm nữa, chúng ta sẽ có một mạch đẹp hơn cái mà ta sẽ có nếu vẽ bằng tay.

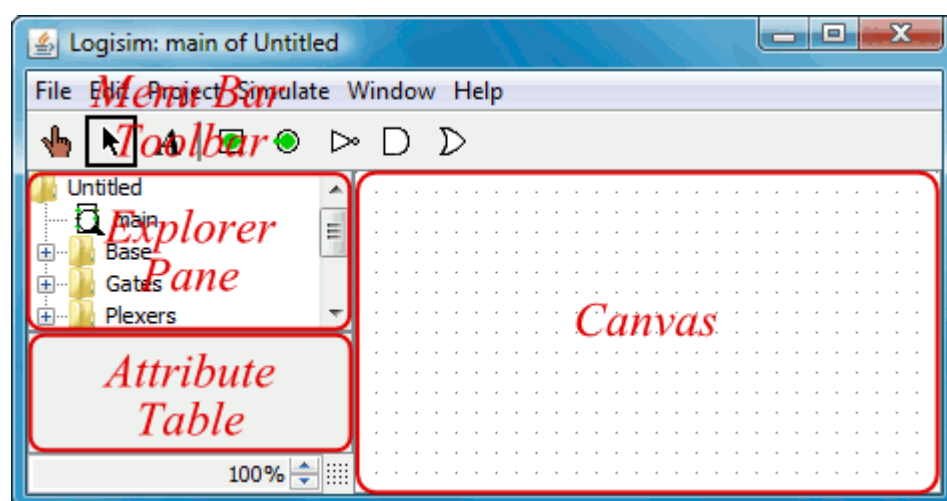
Enjoy your circuit-building!

1.1 Bước 0: Tự định hướng chính bản thân bạn

Khi khởi động Logisim, bạn sẽ thấy một cửa sổ gần giống hình bên dưới. Một vài chi tiết có thể khác nhau một chút vì bạn có thể sử dụng hệ thống khác so với của tôi.



Logisim được chia thành 3 phần, *explorer pane*, *attribute table* (bảng thuộc tính) và *canvas*. Bên trên 3 phần này là thanh menu (*menu bar*) và thanh công cụ (*toolbar*).

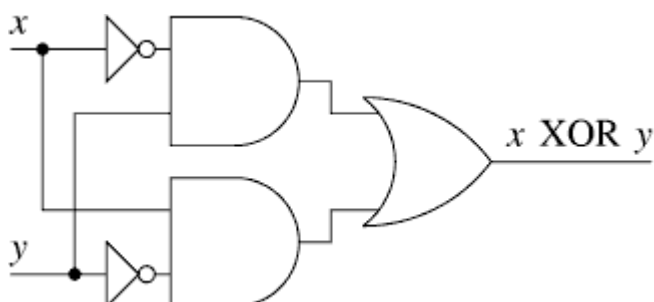


Chúng ta có thể nhanh chóng bỏ qua *explorer pane* và *attribute table*: chúng ta sẽ không kiểm tra chúng trong khuôn khổ bản hướng sử dụng dẫn này và bạn chỉ có thể bỏ qua chúng. Cùng với đó, *menu bar* sẽ được các bạn tự khám phá.

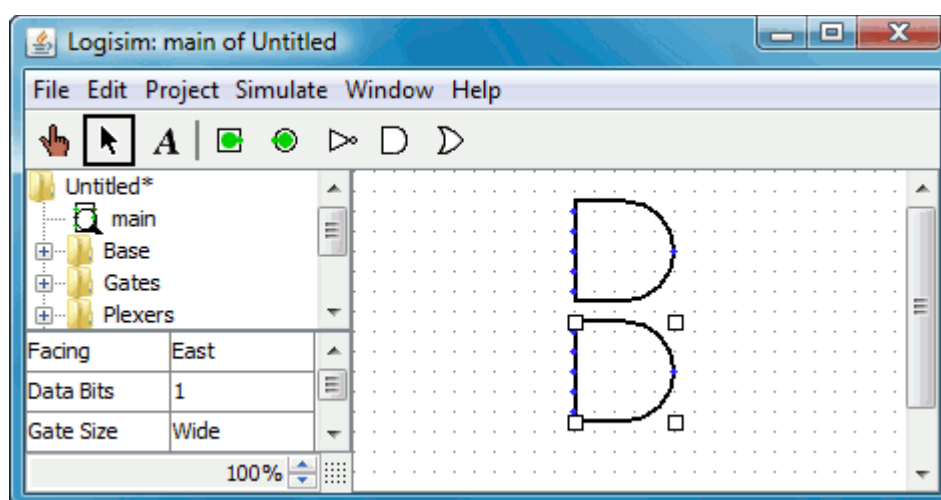
Còn lại *toolbar* và *canvas*. *Canvas* là nơi bạn có thể vẽ mạch của mình và *toolbar* bao gồm các công cụ mà bạn sẽ sử dụng để hoàn thành nó.

1.2 Bước 1: Thêm các cổng



Hãy nhớ lại rằng chúng ta đang cố gắng xây dựng mạch sau trên Logisim.

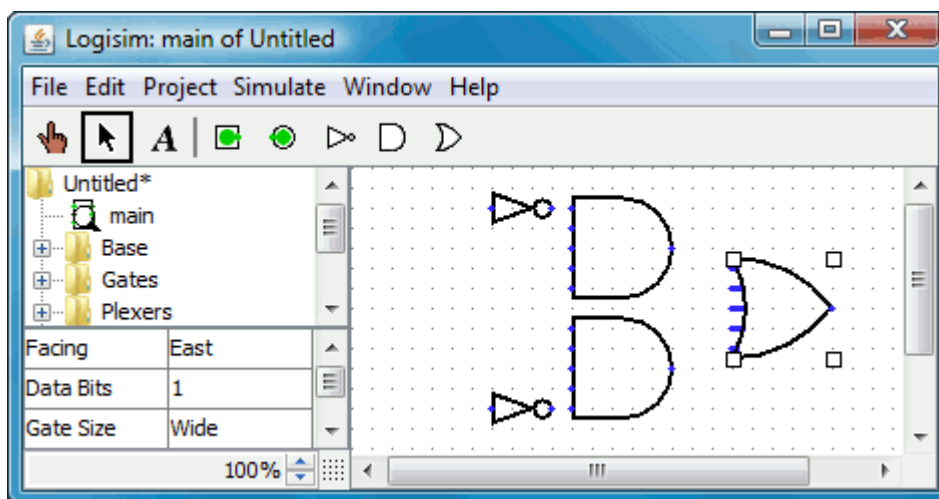


Tôi khuyến cáo việc xây dựng mạch bằng cách chèn các cổng vào trước giống như sự sắp xếp khung xương và sau đó mới liên kết chúng lại bằng dây nối. Việc trước tiên ta sẽ làm là chèn thêm 2 cổng AND. Kích chuột vào công cụ AND (AND tool) trên thanh công cụ. Sau đó kích chuột vào nơi mà bạn muốn cổng AND thứ nhất đi đến. Hãy chắc chắn rằng bạn đã dành đủ chỗ cho phần bên trái. Sau đó bạn tiếp tục kích chuột vào AND tool một lần nữa và đặt cổng AND thứ hai bên dưới cổng thứ nhất.



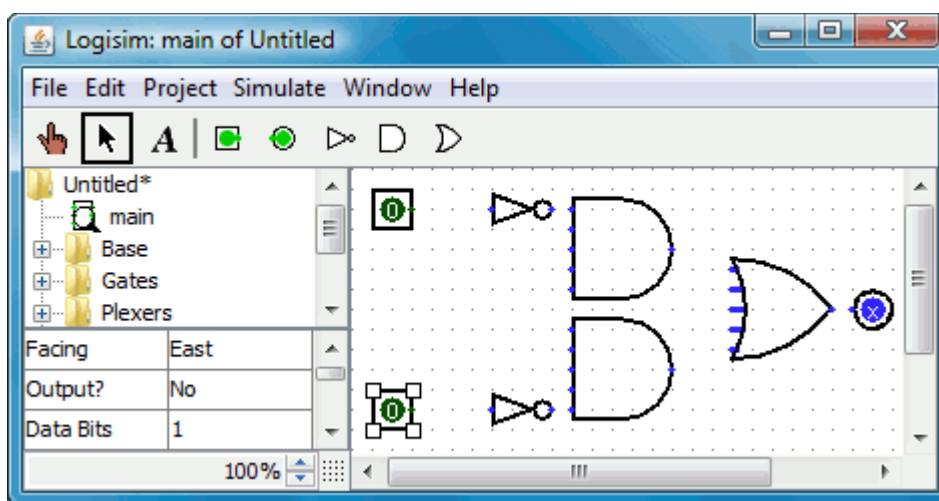
Đề ý đến 5 dấu chấm ở phía bên trái của cổng AND. Chúng là những điểm mà dây nối có thể được gắn. Tình cờ chúng ta sẽ chỉ dùng 2 trong số chúng cho mạch XOR của chúng ta tuy nhiên đối với các mạch khác, bạn sẽ nhận ra rằng có nhiều hơn 2 dây trong một cổng AND là hữu dụng.

Bây giờ, thêm 1 cổng khác vào mạch. Kích vào OR tool (); , sau đó kích vài nơi bạn muốn đặt nó và đặt 2 cổng NOT trên bản vẽ sử dụng NOT tool (.



Tôi để dành 1 ít khoảng trống giữa các cổng NOT và các cổng AND, nếu bạn muốn, bạn có thể đặt chúng ngược nhau và tiết kiệm công sức liên kết chúng lại với nhau bằng dây nối sau này.

Bây giờ chúng ta muốn thêm 2 giá trị đầu vào x và y vào bản vẽ. Chọn *Input tool* (□) và đóng các chốt xuống. Bạn cũng nên đặt một chốt đầu ra bên cạnh cổng đầu ra của OR bằng cách dùng *Output tool* (●). (Một lần nữa, tôi chưa lại một chút khoảng trống giữa cổng OR và chốt đầu ra, nhưng bạn nên đặt chúng ngay bên cạnh nhau).



Nếu bạn không thích nơi mà bạn đặt phần tử nào đó, bạn có thể chọn nó bằng *Edit tool* (↔) và thả nó vào phần bản vẽ. Hoặc bạn có thể xóa hoàn toàn phần tử đó bằng cách chọn *Delete* trên bảng chỉnh sửa (*Edit menu*) hoặc ấn phím Delete.

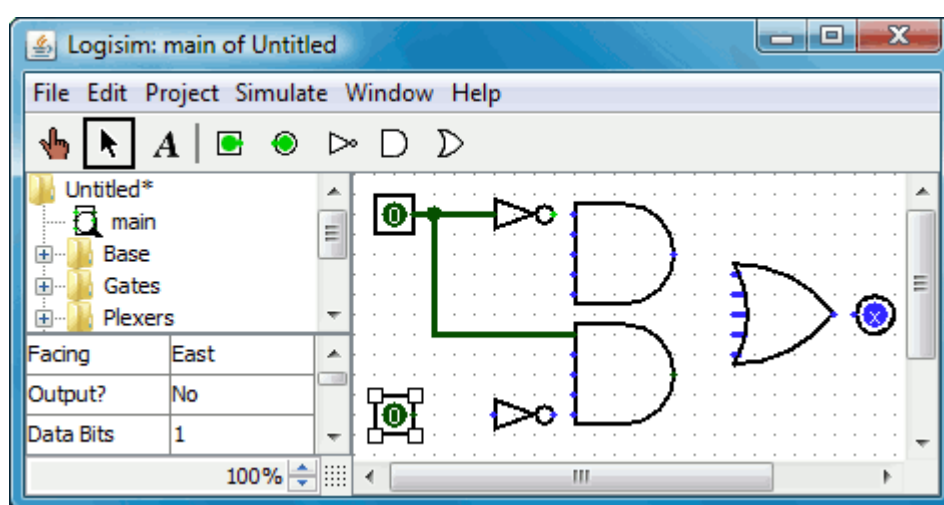
Khi bạn đặt bất cứ thành phần nào của mạch, bạn sẽ nhận thấy rằng ngay khi thành phần đó được đặt vào, Logisim quay lại *Edit tool* thể nên bạn có thể di chuyển thành phần mới được đặt vào đó hoặc kết nối nó với các thành phần khác bằng cách tạo ra các dây nối. Nếu như bạn muốn thêm vào một bản sao của thành phần mới thêm vào, một phím tắt là tổ hợp Ctrl + D để nhân đôi thành phần được chọn. (Một vài máy tính sử dụng phím khác cho *menus*, ví dụ như Command key trên máy Macintoshes. Bạn có thể bấm phím đó với phím “D”).

1.3 Bước 2: Thêm dây nối

Sau khi đã có tất cả các thành phần được thiết lập trên bản vẽ, bạn đã có thể bắt đầu thêm các đoạn dây nối. Chọn *Edit Tool* (☞). Khi con trỏ ở trên điểm có thể nối dây, một vòng tròn nhỏ màu xanh sẽ bao quanh điểm đó. Nhấn chuột ở điểm đó và kéo dây đến nơi bạn muốn.

Logisim khá thông minh trong việc chèn dây nối. Bất cứ khi nào một dây kết thúc trên dây khác, Logisim sẽ tự động nối chúng lại với nhau. Bạn cũng có thể kéo dài hoặc thu gọn một dây nối bằng cách kéo một trong hai đầu của dây, sử dụng *writing tool*.

Dây nối trong Logisim chỉ có thể nằm ngang hoặc dọc. Để có thể nối đầu vào ở trên với cổng NOT và cổng AND, tôi cần dùng thêm 3 dây khác nhau.

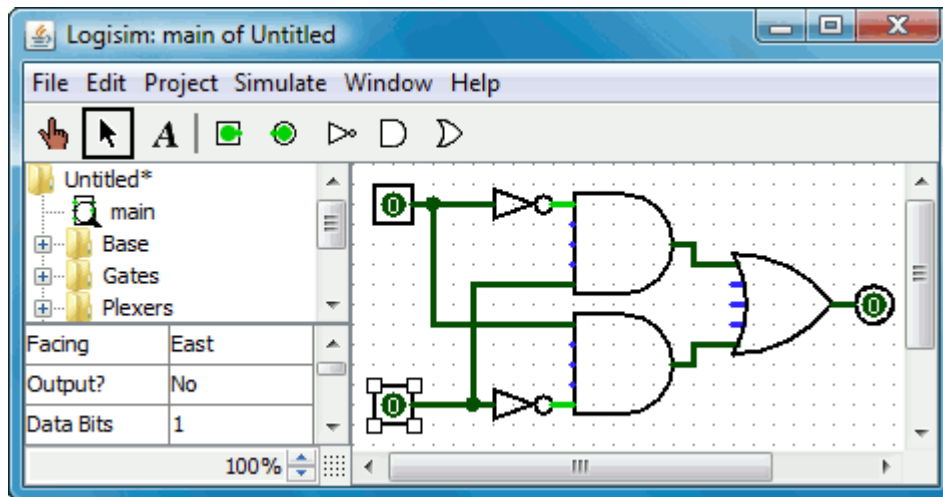


Logisim tự động nối các dây với các cổng và các dây với nhau. Trong đó gồm có việc tự động vẽ chấm tròn ở phần giao nhau hình chữ T của các dây như trên hình, thể hiện rằng các dây đã được nối với nhau.

Khi vẽ các dây nối, bạn có thể thấy một vài dây màu xanh da trời hoặc dây màu xám. Màu xanh da trời thể hiện rằng giá trị ở một điểm là không xác định, màu xám thể hiện rằng dây chưa được nối với bất cứ phần tử nào. Đây không phải là vấn đề lớn trong quá trình xây dựng mạch của bạn. Tuy nhiên khi bạn hoàn thành mạch của mình, các dây không được có màu xanh hoặc xám. (Chân chưa được kết nối của cổng OR sẽ vẫn có màu xanh, điều đó là chấp nhận được).

Nếu bạn có một dây màu xanh hay xám khi mà bạn cho rằng tất cả các dây đã được nối với nhau, thì cũng có nghĩa là đã có nhầm lẫn ở đâu đó. Việc nối dây ở đúng chỗ là rất quan trọng. Logisim vẽ các dấu chấm nhỏ trên các phần tử để chỉ ra nơi các dây có thể được nối tới. Khi bạn thực hiện, bạn sẽ thấy những dấu chấm chuyển từ màu xanh sang màu xanh nhạt hoặc xanh thẫm.

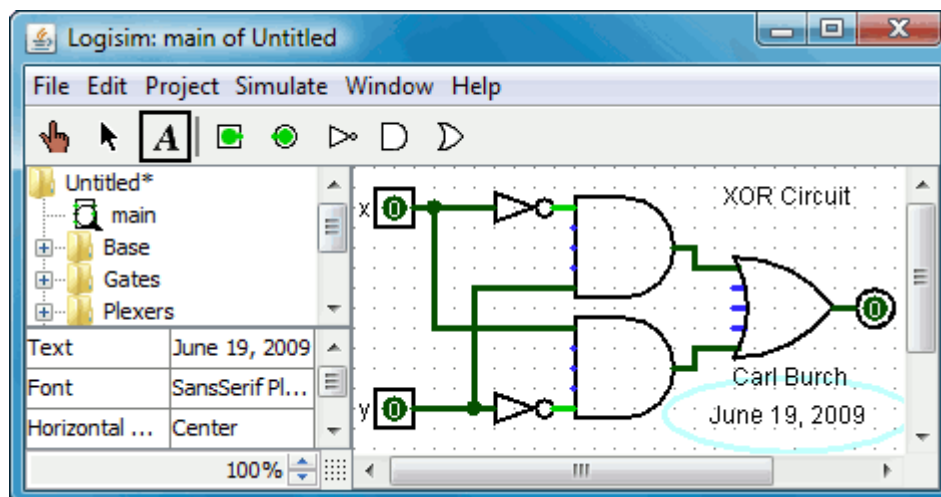
Khi tất cả các dây đã được kết nối, chúng sẽ có màu xanh nhạt hoặc xanh thẫm.



1.4 Bước 3: Chèn text

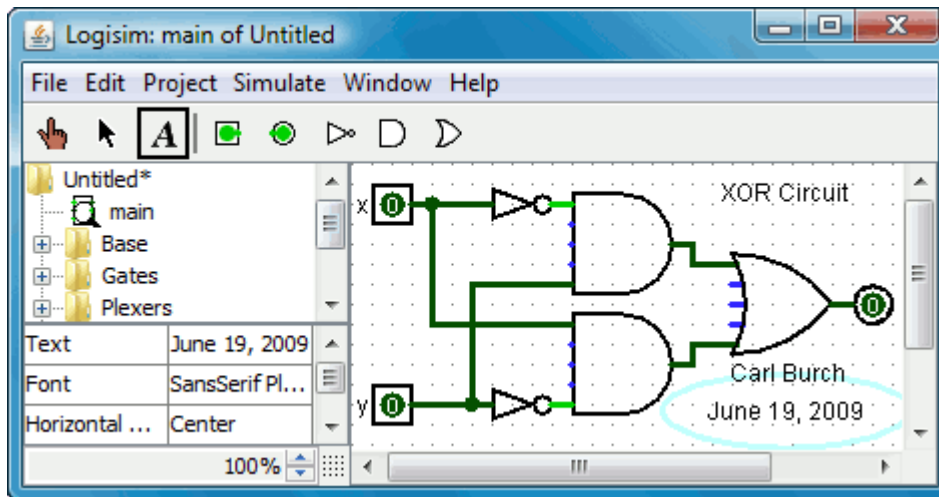
Để mạch hoạt động được, việc chèn text là không cần thiết; tuy nhiên nếu bạn muốn giới thiệu mạch của bạn cho người khác, các nhãn sẽ giúp bạn thể hiện mục đích của các phần nhỏ trong mạch của bạn.

Chọn *text tool* (A). Bạn có thể kích chuột vào chốt đầu vào và bắt đầu đặt cho nó một cái nhãn. (Sẽ là tốt hơn nếu bạn kích trực tiếp vào chốt đầu vào chứ không phải nơi bạn muốn đoạn text đi đến bởi vì nhãn sẽ di chuyển theo chốt). Bạn có thể làm tương tự đối với chốt đầu ra. Hoặc bạn có thể chỉ kích chuột vào vị trí cũ và gỡ text để đặt nhãn ở bất cứ nơi nào khác.



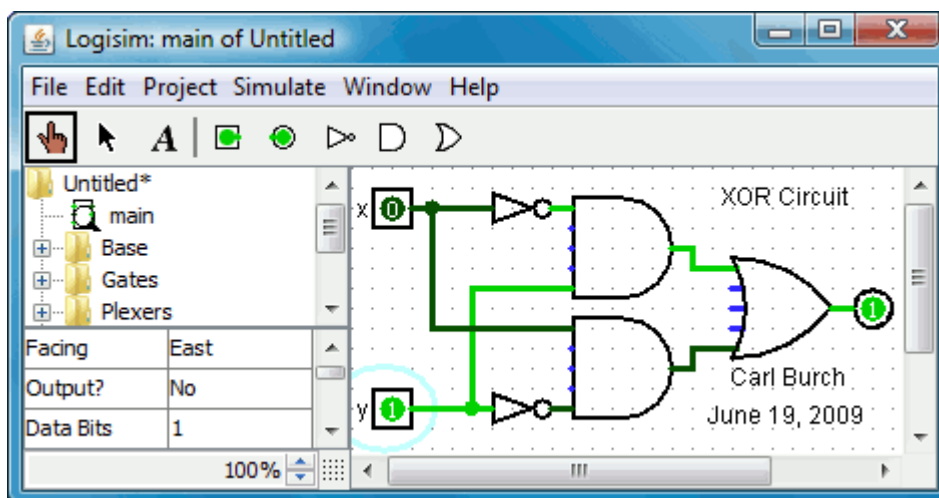
1.5 Bước 4: Kiểm tra mạch

Bước cuối cùng là kiểm tra lại mạch để chắc chắn rằng nó thực sự làm những thứ mà chúng ta dự định. Logisim đã mô phỏng mạch. Hãy chú ý xem chúng ta đang ở đâu



Lưu ý rằng các cả 2 chốt đầu vào và chốt đầu ra đều đang chứa các số 0. Điều đó nói lên rằng mạch đó đã tính ra giá trị 0 khi cả hai giá trị đầu vào đều là 0.

Bây giờ ta sẽ thử một tổ hợp giá trị đầu vào khác. Chọn *poke tool* (👉) và thay đổi giá trị đầu vào bằng cách kích vào đầu vào đó. Mỗi lần bạn thay đổi đầu vào, giá trị của nó sẽ được chốt lại. Chúng ta đầu tiên sẽ ấn vào đầu vào bên dưới



Khi bạn thay đổi giá trị đầu vào, Logisim sẽ chỉ ra cho bạn giá trị nào di chuyển theo dây nối bằng cách vẽ chúng màu xanh nhạt cho giá trị 1 và xanh thẫm (gần như màu đen) cho giá trị 0. Bạn cũng có thể nhận thấy rằng giá trị đầu ra đã chuyển thành 1.

Chúng ta đã kiểm tra 2 dòng đầu tiên của bảng chân lý, và các giá trị đầu ra đều thỏa mãn các giá trị được định trước.

x	y	$x \text{ XOR } y$
0	0	0
0	1	1
1	0	1
1	1	0

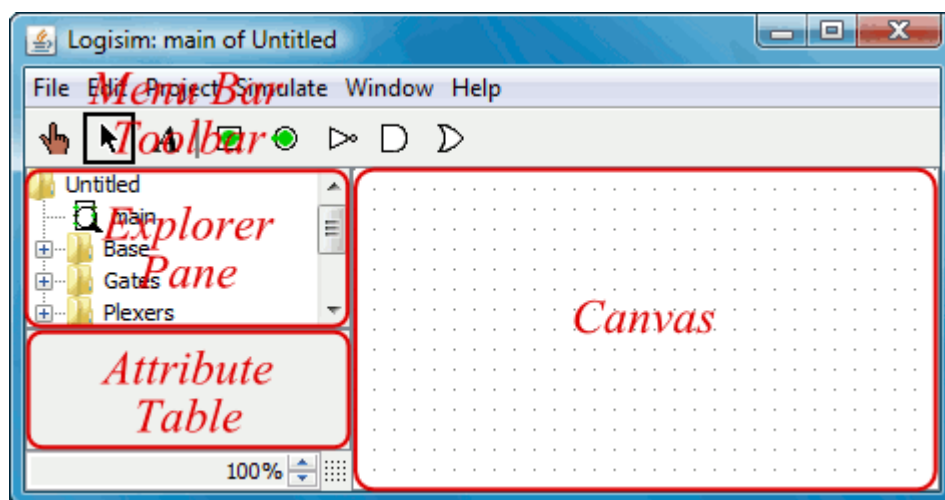
Bằng cách ấn vào các công tắc trên các cách kết hợp khác, chúng ta có thể xác minh 2 dòng còn lại. Nếu các giá trị đều thỏa mãn thì mạch đã được hoàn thành và có thể hoạt động được.

Để hoàn thành công việc, bạn có thể sẽ muốn lưu lại hay in mạch của mình ra. *File menu* cho phép bạn làm việc đó và đương nhiên cũng cho phép bạn thoát khỏi Logisim. Nhưng tại sao lại thoát vào lúc này?

Lúc này bạn đã hoàn thành bài hướng dẫn, bạn có thể thử nghiệm Logisim bằng cách xây dựng các mạch của riêng bạn. Nếu muốn xây dựng các mạch với các tính năng phức tạp hơn, bạn nên xem thêm phần còn lại của mục *Help* để biết có thể làm được những gì khác. Logisim là một công cụ mạnh, nó cho phép bạn xây dựng và thử nghiệm các mạch rất lớn; quá trình từng bước một này mới chỉ phác họa nên lớp bề mặt của Logisim.

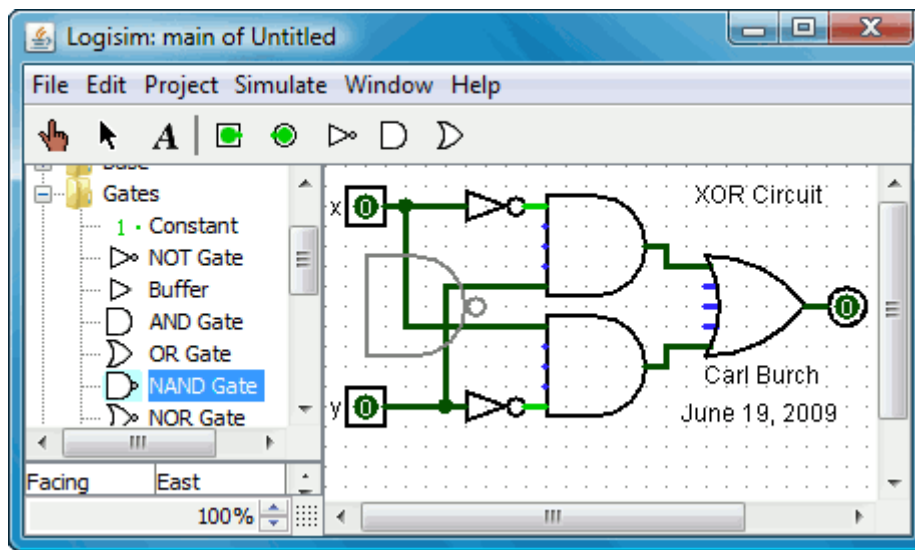
2. THƯ VIỆN VÀ CÁC THUỘC TÍNH

Trong mục này, chúng ta sẽ xem xét cách sử dụng hai lĩnh vực chính khác của cửa sổ Logisim, đó là *explorer pane* và *attribute table*.



2.1 Explorer pane

Logisim sắp đặt các công cụ trong **các thư viện**. Chúng được thể hiện như các thư mục trong *explorer pane*; để tiếp cận các thành phần của thư viện, bạn chỉ cần phải kích đúp vào thư mục tương ứng. Dưới đây, tôi đã mở thư viện các Cổng và chọn công cụ NAND trong đó. Bạn có thể thấy rằng Logisim bây giờ đã có thể thêm các cổng NAND vào mạch.



Nếu bạn nhìn qua các lựa chọn trong thư viện Cổng, bạn sẽ nhận thấy rằng chúng ta không cần phải tự xây dựng mạch XOR như lúc trước: Nó đã được dựng sẵn trong Logisim.

Khi bạn tạo ra một dự án, nó sẽ tự động bao gồm một vài thư viện:

- Base (nền móng): Các công cụ được tích hợp để sử dụng Logisim.
- Gates(các cổng): Các thành phần biểu diễn các chức năng số đơn giản.
- Plexers: Các thành phần ghép nối phức tạp hơn như bộ dồn kênh (multiplexer), bộ giải mã (decoder).
- Arithmetic (số học): Các thành phần biểu diễn số học.
- Memory (bộ nhớ): Các thành phần nhớ dữ liệu như các flip-flop, thanh ghi và RAM.

Logisim cho phép ta thêm các thư viện mới, thông qua thẻ *Load Library* trong mục Project. Bạn có thể thấy rằng Logisim có 3 lớp thư viện.

- **Built-in libraries (thư viện dựng sẵn)** là các thư viện được phân phối cùng với Logisim. Chúng được xác minh trong [Library Reference](#). (tham khảo thư viện).
- **Logisim libraries (thư viện Logisim)** là công trình được dựng bên trong Logisim và được lưu vào ổ đĩa giống như công trình Logisim. Bạn có thể phát triển một tập hợp các mạch trong một công trình đơn lẻ (như đã được giới thiệu tổng mục [Subcircuits](#) của tập hướng dẫn này) sau đó sử dụng tập hợp mạch đó như một thư viện cho các công trình khác.
- **JAR libraries (các thư viện JAR)** Là các thư viện được phát triển trong Java nhưng không được phân phối cùng với Logisim. Bạn có thể download thư viện JAR đã được viết bởi những người khác, hay bạn có thể viết thư viện của riêng bạn như đã được giới thiệu trong mục [JAR Libraries](#) . Phát triển một thư viện JAR khó hơn nhiều so với phát triển một thư viện Logisim, tuy nhiên các thành phần có thể thú vị hơn rất nhiều, bao

gồm những thứ như các thuộc tính và sự tương tác với người sử dụng. Các thư viện dựng sẵn (không giống Base) được viết sử dụng giao diện chương trình ứng dụng (API) giống với các thư viện JAR có thể sử dụng nên chúng biểu hiện một cách xác đáng hệ các chức năng mà các thư viện JAR có thể hỗ trợ.

Một số thư viện JAR được phân phối mà không có thông tin về lớp Java ta sẽ bắt đầu cùng. Khi nạp JAR, Logisim sẽ nhắc bạn gõ vào một tên lớp. Tên lớp đó nên được cung cấp bởi người phân phối file JAR cho bạn.

Để gỡ bỏ một thư viện, chọn *Unload Library...* từ mục *Project*. Logisim sẽ ngăn bạn khỏi việc gỡ bỏ các thư viện chứa các thành phần được sử dụng trong mạch, việc đó sẽ xuất hiện trên thanh công cụ hoặc được ánh xạ đến một nút bấm của con chuột.

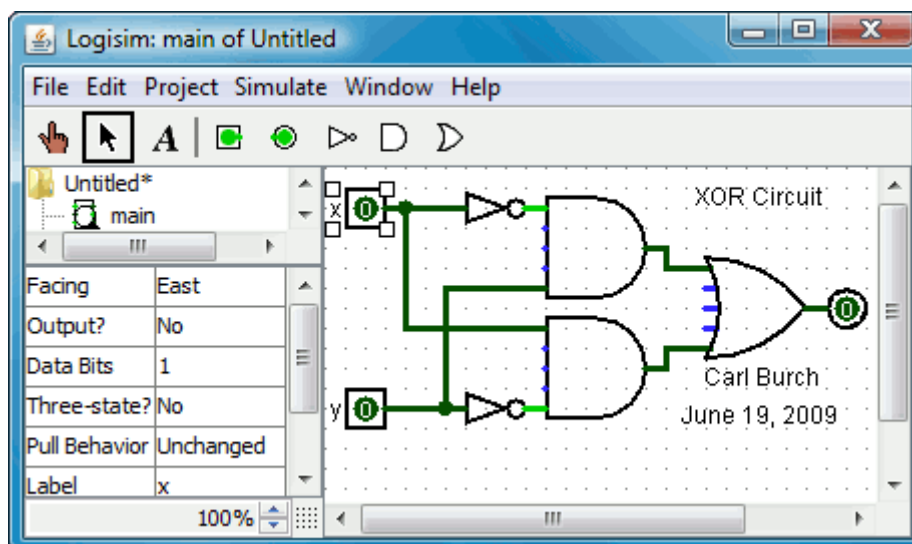
Một cách tình cờ, một thư viện về mặt kỹ thuật chỉ chứa các công cụ mà không chứa các thành phần. Vì thế trong thư viện nền (Base library), bạn sẽ thấy *Poke Tool* (👉), *Edit Tool* (🖱️) và các công cụ khác, chúng không trực tiếp ứng với các thành phần riêng rẽ nào. Hầu hết các thư viện chỉ chứa các công cụ dùng cho việc thêm các thành phần riêng rẽ; tất cả các thư viện dựng sẵn ngoài thư viện nền (Base library) đều giống như vậy.

2.2 Bảng thuộc tính

Nhiều thành phần có các **thuộc tính** riêng, đó các là đặc điểm dùng cho việc cấu hình cách một thành phần chạy hay xuất hiện. **Bảng thuộc tính** được dùng cho việc quan sát và trưng bày giá trị thuộc tính của một thành phần.

Để chọn các thuộc tính của một thành phần mà bạn muốn quan sát, sử dụng *Edit tool* (🖱️) kích chuột vào thành phần đó. (Bạn cũng có thể kích chuột phải hay dùng tổ hợp Ctrl+ chuột trái vào thành phần đó và chọn thẻ *Edit tool* (🖱️) từ cửa sổ hiện ra. Cùng với đó, việc thao tác với một thành phần qua *Poke tool* (👉) hay *Text tool* (A) cũng sẽ thể hiện các thuộc tính của thành phần đó.

hình dưới đây mô tả những gì diễn ra khi ta chọn cổng đầu vào phía trên của mạch XOR rồi kéo màn hình xuống phần chọn font cho tiêu đề. (Label Font attribute)



Để chỉnh sửa một giá trị thuộc tính, kích chuột vào giá trị đó. Giao diện để chỉnh sửa một thuộc tính sẽ phụ thuộc vào chính thuộc tính mà bạn đang thay đổi; trong trường hợp chỉnh sửa font cho tiêu đề, một bảng hội thoại sẽ hiện ra cho việc chọn font mới; tuy nhiên một vài thuộc tính (như vị trí tiêu đề) sẽ hiện ra một menu nhỏ hơn mà bạn sẽ lựa chọn giá trị từ đó.

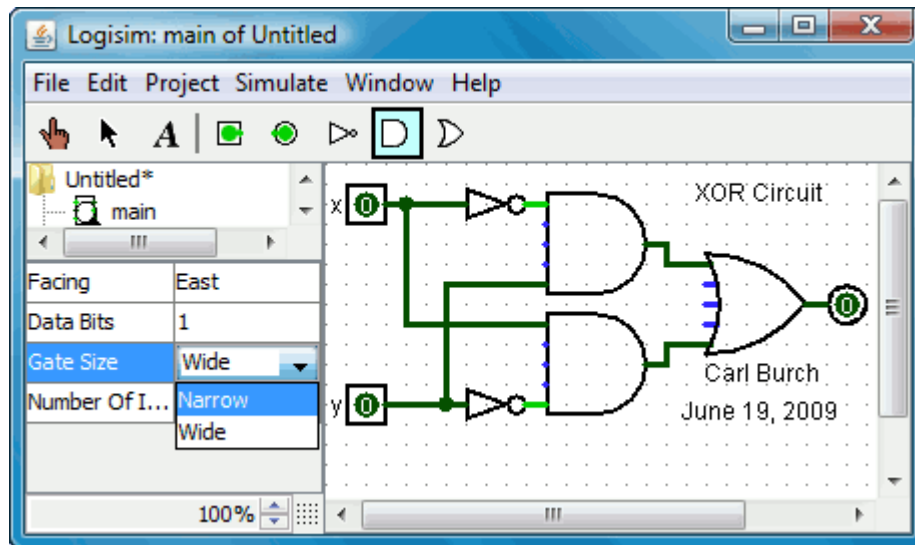
Mỗi loại thành phần có một tập hợp các thuộc tính khác nhau; để biết được chúng có ý nghĩa gì, bạn có thể xem tài liệu thích hợp ở [Library Reference](#).

Nếu như bạn đã lựa chọn nhiều thành phần qua công cụ chỉnh sửa (*Edit tool*), bảng thuộc tính sẽ hiện ra các thuộc tính chung của tất cả các thành phần đó (ngoại trừ các dây). Nếu các thành phần được chọn không có một giá trị thuộc tính nào có chung cho tất cả, phần hiện ra sẽ trống. Bạn có thể 1 lần thay đổi giá trị thuộc tính của tất cả các thành phần được chọn thông qua bảng thuộc tính.

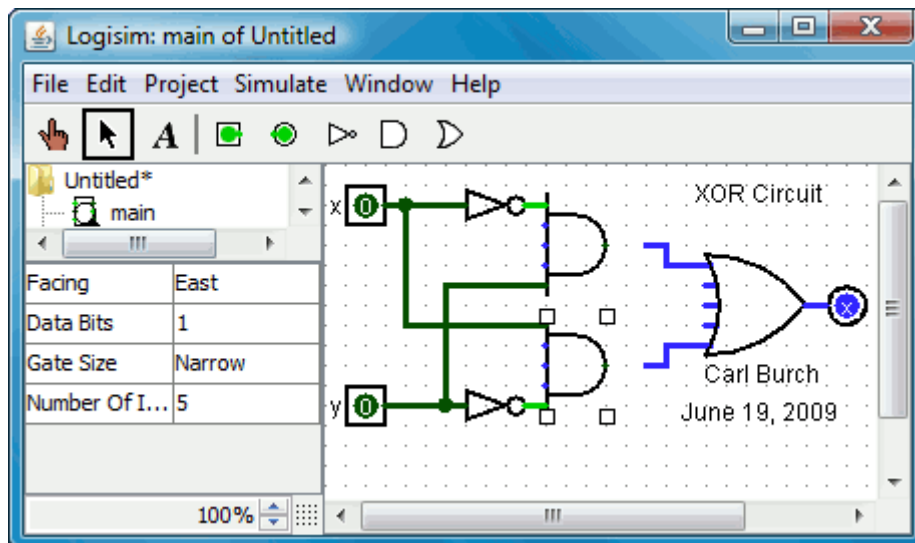
2.3 Thuộc tính của công cụ

Tất cả các công cụ để thêm các thành phần cho mạch đều có một tập hợp các thuộc tính, chúng được truyền tiếp cho các thành phần tạo bởi công cụ đó, mặc dù các thuộc tính của các thành phần có thể được thay đổi sau mà không làm ảnh hưởng thuộc tính của công cụ. Khi bạn lựa chọn một công cụ, Logisim sẽ thay đổi bảng thuộc tính để thể hiện các thuộc tính của công cụ đó.

Ví dụ như khi bạn muốn thiết lập các cổng AND nhỏ hơn. Ngay từ bây giờ, mỗi khi chúng ta chọn công cụ AND, Logisim sẽ tạo ra một cổng AND lớn. Tuy nhiên nếu chúng ta chỉnh sửa thuộc tính kích cỡ cổng ngay sau khi lựa chọn công cụ (trước khi đặt các cổng AND của nó vào mạch), chúng ta sẽ thay đổi các thuộc tính cho công cụ để sau đó các cổng AND được chèn vào thông qua công cụ sẽ tỉ mỉ hơn.



Bây giờ, chúng ta có thể xóa hai cổng AND đã tồn tại và chèn vào 2 cổng AND mới vào vị trí cũ. Lần này, chúng sẽ được thu gọn. (Nếu bạn chọn giảm số lượng đầu vào xuống 3, cổng AND sẽ không có các phần mở rộng ở phía bên trái. Nhưng bạn sẽ phải nối lại các dây nối trong mạch để các dây liên kết được với phía trái của các cổng AND).



Đối với một số công cụ, các biểu tượng công cụ sẽ thể hiện một giá trị vài thuộc tính. Ví dụ công cụ chốt có biểu tượng giống với giá trị mà thuộc tính bề mặt thể hiện.

Các công cụ trong thanh công cụ đều có một bộ thuộc tính tách rời từ các công cụ tương ứng trong *explorer pane*. Vì thế, cho dù chúng ta có thay đổi công cụ AND trong thanh công cụ để tạo ra các cổng AND thu gọn, công cụ AND trong thư viện cổng sẽ vẫn tạo ra các cổng AND lớn trừ khi bạn thay đổi luôn cả các thuộc tính của nó.

Trên thực tế, các công cụ chốt đầu vào và đầu ra trên thanh công cụ mặc định đều là các trường hợp của công cụ chốt (Pin) trong thư viện nền, tuy nhiên các bộ thuộc tính là khác nhau. Biểu tượng của công cụ chốt (Pin tool) được vẽ thành một vòng tròn hay một hình vuông tùy theo giá trị đầu ra của nó.

Logisim cung cấp một phím tắt rất tiện lợi cho việc thay đổi thuộc tính hướng (Facing tool), thuộc tính này điều khiển hướng mà các thành phần sẽ quay đến: Việc gõ một phím mũi tên khi đang chọn công cụ này sẽ tự động thay đổi hướng của thành phần.

3. CÁC MẠCH CON

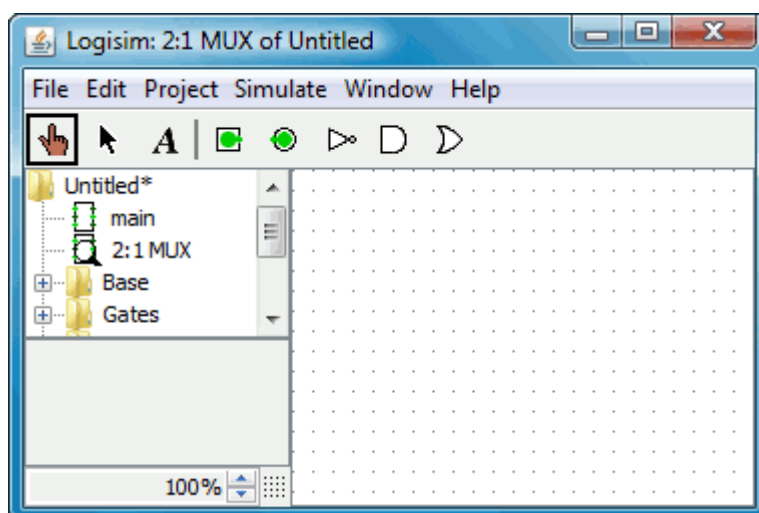
Khi bạn xây dựng các mạch ngày càng phức tạp hơn, bạn sẽ muốn xây dựng các mạch nhỏ hơn mà có thể sử dụng nhiều lần như một môđun được lồng vào trong các mạch lớn hơn. Trong Logisim, một mạch nhỏ được sử dụng trong mạch lớn hơn được gọi là **mạch con**.

Nếu bạn đã quen với lập trình, bạn cũng sẽ quen với khái niệm chương trình con (còn được gọi là *thủ tục*, *hàm* hay *phương pháp* trong các ngôn ngữ khác nhau). Khái niệm về mạch con cũng tương tự như khái niệm về chương trình con trong lập trình, nó được sử dụng cho các mục đích giống chương trình con: Để chia nhỏ một công việc lớn thành các phần nhỏ hơn, tiết kiệm công sức định nghĩa lại các phần giống nhau và để cho việc sửa lỗi dễ dàng, thuận tiện hơn.

3.1 Tạo mới các mạch

Tất cả các công trình trong Logisim đều là một thư viện các mạch. Ở dạng đơn giản nhất, mỗi công trình chỉ có một mạch (mặc định gọi là “*main*”) nhưng có thể thêm vào một cách dễ dàng: Chọn “*Add Circuit*” (chèn mạch)...trong thẻ Project và đặt một cái tên bất kỳ cho mạch mới mà bạn muốn tạo ra.

Giả sử ta muốn xây dựng một bộ dồn kênh 2-to-1 với tên là “2:1 MUX”. Sau khi thêm mạch, Logisim sẽ hiện ra như sau:

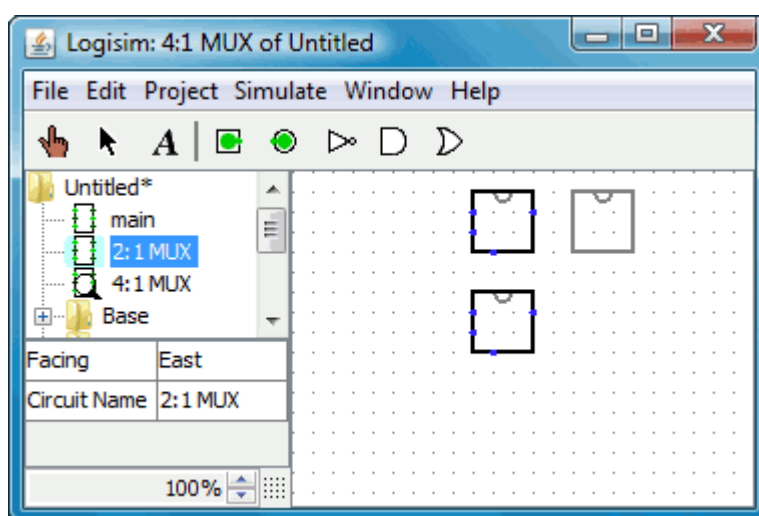


Trong phần *explorer pane*, bạn có thể thấy rằng công trình hiện tại bao gồm hai mạch, “main” và “2:1 MUX”. Logisim vẽ hình một cái kính lúp bên trên biểu tượng của mạch đang xét; tên của mạch đang xét đó cũng sẽ hiện lên trên thanh tiêu đề.

Sau khi chỉnh sửa mạch cho đúng với một bộ dồn kênh 2:1, ta sẽ có một mạch như sau:

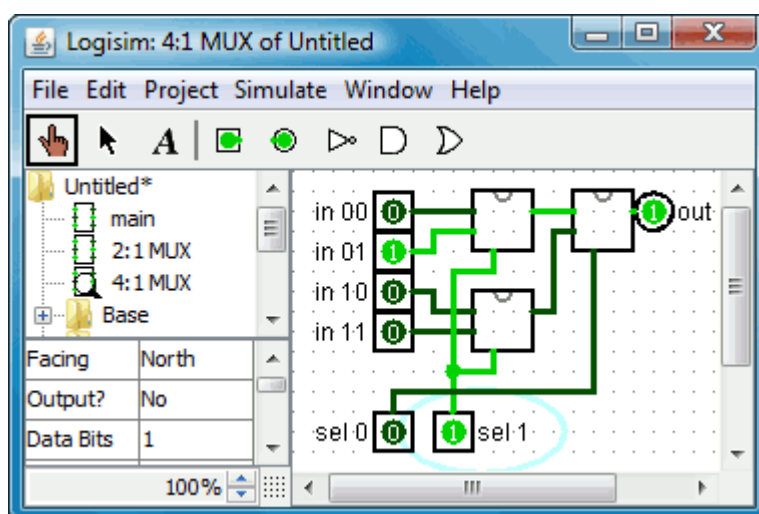
3.2 Sử dụng mạch con

Giả sử chúng ta muốn xây dựng một bộ dồn kênh 4-to-1 sử dụng các bộ dồn kênh 2-to-1 mà ta đã xây dựng. Đầu tiên ta sẽ tạo mới một mạch với tên gọi “4:1 MUX”. Để chèn các bộ dồn kênh 2-to-1 vào mạch, kích vào 2:1 MUX *một lần* trên *explorer pane* để chọn nó như một công cụ, sau đó chúng ta có thể chèn các bản sao của nó, được thể hiện dưới dạng các hộp nhỏ, bằng cách kích chuột vào bản vẽ.



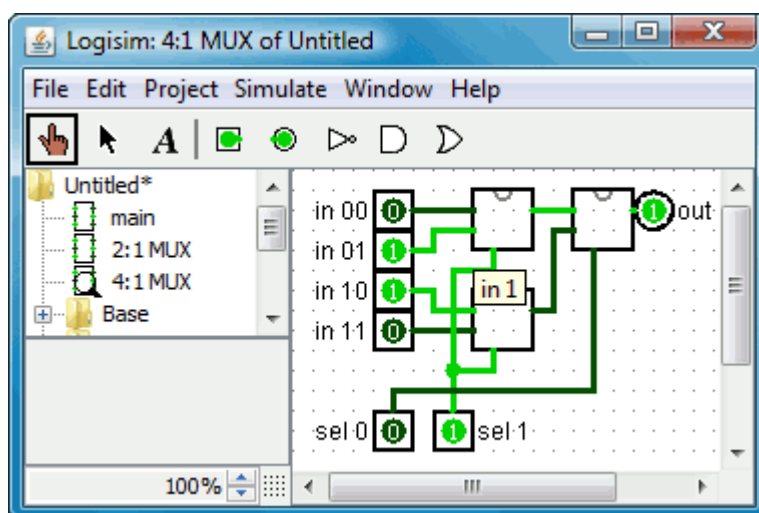
Nếu bạn kích đúp vào mạch 2:1 MUX trên *explorer pane* của sổ chương trình sẽ chuyển thành chỉnh sửa mạch 2:1 MUX.

Sau khi dựng mạch, chúng ta sẽ có như hình sau:



Mạch dồn kênh 4:1 của ta sử dụng 3 bản sao của bộ dồn kênh 2-to-1, mỗi bản sao được vẽ như một cái hộp với các chấm dọc theo cạnh của hộp. Các dấu chấm trong hộp này tương ứng với các chốt đầu vào và đầu ra trong mạch 2:1 MUX. 2 dấu chấm ở mặt phía Tây của hộp ứng với 2 chốt quay mặt về hướng Đông trong mạch 2:1 MUX; chấm ở mặt phía đông ứng với chốt quay sang hướng tây của mạch 2:1 MUX (tính cờ cũng là một chốt đầu ra); và chấm ở mặt phía nam của hộp ứng với chốt quay sang hướng bắc của mạch 2:1 MUX. Thứ tự của 2 dấu chấm trên mặt phía tây của hộp tương ứng với thứ tự từ trên xuống dưới trong thiết kế mạch con. (nếu cũng có nhiều chấm nhỏ trên mặt bắc và nam của hộp, chúng sẽ tương ứng với thứ tự phải trái trong mạch con).

Nếu các chốt trong bảng mạch con có nhãn kết hợp với chúng, Logisim sẽ hiển thị nhãn đó trong **tip** (một khung chữ tạm thời) khi người dùng trỏ chuột vào vị trí tương ứng trong phần tử mạch con. (nếu bạn cảm thấy các tip này bất tiện, bạn có thể tắt bỏ chúng thông qua Project/Options/Canvas).



Một vài các thành phần khác cũng sẽ hiển thị các tip: một số chốt của một flip-flop dựng sẵn, trỏ chuột vào nó sẽ hiện ra dòng chú giải về công việc của chốt.

Một cách ngẫu nhiên, tất cả các chốt đối với một mạch chỉ có thể là hoặc đầu vào hoặc đầu ra. Một số các chip được sản xuất có các chân đóng vai trò là đầu vào trong một số trường hợp và đóng vai trò là đầu ra trong các trường hợp khác; bạn không thể xây dựng các chip đó trong Logisim (ít nhất là đối với phiên bản hiện thời).

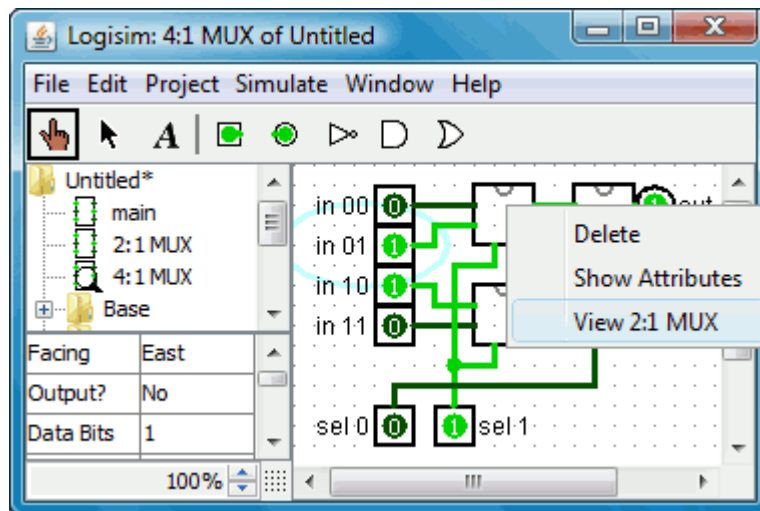
Logisim sẽ duy trì thông tin trạng thái khác nhau cho tất cả các mạch con xuất hiện trong mạch. Ví dụ một mạch có một flip-flop và mạch đó được sử dụng nhiều lần như một mạch con, khi đó flip-flop của mỗi một mạch con sẽ có giá trị riêng của nó khi mô phỏng mạch lớn hơn.

Bây giờ chúng ta có một bộ dồn kênh 4-to-1 đã được định nghĩa, chúng ta có thể sử dụng nó trong các mạch khác. Logisim không giới hạn một mạch có thể được lồng vào sâu đến đâu- mặc dù nó sẽ ngăn chặn các mạch được lồng chính bên trong chúng.

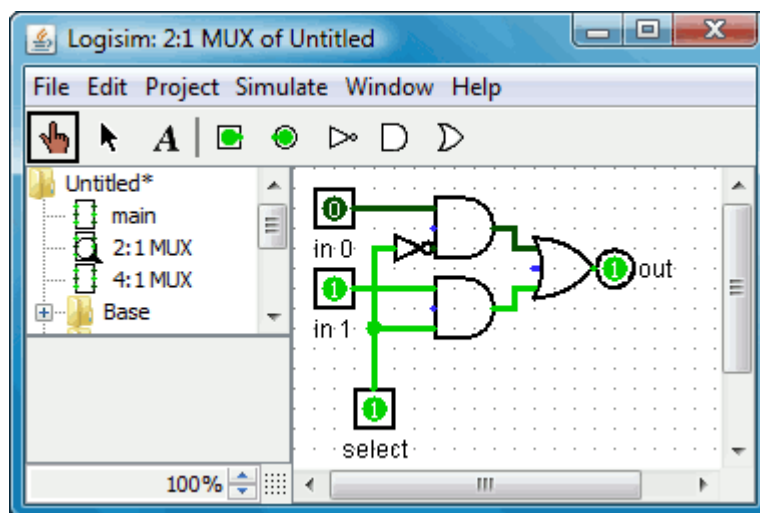
Chú ý: Không có gì là sai khi chỉnh sửa một mạch mà đang được sử dụng như một mạch con; trên thực tế điều đó là rất phổ biến. Bạn hãy nhớ rằng bất cứ một thay đổi nào trên các chốt của mạch (chèn, xóa hay di chuyển chúng) cũng sẽ sắp xếp lại chúng trong mạch lớn. Vì thế nếu bạn thay đổi một chốt nào đó của một mạch, bạn cũng sẽ cần phải chỉnh sửa tất cả các mạch sử dụng mạch đó như một mạch con.

3.3 Sửa lỗi các mạch con

Khi bạn kiểm tra các mạch, bạn sẽ thường phát hiện ra lỗi. Để tìm ra lỗi sai, việc kiểm tra những gì đang diễn ra trong các mạch trong khi vận hành bằng mạch tổng thể giúp bạn. Bạn có thể thực hiện việc đó khi đang xem xét mạch tổng thể bằng cách mở cửa sổ pop-up của mạch con (kích chuột phải hoặc tổ hợp Ctrl+ chuột trái vào hộp) sau đó bạn chọn mục *View*.



Sau khi bạn chọn như vậy, cửa sổ quan sát sẽ chuyển sang mạch con.



Chú ý rằng giá trị của các chốt trong mạch con trùng với các giá trị được truyền cho chúng trong mạch chứa chúng.

Trong khi đối với mạch con, bạn có thể thay đổi nó theo bất cứ cách nào bạn muốn; bất cứ thay đổi nào của các giá trị của chốt cũng sẽ được truyền cho mạch chứa. (Nếu bạn thử chuyển giá trị

của một chốt sử dụng *Poke Tool*. Logisim sẽ mở ra một khung hội thoại để hỏi bạn có muốn tạo mới một trạng thái hay không; trả lời Yes sẽ tách trạng thái được xem xét với mạch con khỏi trạng thái của mạch bên ngoài, trong khi nó câu trả lời No sẽ kết thúc yêu cầu thay đổi giá trị).

Bạn sẽ kết thúc việc xem xét và chỉnh sửa mạch tổng thể bằng cách kích đúp vào *explorer pane*, hay thông qua của sổ con *Go Out To State* trong phần mô phỏng (*Simulate menu*)

3.4 Thư viện Logisim

Mỗi một công trình Logisim đều tự động là một thư viện có thể được nạp vào các công trình Logisim khác: Chỉ cần lưu nó trong một file và sau đó nạp thư viện khi đang ở trong một công trình khác. Tất cả các mạch được định nghĩa trong mạch đầu tiên sẽ có như một mạch con cho công trình thứ hai. Tính năng này cho phép bạn tái sử dụng các thành phần hay dùng trong các công trình và chia sẻ các thành phần ưa thích với người khác.

Mỗi một công trình đều có một mạch chính(*main circuit*) định sẵn, mạch chính này có thể được thay đổi thành mạch hiện hành thông qua tùy chọn *Set As Main Circuit* trong phần *Project*. Ý nghĩa duy nhất của mạch chính là nó sẽ được hiện ra trước tiên khi bạn mở công trình đó ra. Tên mặc định của mạch khi mới được tạo ra (“*main*”) không quan trọng và bạn có thể đổi tên hoặc xóa mạch đó một cách tự do.

Với một thư viện Logisim được nạp, bạn sẽ được phép xem các mạch đó và quản lý các trạng thái của chúng, tuy nhiên Logisim sẽ ngăn không cho bạn thay đổi thiết kế của mạch cũng như các dữ liệu khác được lưu trữ trong file.

Nếu bạn muốn sửa một mạch trong thư viện Logisim được nạp vào, bạn sẽ cần mở *rieend* nó trong Logisim. Ngay khi bạn lưu nó, công trình khác sẽ tự động nạp bản chỉnh sửa; nhưng nếu nó không thực hiện chức năng này, bạn có thể kích chuột phải vào thư mục *library* trong *explorer pane* và chọn *Reload Library*.

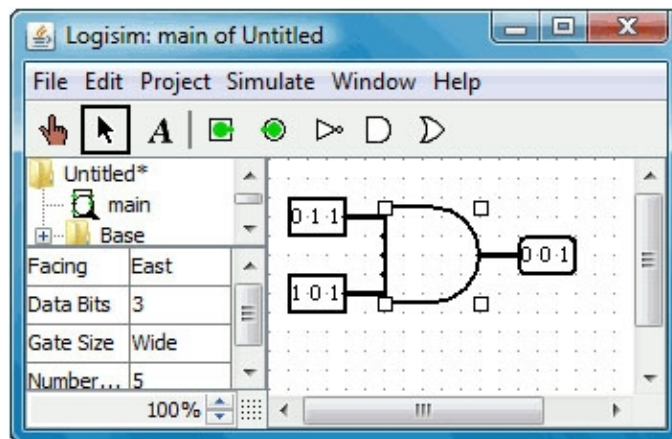
4. DÂY

Trong các mạch điện Logisim đơn giản, hầu hết các dây chỉ mang 1 bit, nhưng Logisim cũng cho phép bạn kết hợp các dây vào để mang những giá trị nhiều bit. Số lượng bit đi qua dây được gọi là độ rộng bit.

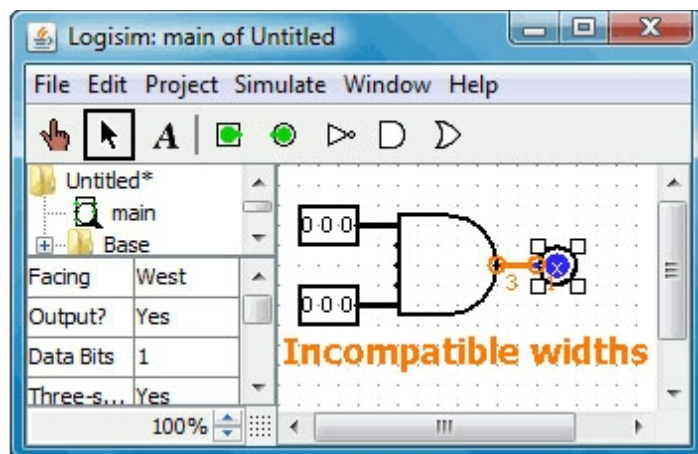
4.1 Tạo các bó

Mọi đầu vào và đầu ra trên một mạch đều có 1 chiều rộng bit kết hợp với nó. Thường thì chiều rộng bit là 1, và không có cách nào để thay đổi nó, nhưng nhiều thành phần trong Logisim bao gồm các thuộc tính cho phép bạn tùy chỉnh độ rộng bit đầu vào và đầu ra của chúng.

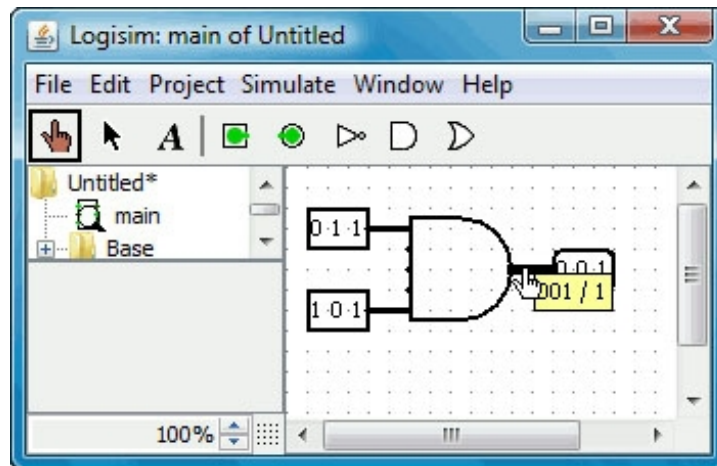
Ảnh chụp màn hình minh họa dưới đây là một mạch đơn giản cho việc tìm kiếm các bitwise AND của hai đầu vào ba bit. Chú ý làm thế nào để 3 bit đầu ra là AND của 2 đầu vào. Tất cả các thành phần đã được điều chỉnh để phù hợp với dữ liệu 3 bit thông qua những data bits thuộc tính. Ảnh chụp màn hình biểu thị các thuộc tính của cổng AND, bao gồm các data bits thuộc tính của 3.



Tất cả các thành phần trong Logisim xác định một chiều rộng bit cho mỗi đầu vào và đầu ra. Ngược lại, chiều rộng bit của dây là không xác định. Thay vào đó, chiều rộng của dây điều chỉnh theo các thành phần mà nó được đính kèm. Nếu một dây kết nối hai thành phần được yêu cầu mà độ rộng bit khác nhau, Logisim sẽ báo cáo "độ rộng không tương thích" và chỉ ra các vị trí vi phạm bằng màu da cam. Ở hình dưới, chiều rộng bit của đầu ra đã bị thay đổi thành 1, vì vậy Logisim báo cáo rằng không thể kết nối một giá trị 3 bit với 1 giá trị 1 bit



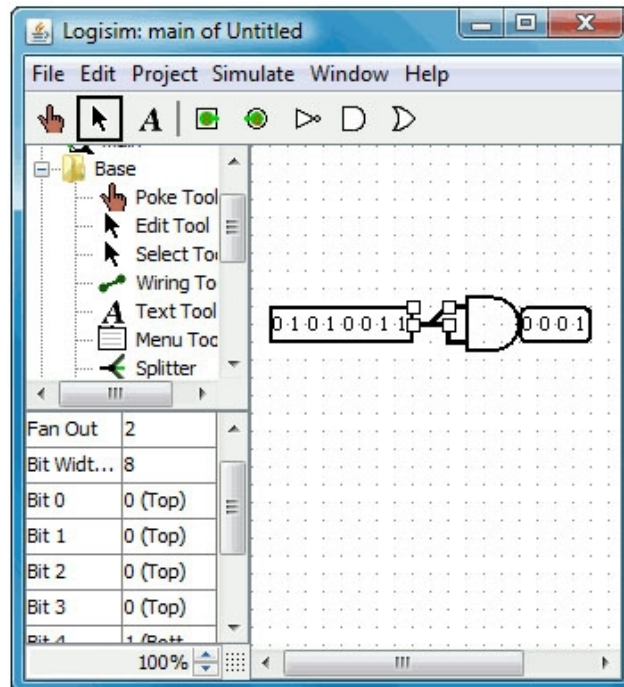
Các dây kết nối các địa điểm không tương thích (màu da cam) không mang giá trị. Với những dây một bit bạn có thể thấy ngay giá trị mà dây mang vì Logisim tô các dây với màu xanh lá cây nhạt hoặc xanh lá cây đậm tùy theo giá trị. Nó không hiển thị các giá trị của những dây mang nhiều bit: chúng thường được tô đơn giản bằng màu đen. Tuy nhiên bạn có thể biết giá trị của một sợi dây bằng cách nhấn vào nó sử dụng công cụ poke.



4.2 Dụng cụ rẽ nhánh

Khi bạn làm việc với những giá trị mang nhiều bit, bạn thường muốn chuyển các bit khác nhau theo những hướng khác nhau. Công cụ splitter trong thư viện cơ sở cho phép bạn thực hiện việc này.

Ví dụ, ta muốn xây một mạch tính AND của 2 nibble của chính 8 bit đầu vào của nó (bốn bit trên và bốn bit thấp hơn). Chúng ta sẽ có một giá trị 8 bit từ các nút đầu vào, và ta muốn chia nó thành 2 giá trị 4 bit. Trong mạch dưới đây chúng tôi đã sử dụng một dụng cụ rẽ nhánh (splitter) để làm điều này: đầu vào 8 bit đến splitter (tương tự như chân của con chim) rồi bị phân ra thành 2 giá trị 4 bit, và được đưa vào các cổng AND rồi từ đó được ra đầu ra.



Trong ví dụ này, splitter thực sự chia một giá trị đầu vào thành nhiều giá trị rồi gửi đi. Nhưng splitter cũng có thể kết hợp nhiều giá trị vào thành một giá trị duy nhất. Trong thực tế, những splitter này là không có định hướng (non-directional): chúng có thể gửi các giá trị theo một đường vào một thời điểm nào đó, nhưng sẽ theo một đường khác vào một thời điểm sau đó, và chúng thậm chí có thể gửi cả 2 đường một lúc. Như trong ví dụ dưới đây có một giá trị đi về

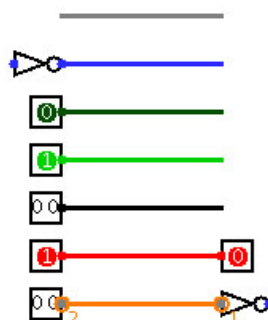
phía đông thông qua hai splitter, sau đó được định tuyến trở lại về phía tây qua 2 splitter đó một lần nữa, và sau đó lại trở về phía đông, nơi nó gặp đầu ra.



Chìa khóa để có thể hiểu được các splitter là những thuộc tính của nó.

4.3 Màu của dây

Dây điện trong Logisim có đầy đủ màu sắc của cầu vồng:



Màu xám: độ rộng bit của dây điện không rõ. Điều này xảy ra bởi vì dây điện không được gắn vào bất kì đầu vào hay đầu ra nào của các thành phần. (Tất cả các yếu tố đầu vào và đầu ra có độ rộng bit được định nghĩa sẵn.)

Màu xanh da trời: dây này để mang những giá trị 1 bit, nhưng không biết được giá trị nó đang mang

Màu xanh lá cây thẫm: mang bit 0

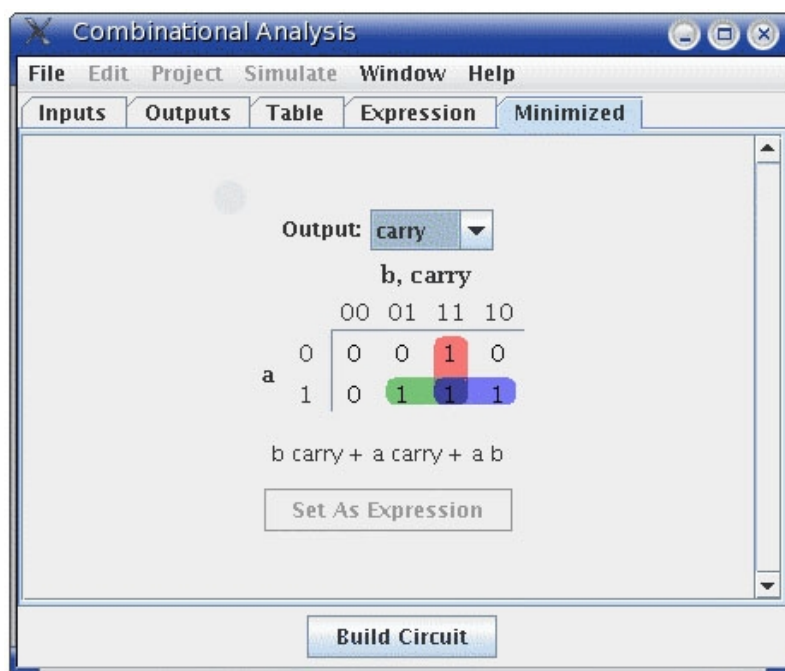
Màu xanh lá cây nhạt: mang bit 1

Màu đen: Dây mang một giá trị nhiều bit. Một số hoặc tất cả các bit có thể không được chỉ định.

Màu đỏ: Dây mang một giá trị lỗi. Điều này thường phát sinh bởi sự mâu thuẫn các giá trị trên dây.

Màu da cam: Các thành phần trên dây không chấp nhận độ rộng bit. Dây màu da cam rất có khả năng bị “hỏng”. Nó không mang giá trị giữa các thành phần.

5. PHÂN TÍCH TỔNG HỢP



Tất cả các loại mạch đều rơi vào 2 mục chính: Trong một mạch tổng hợp, tất cả các mạch đầu ra đều là sự tổng hợp chặt chẽ của các mạch đầu vào hiện tại, trong khi trong mạch tuần tự, một số kết quả đầu ra có thể phụ thuộc vào các yếu tố đầu vào trước đó (chuỗi đầu vào theo thời gian).

Mục các mạch tổng hợp thường đơn giản là sự kết hợp của 2 mạch. Các học viên sử dụng ba dụng cụ chính để tổng hợp các mạch như vậy:

1. Mạch logic
2. Các biểu thức boolean
3. Bảng chân lí

Các mô đun phân tích tổng hợp cho phép bạn chuyển đổi giữa ba dụng cụ này ở mọi hướng. Đó là một cách đặc biệt thuận tiện để tạo ra và tìm hiểu mạch với một số ít bit đầu vào và đầu ra.

5.1 Cách mở bảng phân tích tổng hợp

Phần lớn các mô đun phân tích tổng hợp được truy cập thông qua một cửa sổ duy nhất, nó cho phép bạn xem các bảng chân lý và biểu thức Boolean. Cửa sổ này có thể được mở bằng hai cách.

Thông qua menu Window

Chọn “Combinational Analysis”, và cửa sổ Phân tích tổng hợp sẽ hiện ra. Nếu bạn chưa từng mở cửa sổ này trước đây, cửa sổ này sẽ không hiện ra mạch nào cả.

Chỉ có đúng 1 cửa sổ Phân tích tổng hợp dù bao nhiêu project đang được mở ra, không có cách nào để tạo ra 2 cửa sổ mở ra cùng 1 lúc.

Thông qua menu Project

Từ một cửa sổ chỉnh sửa mạch điện, bạn cũng có thể yêu cầu Logisim phân tích mạch bằng cách chọn “Analyze Circuit” từ menu Project. Trước khi Logisim mở ra cửa sổ, nó sẽ tính toán các biểu thức Boolean và bảng chân lý tương ứng với mạch điện và đặt ở nơi bạn có thể xem.

Để có thể phân tích mạch thành công, mỗi đầu vào phải được gắn với một chốt đầu vào và mỗi đầu ra phải được gắn với một chốt đầu ra. Logisim chỉ phân tích những mạch với nhiều nhất 8 đầu vào hoặc 8 đầu ra, và chúng đều phải là những chốt 1 bit. Nếu không bạn sẽ nhìn thấy thông báo lỗi và cửa sổ sẽ không mở ra.

Trong xây dựng các biểu thức Boolean tương ứng với một mạch, Logisim đầu tiên sẽ cố gắng xây dựng một biểu thức Boolean tương ứng chính xác với các cổng trong mạch. Nhưng nếu mạch điện sử dụng các thành phần không phải là cổng (như multiplexer) nó sẽ bật lên một hộp thoại cho bạn biết không thể thu được biểu thức Boolean, và thay vào Logisim sẽ đưa ra những biểu thức dựa trên bảng chân lý.

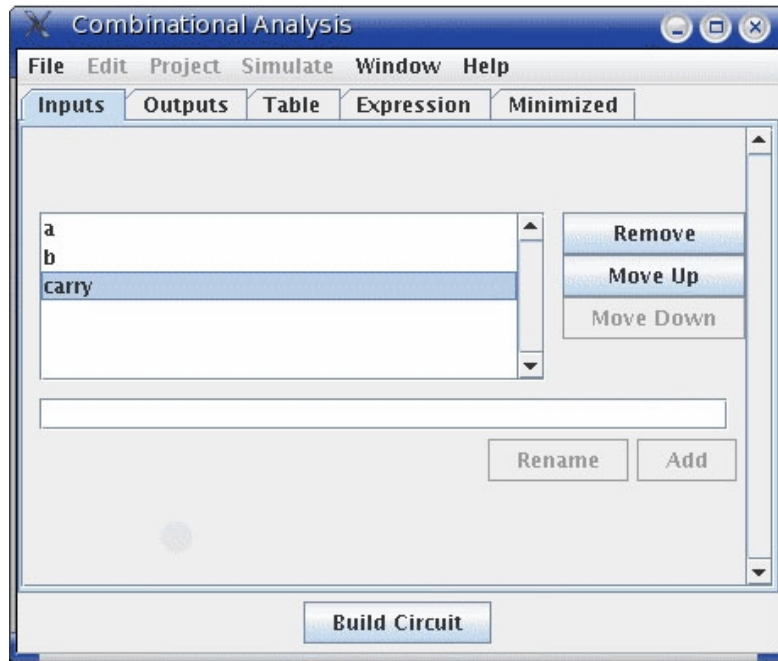
Những hạn chế

Logisim sẽ không phát hiện các mạch tuần tự. Nếu bạn bắt nó phân tích một mạch tuần tự, nó vẫn sẽ tạo ra một bảng sự thật và biểu thức Boolean tương ứng nhưng sẽ không chính xác. Vì thế không nên sử dụng bừa bãi hệ thống Phân tích tổng hợp. Chỉ sử dụng nó khi bạn chắc chắn rằng mạch bạn đang phân tích thực sự là mạch tổng hợp.

Logisim có thể sẽ làm những thay đổi không mong muốn với mạch điện. Hệ thống Phân tích tổng hợp yêu cầu mỗi đầu vào và đầu ra có một tên (gán nhãn) duy nhất mà phù hợp với các quy tắc đặt tên của Java (mỗi kí tự phải là chữ cái hoặc số, và kí tự đầu tiên phải là chữ cái, không được có dấu cách giữa các kí tự). Nó sẽ cố gắng sử dụng các nhãn đang tồn tại của các chốt, và sẽ sử dụng một danh sách các giá trị mặc định nếu nhãn không tồn tại. Nếu một nhãn được đặt không hợp lệ, Logisim sẽ cố gắng trích ra một tên hợp lệ từ nhãn đó.

5.2 Sửa đổi bảng chân lý

Một cửa sổ “Combinational Analysis” sẽ bao gồm 5 thẻ



Phần này miêu tả 3 thẻ Inputs, Output và Table. Phần tiếp theo sẽ tiếp tục miêu tả 2 thẻ còn lại Expression và Minimized.

Thẻ Inputs và Outputs

Thẻ Inputs cho phép bạn xem và sửa danh sách các yếu tố đầu vào. Để thêm đầu vào mới, gõ nó vào phần dưới cùng của cửa sổ, và nhấp vào Add. Nếu bạn muốn đổi tên một đầu vào, chọn nó và nhấp vào Rename.

Để loại một input, bạn nhấp vào nó và chọn Remove. Bạn cũng có thể sắp xếp lại các đầu vào bằng cách sử dụng nút Move Up hoặc Move Down.

Mọi việc làm tác động đến bảng chân lí ngay lập tức.

Thẻ Output làm việc cũng như thẻ Inputs, ngoại trừ việc nó làm việc với các yếu tố đầu ra.

Thẻ Table

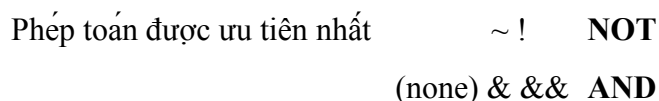
Trong thẻ này chỉ có bảng chân lí hiện thời, với đầu vào tạo thành các cột bên trái và đầu ra tạo thành các cột bên phải.

Bạn có thể chỉnh sửa các giá trị hiện tại xuất hiện trong các cột đầu ra bằng cách nhấp vào giá trị của đầu vào. Các giá trị sẽ xoay vòng xung quanh 0, 1 và x (nghĩa là “không quan tâm”). Chúng ta sẽ thấy trong các phần tiếp theo, bất kì giá trị “không quan tâm” nào cũng sẽ cho phép tính toán các biểu thức rút gọn linh hoạt hơn.

Bạn cũng có thể điều hướng và chỉnh sửa bảng chân lí bằng cách sử dụng bàn phím. Và bạn có thể sao chép và dán giá trị sử dụng clipboard.

5.3 Tạo các biểu thức

The Expression



^ **XOR**

Phép toán được ưu tiên thấp nhất + | || **OR**

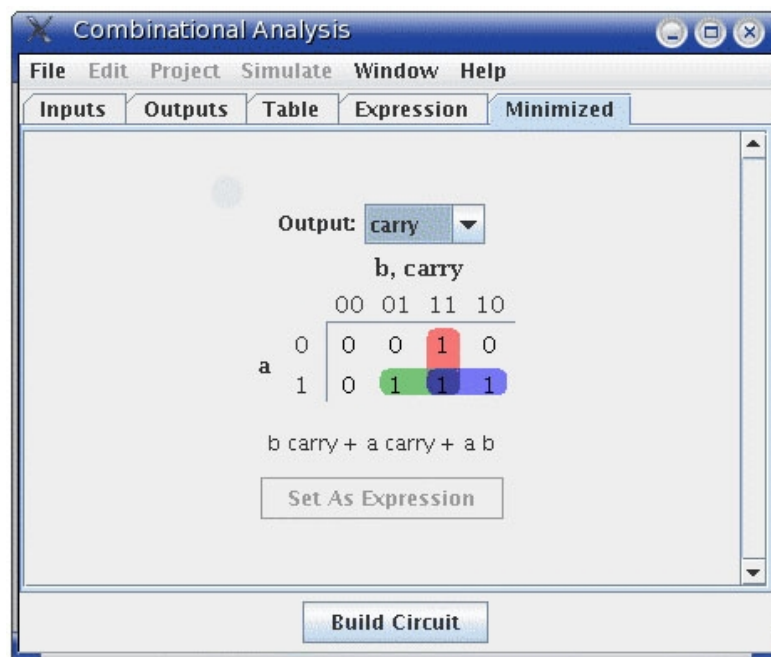
Các cách diễn đạt sau đây đều đúng cho cùng 1 biểu thức boolean. Bạn có thể kết hợp các toán tử

$\sim a \ (b + c)$

$!a \ \&\& \ (b \ || \ c)$

NOT a AND (b OR c)

The Minimized



The Minimized hiển thị các biểu thức rút gọn liên quan đến một cột của bảng chân lí. Bạn có thể chọn xem biểu thức rút gọn của các output bằng bộ chọn ở đầu trang..

Nếu có nhỏ hơn 5 biến đầu vào, một bản đồ Karnaugh sẽ hiện ra . Bạn có thể nhấp vào bảng để thay đổi giá trị của bảng chân lí.

Ở dưới bản đồ chính là biểu thức boolean của biến đầu ra, hiển thị như ở trong thẻ Expression. Nếu có nhiều hơn 4 biến đầu vào, bản đồ Karnaugh sẽ không hiện ra, nhưng biểu thức rút gọn vẫn được tính (Logisim sử dụng thuật toán Quine-McCluskey để tính biểu thức rút gọn).

Nút “Set As Expression” cho phép bạn chọn biểu thức rút gọn cho các biến. Nút này thường là không cần thiết, vì sử dụng thẻ Expression sẽ thuận tiện hơn.

5.4 Tạo mạch

Nút Build Circuit sẽ xây dựng một mạch có các cổng tương ứng với các biểu thức được chọn cho mỗi đầu ra. Các cổng đầu vào và đầu ra sẽ hiển thị đúng theo thứ tự từ trên xuống dưới như ở trong thẻ Inputs và Outputs. Nói chung, các mạch được xây dựng sẽ rất đẹp; quả thật vậy, một trong những ứng dụng của mô đun Phân tích tổng hợp là để thiết kế lại các mạch vẽ xấu.

Khi bạn bấm vào nút Build Circuit, một hộp thoại sẽ xuất hiện cho bạn chọn những project mà cần mạch đó và tên mà bạn muốn đặt cho nó. Nếu bạn sử dụng tên của một mạch đã tồn tại thì mạch cũ đó sẽ bị thay thế (sau khi Logisim hỏi bạn có muốn thay thế không)

Hộp thoại Build Circuit có 2 mục mà bạn có thể chọn. Mục “Two-Input Gates Only” để chỉ định bạn muốn tất cả các cổng đều có 2 đầu vào (ngoại trừ những cổng NOT). Mục “Use NAND Gates Only” thể hiện bạn muốn xây dựng một mạch chỉ gồm toàn các cổng NAND. Bạn có thể chọn cả 2 mục. Nhưng Logisim không thể xây dựng một mạch toàn cổng NAND với những biểu thức có toán tử XOR. Khi đó mục “Use NAND Gates Only” sẽ bị cấm.

6. GIỚI THIỆU CÁC MENU

Phần này sẽ trình bày về 6 menu chính của Logisim

Menu File

Menu Edit

Menu Project

Menu Simulate

Menu Window

Menu Help

6.1 Menu File

New

Mở ra một project mới trong một cửa sổ mới. Project đó sẽ được khởi tạo như mẫu đang được chọn

Open ...

Mở một project đã tồn tại trong một cửa sổ mới.

Close

Đóng tất cả các cửa sổ liên quan đến project đang được xem

Save

Lưu lại project đang được xem (ghi đè lên file cũ)

Save as...

Lưu project đang được xem thành một file khác (không ghi đè lên file cũ)

Export Image..

Tạo ra file ảnh tương ứng với mạch.

Print...

In mạch.

Preference...

Hiển thị các cửa sổ ứng dụng ưu tiên. (trong hệ điều hành Mac OS, nó sẽ xuất hiện trong menu Logisim)

Exit

Đóng tất cả các project và thoát khỏi Logisim (trong hệ điều hành Mac OS, nó sẽ có tên là Quit trong menu Logisim)

6.2 Menu Edit

Undo XX

Hủy các thao tác được thực hiện gần đây nhất. Lưu ý rằng nó không bao gồm các thay đổi liên quan đến trạng thái của mạch (như các thao tác được thực hiện bởi công cụ Poke)

Cut

Loại bỏ các thành phần đang được chọn từ các mạch vào clipboard của Logisim.

Lưu ý: Clipboard của Logisim hoàn toàn riêng biệt với clipboard của hệ thống tổng thể. Vì thế nên các thao tác cắt / sao chép / dán sẽ không làm việc trên các ứng dụng khác, thậm chí cả các bản Logisim khác đang chạy. Nhưng nếu bạn đang có nhiều project đang mở dưới cùng một Logisim, bạn hoàn toàn có thể cắt / sao chép / dán giữa các project đó.

Copy

Sao chép các thành phần đang được chọn trong mạch vào clipboard của Logisim. (các lưu ý như ở trong mục Cut)

Paste

Dán các thành phần trong clipboard của Logisim đến phần đang được chọn (các lưu ý như ở trong mục Cut).

Delete

Xóa bỏ hoàn toàn các phần đang được chọn trong mạch, và không đưa đến clipboard.

Select All

Chọn toàn bộ các thành phần trong mạch hiện tại

6.3 Menu Project

Add Circuit...

Thêm một mạch mới vào project hiện tại. Logisim sẽ đề nghị bạn đặt tên cho mạch, tên đó không được trùng với bất kỳ mạch nào trong project.

Load Library

Tải một thư viện vào project. Bạn có thể 3 loại thư viện (đề cập đến trong mục khác)

Unload Libraries...

Gỡ bỏ các thư viện hiện tại trong mạch, Logisim sẽ không cho phép bạn gỡ bất cứ thư viện nào đang được dùng.

Analyze Circuit

Xây dựng bảng chân lí và các biểu thức boolean tương ứng với mạch hiện tại, hiển thị chúng ở cửa sổ Phân tích tổng hợp. Quá trình phân tích chỉ thực hiện được cho những mạch tổng hợp. Một bản mô tả đầy đủ của quá trình phân tích sẽ được hiển thị trong bảng Phân tích tổng hợp.

Rename Circuit...

Đổi tên mạch đang được hiển thị.

Move Circuit Up

Chuyển mạch đang được hiển thị lên một dòng trong danh sách các mạch của project ở cửa sổ explorer.

Move Circuit Down

Chuyển mạch đang được hiển thị xuống một dòng trong danh sách các mạch của project ở cửa sổ explorer.

Set As Main Circuit

Thiết lập mạch đang được hiển thị trở thành mạch chính trong project (menu này sẽ bị khóa nếu mạch hiện tại đã là mạch chính rồi). Sự khác biệt duy nhất của mạch chính là nó sẽ là mạch được hiển thị đầu tiên khi một project được mở.

Remove Circuit

Xóa bỏ mạch đang được hiển thị khỏi project. Logisim sẽ không cho bạn gỡ bỏ những mạch đang được sử dụng như những mạch con của các mạch khác, và nó cũng sẽ không cho bạn xóa bỏ mạch điện cuối cùng còn lại trong project.

Show Project Toolbar

Khi được đánh dấu, nó sẽ hiển thị một thanh công cụ nhỏ phía trên cửa sổ explorer, cho phép người dùng dễ dàng thực hiện các thao tác với các mạch như: thêm, đổi tên, sắp xếp lại và xóa bỏ một mạch từ project.

Options...

Mở cửa sổ các tùy chọn của project (Project Options).

6.4 Menu Simulate

Simulate Enabled

Nếu được đánh dấu, các mạch điện sẽ chạy (“live”). Mục này sẽ tự động loại bỏ đánh dấu nếu phát hiện sự dao động trong mạch.

Reset Simulation

Xóa mọi thứ trong trạng thái hiện tại của mạch, nó như là bạn mới mở lại 1 file.

Step Simulation

Thực hiện bước tiếp theo trong bộ mô phỏng.

Go Out To State

Khi bạn kiểm tra kỹ lưỡng trạng thái của một mạch con thông qua menu pop-up của nó, mục này sẽ liệt kê các mạch ở trên mạch đang được xem.

Go In To State

Nếu bạn đã từng kiểm tra kỹ lưỡng trạng thái của một mạch con, rồi sau đó muốn quay lại mạch đó. Menu này sẽ liệt kê tất cả những mạch phụ của mạch hiện thời.

Tick Once

Menu này sẽ làm cho đồng hồ chạy từng bước một. Nó rất là hữu dụng khi bạn muốn chạy đồng hồ bằng tay, đặc biệt là khi đồng hồ không thuộc cùng mạch mà bạn đang xử lý

Tick enable

Khởi động đồng hồ chạy tự động. Menu này chỉ chạy được khi trong mạch có thiết bị đồng hồ (trong thư viện Base)

Tick Frequency

Cho phép bạn chọn tần số tick. Ví dụ, 8 Hz nghĩa là đồng hồ sẽ tick 8 lần trong 1 giây. Tick là đơn vị đo tốc độ của đồng hồ.

Lưu ý rằng tốc độ chu kỳ đồng hồ sẽ chậm hơn so với tốc độ tick.

Đồng hồ nhanh nhất có thể sẽ có một chu kì 1 tick lên và một chu kì 1 tick xuống. Đồng hồ đó sẽ có chu kì lên/xuống là 4Hz nếu tốc độ của đồng hồ là 8 Hz

Logging...

Vào mô đun đăng nhập

6.5 Menu Window

Minimize

Thu nhỏ cửa sổ hiện thời.

Maximize (Zoom trên MacOS)

Phục hồi kích cỡ của cửa sổ.

Combinational Analysis

Hiển thị cửa sổ Phân tích tổng hợp.

Individual window titles

Hiển thị cửa sổ tương ứng lên phía trước.

6.6 Menu Help

Tutorial

Mở hệ thống hướng dẫn trực tuyến, phần Hướng dẫn cho người mới bắt đầu trong mục Hướng dẫn sử dụng Logisim.

User's Guide

Mở hệ thống hướng dẫn trực tuyến, mục “Guide to Being a Logisim User”.

Library Reference

Mở hệ thống hướng dẫn trực tuyến phần Thư viện.

About ...

Mở ra một cửa sổ ghi phiên bản. (Trong hệ điều hành Mac OS, mục này ở trong menu Logisim) .

7. CÁC THÀNH PHẦN CỦA BỘ NHỚ

Thành phần RAM và ROM là 2 thành phần hữu dụng nhất trong thư viện tích hợp của logicsim. Tuy nhiên, do số lượng thông tin mà nó có thể lưu trữ, chúng cũng là 2 thành phần phức tạp nhất.

Tư liệu về cơ chế hoạt động của chúng trong mạch có thể tìm thấy trong mục RAM và ROM của library references. Mục này của user's guide giải thích về giao diện cho phép người sử dụng quan sát và chỉnh sửa nội dung bộ nhớ.

7.1 Truy cập bộ nhớ

Bạn có thể điều khiển bộ nhớ bằng công cụ Poke, nhưng giao diện cho nó còn một vài hạn chế bởi không gian làm việc: cho những chỉnh sửa đơn giản nhất, bạn sẽ thấy the intergrated hex editors thuận tiện hơn rất nhiều.

Tuy nhiên, để quan sát và chỉnh sửa trong mạch, công cụ Poke có 2 chế độ hoạt động: bạn có thể chỉnh sửa địa chỉ hiển thị, và bạn có thể chỉnh sửa các giá trị đơn lẻ.

Để chỉnh sửa địa chỉ hiển thị, click vào vùng ngoài hình chữ nhật. Logisim sẽ vẽ một

- + gõ các chữ số của hệ 16 sẽ thay đổi các địa chỉ top tương ứng .
- + gõ Enter sẽ xuống một dòng
- + gõ backspace sẽ lên một dòng
- + gõ spacebar sẽ cuộn xuống 1 trang (4 dòng)

Để chỉnh sửa một giá trị cụ thể, click vào giá trị đó trong vùng chữ nhật hiển thị. Logicsim sẽ vẽ một hình chữ nhật xung quanh địa chỉ đó.

- + gõ các chữ số của hệ 16 sẽ thay đổi giá trị ở địa chỉ đang được chỉnh sửa.
- + gõ enter sẽ chuyển tới chỉnh sửa giá trị tiếp theo ngay dưới nó (xuống một dòng).
- + gõ backspace sẽ chuyển tới chỉnh sửa giá trị của địa chỉ trước đó.
- + gõ spacebar sẽ chuyển tới chỉnh sửa giá trị của địa chỉ kế tiếp.

7.2 Pop-up menus and files

Một trình đơn bật lên cho bộ nhớ gồm có thêm 4 lựa chọn bên cạnh các lựa chọn thông thường cho tất cả các thành phần.

- +Edit content: đưa ra một trình soạn thảo hệ 16 để chỉnh sửa nội dung bộ nhớ
- +Clear content: xác lập lại giá trị của tất cả các ô nhớ về 0.
- +Load Image: thiết lập lại tất cả các giá trị của phần bộ nhớ dựa trên các giá trị được tìm thấy trong một tệp sử dụng khuôn thức sẽ được diễn giải dưới đây.
- + Save Image: Lưu trữ tất cả các giá trị của bộ nhớ vào một tệp sử dụng khuôn thức như dưới đây.

Các định dạng tệp sử dụng cho các tệp ảnh rất đơn giản, nó cho phép bạn viết một chương trình, như một chương trình assembler, tạo ra các bộ nhớ ảnh mà có thể sau đó truyền vào trong bộ nhớ. Một ví dụ của khuôn thức này: nếu chúng ta có một bộ nhớ 256 bytes mà 5 bytes đầu là: 2, 3, 0, 20 và -1 và tất cả các giá trị còn lại bằng 0 thì chúng ta sẽ có một tệp ảnh như sau:

```
V2.0 raw
02
03
00
14
ff
```

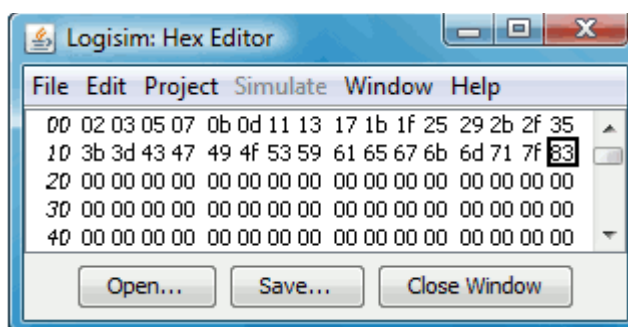
dòng đầu tiên xác định khuôn thức tệp đang được sử dụng (hiện tại, chỉ một định dạng tệp được tìm thấy). Các giá trị tiếp theo liệt kê các giá trị trong hệ 16, bắt đầu từ địa chỉ 0, bạn có thể đặt vài giá trị như thế vào cũng một dòng. Logicsim sẽ tự cho rằng, các giá trị ko được liệt kê bằng 0.

Tệp ảnh có thể sử dụng run-length encoding, ví dụ thay vì liệt kê giá trị 00 16 lần trong một hàng, tệp đó có thể truyền vào 16 * 00 thay vì lặp lại 00 mười sáu lần. Chú ý rằng số lần lặp lại

được viết trong hệ cơ số 10. Tập được tạo ra bởi logicsim sẽ sử dụng run-length encoding để chạy qua ít nhất 4 giá trị.

7.3 Hex editor

Logisim bao gồm một hex editor tích hợp để theo dõi và chỉnh sửa nội dung bộ nhớ. Để sử dụng, chọn thẻ memory trong giao diện, chọn edit contents... Đối với thành phần ROM, bạn có thể truy cập hex editor bằng cách click vào các giá trị tương ứng.



Những số in nghiêng nằm ở cột ngoài cùng bên trái, hiển thị giá trị địa chỉ của bộ nhớ, viết trong hệ 16. Các số khác hiển thị các giá trị bắt đầu từ giá trị ô nhớ đó, hex editor có thể hiển thị 4, 8 hoặc 16 giá trị trên một dòng, phụ thuộc vào kích cỡ của cửa sổ. Để tiện cho việc đếm, giữa nhóm 4 giá trị có một khoảng trắng lớn ngăn cách.

Bạn có thể di chuyển đến các ô giá trị bằng thanh cuộn hoặc sử dụng các phím di chuyển để thay đổi giá trị của ô nhớ bằng cách gõ vào các giá trị hệ 16.

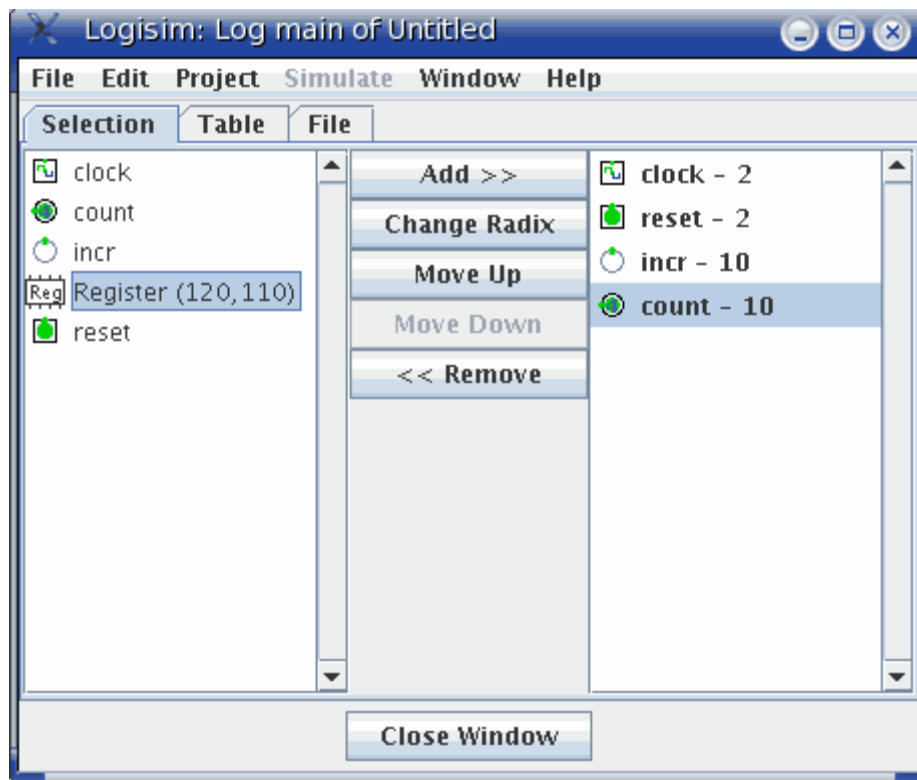
Bạn có thể chọn một vùng các giá trị bằng cách kéo thả chuột, shift-clicking hoặc giữ shift khi dùng phím di chuyển. Các giá trị có thể được sao chép và dán bằng lựa chọn edit. Bảng ghi giá trị này có thể được chuyển sang các ứng dụng khác.

8. LOGGING

Khi thử một mạch điện lớn và phức tạp, đôi khi bạn cần lưu lại các trạng thái hoạt động của nó. Đây là mục đích thiết lập nên của chế độ logging trong logicsim, cho phép người sử dụng chọn các thành phần mà giá trị của chúng nên được ghi lại. Một cách tùy ý, bạn có thể chọn một tập để lưu các giá trị ghi đó.

Chú ý: đây là giai đoạn quan trọng nhất, tuy có phần nhàm chán nhưng nó rất thuận tiện trong việc chỉnh sửa sau này. Nếu người sử dụng không chèn thêm các comments thì bản ghi sẽ không thay đổi.

Bạn có thể vào chế độ logging qua menu Simulate trên khay hệ thống. Một khay hệ thống gồm 3 thẻ hiện ra.

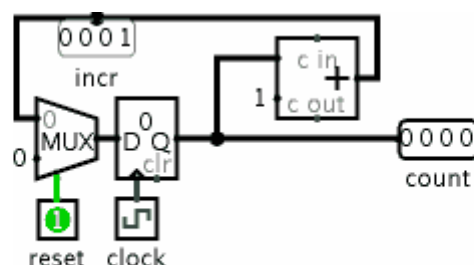


Mỗi project chỉ có một cửa sổ logging. Vì vậy, khi người sử dụng chuyển đến một mạch khác trong cùng một project, cửa sổ logging sẽ tự động chuyển sang chế độ logging của mạch đó.

Chú ý khi chế độ logging chuyển sang ghi chép cho một mô phỏng khác, nó sẽ kết thúc và ghi vào một tệp. Khi bạn quay trở lại, nó sẽ ghi nhớ cấu hình của mô phỏng đó, nhưng người sử dụng sẽ phải khởi động lại chế độ logging này.

8.1 Thẻ lựa chọn (the selection tab)

Thẻ selection cho phép người sử dụng chọn giá trị nào để ghi vào bản log. Cửa sổ dưới đây tương ứng với mạch sau





Thẻ selection được chia làm 3 vùng làm việc: vùng đầu tiên (ngoài cùng bên trái) liệt kê tất cả các thành phần trong mạch mà giá trị của chúng có thể được ghi lại vào log. Trong các thư viện tích hợp sẵn, các loại thành phần sau có thể được logging:

Thư viện cơ sở: Pin, Probe và thành phần Clock.

Thư viện bộ nhớ: tất cả các thành phần

Đối với các thành phần có nhãn đi cùng thì tên của chúng tương ứng với các nhãn đó; tên của các thành phần khác đặc trưng cho loại của chúng và vị trí của chúng trong mạch. Các mạch con cũng hiện tên trong cột này, chúng không thể bị ghi lại, nhưng các thành phần thích hợp bên trong chúng thì có thể. Chú ý rằng các thành phần RAM yêu cầu người sử dụng chọn địa chỉ phần bộ nhớ nên được ghi lại, nó cho phép ghi lại cho chỉ 256 địa chỉ đầu tiên.

Vùng ngoài cùng bên phải liệt kê các thành phần đã được lựa chọn. Nó cũng chỉ ra cơ sở mà các giá trị đa bit được ghi vào, cơ sở này không có tác dụng cụ thể đối với các giá trị đơn bit.

Cột giữa gồm các nút cho phép điều khiển các mục trong cột selection

Add: thêm các mục đang được chọn trong cột trái vào cột phải

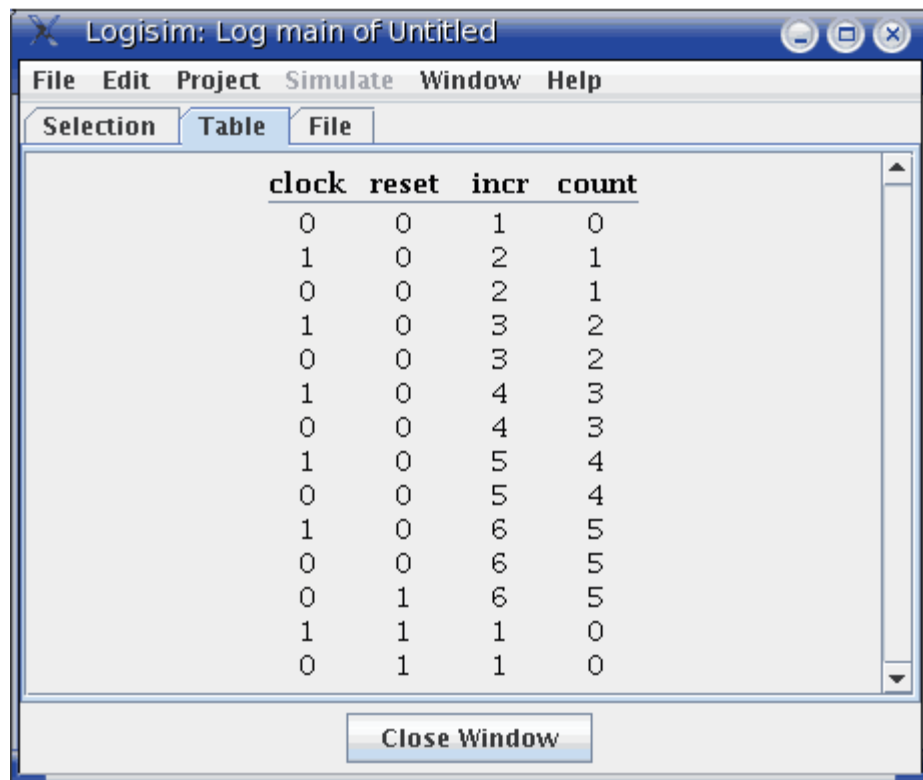
Change radix: thay đổi các cơ sở của các thành phần trong cột phải giữa các hệ: nhị phân, thập phân và 16

Move up: di chuyển lên

Move down: di chuyển xuống

Remove: xóa thành phần đang được chọn khỏi cột phải

8.2 Thẻ table



The screenshot shows a window titled "Logisim: Log main of Untitled" with a menu bar (File, Edit, Project, Simulate, Window, Help) and three tabs (Selection, Table, File). The "Table" tab is active, displaying a table of simulation data. The table has four columns: clock, reset, incr, and count. The data is as follows:

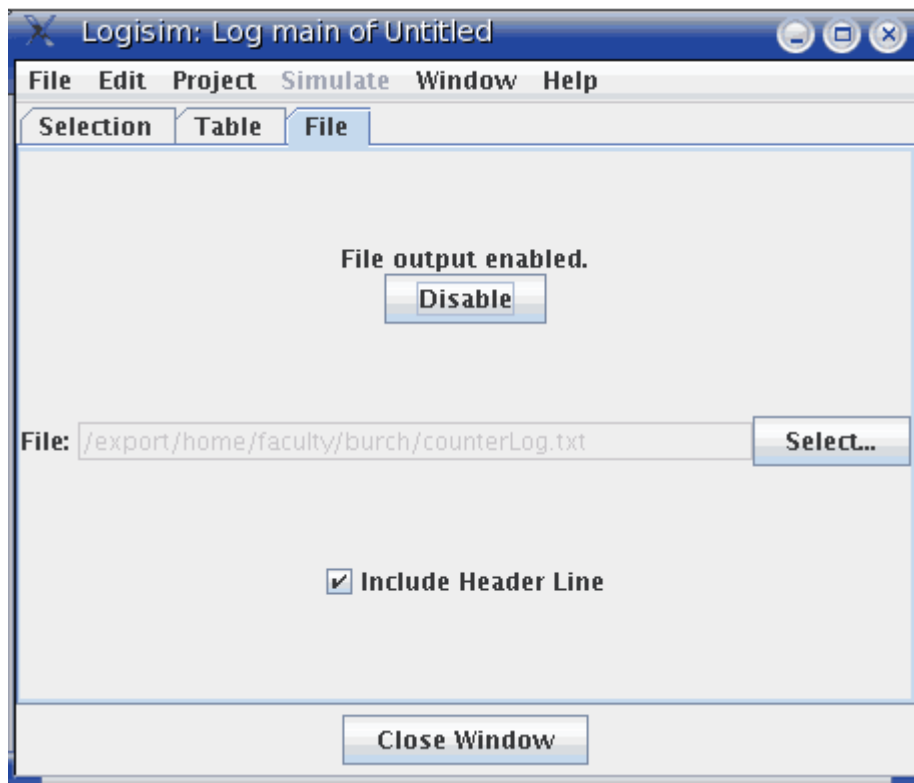
clock	reset	incr	count
0	0	1	0
1	0	2	1
0	0	2	1
1	0	3	2
0	0	3	2
1	0	4	3
0	0	4	3
1	0	5	4
0	0	5	4
1	0	6	5
0	0	6	5
0	1	6	5
1	1	1	0
0	1	1	0

At the bottom of the window is a "Close Window" button.

Mỗi thành phần trong cột selection sẽ có một cột tương ứng. Mỗi hàng hiển thị một snapshot sau khi sự truyền giá trị được thực hiện. Các dòng giống nhau sẽ ko được thêm vào trong log. Chú ý rằng chỉ có khoảng 400 hàng được hiển thị. Một số hàng có thể có các phần trống nếu các thành phần tương ứng không nằm trong vùng lựa chọn khi mà hàng đó được tính toán.

8.3 Thẻ tệp

Thẻ tệp cho phép người sử dụng xác định một tệp để lưu các log vào



Trên cùng là một chỉ thị cho biết việc lưu log vào file có đang được thực hiện hay không và một nút cho phép bật hoặc tắt nó. (chú ý rằng người sử dụng không thể enable nó cho đến khi một file được xác định ở phía dưới.) Nút cho phép người sử dụng tạm dừng và khởi động lại việc ghi chép. Khi người sử dụng chuyển vào cửa sổ của project để quan sát sự giả lập khác, quá trình log sẽ tự động dừng lại; nếu người sử dụng quay lại và muốn tiếp tục log, họ phải thiết lập lại bằng tay sử dụng nút trên cùng.

ở giữa là chỉ thị cho biết tệp nào đang được ghi vào. Để thay đổi nó, sử dụng nút Select. Khi chọn một tệp, chế độ logging sẽ được tự động khởi động. Nếu chọn một tệp đã có sẵn, logicsim sẽ hỏi nếu người sử dụng muốn ghi đè lên hay mở rộng tệp về phía cuối.

dưới cùng người sử dụng có thể điều khiển một dòng tệp header có nên đặt vào trong một tệp để chỉ ra mục nào được chọn hay không. Nếu header được thêm vào, một dòng header mới sẽ được đặt vào tệp ngay khi các mục thay đổi.

Định dạng tệp

Các ghi chép được đặt vào trong tệp dưới hình thức gần giống với những gì hiện lên trong thẻ Table. (Một điểm khác là bất cứ header lines nào sẽ chứa đường dẫn tới các đối tượng nằm trong các mạch phụ.) Định dạng này được thiết kế đơn giản để người sử dụng có thể đưa nó vào trong các chương trình khác để xử lý, như là Python/perl hoặc các chương trình bảng tính.

Vì thế các bản ghi có thể xử lý tệp cùng lúc khi logicsim đang chạy, logicsim sẽ nạp bản ghi mới vào đĩa cứng sau mỗi 500ms. Chú ý rằng Logicsim cũng có thể đóng một cách gián

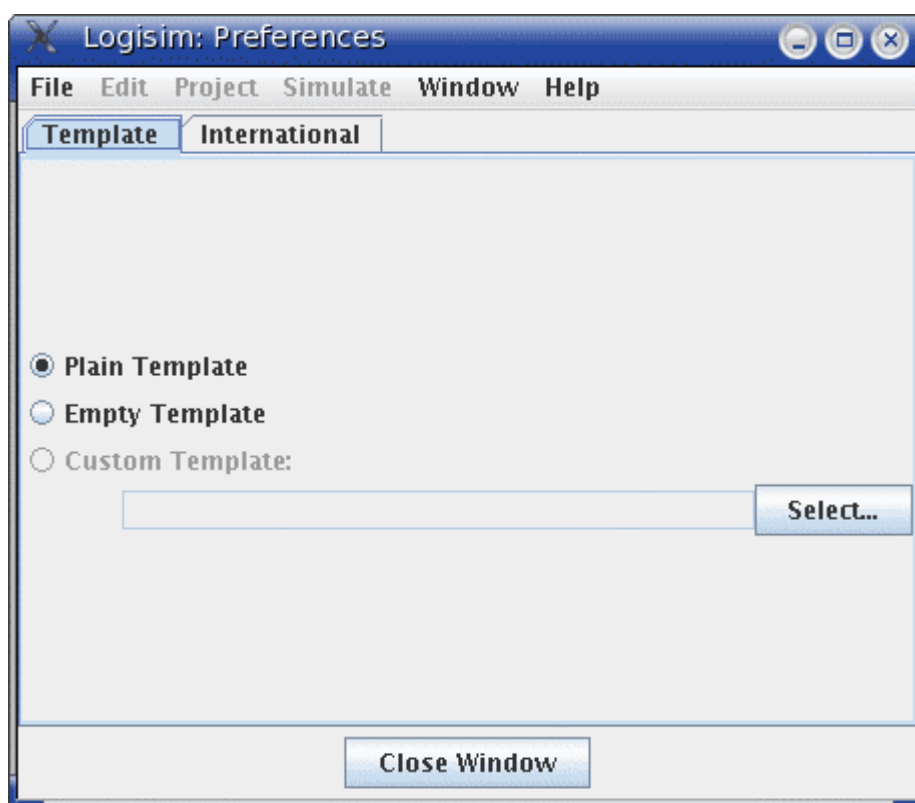
đoạn hoặc mở lại các tệp trong thời gian chạy, đặc biệt khi một vài giây trôi qua mà không có thêm bản ghi nào được nạp vào.

9. APPLICATION PREFERENCES

Logisim hỗ trợ hai loại tùy chọn cấu hình là application preferences và project options. Application preferences hiển thị tất cả các project đang mở còn project options chỉ làm việc với một project cụ thể. Phần này sẽ nói về application preferences còn project options sẽ được nói tới trong phần khác

Bạn có thể xem và chỉnh sửa application preferences bằng cách vào File → Preferences. Một cửa sổ sẽ hiện ra với hai thẻ(tab). Chúng ta sẽ giới thiệu từng thẻ một và sau đó tìm hiểu cách cấu hình từ dòng lệnh.

9.1 Thẻ Template



Một mẫu là một file của logisim mà được khởi động mỗi khi tạo ra một project mới. Ngoài ra, nếu bạn có sẵn một file logisim với môi trường làm việc đã được chỉnh sửa, bạn có thể sử dụng nút Revert All to Template để đưa về mặc định trong cửa sổ để sửa Project Options

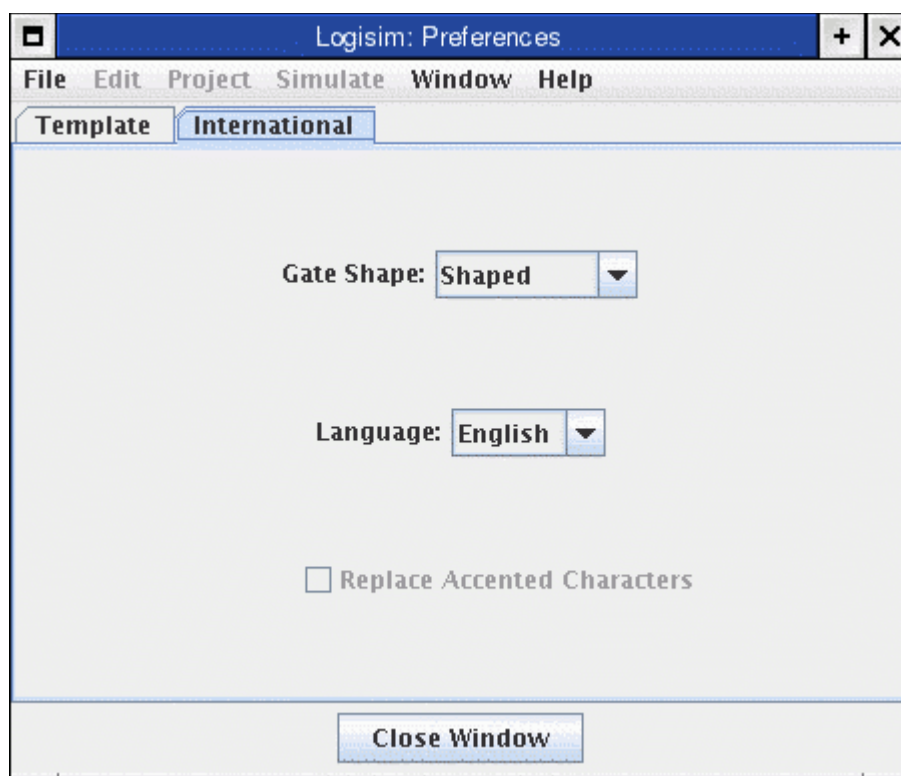
Mặc dù những mẫu rất hiệu quả khi sử dụng ở những trường hợp khác, chúng hợp lý nhất khi sử dụng trong lớp học, nơi mà những hướng dẫn viên có thể đưa cho sinh viên những mẫu để bắt

đầu làm việc. Việc này đặc biệt thích hợp nếu lớp học sử dụng logisim một cách thường xuyên với các tính năng nâng cao. Trong trường hợp đó, thiết lập mặc định tỏ ra không hiệu quả. Mẫu cũng có thể dùng trong lớp khi mà hướng dẫn viên cho phép sinh viên nộp bài mà họ tự thiết lập môi trường làm việc

Mặc định, tùy chọn **Plain Template** sẽ được chọn, sử dụng mẫu mặc định của logisim. Nếu bạn muốn một thiết lập cơ bản nhất, bạn có thể chọn **Empty Template**. Nhưng nếu bạn muốn một file khác sử dụng mẫu, chọn **Select** và chọn mẫu sau đó chọn **Custom template**.

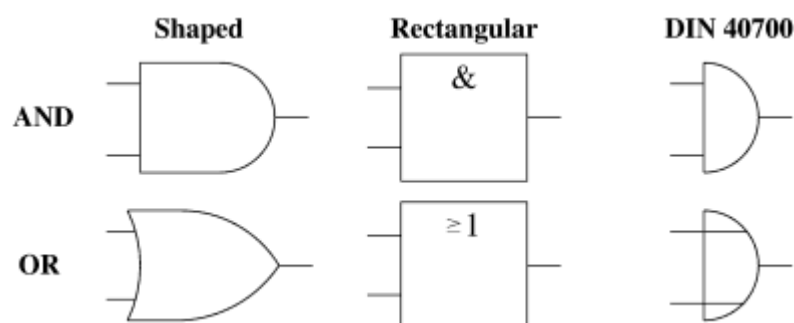
9.2 Thẻ International

Thiết lập này hỗ trợ tùy chọn để Logisim có thể sử dụng được ở nhiều nước



Thẻ này có ba lựa chọn:

- **Hình dạng cổng:** logisim hỗ trợ ba chuẩn để vẽ cổng: cổng định hình dạng, cổng dạng chữ nhật, và cổng kiểu DIN 40700. Bảng sau đây thể hiện sự khác biệt giữa chúng



Bởi vì hình dạng định sẵn thì phổ biến ở Mỹ trong khi kiểu dáng hình chữ nhật thì lại phổ biến ở Châu Âu, một số người quy định những kiểu này theo vùng của họ. Bởi vậy, từ shaped(hình dáng định sẵn) và rectangular(hình chữ nhật) được ưa thích dùng. Chuẩn DIN 40700 là chuẩn dùng cho biên tập nội dung tương tự (analog) và nội dung số (digital) được đưa ra bởi tổ chức chuẩn của Đức tên là DIN. DIN đưa ra chuẩn hình chữ nhật cho các thành phần số vào năm 1976 nhưng một số kỹ sư vẫn thích dùng những kiểu cũ hơn nhưng số này ngày càng ít đi.

Logisim không tuân theo chặt chẽ một chuẩn nào. Nó đi theo hướng để có thể chuyển đổi giữa các chuẩn. Một số trường hợp, cổng có hình dáng sẵn thì thường vuông hơn kích thước được định ra bởi chuẩn IEEE. Và, mặc dù cổng XOR và XNOR cần phải có chiều rộng giống với cổng OR và NOR với kiểu hình chữ nhật. Chúng không được tạo ra như thế vì sự khó khăn trong việc tạo ra hình dáng của cổng XOR.

Hiện nay, cổng OR chữ nhật thường được vẽ với một ký hiệu “>0” hơn là ký hiệu chuẩn được đề cập ở phần trên bởi vì rất nhiều hệ thống không hỗ trợ đầy đủ bộ mã Unicode.

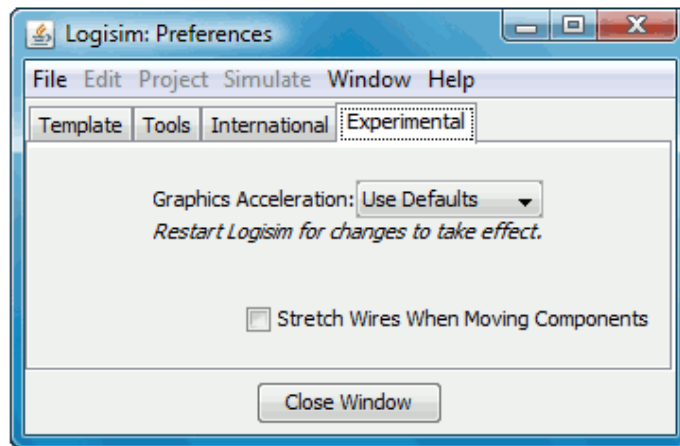
Ngôn ngữ: Thay đổi giữa các ngôn ngữ. Phiên bản hiện tại hỗ trợ cả Tiếng Anh và tiếng Tây Ban Nha. Phân dịch sang tiếng Tây Ban Nha được đóng góp bởi Pablo Leal Ramos, Tây Ban Nha.

Chúng tôi hoan nghênh tất cả mọi người dịch logisim sang các ngôn ngữ. Nếu bạn thấy thích thú với việc này, hãy liên hệ với tôi, Carl Burch. Tôi sẽ thông báo cho bạn việc phải làm, gửi cho bạn hướng dẫn và việc này không đòi hỏi hiểu biết về ngôn ngữ JAVA.

Thay đổi các ký tự: Một số hệ thống hỗ trợ không tốt cho các ký tự (như ñ hoặc ö) mà không có trong bộ mã ASCII 7 bit. Khi việc này được kiểm tra, logisim sẽ thay thế tất cả những ký tự trên với những ký tự tương ứng của bộ mã ASCII 7 bit. Phần đánh dấu được ẩn vì hiện nay ngôn ngữ hiện tại không có một dị bản nào khác.

9.3 Thẻ Experimental

Những thiết lập này liên quan đến những tính năng thí nghiệm mà được đưa vào dựa trên những phản hồi của người dùng.



- **Tăng tốc đồ họa:** Một số người sử dụng logisim phản ánh rằng Logisim xử lý đồ họa tương đối chậm. Một người dùng tiến hành thêm lệnh `Dsun.java2d.d3d = true` trong phần dòng lệnh để chỉ ra rõ ràng điều này. Bạn có thể điều khiển việc này bằng cách sử dụng hộp thực đơn xổ xuống. Tôi chưa từng xem xét vấn đề tốc độ của logisim, vì thế không thể đánh giá một cách chính xác được. Tôi tin tưởng vào những phản hồi của người dùng về việc này. Một điều lưu ý rằng nó không có tác dụng khi bạn chưa khởi động lại logisim.
- **Kéo dài dây nối:** Khi các thành phần được di chuyển theo chiều ngang hoặc dọc. Logisim tự động thêm dây nối để bảo toàn sự kết nối. Những phiên bản trong tương lai có thể đặt thuộc tính này là mặc định mà không thể tắt đi được nhưng tôi muốn cho người dùng cơ hội để thử và đề xuất ý kiến về nó.

9.4 Tùy chọn Command-line

Bạn có thể thiết lập phần application preferences thông qua dòng lệnh. Việc này thực sự hiệu quả khi bạn muốn khởi động logisim với thiết lập giống nhau cho mọi sinh viên ngay cả khi các sinh viên trước đã thay đổi một số thiết lập.

Cú pháp của việc thiết lập bằng dòng lệnh như sau:

```
java -jar jarFileName [options] [filenames]
java -jar jarFileName [thiết lập] [tên file]
```

Việc thêm file vào dòng lệnh sẽ được mở bởi những cửa sổ khác nhau trong logisim.

Ví dụ sau đây mô tả việc bật Logisim với những thiết lập cơ bản

```
java -jar jarFileName -plain -gates shaped -locale en
```

Các tùy chọn hỗ trợ

```
-plain
-empty
```

`-template templateFile`

Thiết lập các mẫu để dùng logisim

`-gates [shaped|rectangular]`

Thiết lập dạng cổng được dùng.

`-locale localeIdentifier`

Thiết lập ngôn ngữ được dùng. Ở đây hỗ trợ hai loại là tiếng Anh và Tây Ban Nha

`en` English

`es` Spanish

`-accents [yes|no]`

Tùy chọn này chỉ được dùng cho những ngôn ngữ sử dụng kí tự ngoài bảng mã 7 bit ASCII. Những kí tự ngoài bảng mã sẽ được thay thế bằng những kí tự tương ứng. Điều này rất thuận lợi cho những hệ thống không hỗ trợ các loại kí tự này.

`-nosplash`

Tắt cửa sổ hiện thị đang mở Logisim khi bắt đầu khởi động

`-help`

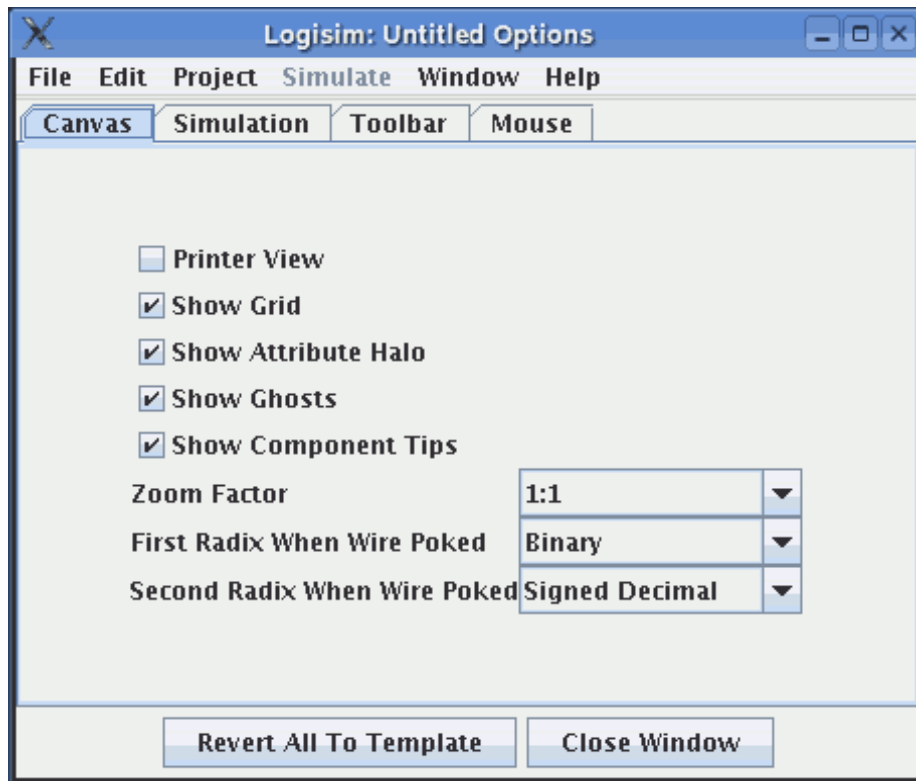
Hiện thị tổng quát về các lệnh trong phần thiết lập dòng lệnh

`-version`

Hiện thị phiên bản của logisim

10. PROJECT OPTION

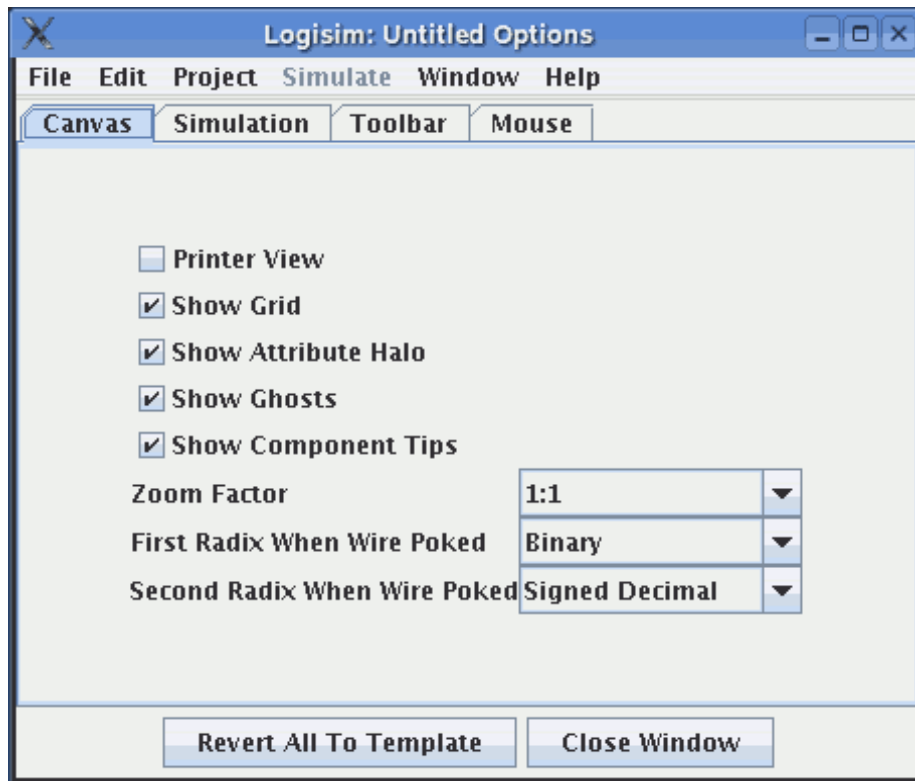
Bạn có thể xem và chỉnh sửa phần project option bằng cách vào Project➔Option



Ở dưới cùng là nút lệnh **Revert All to Template** để đưa tất cả các thiết lập và thuộc tính về mẫu hiện tại

10.1 Thẻ Canvas

Thẻ Canvas cho phép thiết lập các thông số cho vùng để vẽ mạch.

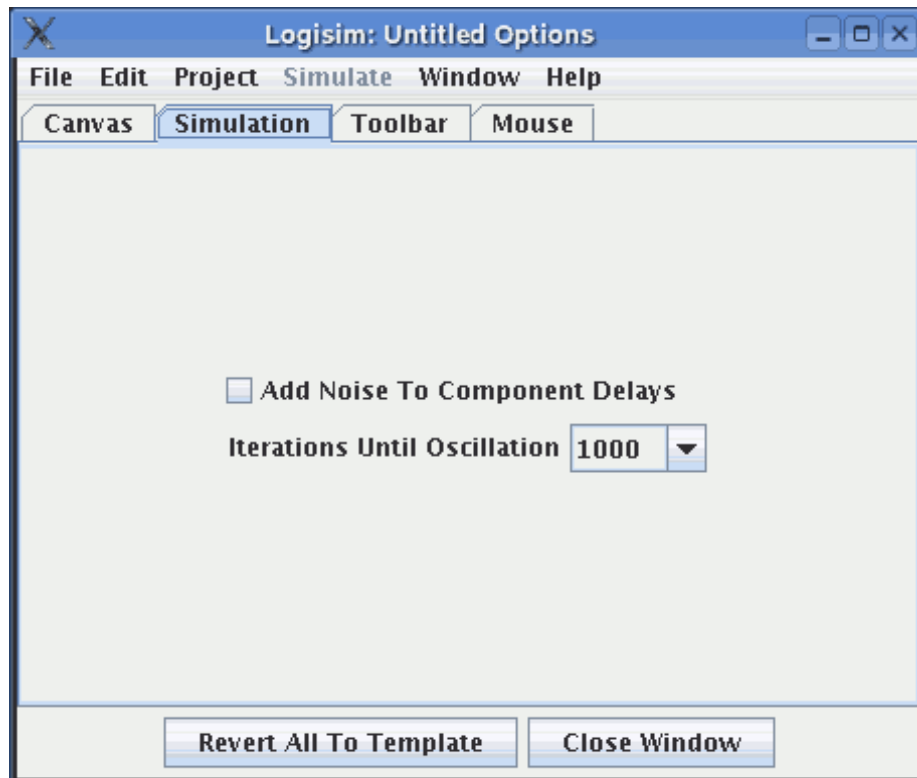


Có tám lựa chọn:

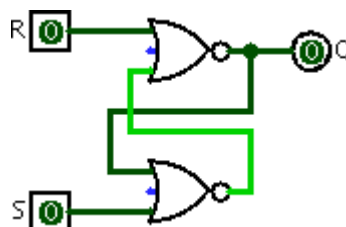
- **Printer View:** Xác định xem mạch có được hiển thị trên màn hình giống như qua máy in không. Thông thường, Logisim hiển thị mạch trên màn hình với chỉ dẫn để biết trạng thái của mạch và nó cũng hiển thị một vài hướng dẫn về giao diện các thành phần. Còn máy in thường bỏ qua những chỉ dẫn về trạng thái và những chỉ dẫn về giao diện các thành phần.
- **Show Grid:** Xác định xem có hiển thị các dấu chấm trên nền khu vực vẽ hay không.
- **Show Attribute Halo:** Xác định xem có hiển thị hình ovan xung quanh các thuộc tính hay công cụ mà đang các thuộc tính của nó đang được hiển thị ở bảng thuộc tính.
- **Show Ghosts:** Xác định xem viền màu xám có được hiển thị quanh các thành phần khi di chuột qua vùng vẽ mạch. Ví dụ, nếu bạn chọn cổng AND và di chuột vào cửa sổ nhưng không click, một viền màu xám của cổng AND sẽ xuất hiện.
- **Show component Tips:** Xác định có hiển thị “tool tips” tạm thời khi chuột di qua các thành phần của mạch.
- **Zoom Factor:** cho phép bạn phóng to hay thu nhỏ hình ảnh bởi một tỉ lệ nào đó.
- **First Radix When Wire Poked:** thiết lập các giá trị sẽ được hiển thị như thế nào khi một dây nối được chọn sử dụng công cụ Poke. Khi được chọn, các giá trị sẽ được hiển thị tạm thời cho đến khi người dùng click chuột vào vùng khác.
- **Second Radix When Wire Poked:** Thiết lập phần thứ hai của giá trị sẽ được hiển thị như thế nào.

10.2 Thẻ Simulation

Thẻ Simulation cho phép thiết lập các thuật toán cho việc giả lập các mạch. Các tham số được áp dụng cho tất cả các mạch đang được giả lập trong cùng một cửa sổ, ngay cả khi các mạch đó tồn tại ở một thư viện khác trong project.



- **Add Noise To Component Delays:** cho phép bạn bật hoặc tắt các nhiễu ngẫu nhiên mà được thêm vào sự trễ của các thành phần. Sự mô phỏng bên trong sử dụng một đồng hồ ẩn cho sự giả lập này, để cho thực tế, các thành phần (trừ dây nối và bộ chia) đều có một độ trễ nhất định giữa việc chúng nhận input và trả ra output. Nếu thiết lập này được chọn, Logisim sẽ thường xuyên tạo ra các thành phần nhận tín hiệu click chuột chậm hơn một chút. Sự ngẫu nhiên này được thêm vào để giải quyết vấn đề của mạch chốt. Nếu không có độ nhiễu ngẫu nhiên, mạch sẽ dao động khi cổng làm việc như nhau. Nhưng khi nhiễu ngẫu nhiên được thêm vào, một cổng sẽ làm việc nhanh hơn cổng còn lại.

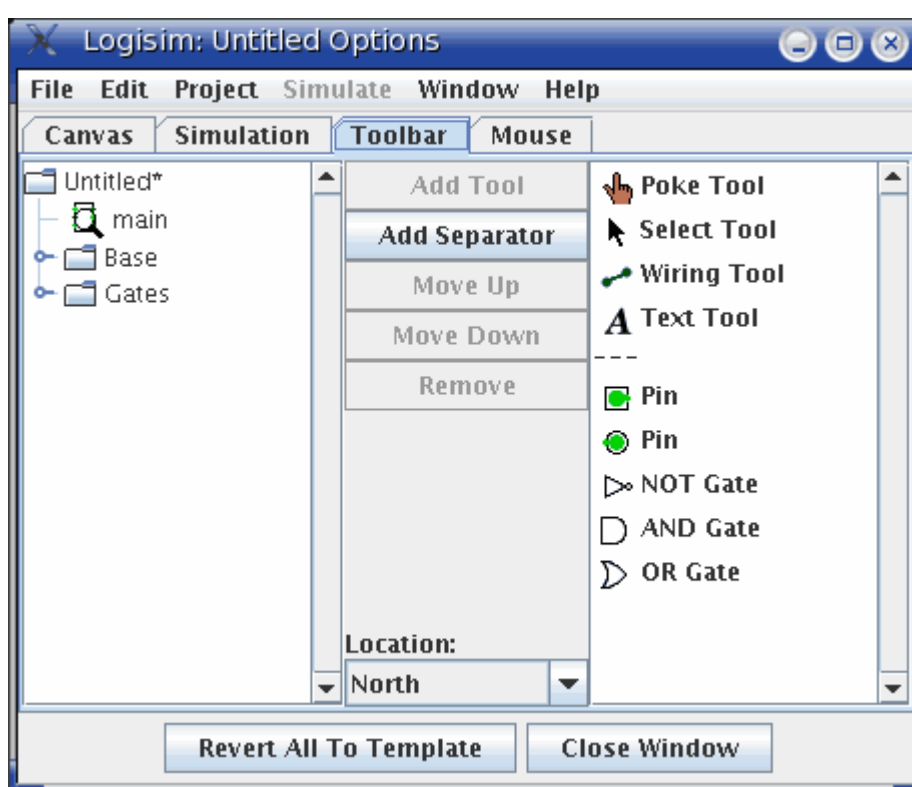


Tôi khuyến cáo nên bật thiết lập này. Vì sự thêm vào độ nhiễu này là tương đối mới vì vậy chúng có thể dẫn đến một số lỗi trong khi giả lập mạch.

Iterations Until Oscillation: Cho phép bạn thiết lập thời gian bao lâu để mô phỏng mạch trước khi quyết định nó đang dao động. Số trong hộp mô tả số lượng clicks của đồng hồ ẩn bên trong (công bình thường chỉ cần một click). Con số mặc định là 1000 là đủ tốt cho hầu hết các mục đích ngay cả khi mạch là lớn. Nhưng bạn có thể tăng con số này lên nếu bạn đang làm việc với một mạch mà Logisim báo là dao động bị sai. Có thể tưởng tượng rằng một mạch là kết hợp của cách mạch chốt có thể tạo ra lỗi. Bạn cũng có thể phải giảm con số này xuống khi bạn làm việc với một mạch mà có xu hướng dao động và bạn đang sử dụng một bộ xử lý chậm chạp.

10.3 Thẻ Toolbar

Thẻ Toolbar cho phép bạn thiết lập những công cụ sẽ xuất hiện trên thanh công cụ.



Bên trái là một cửa sổ liệt kê tất cả các công cụ, và danh sách bên phải là những công cụ hiện đang được hiển thị trên thanh công cụ. "---" là kí hiệu của dấu ngăn cách hiển thị như một khoảng trắng nhỏ. Ở giữa là năm nút bấm chọn là một thực đơn sổ xuống:

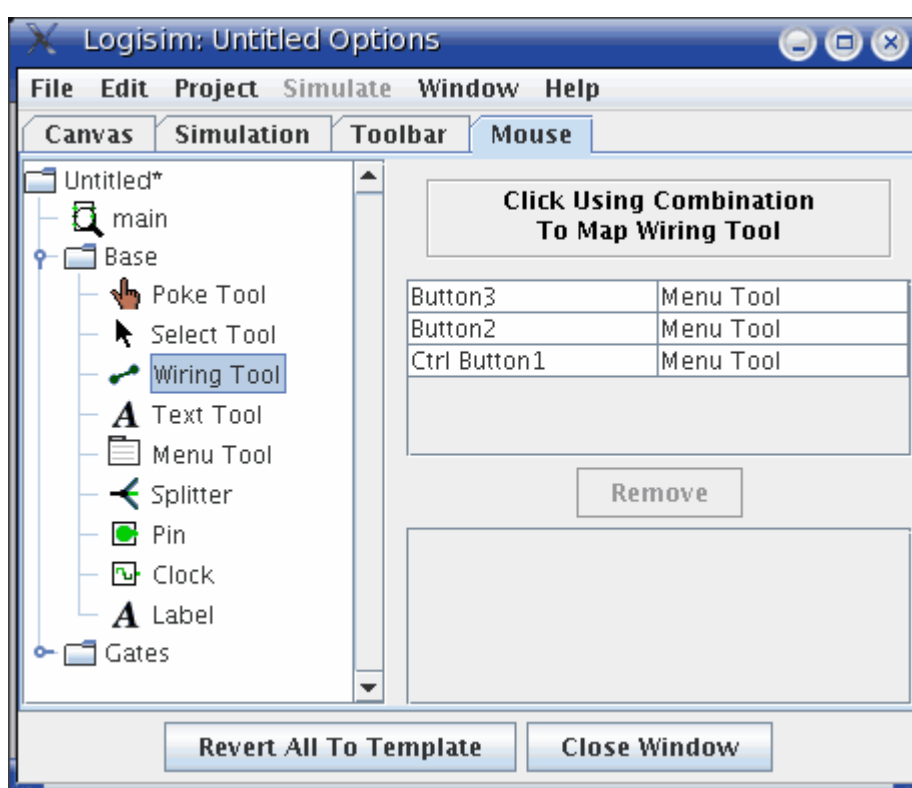
- **Add Tool:** Thêm công cụ đang được chọn ở cửa sổ bên trái vào cuối thanh công cụ
- **Add Separator:** Thêm dấu ngăn cách vào cuối thanh công cụ.
- **Move Up:** chuyển công cụ đang được chọn lên một vị trí trong thanh công cụ
- **Move Down:** Chuyển công cụ đang được chọn xuống một vị trí trong thanh công cụ
- **Remove:** Xóa công cụ được chọn khỏi thanh công cụ
- **Location:** Thiết lập vị trí của thanh công cụ trong cửa sổ logisim. Thanh công cụ có thể đặt ở một trong bốn viền của cửa sổ làm việc được kí hiệu là Bắc, Nam, Đông, Tây. Chúng cũng có thể đặt thuộc tính ẩn hoặc có thể đặt ở giữa tức là ở bên trái của khu vực vẽ nhưng ở bên phải của cửa sổ các đối tượng và bảng thuộc tính.

Các thuộc tính của các công cụ không được hiển thị trong cửa sổ này, để làm việc đó, bạn có thể xem và sửa nó trong cửa sổ chính của chương trình.

10.4 Thẻ Mouse

Mặc định, khi bạn click chuột vào khu vực vẽ của Logisim, công cụ đang được chọn sẽ được sử dụng. Nếu bạn click chuột phải hoặc phím điều khiển, một cửa sổ công cụ sẽ hiện lên cho những thành phần hiện tại đang ở bên dưới trỏ chuột.

Logisim cho phép bạn thay đổi thiết lập này, để bạn không phải dùng tới thanh công cụ hoặc cửa sổ chứa các thành phần nhiều lần. (Điều này rất thuận tiện nếu bạn thuận tay trái). Mỗi một tổ hợp của phím chuột và một phím điều khiển (Shift, Control) sẽ được thay thế cho một công cụ khác nhau. Thẻ mouse cho phép bạn thiết lập các thay thế này.



- Ở bên trái là một cửa sổ nơi bạn có thể chọn các công cụ bạn muốn thay thế bằng.
- Ở phía trên bên phải là một hình chữ nhật, bạn click vào đó để sử dụng tổ hợp chuột, phím thay thế cho một thành phần. Ví dụ, bạn muốn tạo một dây nối mới bằng việc sử dụng cách giữ phím shift, đầu tiên bạn chọn công cụ Wiring trong phần cửa sổ bên trái; sau đó bạn click + shift vào dòng chữ “Click Using Combination to Map Wiring Tool”. Nếu tổ hợp phím chuột bạn chọn đã được sử dụng, khi đó nó sẽ thay thế cho các trước đó.
- Bên dưới khu vực này là danh mục các thành phần đã được dùng tổ hợp chuột và phím này. Lưu ý rằng bất cứ tổ hợp nào mà không được liệt kê sẽ được dùng bình thường bằng cách chọn.

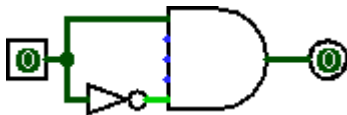
- Bên dưới là một nút bấm Remove cho phép bạn xóa tổ hợp phím chuột đang được chọn ở bảng liệt kê phía trên. Trong tương lai, tổ hợp chuột phím sẽ có thể thay thế cho bất kì công cụ nào đang được chọn ở thanh công cụ hay cửa sổ chứa các thành phần.
- Bên dưới là danh sách của các thuộc tính của công cụ đang được chọn trong danh sách của các tổ hợp đã được tạo. Mỗi tổ hợp này có một thuộc tính riêng, khác với thuộc tính được sử dụng trên thanh công cụ và cửa sổ chứa các thành phần. Bạn có thể thay đổi các thuộc tính ở đây.

11. SỰ TRUYỀN GIÁ TRỊ

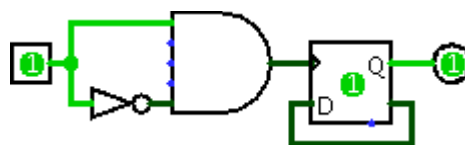
Thuật toán của Logisim dùng cho việc mô phỏng sự truyền đi của các giá trị trong mạch hiếm khi làm bạn phải lo lắng. Chúng đủ phức tạp để có thể mô tả sự trễ của các cổng, nhưng không đủ thực tế để có thể mô tả những hiện tượng khó lường hạn như sự thay thay đổi thường xuyên điện áp.

11.1 Sự trễ của cổng

Dưới đây là một ví dụ cho thấy sự tinh vi của thuật toán sử dụng trong logisim.



Rõ ràng rằng output luôn là 0. Nhưng trong thực tế cổng NOT không đáp ứng ngay lập tức khi nhận được input, và cũng như vậy trong Logisim. Do vậy, khi input của mạch chuyển từ 0 sang 1, cổng AND sẽ nhận hai input là 1 và do đó sẽ trả ra output là 1 trong một khoảng thời gian ngắn. Bạn không nhìn thấy điều đó trên màn hình nhưng kết quả này có thể thấy rõ qua việc sử dụng output của cổng AND làm input cho đồng hồ của một flip-flop D.



Khi output của cổng AND xuất hiện sẽ ngay lập tức đi vào flip-flop D. Do đó, giá trị sau khi qua flip-flop D sẽ thay đổi mỗi lần output của cổng AND chuyển từ 0 sang 1.

Mỗi thành phần đều có sự trễ bên trong. Các thành phần phức tạp trong Logisim thường có độ trễ lớn hơn. Tuy nhiên độ trễ một phần nào đó không biết trước được do đó có thể không diễn tả đúng so với trong thực tế.

Thỉnh thoảng, với tần số không đều, Logisim sẽ thêm độ trễ vào sự xử lý của của một thành phần. Việc này là để mô phỏng sự không đều trong mạch thực. Trường hợp đặc biệt, một chốt R-S xử dụng hai cổng NOR sẽ hoạt động mà không cần đến sự trễ ngẫu nhiên vì hai cổng này

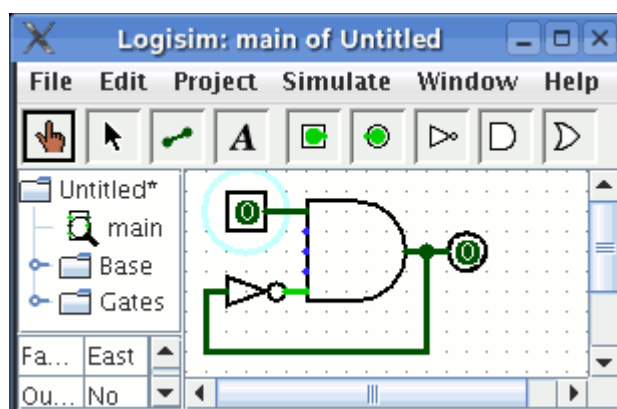
xử lý input ở cùng một thời điểm. Sự thêm độ trễ này có thể thay đổi ở thẻ **Simulation** trong Project Options.

Đứng từ góc độ kĩ thuật, không khó để xử lý vấn đề độ trễ trong các mạch đơn. Xử lý vấn đề độ trễ của cổng trong một mạch con phức tạp hơn một chút. Logisim đang cố gắng để diễn tả một cách chính xác nhất.

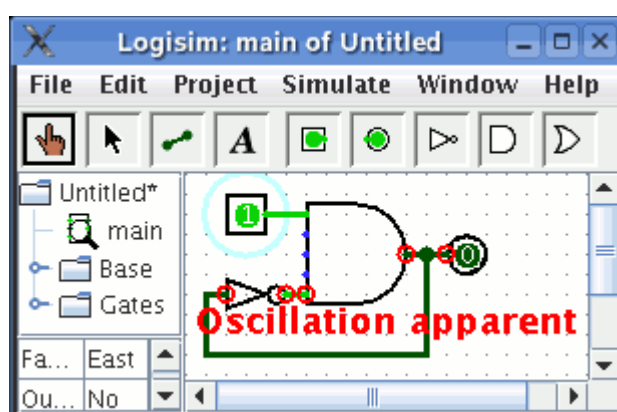
Lưu ý rằng tôi không nói về việc Logisim luôn luôn luôn mô tả đúng sự trễ của các cổng. Nhưng ít nhất Logisim sẽ cố gắng một cách tốt nhất có thể.

11.2 Lỗi dao động

Thuật toán truyền đi, thường làm việc một cách hiệu quả mà không xảy ra lỗi, sẽ trở nên dễ thấy hơn khi bạn tạo một mạch và cho nó dao động.



Mạch này đang ở trạng thái không thay đổi(tĩnh). Nhưng nếu bạn đổi input sang 1, mạch sẽ tạo ra một vòng lặp vô hạn. Sau một thời gian, Logisim sẽ thông báo rằng mạch này đang dao động.



Nó sẽ hiển thị giá trị ở thời điểm Logisim dừng chạy mạch. Giá trị này(trong hình vẽ) có thể sai. Cổng AND cho ra output 1 mặc dù một input của nó là 0 nhưng nó có thể do cổng NOT có input và output đều là 1.

Logisim khoanh tròn bằng màu đỏ ở nơi mà có thể tạo ra sự dao động. Nếu một điểm được đánh dấu nằm trong một mạch con, Logisim sẽ vẽ mạch con đó trong viền màu đỏ.

Khi Logisim phát hiện ra sự dao động, nó sẽ tắt toàn bộ sự mô phỏng. Bạn có thể bật lại sự mô phỏng bằng cách bật thiết lập Simulation Enabled trong thực đơn Simulate.

Logisim phát hiện sự dao động bằng một cách tương đối đơn giản. Nếu mạch mô phỏng có nhiều thứ được lặp lại, chúng sẽ dừng lại vào thông báo dao động. (Những điểm mà Logisim phát hiện ra là tham gia vào sự dao động là những điểm mà được chạy qua trong 25% cuối cùng của sự lặp lại). Vì vậy, nó có thể nhầm lẫn trong việc báo có sự dao động, đặc biệt khi bạn làm việc với một mạch rất lớn. Tuy nhiên đó có lẽ phải là một mạch lớn hơn tất cả các mạch mà tôi đã xây dựng bởi logisim. Trong bất cứ trường hợp nào, nếu bạn chắc rằng thông báo đó là không đúng, bạn có thể thiết lập lại số vòng lặp lại trước khi sự dao động xảy ra ở thẻ Simulation trong Project Options.

11.3 Hạn chế

Thuật toán truyền tin của Logisim đủ phức tạp cho các mục đích về dạy và học những không đủ phức tạp cho việc chế tạo mạch trong công nghiệp. Sự thiếu sót của thuật toán này thể hiện ở những điểm sau:

- Logisim không thể mô phỏng được các mạch con mà các chân có thể vừa làm input vừa làm output. Các thành phần được xây dựng bởi Java có thể có các chân như vậy: Trong thư viện có sẵn, thư viện bộ nhớ mạch RAM có một chân D mà có thể vừa làm input vừa làm output.
- Logisim dừng sự mô phỏng lại khi sự lặp lại vượt quá một con số định sẵn và thông báo có sự dao động. Có thể thấy rằng, một mạch lớn mà như vậy chưa phải là dao động sẽ gặp vấn đề.
- Ngoài vấn đề về độ trễ của cổng, Logisim không quan tâm đến thời gian của sự của độ trễ trong các phần tử. Sự trễ ngẫu nhiên này trong một số phần tử có thể làm cho giống với thực tế hơn nhưng chưa thể mô phỏng thực sự giống với thực tế.
- Logisim không phân biệt sự khác nhau giữa các mức điện áp: một bit chỉ có thể là bật, tắt, không xác định hoặc bị lỗi.
- Còn một số thiếu sót nữa mà tôi bỏ qua ở đây vì chúng tương đối khó hiểu và nếu bạn có hiểu thì Logisim cũng sẽ không đi theo hướng khắc phục nó. Ví dụ, tôi có một người bạn làm việc cho một công ty chuyên sản xuất chip, và công việc của anh ấy là về các “bubbles” trong chip theo công nghệ dây nano có thể dẫn tới sự mất kết nối đột ngột.
- Tôi không phải là một người thiết kế mạch chuyên nghiệp, vì vậy có thể có một số lỗi trong phương pháp truyền tải dữ liệu mà tôi chưa biết. Tôi rất mong có được sự đóng góp của những người chuyên nghiệp.