



Bộ môn Kỹ thuật Máy tính, Viện Công nghệ thông tin và Truyền thông, ĐHBKHN
www.dce.hust.edu.vn

Thiết kế IC

Msc. Nguyễn Đức Tiến

tiennd@soict.hust.edu.vn

+84-91-313-7399

Mục tiêu học phần

- Nắm bắt được kiến thức cơ bản và ứng dụng của các IC lập trình mô tả phần cứng được.
- Nắm bắt được các nguyên lý lập trình mô tả phần cứng và các cấu trúc lập trình cơ bản, minh họa bằng ngôn ngữ VHDL..
- Có kiến thức và kỹ năng thiết kế mạch số
- Hiểu biết về nguyên lý hoạt động phần cứng của bộ vi điều khiển và có khả năng thiết kế bộ vi điều khiển.

Tuyển dụng

● <https://www.facebook.com/dolphin.jobs/>



INTERNSHIP PROGRAM
"Find a Way to be Silicon Expert"

Dolphin Technology

Trong năm 2017, chúng tôi mở ra cơ hội phát triển nghề nghiệp cho các bạn sinh viên với Chương trình "Internship Program" vào các vị trí như sau:

Thiết kế số

Mô tả: Được đào tạo về các ngôn ngữ lập trình phần cứng: Verilog, SystemVerilog
Được tham gia vào quá trình phát triển Semiconductor IP
Làm việc với các công cụ CAD: Magma, Talus Design, Vortex, Power, Quartz; Synopsys Design Compiler/IC Compiler

Kiểm chứng Thiết kế

Mô tả: Được đào tạo về Kiểm tra hệ thống sử dụng SystemVerilog, UVM/OVM, VMM
Tích hợp IP và kiểm tra sử dụng giao tiếp chuẩn công nghiệp: AMBA, IBM CoreConnect, OCP, Wishbone
Thiết kế và kiểm tra Bus Functional Models
Kiểm tra chức năng Semiconductor IP
Làm việc trên các công cụ kiểm tra: Mentor Graphic Modelsim/Questasim, Synopsys

Hồ sơ đăng ký:
Thư ngỏ, sơ yếu lí lịch bằng tiếng Anh, nêu rõ kinh nghiệm nghiên cứu, làm việc (nếu có)
Chứng nhận kết quả học tập đến thời điểm hiện tại
Chứng chỉ, bằng cấp khác (nếu có)

Nhận hồ sơ từ ngày 06/04/2017 đến 20/04/2017 vào địa chỉ email jobs@dolphin-vc.com

Quy định của Chương trình
Những bạn tham gia chương trình thực tập tại Dolphin sẽ được đào tạo và làm việc tại môi trường của Dolphin trong khoảng thời gian hợp lý được thỏa thuận giữa Công ty và bạn nhằm đảm bảo việc học tập của bạn tại trường. Sinh viên sẽ được nhận phụ cấp trong quá trình thực tập bao gồm: phí đi lại, tiền gửi xe tại nhà, ăn trưa hoặc tối, ... Khi ra trường sẽ làm việc chính thức tại Công ty. Ngoài ra Sinh viên thực tập tại Công ty tốt nghiệp loại giỏi sẽ nhận được phần thưởng trị giá gấp đôi phụ cấp thực tập/tháng

Thể lệ của Chương trình
Thí sinh sẽ tham gia Cuộc thi Thực tập tại Dolphin để được tham gia chương trình thực tập.

Đối tượng tham gia:
- Sinh viên năm 4, 5 các ngành Điện tử, Máy tính, Công nghệ thông tin
- CPA ≥ 2,8, hoặc có khả năng lập trình thành thạo

Cuộc thi bao gồm 2 vòng như sau:
Vòng 1:
Thí sinh đáp ứng được yêu cầu về đối tượng tham gia Chương trình sẽ được xem xét hồ sơ và gọi vào vòng thi viết.
Vòng 2:
Những thí sinh vượt qua vòng thi viết sẽ được gọi đến phỏng vấn.

Liên hệ:
Nguyễn Nhistinguynh
Phòng Nhân sự, Dolphin Technology Vietnam Center
Địa chỉ: Tầng 2, Tòa nhà Lilama, 124 Minh Khai, Hai Bà Trưng, Hà Nội
Điện thoại: +84-4-3624-9784
Fax: +84-4-3624-9840



Dolphin Technology Vietnam Center

TUYỂN DỤNG
Tháng 4

10
Kỹ sư layout

SINH VIÊN MỚI TỐT NGHIỆP

- Có đam mê làm việc trong lĩnh vực vi mạch
- Có kiến thức cơ bản về logic gates, flops
- Kinh nghiệm làm việc trên môi trường Linux là một lợi thế
- Đọc hiểu tài liệu tiếng Anh, nói tốt, viết tốt là một lợi thế
- Làm việc chăm chỉ và kỹ năng làm việc nhóm
- Cam kết, năng động, sáng tạo

ỨNG VIÊN CÓ KINH NGHIỆM

- Đã được đào tạo về thiết kế mạch, thiết kế layout cho vi mạch VLSI
- Có kinh nghiệm làm việc với các công cụ CAD (Cadence Virtuoso/Tanner's L-Edit/SPICE)
- Thông thạo với các công cụ kiểm tra vật lý (Calibre DRC/LVS)
- Thông thạo với các công logic và chức năng cơ bản.
- Có kinh nghiệm về Library Characterization, Scripting.
- Kiến thức cơ bản về làm việc trên hệ điều hành Linux

Liên hệ

Hồ sơ bao gồm:

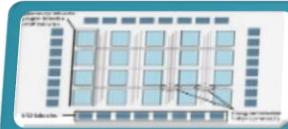
- CV có ảnh bằng tiếng Anh, nêu rõ kinh nghiệm, các dự án nghiên cứu, làm việc (nếu có)
- Bảng điểm, chứng nhận kết quả học tập đến thời điểm hiện tại. Có thể download từ trang web của nhà trường, bao gồm xác nhận có thể nộp sau để đổi chiếu theo yêu cầu của công ty.
- Chứng chỉ, bằng cấp (nếu có)
- Tiêu đề email viết theo mẫu: Vị trí ứng tuyển_Họ và tên
- Chỉ nhận hồ sơ qua email. Hạn nộp hồ sơ đến hết ngày 30/4/2017

Nguyễn Nhistinguynh - HR Department - Dolphin Technology Vietnam Center
Địa chỉ: Tầng 2, Tòa nhà Lilama, 124 Minh Khai, Quận Hai Bà Trưng, Hà Nội
Email: jobs@dolphin-vc.com. Điện thoại: +84 43 624 9784 Fax: +84 43 624 9840

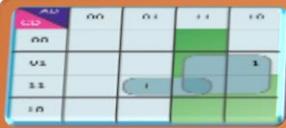
Tuyển dụng

Vị trí:	Kỹ sư phát triển FPGA (VIETTEL)
Số lượng:	05 (6/2017)
Thông tin tuyển dụng	<ul style="list-style-type: none">Nghiên cứu các thuật toán xử lý số.Nghiên cứu các giao thức truyền dữ liệu số.Xây dựng giải pháp thực hiện các thuật toán xử lý số và giao thức truyền dữ liệu trên nền tảng FPGA.Mô phỏng các thuật toán xử lý số sử dụng công cụ Matlab.Lập trình hiện thực hóa (coding / debug) các thuật toán xử lý số và các loại giao thức truyền dữ liệu dùng VHDL hoặc Verilog.Xây dựng các giải pháp test chức năng, test tích hợp đối với các hệ thống có sử dụng FPGA.
Yêu cầu công việc:	<ul style="list-style-type: none">Có kinh nghiệm lập trình FPGA hoặc DSP (fixed point).Ưu tiên ứng viên:<ul style="list-style-type: none">Có kiến thức chuyên sâu về hệ thống viễn thông (2G/3G/4G/CDMA).Có kinh nghiệm thiết kế phần cứng hệ thống vi xử lý: MCU/DSP/FPGA, RAM/ROM, ADC/DAC, Power/Reset, Power amplifier, Interface...Quyền lợi và chính sách:<ul style="list-style-type: none">Mức lương: Từ \$700 – \$1500Hưởng mọi chế độ của NLĐ theo quy định của Luật Lao động VN và quy định chung của Tập đoànLàm việc trong môi trường năng động, chuyên nghiệp, sáng tạoCó mục tiêu và định hướng phát triển nghề nghiệp rõ ràngĐược sáng tạo và thỏa mãn niềm đam mê nghiên cứu khoa học và công nghệYêu cầu chung:<ul style="list-style-type: none">Tốt nghiệp Đại học chính quy loại khá trở lên chuyên ngành CNTT, Điện tử Viễn thông.Có khả năng đọc hiểu tài liệu tiếng Anh. Ưu tiên có chứng chỉ TOEIC, TOEFL hoặc IELTS tương đương với điểm TOEIC – 550 điểm trở lên.Hình thức ứng tuyển: Gửi CV và văn bằng, chứng chỉ khác (nếu có) đến địa chỉ email:vttek.hr@viettel.com.vn.

Nội dung môn học



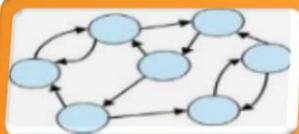
Chế tạo và sản xuất IC



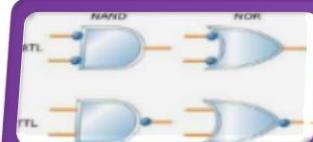
Điện tử số



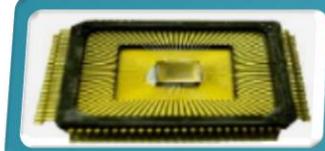
Ngôn ngữ mô tả phần cứng VHDL.



Thiết kế mạch số cơ bản



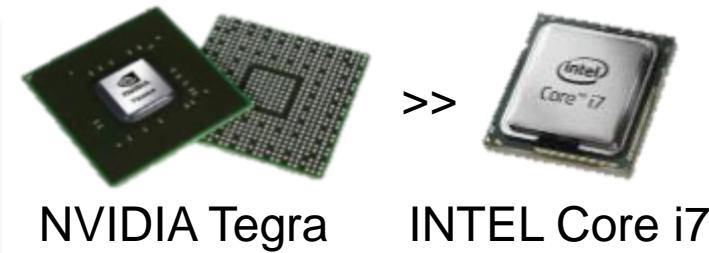
Thiết kế bộ xử lý



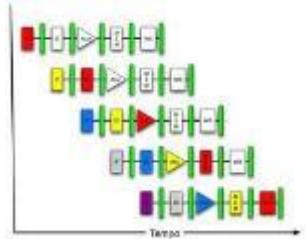
Phụ lục

Ứng dụng thực tiễn

- Cứng hóa các ứng dụng mềm, tạo sản phẩm ưu việt về tốc độ.
- Hiểu rõ nguyên tắc hoạt động của bộ xử lý, làm căn bản giúp việc lập trình phần mềm, lập trình hệ thống trở nên hợp lý, logic hơn.



Đồ họa(Tính toán vector)



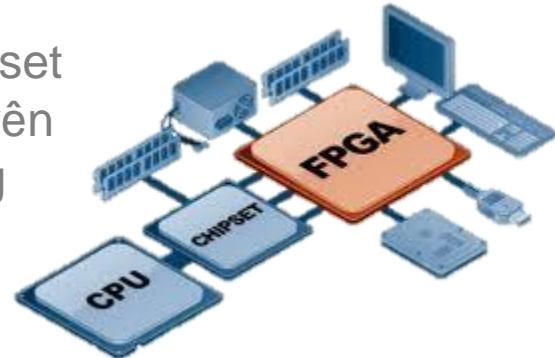
Ứng dụng thực tiễn

- Dễ dàng thiết kế các bo mạch xử lý số.
- Nhanh chóng tạo ra các bộ xử lý mới, mạch số chuyên dụng dựa trên mã nguồn mở và tùy biến theo nhu cầu.
- Khuynh hướng tạo ra các giải pháp tổng thể hardware/software (Oracle & Sun, Google& Motorola Mobility), và các khối xử lý tăng tốc.



QuadPort H/I
HDMI2SDI

Chipset
chuyên
dụng



Card tăng tốc chuyên dụng

Tài liệu tham khảo

- Nguyên lý mạch tích hợp. Tống Văn Ôn. NXB Lao động xã hội. 2007.
- Thiết kế mạch số với VHDL & Verilog. Tống Văn Ôn. NXB Lao động xã hội. 2007.
- Digital VLSI Systems Design. Dr. S. Ramachandran. Springer. 2007.
- Digital Integrated Circuits - A Design Perspective. Jan M. Rabaey, Anantha Chandrakasan, Borivoje Nikolic. Mc Graw Hill.



Tài liệu tham khảo

- Analysis and Design of Digital IC. David A.Hodges, Horace G.jackson, Resve A.Saleh. Mc Graw Hill.
- The Design Warrior's Guide to FPGA. Clive Max Maxfield. 2004.
- Circuit Design with VHDL. Volnei A. Pedroni. MIT Press. 2006.
- Fundamentals Of Digital Logic With VHDL Design 2nd Edition. Stephen Brown, Zvonko Vranesic. McGraw Hill. 2005.

Project tham khảo

- FPGA4U, kit, usb-powered, altera, niosII & linux,
http://fpga4u.epfl.ch/wiki/Main_Page
- SecretBlaze, bộ xử lý RISC 32bit dựa trên MicroBlaze, Xilinx, mã mở VHDL, 5 công đoạn, cache, ngắt.
<http://www.lirmm.fr/~barthe/index.php/page/SecretBlaze.html>
- Các project và module mã nguồn mở
<http://www.opencores.org/>
- Các thủ thuật trên VHDL
<http://vhdlguru.blogspot.com/>

Công cụ sử dụng trong môn học

- Altera Quartus: lập trình HDL và mô phỏng, và nạp kế lén chip FPGA của Altera
- Active-HDL Student Edition: lập trình HDL và mô phỏng trực quan. Miễn phí 30 ngày.
- Libero IDE: xem phân tích RTL, biên dịch HDL, nạp thiế kế lén chip FPGA của Actel

Download

- <ftp://dce.hust.edu.vn/tiennd/thietkeic>

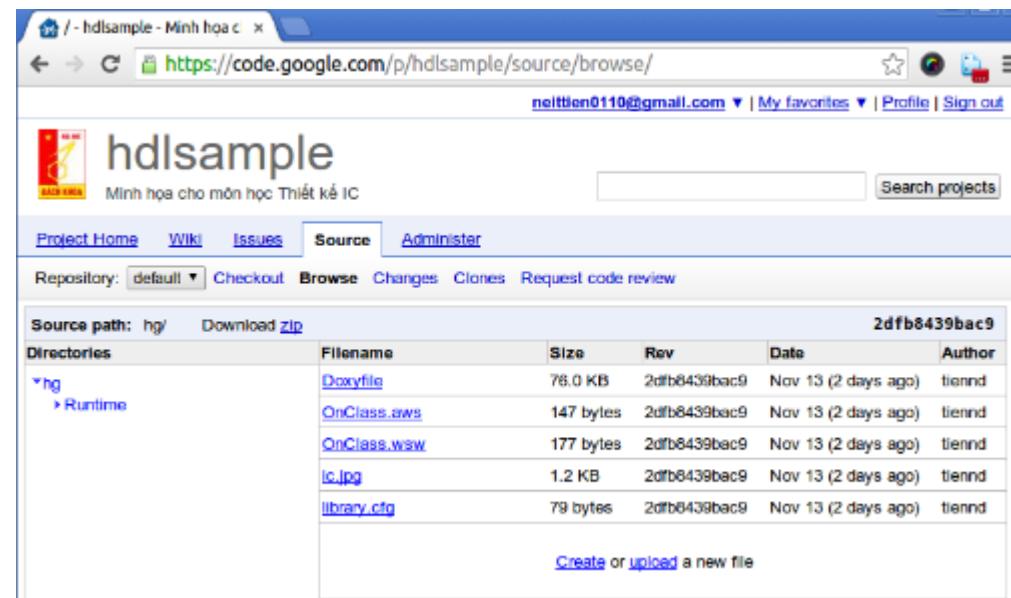


- <http://www.mediafire.com/#09qgoawx2y3id>



Mã nguồn trong slide

- Các ví dụ minh họa trong slide đặt tại Project HDL Sample
<https://github.com/neittien0110/hdlsample>
- Mở Project trên bằng Aldec Active-HDL
- Project trên được lưu trữ bằng Mercurial, hoặc có thể download file zip qua địa chỉ nói trên.

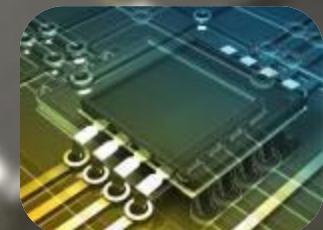




Phần I:

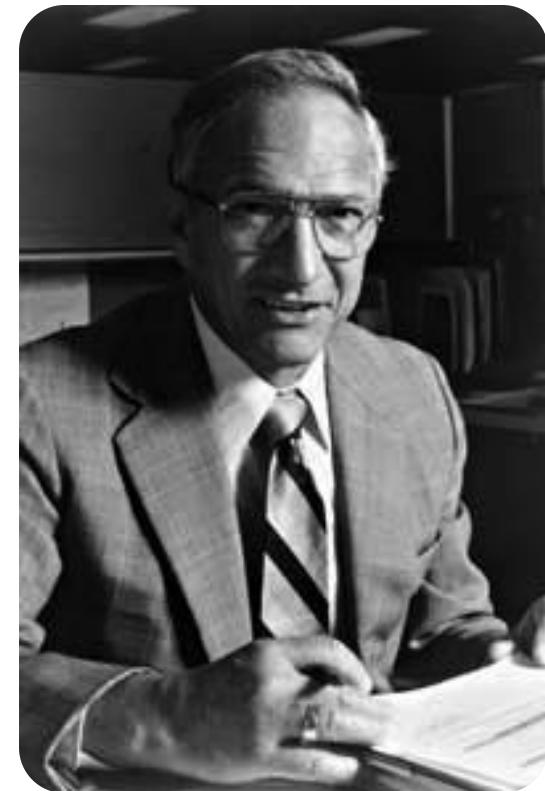
Chế tạo và sản xuất IC

- Tổng quan về IC
- FPGA, ASIC



Robert Noyce, 1927-1990

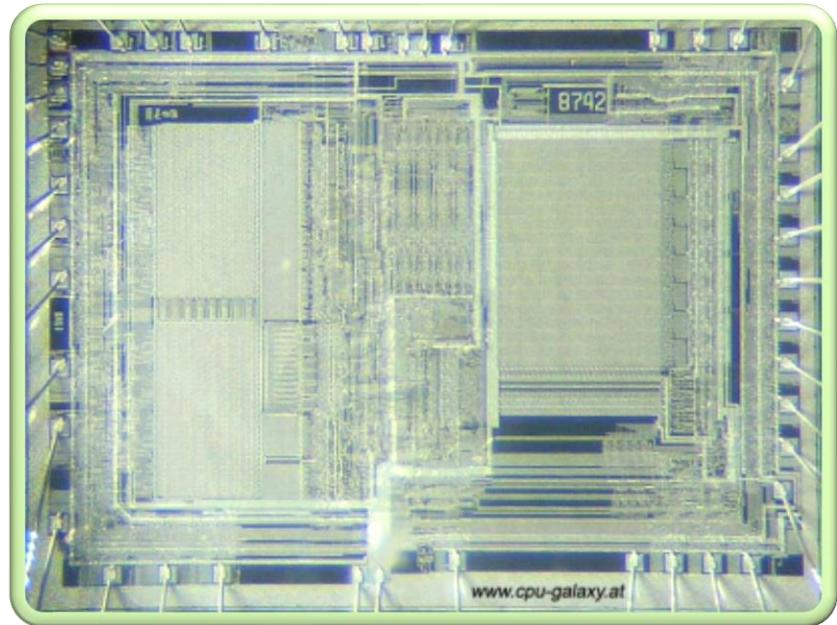
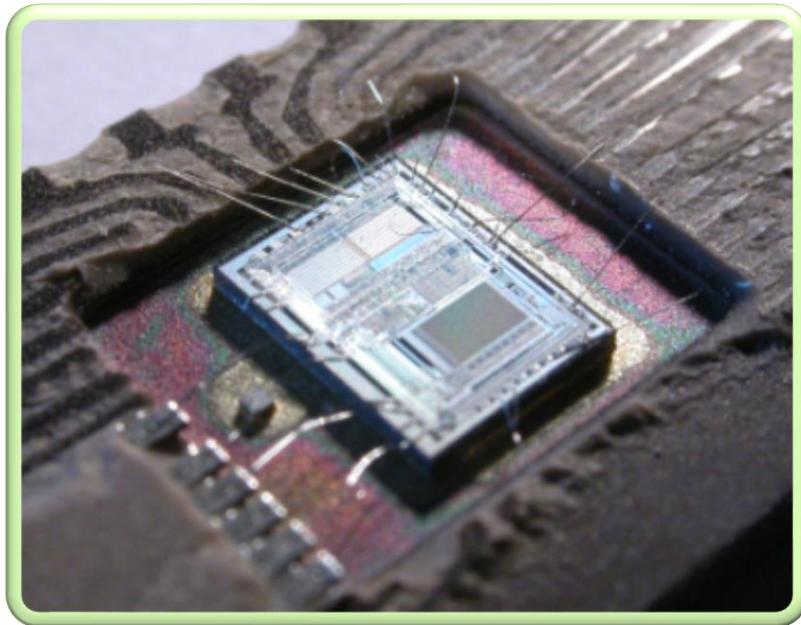
- Được mệnh danh là “Thị trưởng của thung lũng Sillicon”
- Đồng sáng lập Fairchild Semiconductor năm 1957
- Đồng sáng lập Intel 1968
- Đồng phát minh IC



Tham khảo: [hình ảnh IC đầu tiên dạng 2D](#), [IC đầu tiên dạng 3D](#)

IC là gì

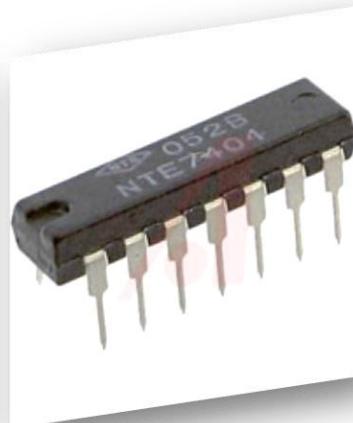
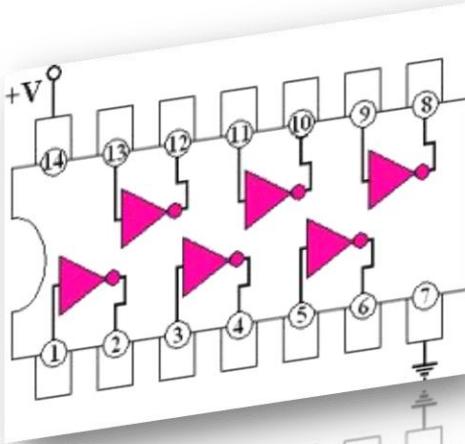
- **IC, Integrated Circuit**, là một mạch điện tử mà tất cả các thành phần đều được đặt trên một đế bán dẫn, không thể tách rời nhau được.



Vi điều khiển Intel 8742: CPU 12MHz, RAM 128B, EPROM 2KB.

Chức năng của IC

- Thực hiện một / vài chức năng điện tử cụ thể.



- Chức năng của IC có thể
 - Cố định
 - Lập trình được
 - Cấu hình được



Thiết kế IC như thế nào?

Viết phần mềm

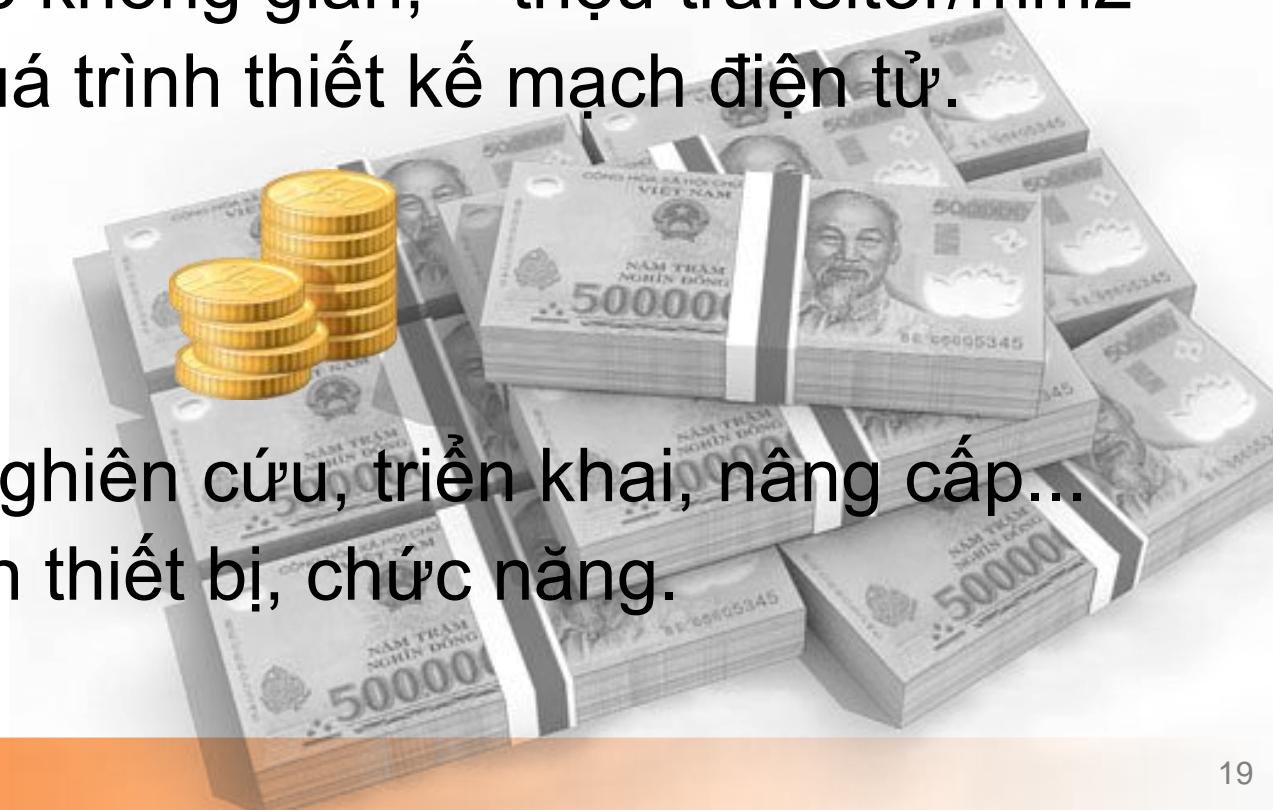
- *Phần mềm mới = Thư viện, Framework + mã nguồn tự viết*
- *Phần mềm thực hiện bởi bộ xử lý CPU/GPU*

Thiết kế IC

- *IC mới = IC chuẩn + custom IC (glue logic)*
- *Thiết kế sẽ thực hiện bằng cách*
 - *Tạo mạch in và hàn gắn các linh kiện cần thiết*
 - *hoặc chạy trên chip FPGA/CPLD*

Lợi ích của các IC - Giá thành

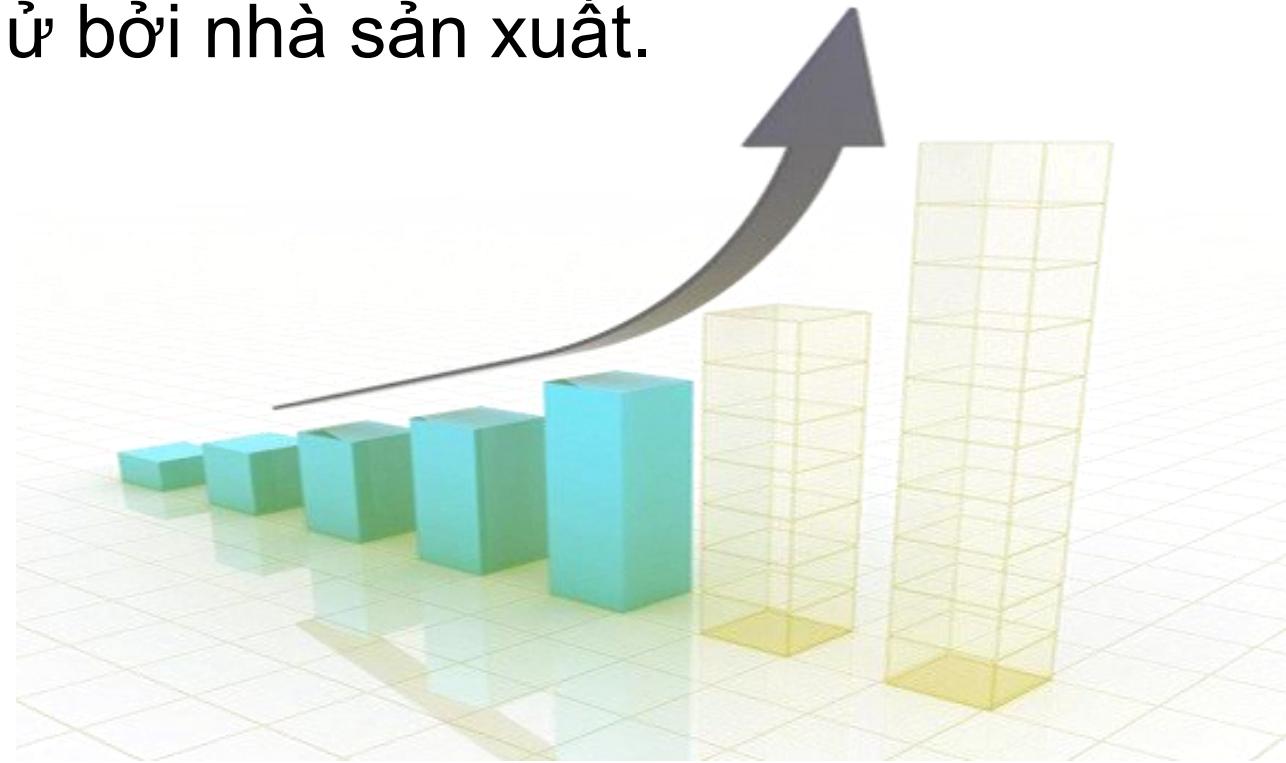
- Tất cả các thành phần của IC được sản xuất đồng thời và hàng loạt, chứ không phải từng IC đơn lẻ.
- Ít tốn linh kiện.
- Được tối ưu về không gian, ~ triệu transistor/mm²
- Module hóa quá trình thiết kế mạch điện tử.



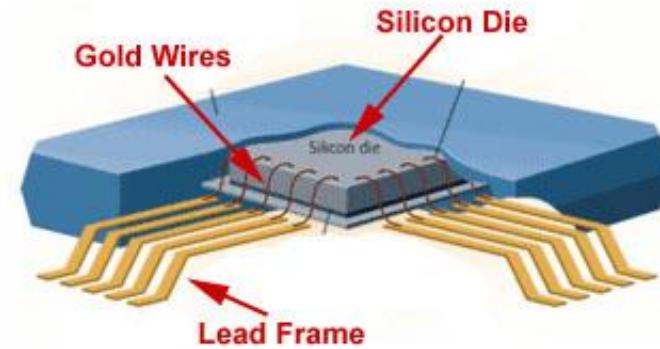
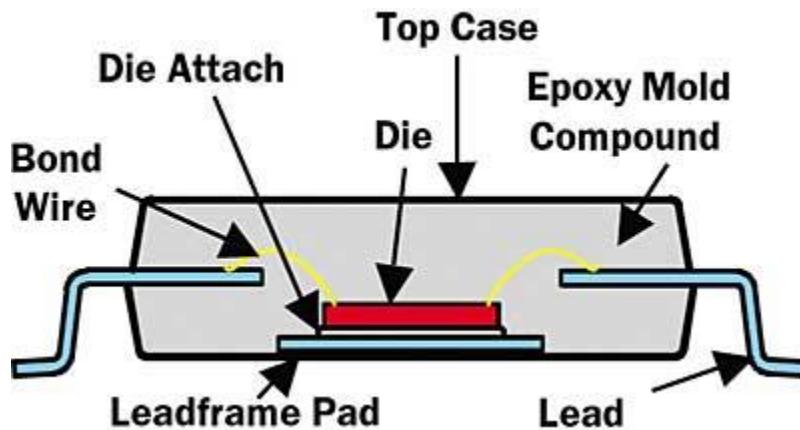
- Giảm chi phí nghiên cứu, triển khai, nâng cấp...
- Giảm giá thành thiết bị, chức năng.

Lợi ích của các IC - Hiệu năng

- Tiêu thụ ít năng lượng.
- Được tối ưu hóa về tốc độ...
- Đồng bộ và tin cậy.
- Được kiểm thử bởi nhà sản xuất.
- Tuổi thọ cao.



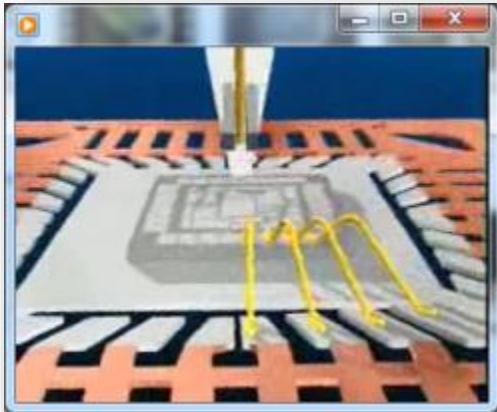
Cấu tạo chung của IC



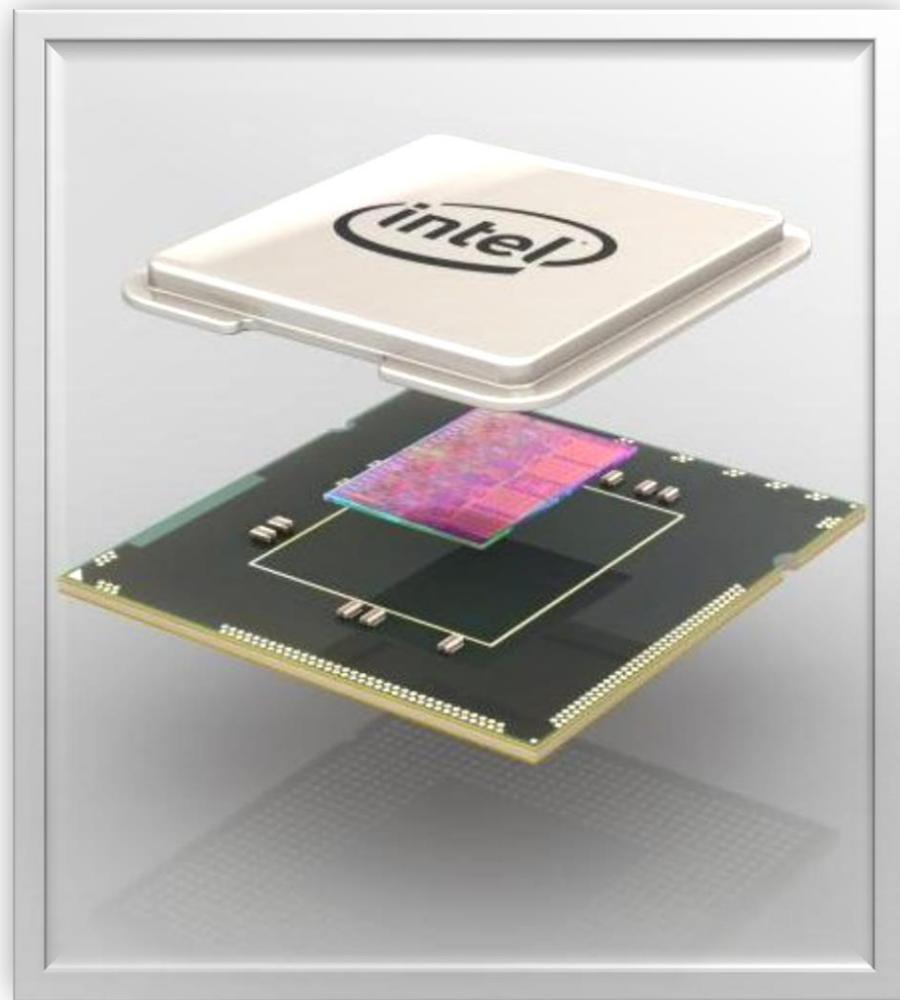
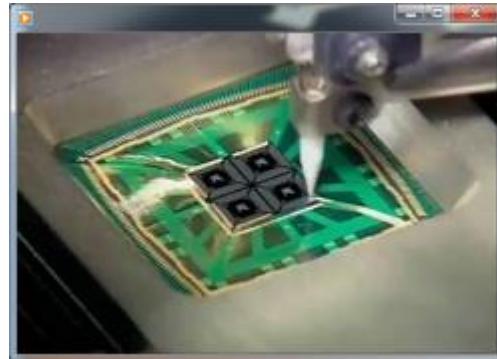
- Die: khuôn, chip silicon.
- Leadframe: khung dẫn, chứa các chân nối ra ngoài
→ rắp ráp lên board dễ hơn
- Wire: nối các chân IO trên die ra các chân tương ứng trên leadframe.
- Vỏ: phủ kín, đóng gói leadframe & die bằng ceramic, plastic... → bảo vệ, tản nhiệt.

Cấu tạo chung của IC

- Liên kết khung dẫn



- Liên kết khung dẫn (2)



Hình dạng vỏ

- Vỏ - package - có nhiều hình dạng khác nhau.

Dual
Inline
Package



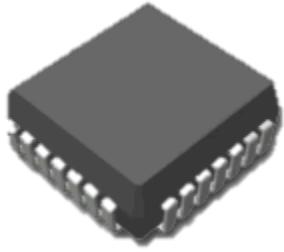
Zigzag
Inline
Package



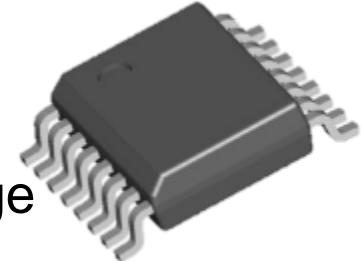
Ball
Grid
Array



Plastic
Leader
Chip
Carrier



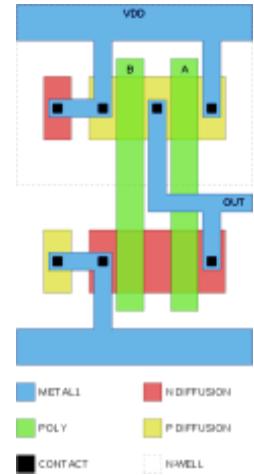
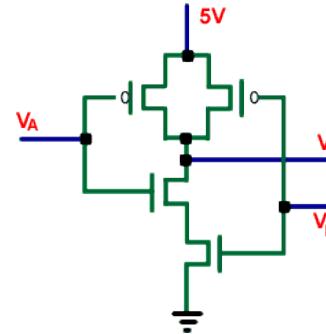
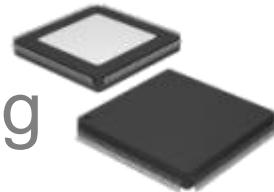
Small
Outline
Package



- Tham khảo một số dạng vỏ

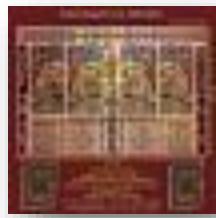
Khuôn – die – chip silicon

- Độ dài mỗi cạnh chừng vài mm.
- Kích thước IC được đo bằng số transistor hoặc số cổng logic, (cổng là đơn vị đo tương ứng với cổng NAND 2 đầu vào).
 - Xilinx Spartan II XC2S200, 540k đ, 200k cổng.
 - Intel Core i7, 6.6m đ, >1 tỷ cổng.
- Cấu thành bởi các transistor CMOS. Một cổng NAND có 4 transistor.



Khuôn – die – chip silicon

- Quy trình xử lý N nm sẽ cho transistor nhỏ nhất có chiều dài N nm. Ví dụ, quy trình 28 nm.
- Kích thước đặc trưng nhỏ nhất $\lambda \approx 1/2$ chiều dài transistor nhỏ nhất.
- Các bộ phận chức năng cấu thành IC được phân vùng rõ ràng. (FGPA hỗ trợ).



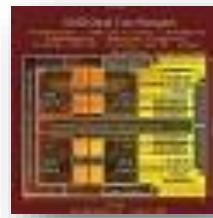
Intel Quad Core



Intel Core i7



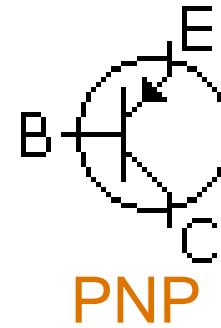
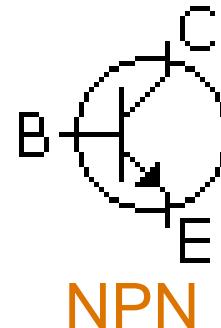
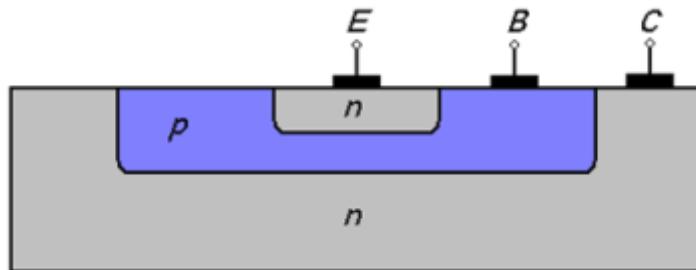
Intel Pentium 5



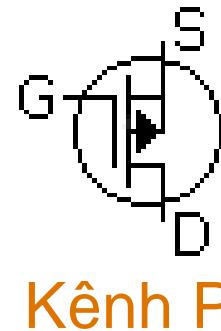
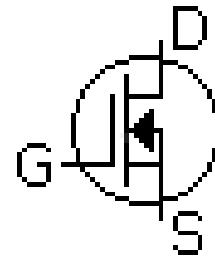
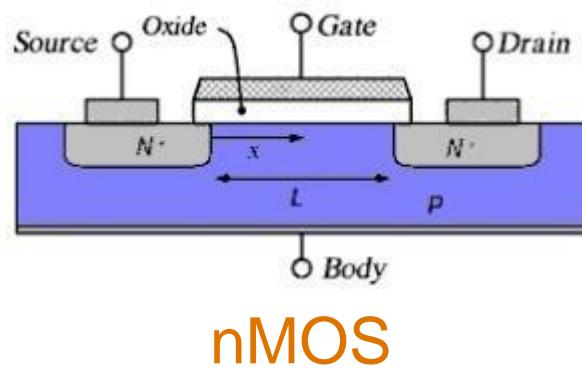
AMD Quad Core

Công nghệ bán dẫn

- Sử dụng các transistor lưỡng cực, Bipolar (1940s)



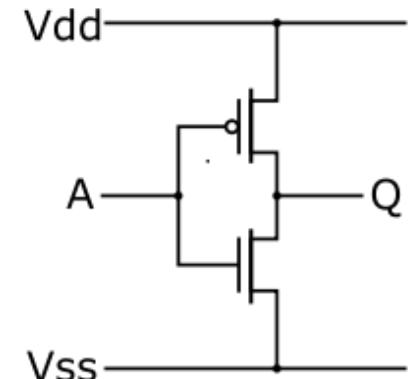
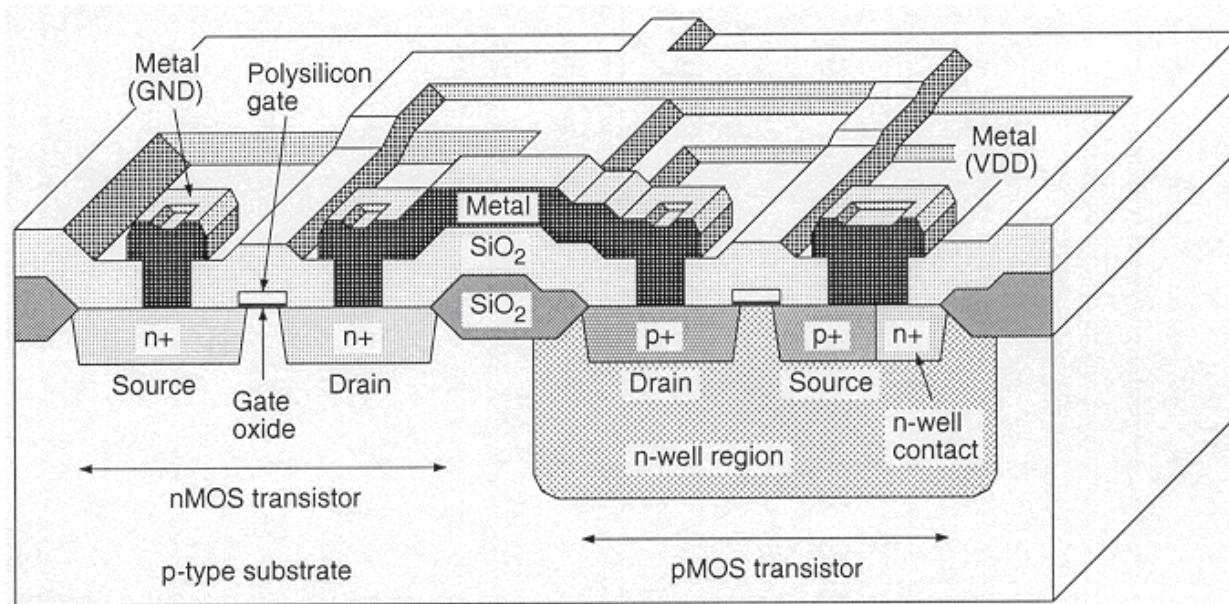
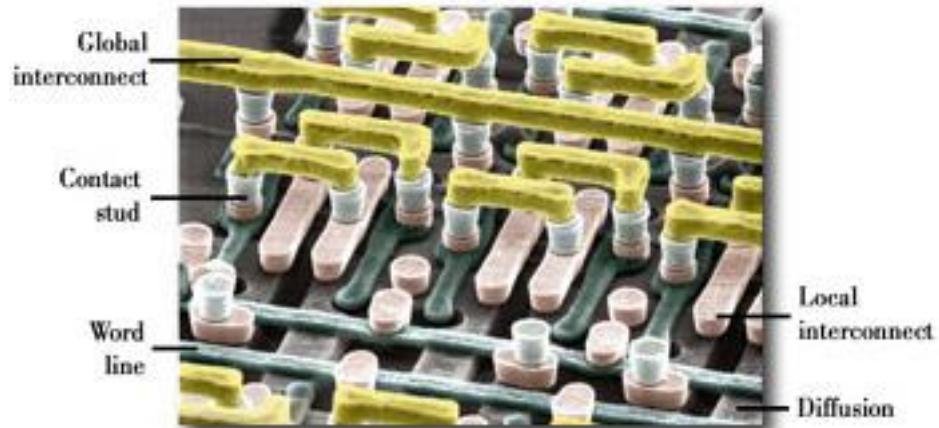
- Sử dụng các transistor có cực cổng kim loại, Metal Oxide Semiconductor (1960s)



Công nghệ bán dẫn

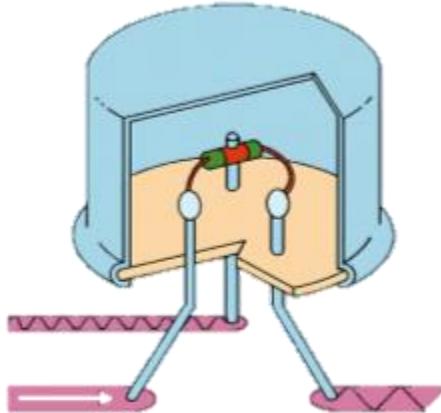
- CMOS,
Complementary MOS

Các lớp
liên kết nối

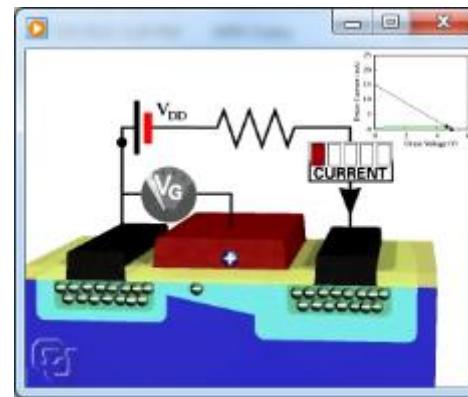


Một số hình ảnh về transistor

- Transistor lưỡng cực



- Transistor MOSFET



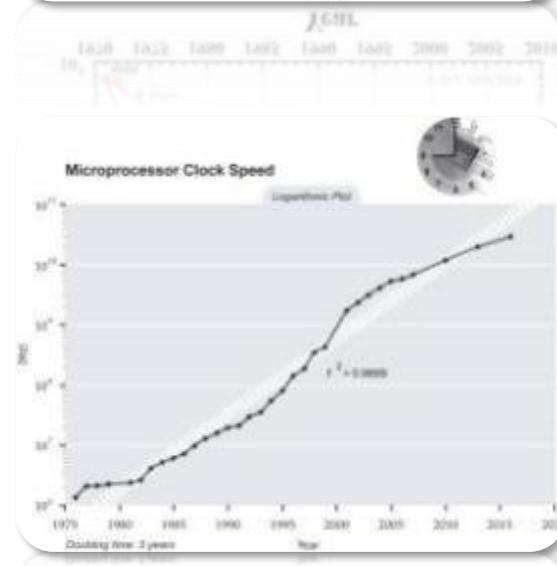
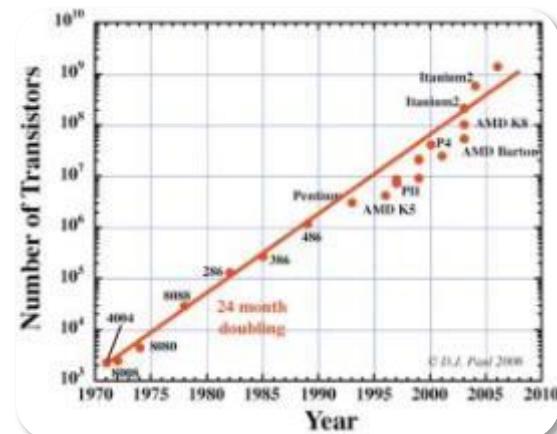
Mật độ tích hợp và hiệu năng

● Mật độ tích hợp: tăng với tốc độ nhanh chóng

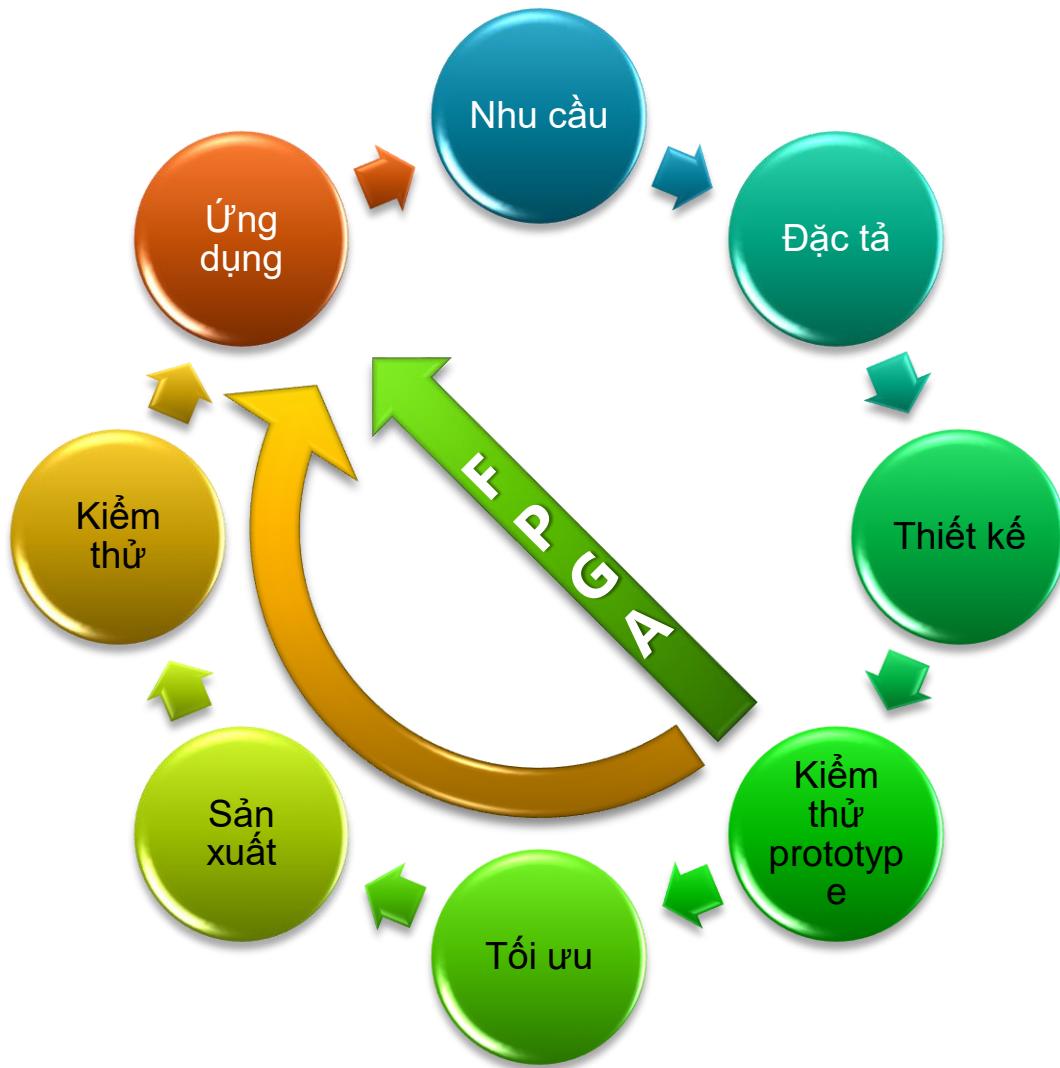
- Cỡ nhỏ (**S**mall **S**cale **I**ntegration)
- Cỡ trung bình (**M**edium **S**cale **I**ntegration)
- Cỡ lớn (**L**arge **S**cale **I**ntegration)
- Cỡ rất lớn (**V**ery **L**arge **S**cale **I**ntegration)

● Hiệu năng:

- Định luật Moore: Tần số xung clock nhân đôi sau mỗi 18 tháng



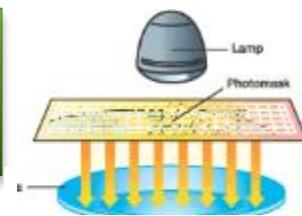
Vòng đời sản phẩm



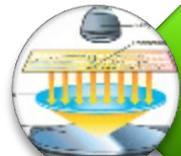
- Nhu cầu từ thực tế, từ các dự đoán trong tương lai. Ví dụ, cần RAM gấp đôi.
- Đặc tả các nhu cầu trên phương diện kỹ thuật. Ví dụ, cần tăng độ rộng bus địa chỉ.
- Thiết kế: mức đỉnh – logic, thiết kế mức RTL, Register Transfer Level
- Tối ưu: về tốc độ, không gian, năng lượng, về công nghệ sản xuất ns.
- Ứng dụng: **Release To Manufacturer**

Quá trình sản xuất IC

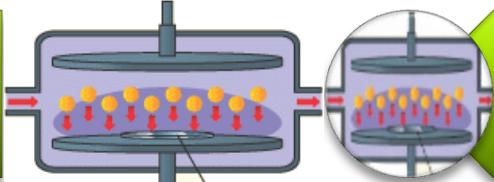
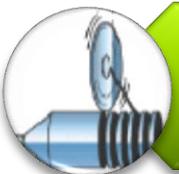
Tạo thỏi silicon



In litô

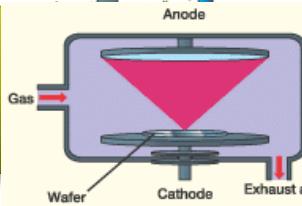
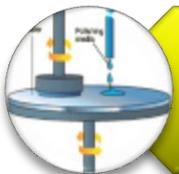


Cắt thành wafer



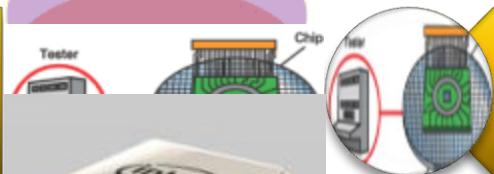
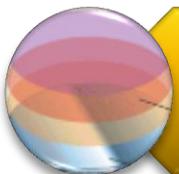
Rửa bề mặt

Mài bóng wafer



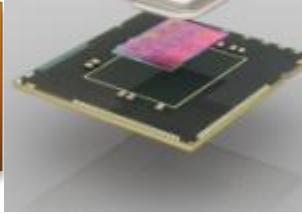
Phủ kim loại

Oxi hóa wafer



Kiểm thử

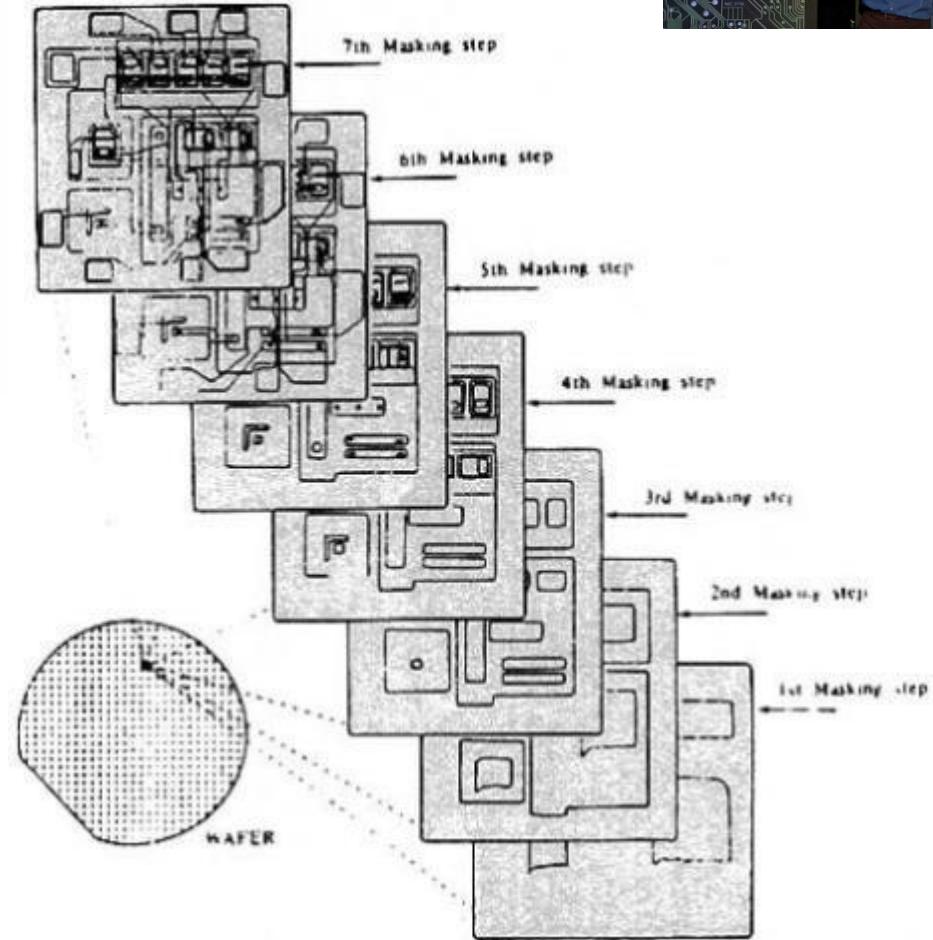
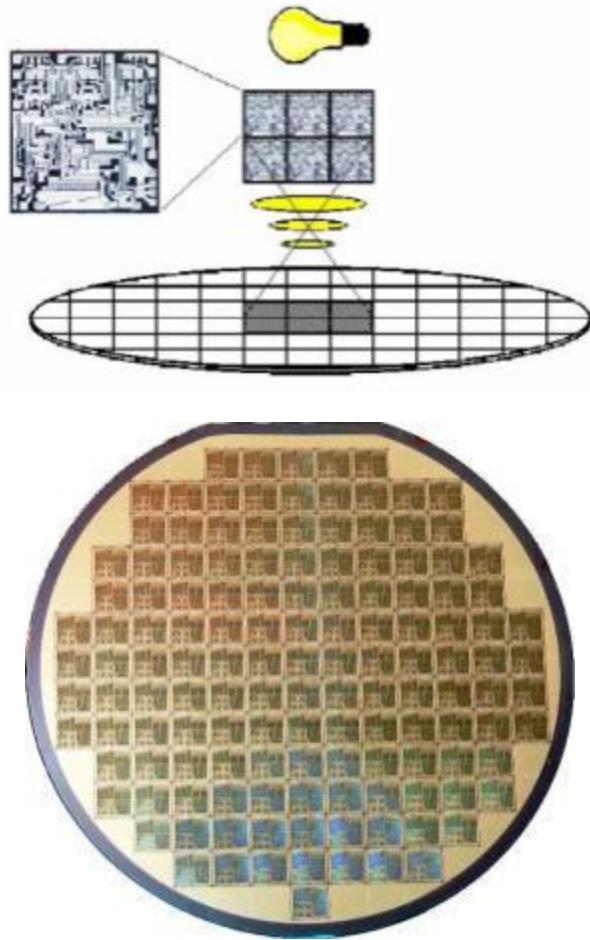
Phủ cản quang



Đóng gói

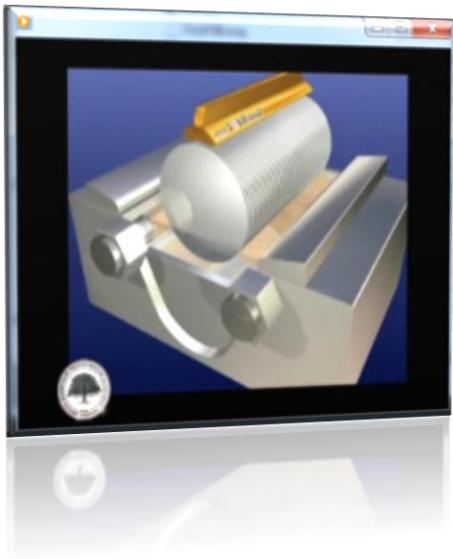


In các lớp vi mạch trên wafer



Quá trình sản xuất IC

- Tạo cắt mài wafer



- Nhà máy



- Cắt và siliicon

- Bài dịch tham khảo



FPGA, ASIC

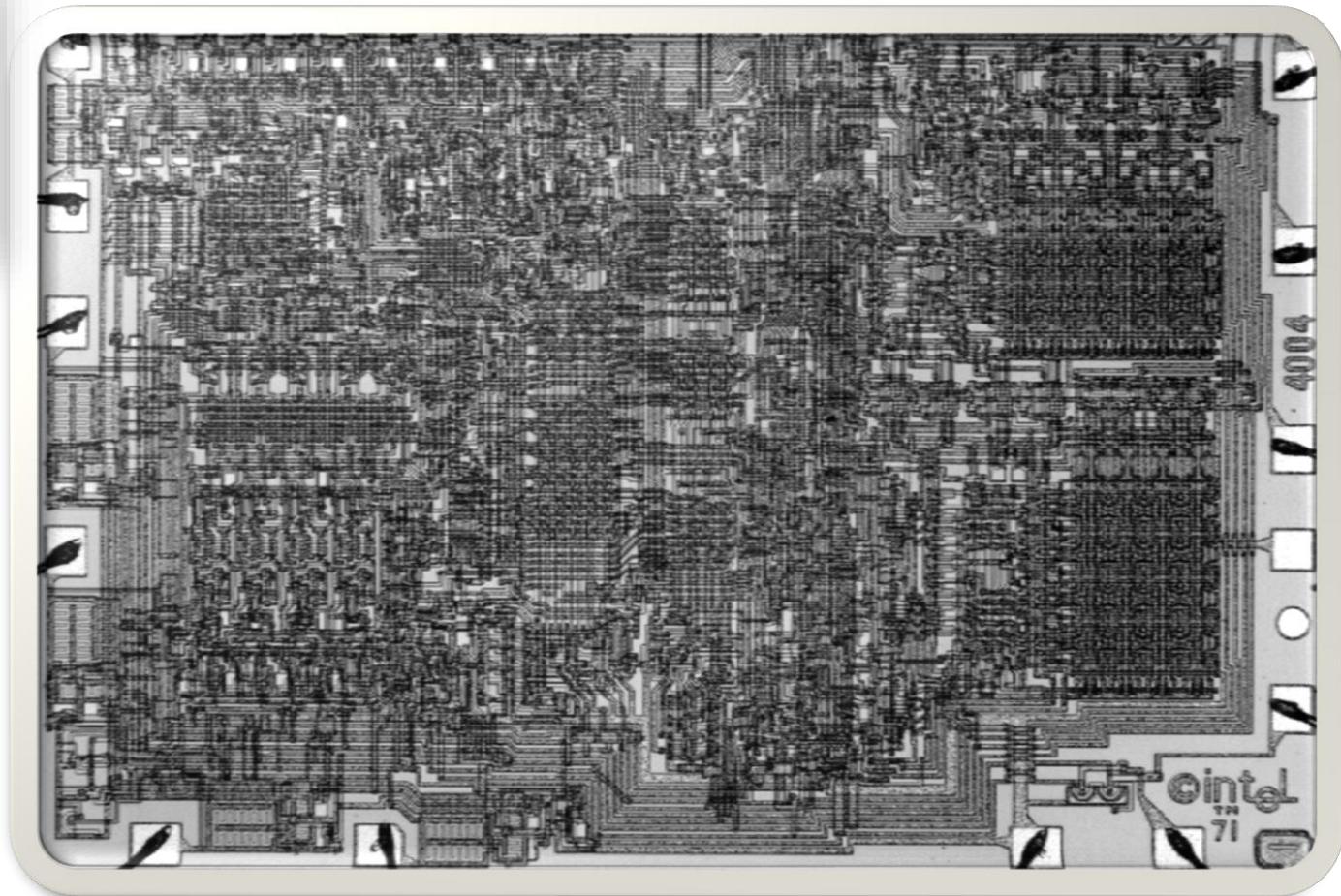
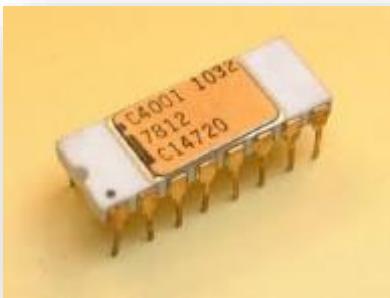
*Field Programmable Gate Array
Application Specific Integrated Circuit*

- Tổng quan
- Kiến trúc
- Qui trình thiết kế FPGA, ASIC
- Giới thiệu công cụ thiết kế và triển khai

Lịch sử

1/5

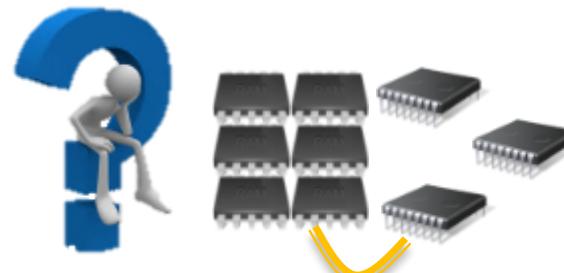
15/11/1971, Intel 4004, khởi đầu kỷ nguyên số



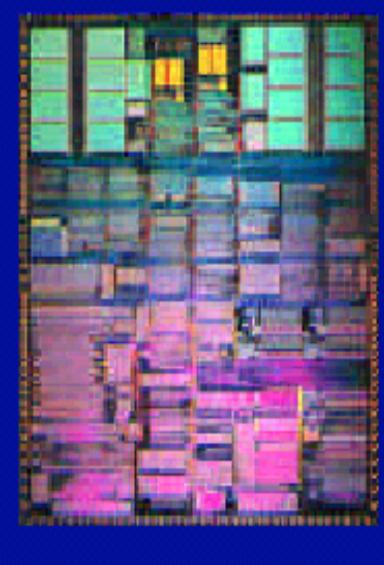
Lịch sử

2/5

- 1980s, VLSI xuất hiện → thiết kế IC theo nhu cầu.



- Độ phức tạp tăng 40% mỗi năm.
- Nhu cầu thiết kế tăng 15% mỗi năm



Mức độ tích hợp của PCB trên mỗi die

Lịch sử

3/5

• Silicon, 2010

Die Area: 2.5x2.5 cm
Voltage: 0.6 V
Technology: 0.07 μm

	Mật độ (Gbits/cm ²)	Thời gian truy xuất (ns)
DRAM	8.5	10
DRAM (logic)	2.5	10
SRAM (cache)	0.3	1.5

	Mật độ (Mgate/cm ²)	Năng lượng (W/cm ²)	Xung đồng hồ (GHz)
Custom	25	54	3
Std. Cell	10	27	1.5
Gate	5	18	1
Single-Mask GA	2.5	12.5	0.7
FPGA	0.3	4.5	0.25

Lịch sử

4/5

- Cuộc cách mạng của các bộ vi xử lý.
 - Điện thoại di động, internet, y học, etc.
- Ngành công nghiệp bán dẫn tăng trưởng từ 21 tỷ USD năm 1985 tới 300 tỷ USD năm 2011



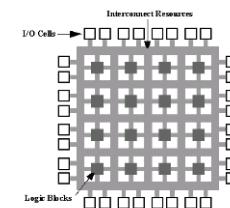
Lịch sử

5/5

- Nhu cầu thiết kế IC ngày càng tăng, chia thành 2 dạng:
 - IC được thiết kế theo nhu cầu chung, số lượng lớn
 - IC được thiết kế theo nhu cầu riêng lẻ (*may đo*) → gọi là **ASIC**, Application Specific IC

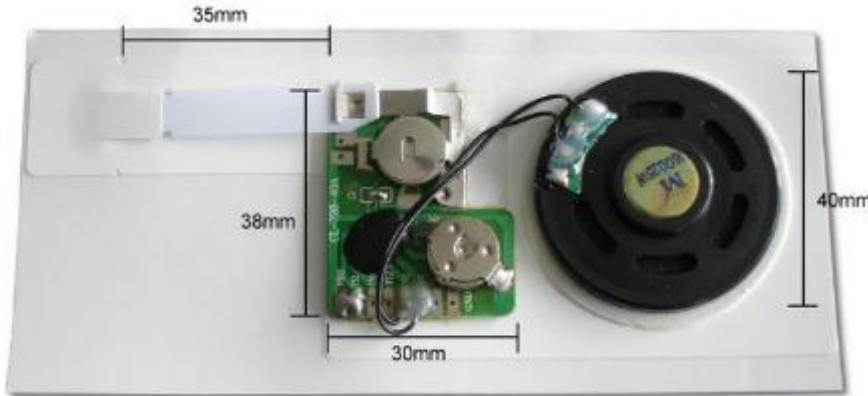
Phân biệt ASIC và IC

- ASIC là IC chuyên dụng.
- Nhưng không phải IC chuyên dụng nào cũng là ASIC.
 - IC chuẩn: ROM, RAM, DRAM.. là IC chuyên dụng nhưng không phải ASIC
 - ASIC: chip cho đồ chơi biết nói, chip cho 1 vệ tinh, chip cho bộ xử lý dưới dạng cell cùng với mạch logic.

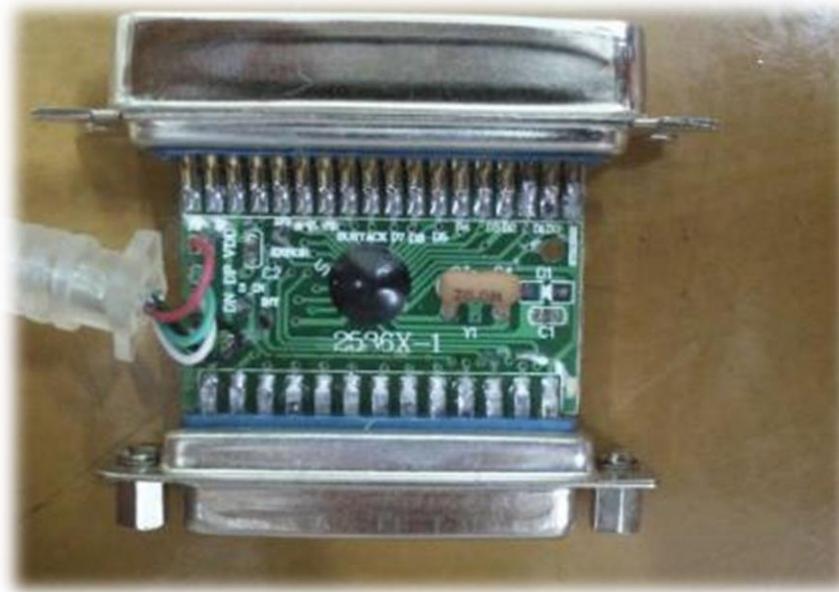
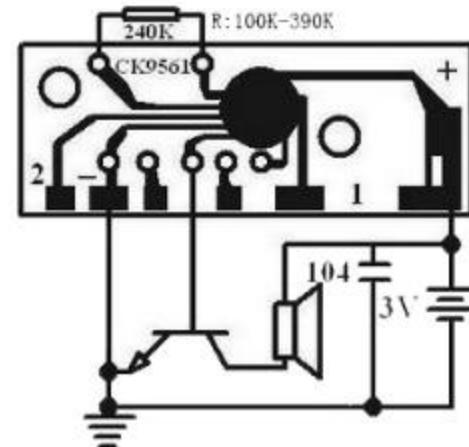


- Qui tắc xác định ASIC: “ASIC là loại IC chuyên dụng không có datasheet tra cứu kèm theo”.

Ví dụ về ASIC



Bưu thiếp nhạc



USB to LPT



Casio FX-300ES



One Wheel

Lợi ích khi sử dụng ASIC

- ASIC đem lại cơ hội sản xuất với số lượng lớn; các bộ phận được tiêu chuẩn hóa để nhanh chóng trở thành sản phẩm thương mại.
 - Giá thành giảm theo số lượng.
 - None Reducing Cost.
 - Quy trình Cost Down trong các nhà máy.
- Hiệu quả kinh tế trong thiết kế
 - Thực hiện prototype nhanh với số lượng thấp.
 - Thiết kế theo nhu cầu, chuyên sâu, số lượng lớn.

Các nguyên tắc thiết kế ASIC

1/6

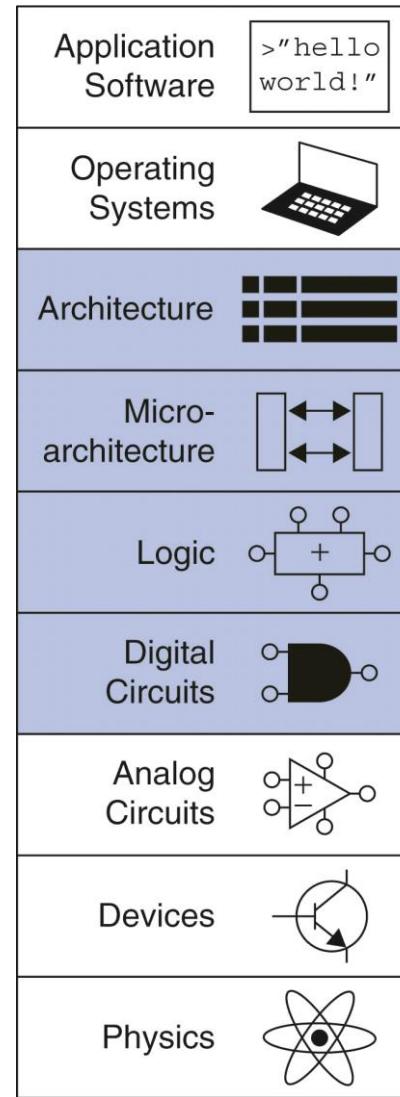
- Trùu tượng hóa
- Quy chuẩn
- Chiến lược thiết kế (*3 chữ Y*)
 - Hierarchy - phân cấp;
 - Modularity – phân chia và hợp nhất;
 - Regularity – nhất quán;
- Sử dụng công cụ CAD để đạt được hiệu quả trong chiến lược thiết kế:
 - Thiết kế mức hệ thống: VHDL
 - Thiết kế mức vật lý, từ VHDL tới silicon, timing closure (Monterey, Magma, Synopsys, Cadence, Avant!)

Nguyên tắc trừu tượng hóa

2/6

- Làm ẩn đi các yếu tố chi tiết không quan trọng

phạm
vi
học
phàn



Nguyên tắc Quy chuẩn

3/6

- Áp dụng các tiêu chuẩn chung trong thiết kế
- Ví dụ: Số hóa các quá trình xử lý
 - Lượng tử hóa các mức điện áp, thay vì giá trị điện áp liên tục
 - Mạch số dễ thiết kế hơn so với mạch tương tự – cho phép xây dựng các hệ thống phức tạp
 - Các hệ thống số đang thay thế dần các hệ thống tương tự (analog) tiền nhiệm:
 - Ví dụ, camera số, truyền hình kỹ thuật số, CD, điện thoại di động, etc

Nguyên tắc về chiến lược

4/6

Hierarchy (Phân cấp)

- Hệ thống được chia thành các module con
- Module con lại được chia thành các module mức thấp hơn nữa

Modularity (Phân chia và hợp nhất)

- Module phải có chức năng rõ ràng
- Module phải có giao diện giao tiếp rõ ràng, hợp chuẩn để có thể ghép với nhau dễ dàng

Regularity (Nhất quán)

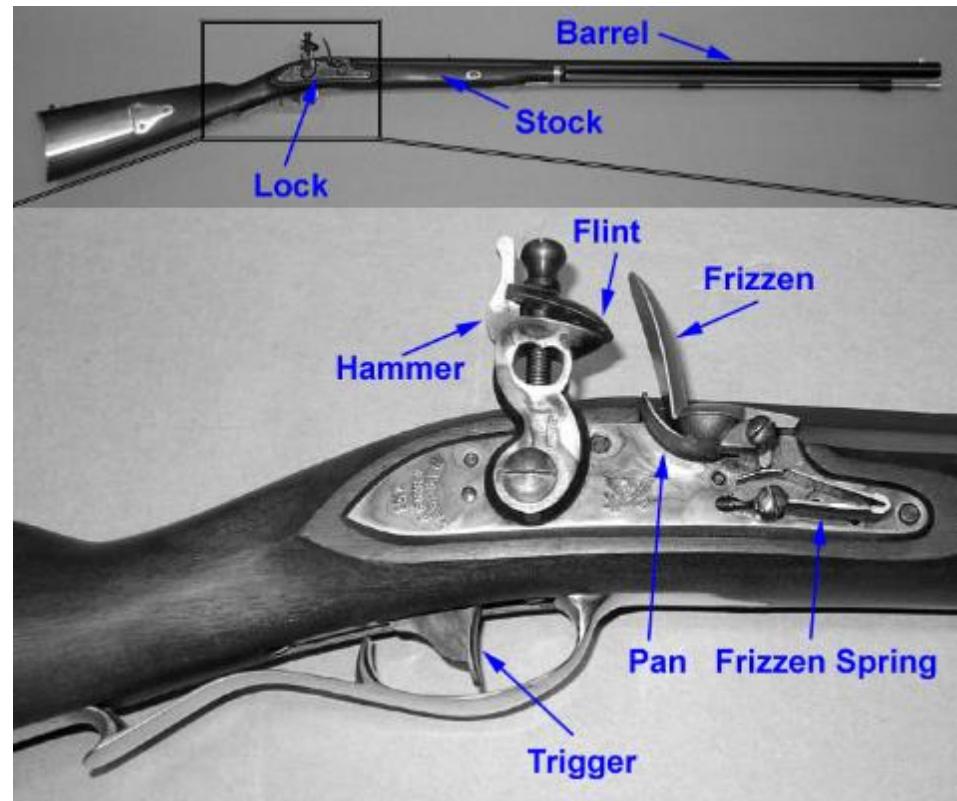
- Có sự tương đồng, đồng nhất, để sao cho các module có thể tái sử dụng

Ví dụ: súng Flintlock

5/6

- Hierarchy (Phân cấp)

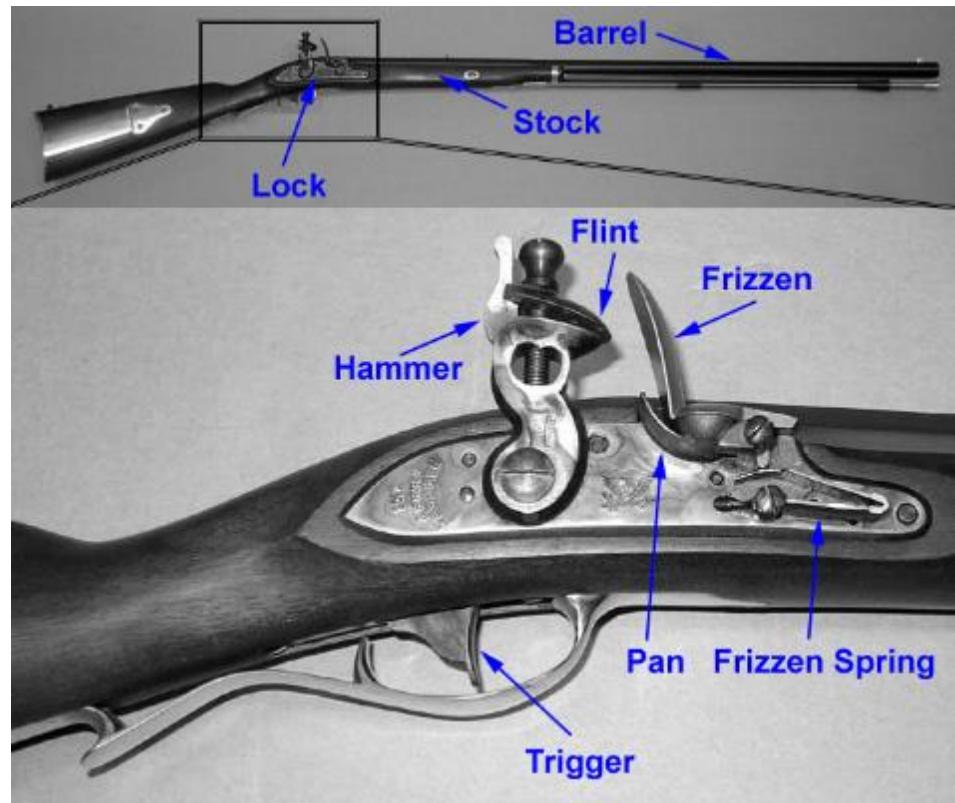
- **Ba bộ phận chính:** khóa, ổ đạn, và nòng
- **Các module để tạo nên khóa:** búa, kim hỏa, frizzen, etc.



Ví dụ: súng Flintlock

6/6

- Modularity (Phân chia và hợp nhất)
 - Chức năng của ổ đạn: chứa đạn, nối giữa nòng và khóa súng.
 - Giao tiếp của ổ đạn: độ dài và vị trí của các lỗ ghép nối
- Regularity (Nhất quán)
 - Các bộ phận đều có thể thay thế được



Cân bằng giữa các yếu tố



- Thiết kế là một quá trình liên tục cân đối các tham số đầu vào, để đạt được hiệu quả mong đợi.



Hiệu năng

- chức năng, thời gian, tốc độ, năng lượng



Kích thước die

- chi phí sản xuất



Thời gian thiết kế

- lập lịch và chi phí nghiên cứu

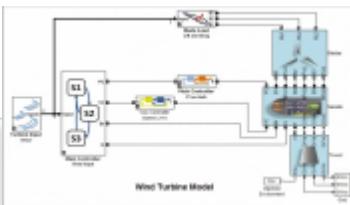


Test và Stability Test

- lập lịch, chi phí nguồn lực, chi phí sản xuất

Quy trình thiết kế chế tạo ASIC

Simulink



$c := a + b;$
 $\text{if } (c = 1) \text{ then } cf := 1;$

Ý tưởng thiết kế

MATLAB
MIDAS
Hypersignal

Đặc tả thiết kế

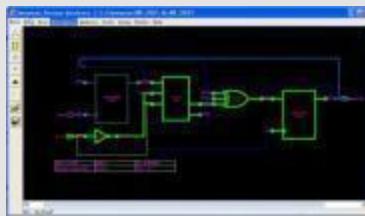
Verilog HDL
VHDL

Thiết kế hệ thống

HDL

Giả lập

Synopsys



Thiết kế mức cồng

Thư viện cell

Tổng hợp

Giả lập

Synopsys

Cadence



Layout

Kiểm tra Layout

Cadence CAD Tools

Novelus



Sản xuất

Kiểm thử chip

Sản phẩm

Test Bench Setup
Signal Generator
Digital Oscilloscope
Spectrum Analyzer
Etc.

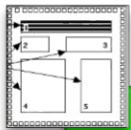
Phân loại ASIC

1/2



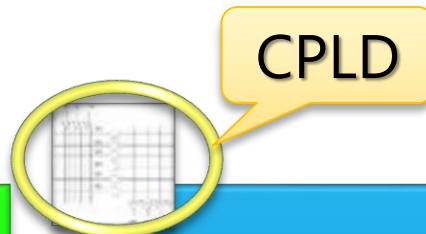
Full Custom IC

- Thiết kế rất phức tạp, hàng trăm man-year
- ~ 25M gate
- Phải sản xuất với số lượng cực lớn mới

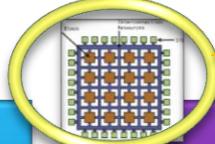


Cell Base IC

- Các cell là Flip-Flop, gate, hoặc bộ xử lý đã được định nghĩa trước, được tái sử dụng lại,
- Sản xuất với số lượng 100k sản phẩm/năm



CPLD



FPGA

Programmable Logic Device

- Dựa trên công nghệ ROM/PROM, cấu tạo chỉ gồm các lớp cổng AND, OR và một số Flip Flop.
- Sản phẩm khá đơn giản với khoảng 1K gate.
- Sản xuất đơn lẻ được

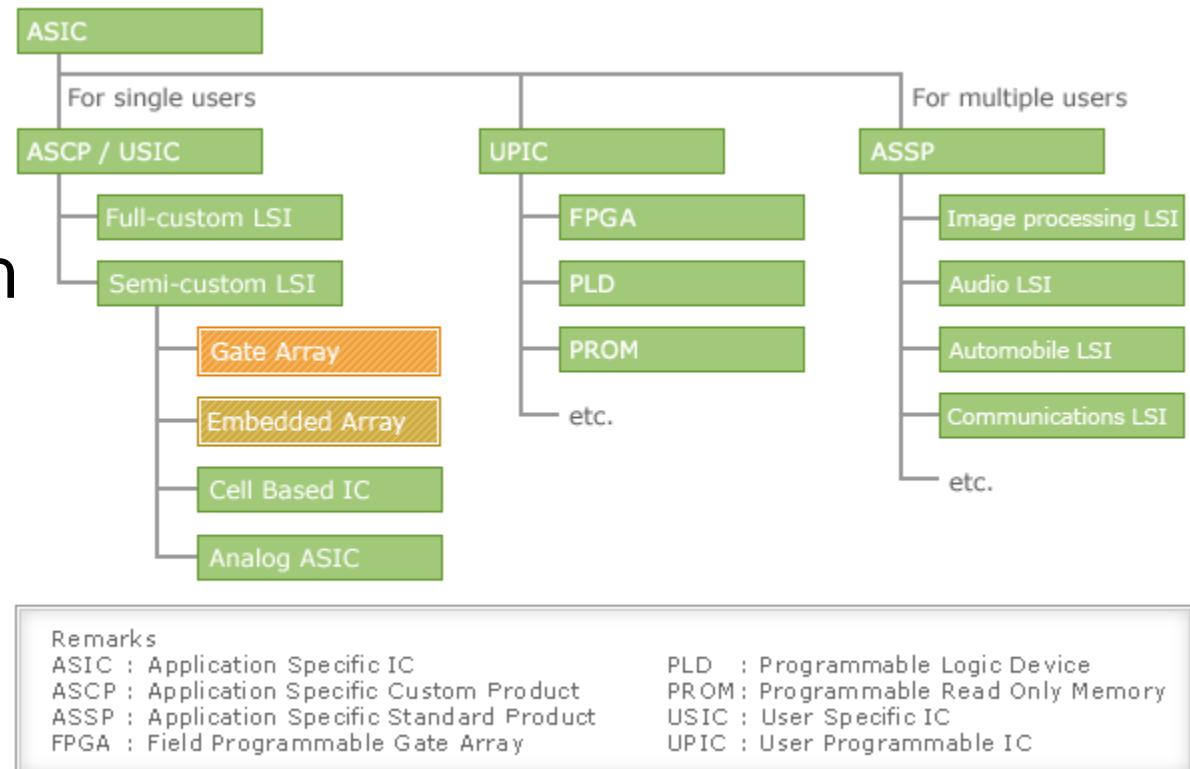
Gate Array

- Các trans tạo sẵn thành mảng, và nhà thiết kế thực hiện việc tạo các liên kết nối giữa chúng bằng cách sử dụng các thư viện cell và CAD.
- Sản xuất đơn lẻ được.

Phân loại ASIC

2/2

- FPGA, PLD, CPLD được xếp vào nhóm các IC lập trình được bởi người dùng



Nhu cầu thứ 1: kỹ thuật

- Một khi chip ASIC càng mạnh thì thiết kế càng tinh vi, phức tạp, càng tiềm tàng những sai sót lớn.
- Các phần mềm hỗ trợ, giả lập chưa thực sự phản ánh hết được hoạt động thực tế của hệ thống.
- Để kiểm tra thiết kế, người kĩ sư buộc phải đặt các nhà sản xuất sản xuất chip đơn lẻ và kiểm tra trên các ứng dụng, môi trường thực → tốn thời gian, và tiền bạc.
→ cần hệ nền kiểm thử phần cứng nhanh chóng.

Từ ASIC tới FPGA

2/4

- Năm 1984, Ross Freeman, Bernard Vonderschmitt, đồng sáng lập công ty Xilinx.
- Năm 1985, Xilinx đưa ra dòng FPGA thương mại đầu tiên, XC2064.
- Năm 2006, Freeman được vinh danh tại National Inventors Hall of Fame vì sáng chế này.
- FPGA dựa trên các công nghệ nền tảng như PROM và PLD, nhưng với kiến trúc mới hiệu quả hơn.
- Các thiết kế IC được thử nghiệm prototype trên các chip FPGA ngay lập tức → tiết kiệm thời gian và tiền bạc.



Ross Freeman đang kiểm tra layout của XC2064 ở dạng bird's-eye view. “Chú hàn lên tường cho anh”?

Nhu cầu thứ 2: số lượng

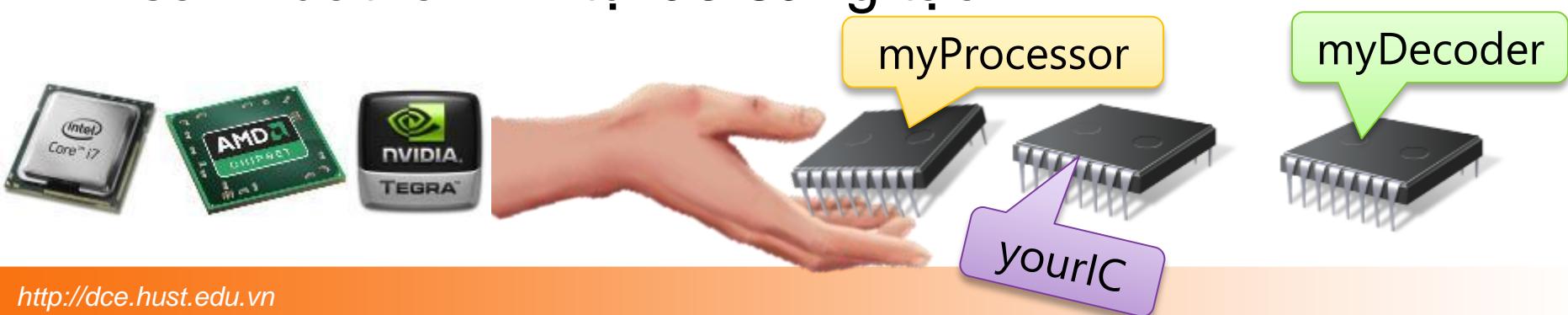
- Các nhà sản xuất lớn chỉ cung cấp các chip thông dụng trên thị trường, với số lượng lớn.
→ bỏ qua nhu cầu về các IC chuyên dụng có số lượng thấp, nhưng tổng nhu cầu thì rất lớn.

Nhu cầu thứ 3: giá thành

- Ban đầu, các chip lập trình được có giá thành khá cao và tốc độ chậm, chỉ sử dụng trong các phòng thí nghiệm.
- Khi công nghệ sản xuất phát triển vượt bậc, thì các chip lập trình được ngày càng mạnh và rẻ
→ ứng dụng đại trà.

Hệ quả: từ phòng thí nghiệm với ứng dụng

- FPGA không chỉ là chip prototype trong các phòng thí nghiệm, mà thực sự trở thành sản phẩm thương mại đại chúng.
- Thị trường phần cứng được cung cấp một dòng sản phẩm có thể thiết kế và sử dụng được ngay.
- Các công ty vừa nhỏ, các kỹ sư hoạt động độc lập, không phải lệ thuộc vào các IC của các nhà sản xuất lớn → tự do sáng tạo.



Custom IC vs FPGA

CustomIC

FPGA

Chi phí chế tạo lớn

Chi phí chế tạo thấp

Tối ưu tốc độ, năng lượng...

Tối ưu tính năng

Sản xuất quy mô lớn

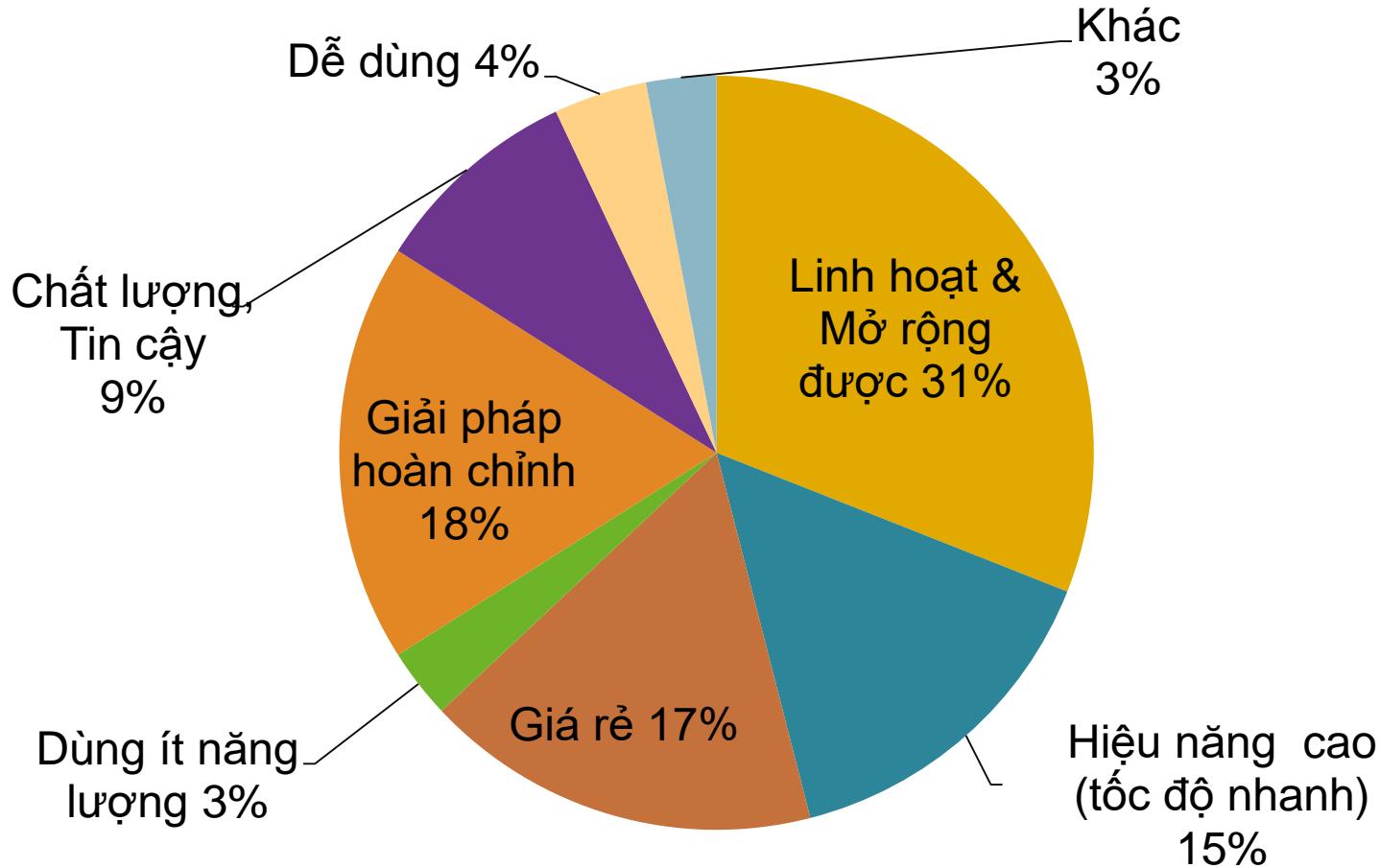
Sản xuất quy mô nhỏ



Tại sao nên dùng FPGA?

1/3

Đánh giá mức độ ảnh hưởng của các yếu tố trong thiết kế IC

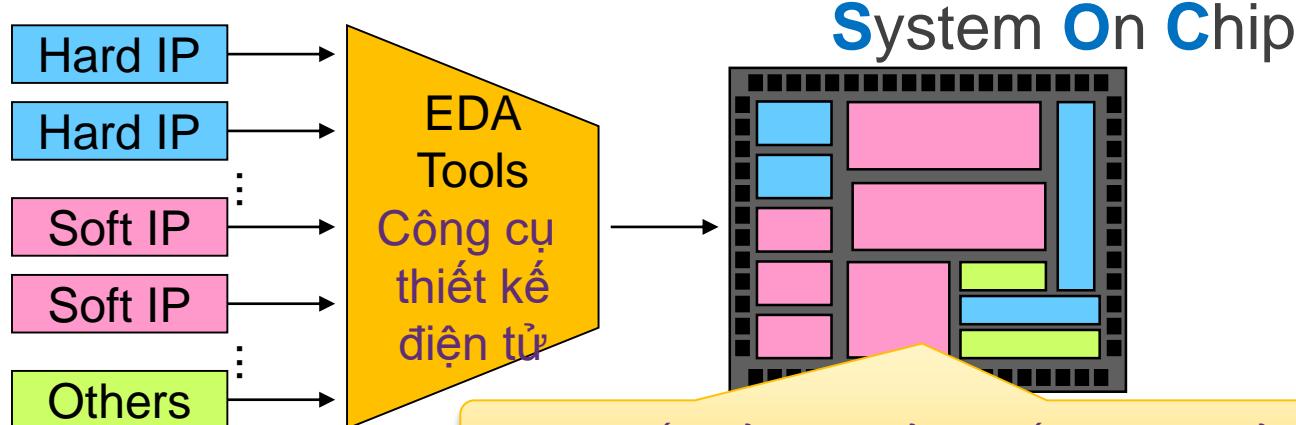


- Yếu tố linh hoạt và mở rộng rất quan trọng

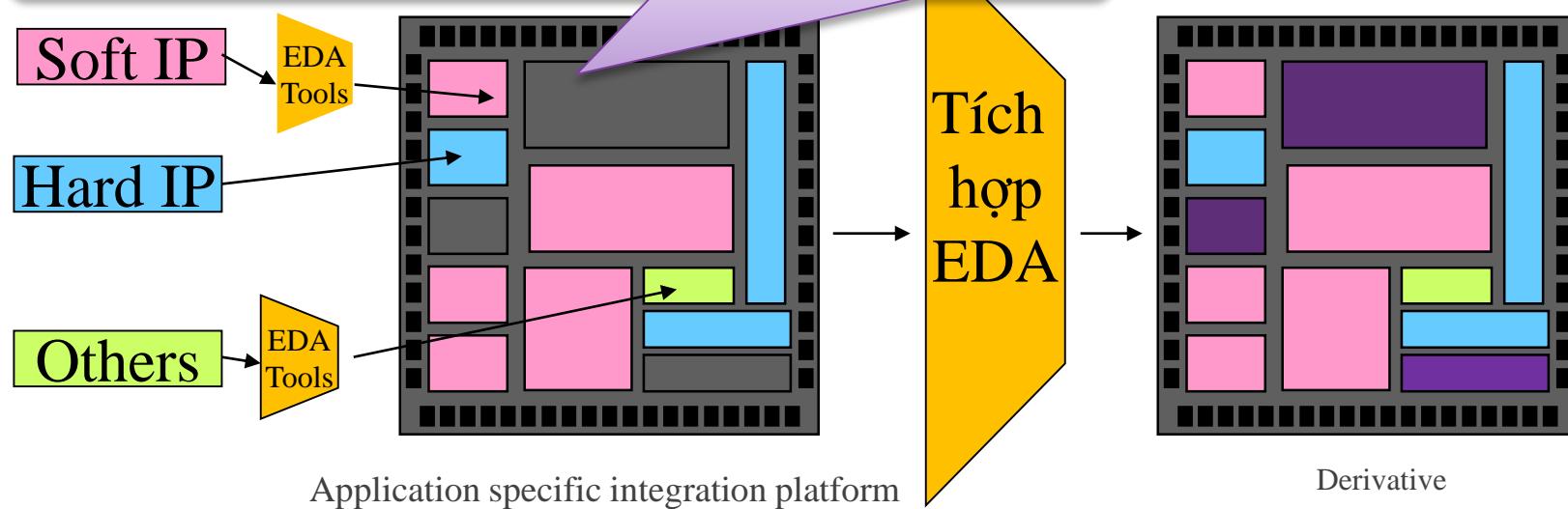
Tại sao nên dùng FPGA?

2/3

Intellectual Property
IP-based



Nhà sản xuất thiết bị tùy biến một phần



Platform-based

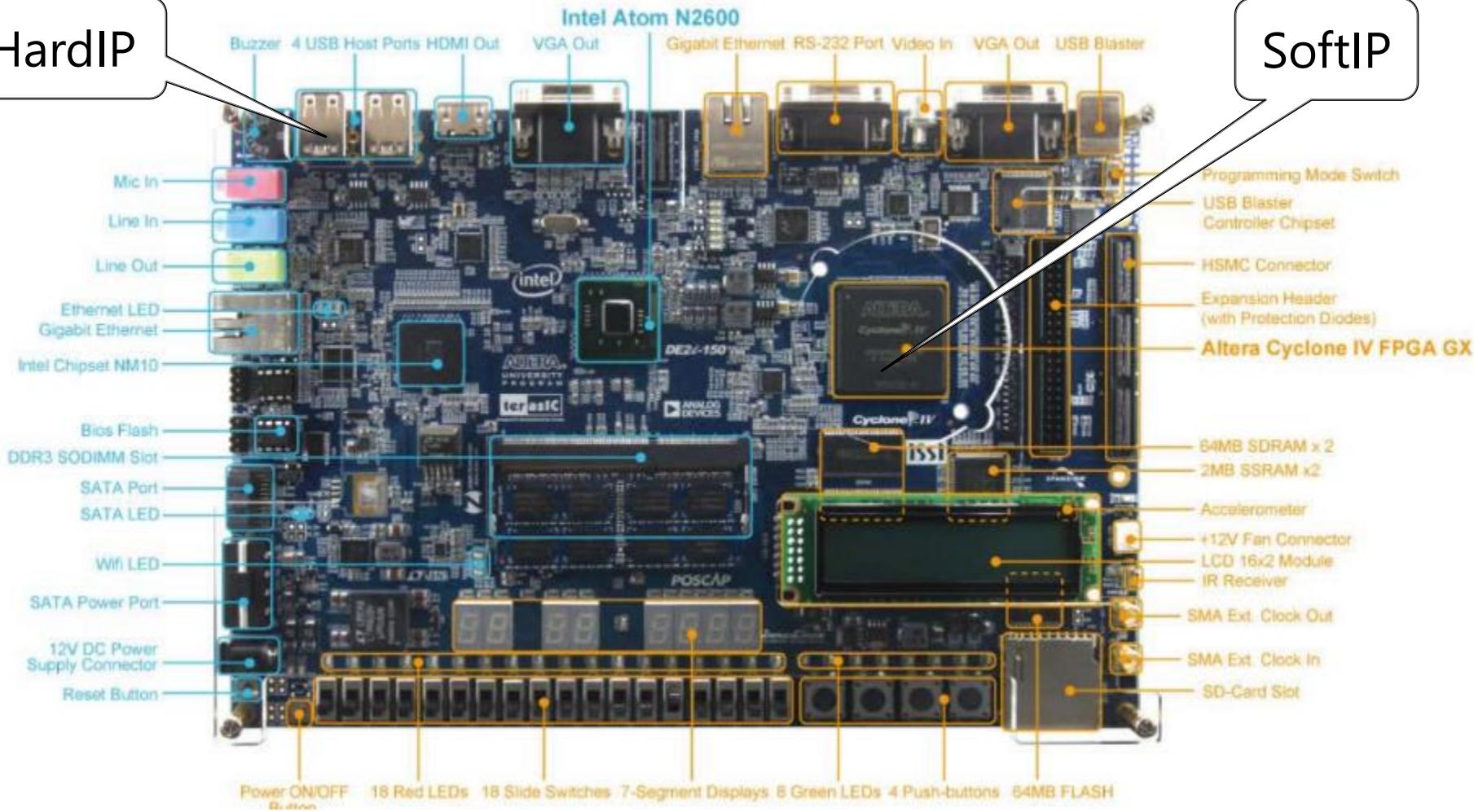
Derivative

Tại sao nên dùng FPGA?

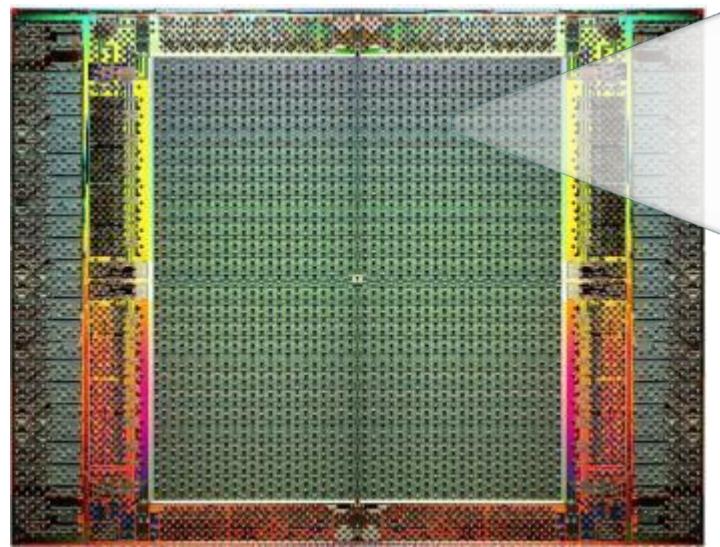
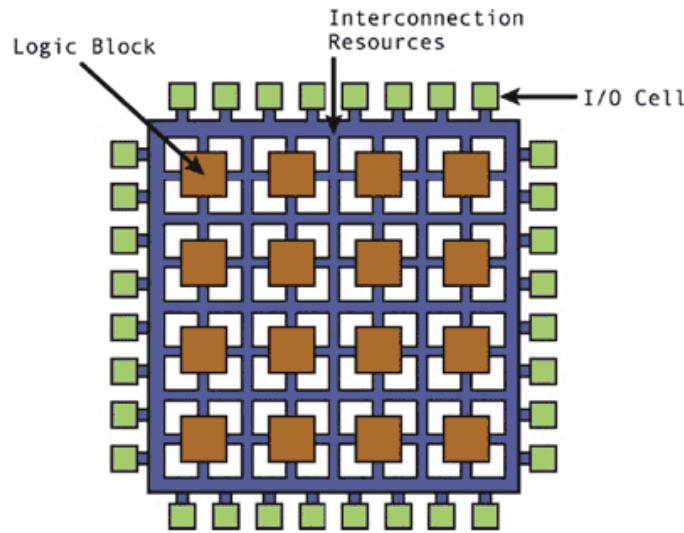
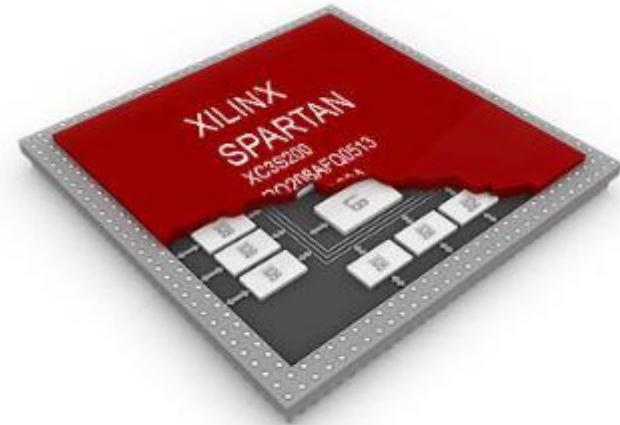
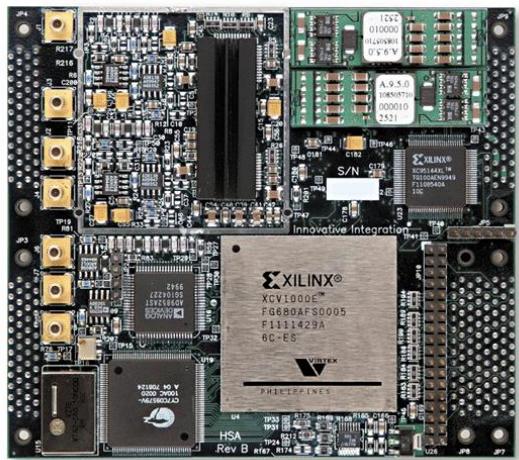
3/3

HardIP

SoftIP



Một số hình ảnh về FPGA



Intel Core i7 die

Đặc trưng FPGA: cấu hình được

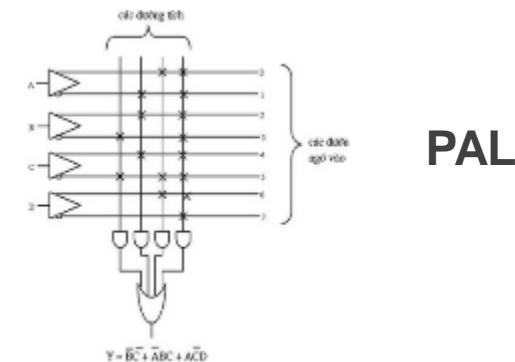
- RCC, Reconfigurable Computing, là các thiết bị cấu hình được. Ví dụ: RAM, ROM, PLA, PAL.

RAM/ROM

Input: Địa chỉ

Output: Giá trị

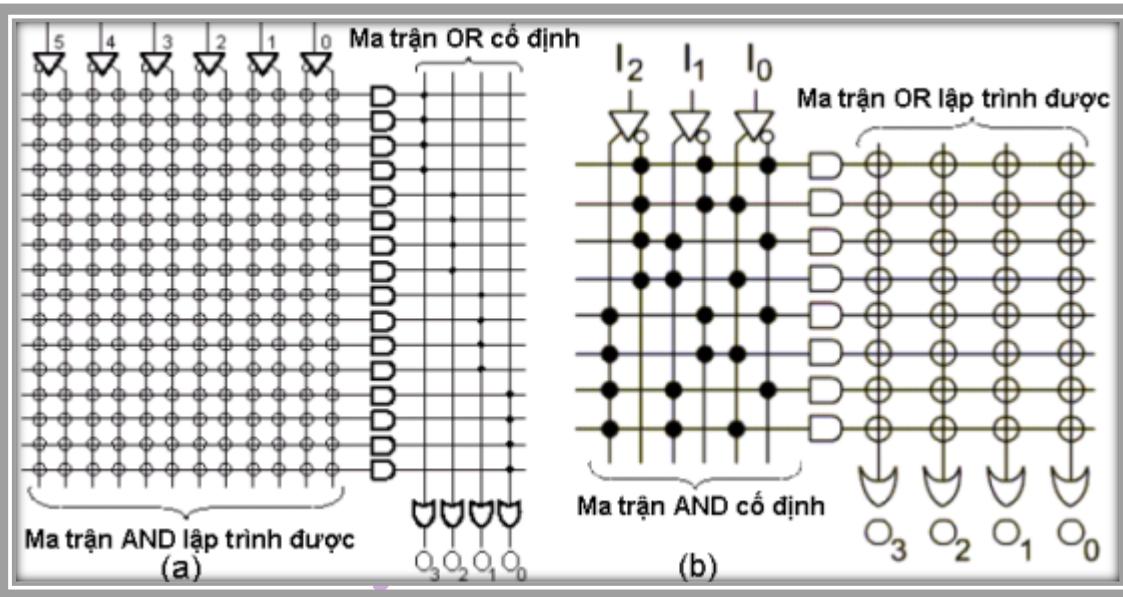
$$\begin{aligned} \text{Giá trị} &= f(\text{Địa chỉ}) \\ y &= f(x) \end{aligned}$$



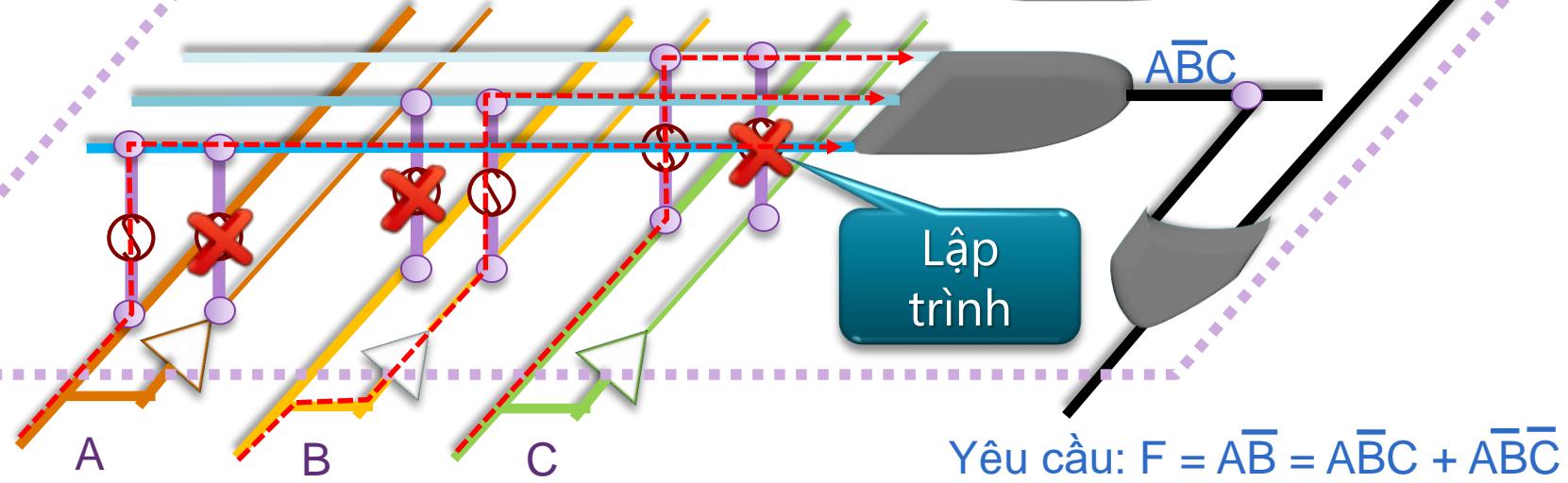
- FPGA với tính chất lập trình được cao, ưu việt, là đại diện của RCC.



Kiến trúc PAL, PROM

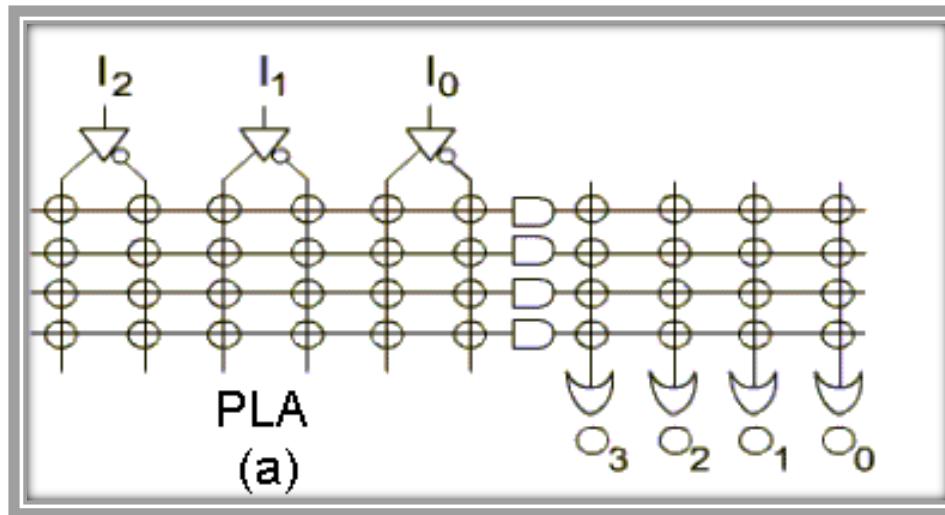


PAL thương mại



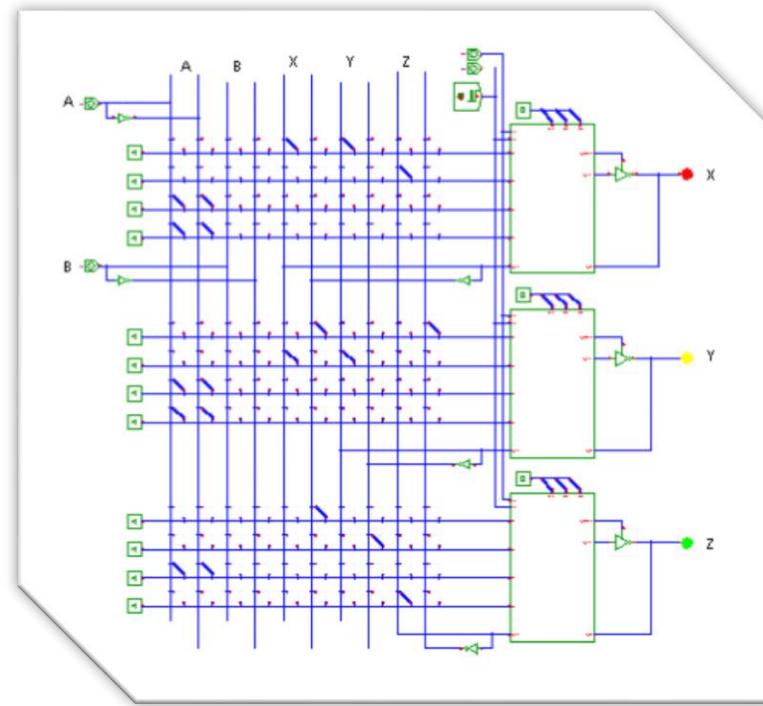
Kiến trúc PLA

- Programable Logic Array, cả ma trận AND và OR đều lập trình được.
 - Tiết kiệm dung lượng ma trận.
 - Bị hạn chế bởi số lượng các cổng AND khi số đầu vào của cổng OR lớn hơn số cổng AND.
 - Trễ truyền lan lớn hơn và mật độ tích hợp nhỏ.



Kiến trúc GAL

- Generic Array Logic nâng cấp từ PAL, gồm một ma trận AND lập trình được (cấu tạo từ EEPROM) và ma trận OR cố định.
- Tuy nhiên, các cổng OR nằm trong các macrocell được nối với flip-flop và các bộ đòn kẽm để có thể chọn tín hiệu ra.
- Tên gọi chung của các thiết bị như PAL, PLA, GAL... là Programable Logic Device



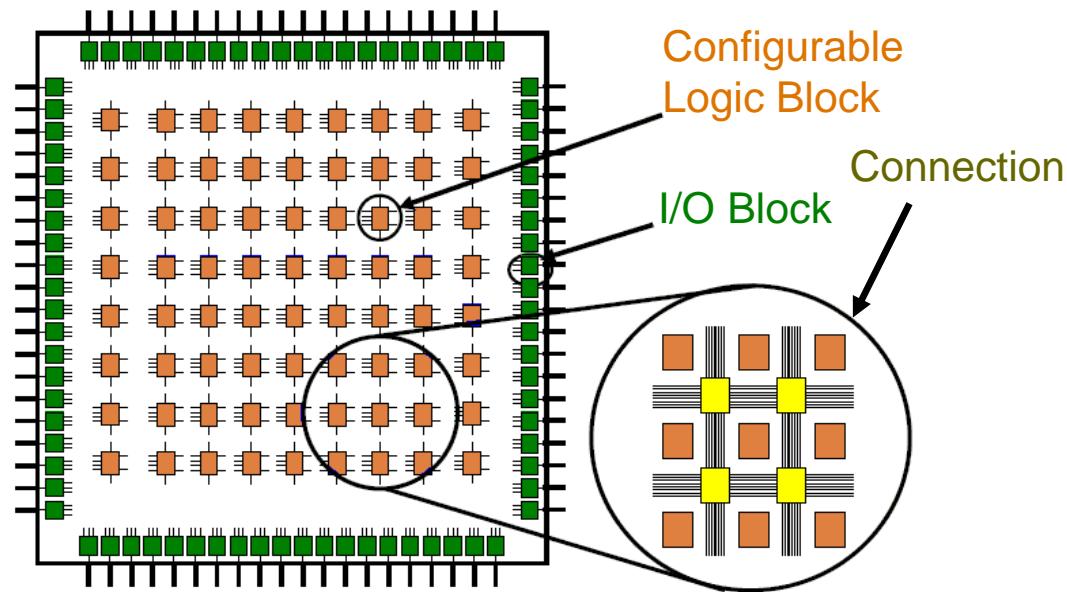
Ví dụ: Dùng GAL điều khiển đèn giao thông:

Kiến trúc FPGA

- FPGA gồm 3 thành phần chính
 - Khối logic – Logic Block (LB): đơn vị xử lý. (hoặc Logic Element)
 - Khối Vào ra – IO cell: giao tiếp với bên ngoài.
 - Liên kết nối – Interconnection: liên kết các đơn vị xử lý.

• Thành phần khác

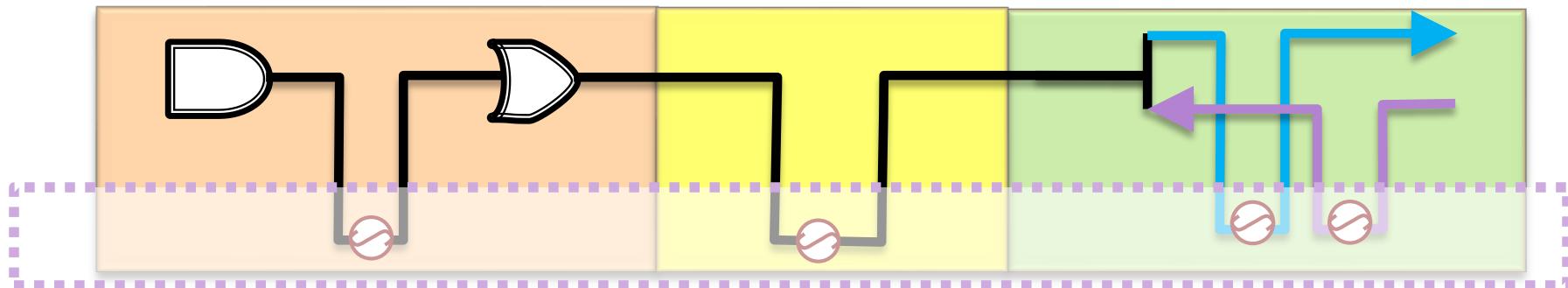
- Buffer
- ClockDII
- ...



Vì sao FPGA lập trình được

1/3

- Cả 3 thành phần: **khối logic**, **khối vào ra**, **liên kết nối**, đều lập trình được.
- **Lập trình cho khối logic** là hành động: “có kết nối hay không phần tử logic A với phần tử logic B?”
- **Lập trình cho khối vào ra** là hành động: “có kết nối hay không đầu ra logic A với chuẩn ngoại vi B?”
- **Lập trình cho liên kết nối** là hành động: “có kết nối hay không **khối logic A** với **khối logic/vào ra B**?”



Vì sao FPGA lập trình được

2/3

- Với FPGA, **lập trình là quá trình định tuyến** giữa các phần tử logic, flipflop... đã được chế tạo cố định sẵn, để thực thi một tác vụ nào đó.
- Một tuyến đều được chế tạo sẵn, và đính kèm một khóa đóng mở. Tuyến được thiết lập hoặc hủy, tương ứng với trạng thái khóa đóng hay mở.
- Mỗi trạng thái của khóa đóng/mở ứng với một bit nhớ trạng thái 0/1 tương ứng.
- Tập hợp các bít nhớ tạo thành bộ nhớ cấu hình cho FPGA.
 - Bảng định tuyến được lưu trữ trong bộ nhớ.
 - Công cụ EDA sẽ dịch các file HDL thành bảng định tuyến.

Vì sao FPGA lập trình được

Phân tích

Ánh xạ các kết nối
vào FPGA cụ thể

HDL

- $c := a + b;$
- if ($c == 1$) then $cf := 1;$

Net List / RTL

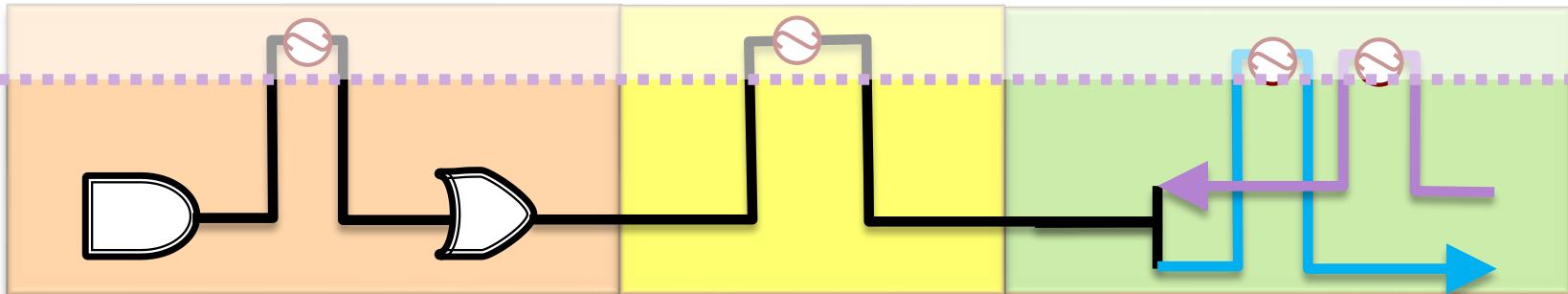
-



BIT file

- 01000100
- 11010101
- 10001001

Bộ nhớ cấu hình

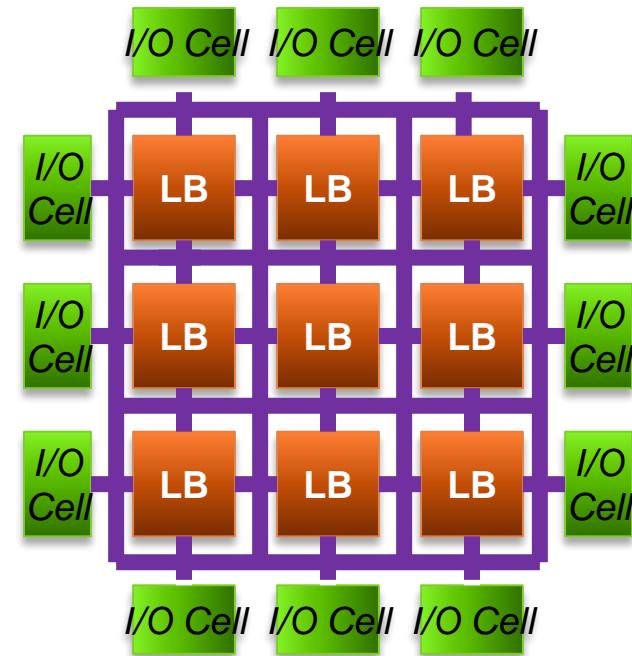


LB: Khối logic lập trình được

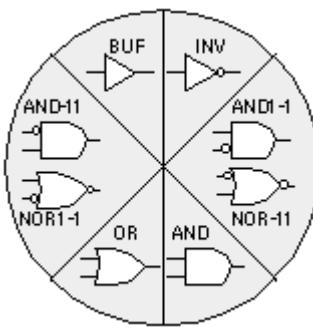
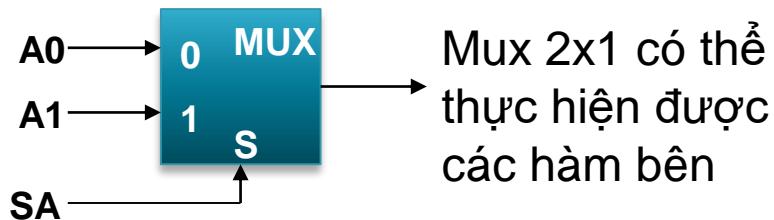
- Tất cả ASIC lập trình được, bao gồm FPGA, đều chứa các khối logic (cell logic) cơ bản giống nhau tạo thành dải.

Có 4 loại khối logic:

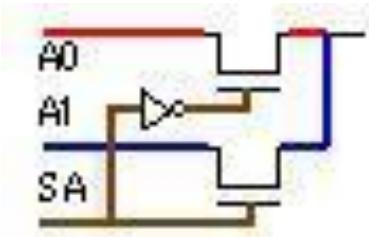
- Dựa vào bảng tìm kiếm (*LUT – Lookup Table*) Xilinx
- Dựa vào bộ ghép kênh (*Multiplexers*) Actel
- Dựa vào PAL/PLA Altera
- Transistor Pairs



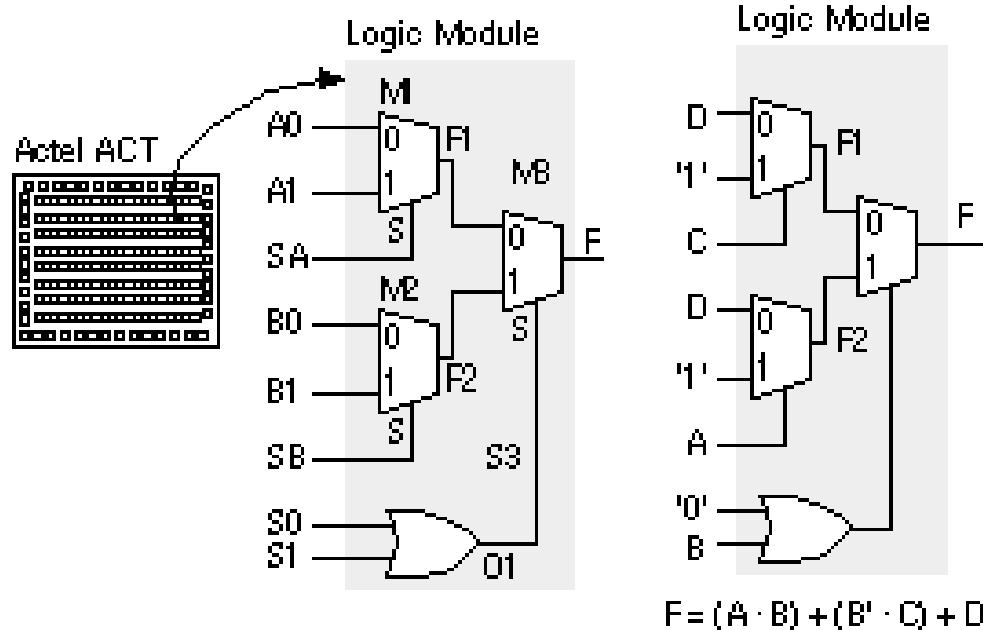
LB: Dựa vào bộ ghép kênh



Biểu diễn mức transistor



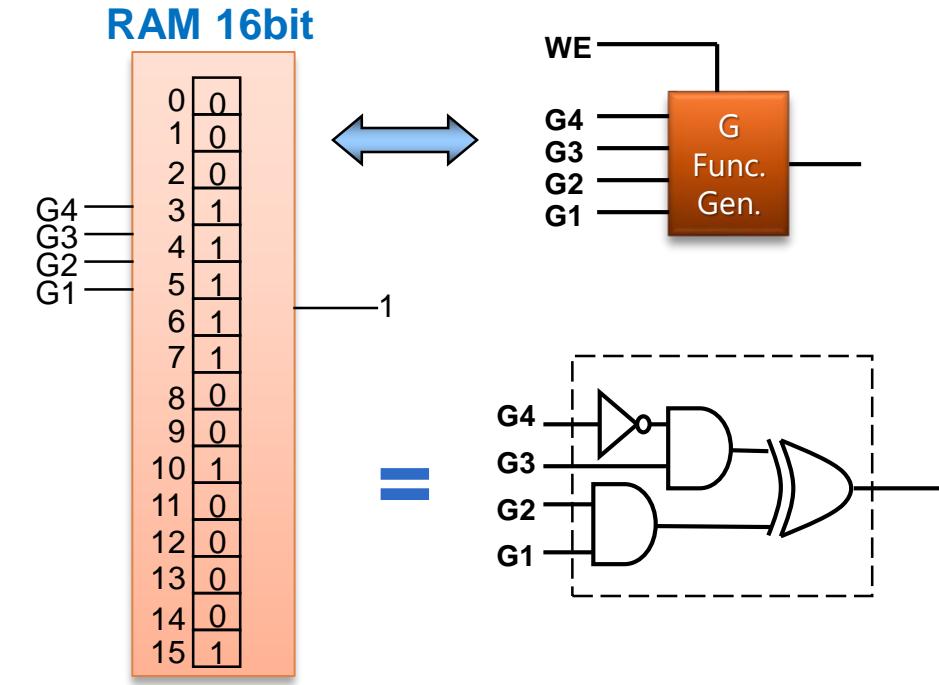
Cấu trúc cell logic ACT 1 (đơn module)



- Cell ACT 1 chỉ có một module logic.
- Cell ACT 2, 3 có nhiều module logic hơn và có Flip Flop riêng.

LB: Dựa vào bảng tìm kiếm

- Bảng tìm kiếm, **LUT**, **Look-Up Table**, là một SRAM có K đầu vào với 2^K bit nhớ, thực hiện được mọi hàm logic có K biến.

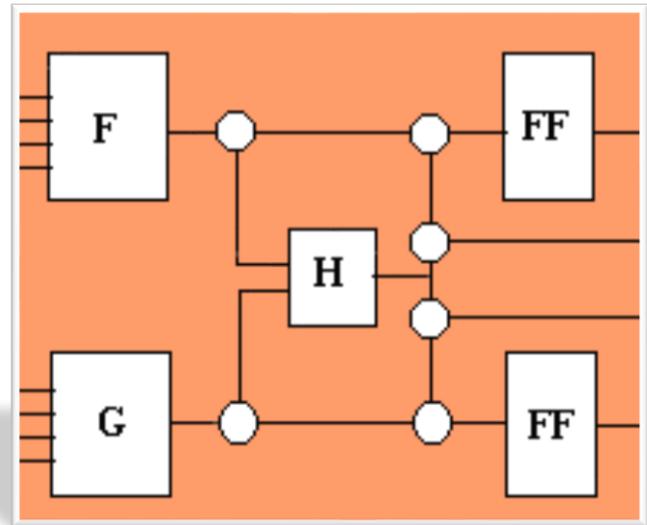


- Thông thường, $K = 4$.
- Tín hiệu ra của một LUT có thể quay trở lại, thành đầu vào của chính LUT đó, hoặc LUT khác.
- Trong một LB, thường có 3 LUT và được gọi là bộ thực hiện hàm F, G và H.

LB: Dựa vào bảng tìm kiếm

2/3

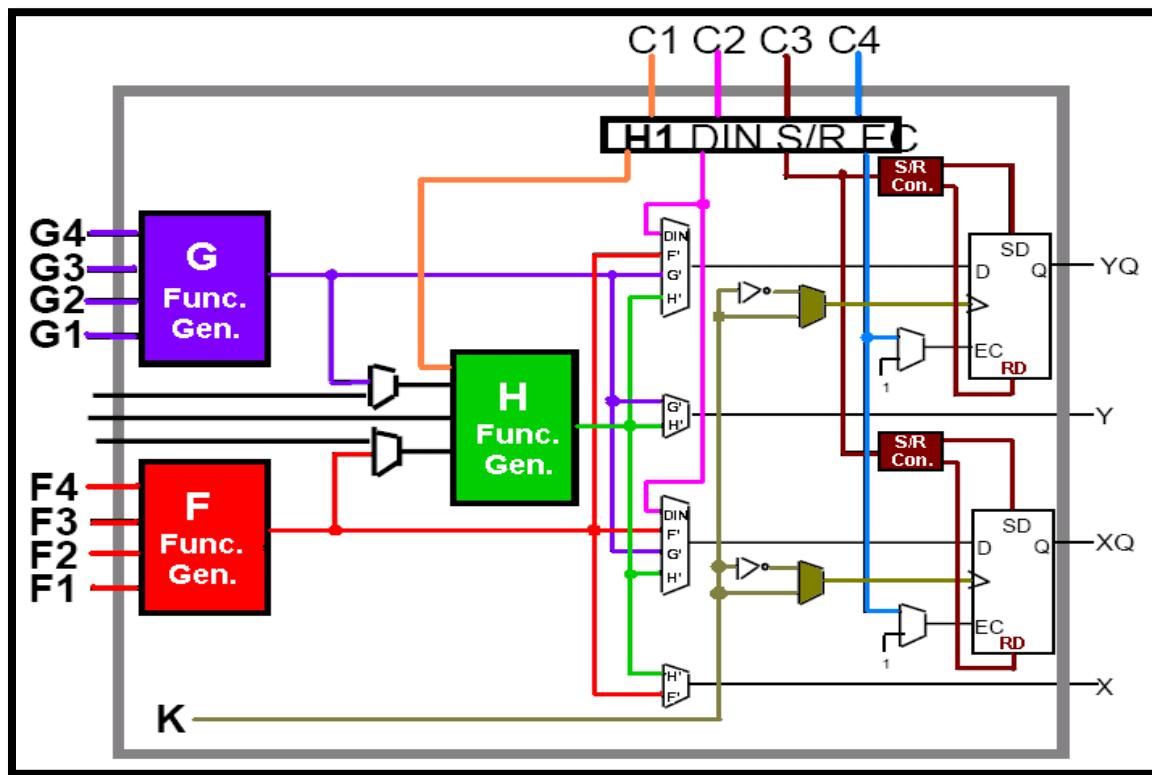
- Các LUT F và G tương đương và độc lập với nhau, thực thi các hàm 4 biến và đưa kết quả tính toán ra ngoài CLB, hoặc nhớ vào FF.
- Nếu phép toán có nhiều hơn 4 biến thì LUT F, G sẽ đưa kết quả tới LUT H để mở rộng thêm.
- Flip-Flop đóng vai trò:
 - Bit nhớ hoặc
 - Chốt dữ liệu
- Hai FF có thể set/reset đồng bộ/không đồng bộ, tích cực theo sườn âm/dương...



Cấu trúc cơ bản của LB dạng LUT

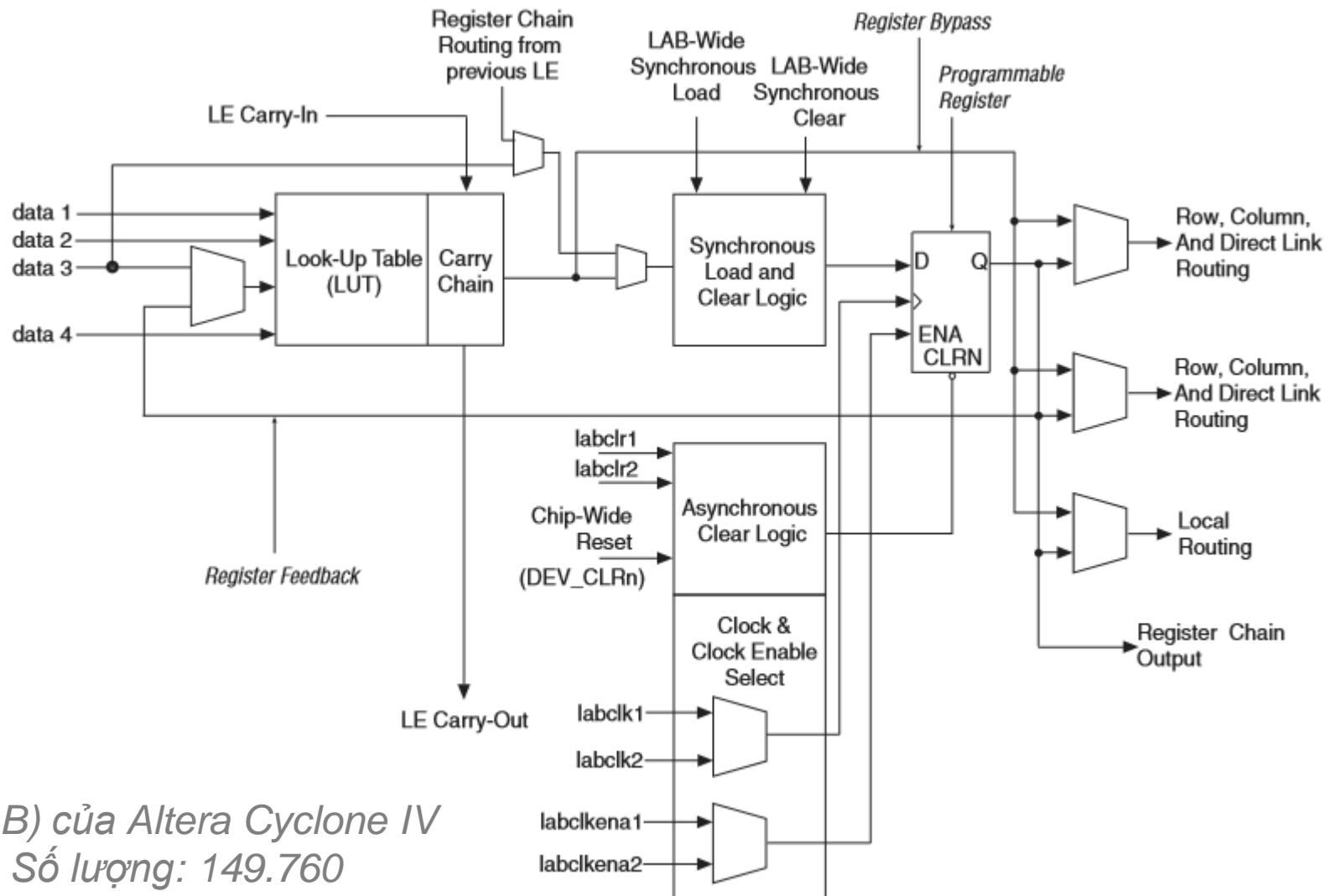
LB: Dựa vào bảng tìm kiếm

- Số liệu đưa vào LB có thể được xử lý bởi các hàm 4 đầu vào, có thể được chốt ở thanh ghi, có thể được chọn kênh, hoặc bởi cả 3 thao tác trên.



Khối logic lập trình
được, CLB, của
XC4000 E/X

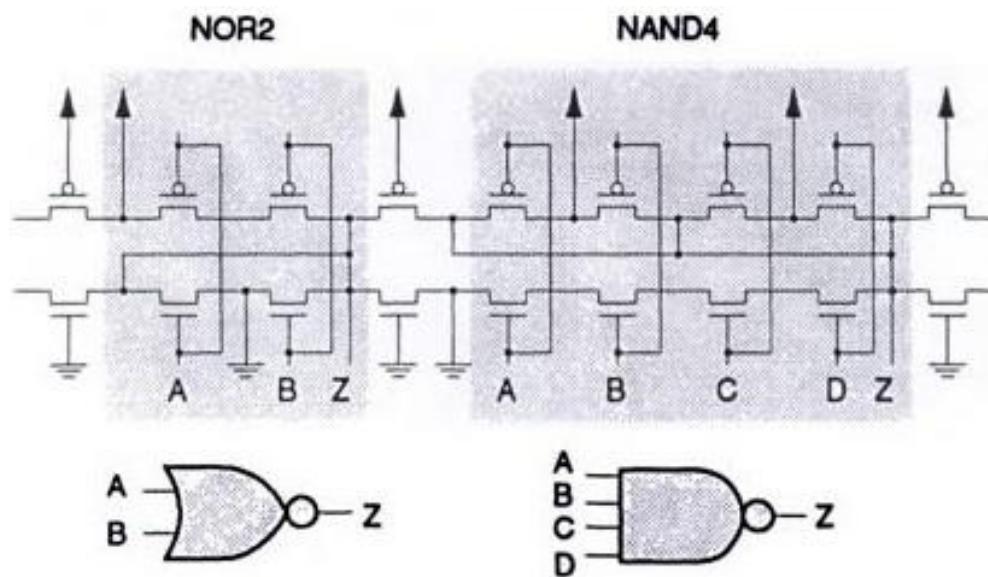
LB: Dựa theo PAL/PLA



LE(LB) của Altera Cyclone IV
Số lượng: 149.760

LB: Dựa theo Transistor Pairs

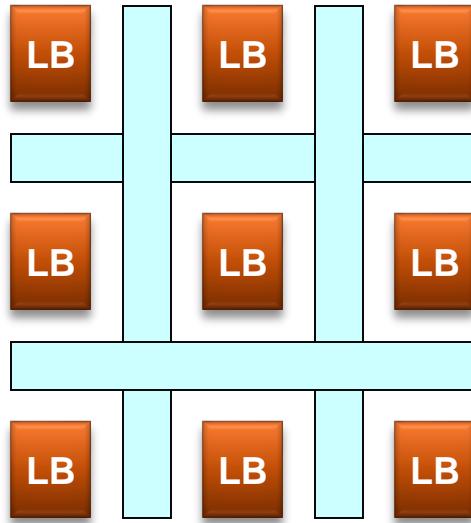
- Còn được gọi là CrossPoint FPGA.
- Số lượng cỗng ít, ~ 4000



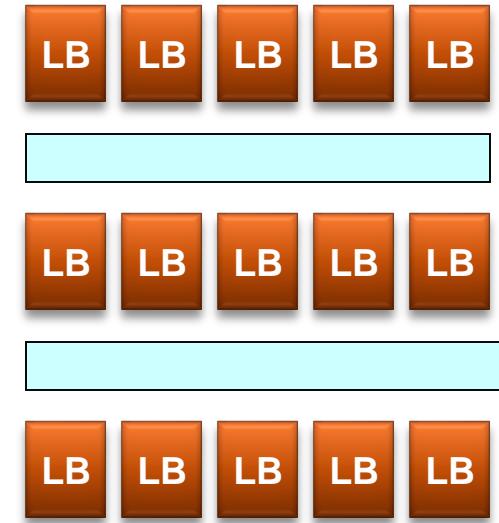
Phân loại FPGA theo độ phức tạp của đơn vị xử lý

- FPGA kế thừa nhiều ý tưởng thiết kế của các sản phẩm trước → sự quen thuộc trong kiến trúc.
- Nhưng số chức năng, mức độ tích hợp, khả năng tính toán của từng đơn vị xử lý trong FPGA có khác nhau, gồm **Coarse** /kɔ:s/ , và **Fine**.
- **Coarse-grained**: Đơn vị xử lý là một tập hợp của các PLD, các khối cấu hình được CLB, thực thi được hàm phức tạp, có yêu cầu tính toán lớn. Ví dụ: Actel Mux, Xilinx LUT.
- **Fine-grained**: Đơn vị xử lý chỉ gồm các khối cấu hình được CLB nhỏ, thực thi các hàm logic đơn giản. Ví dụ Transistor Pairs.

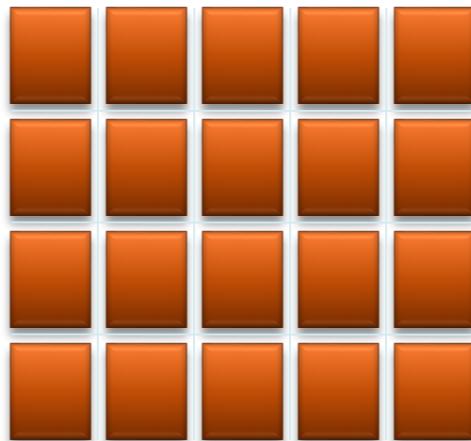
IR: Các loại liên kết nối



Ma trận đối xứng
Symmetrical Array

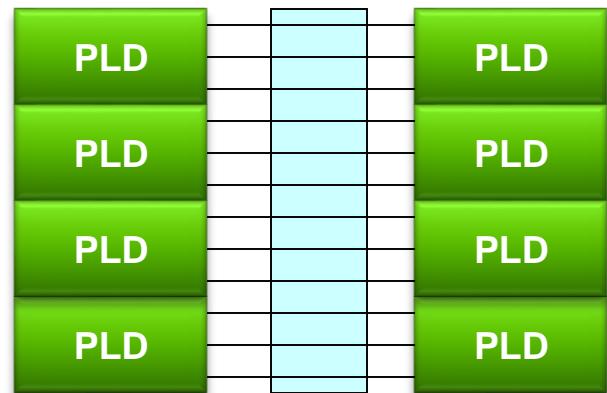


Cấu trúc dòng
Row-based



Sea-of-Gates

PLD phân cấp
Hierarchical (CPLD)



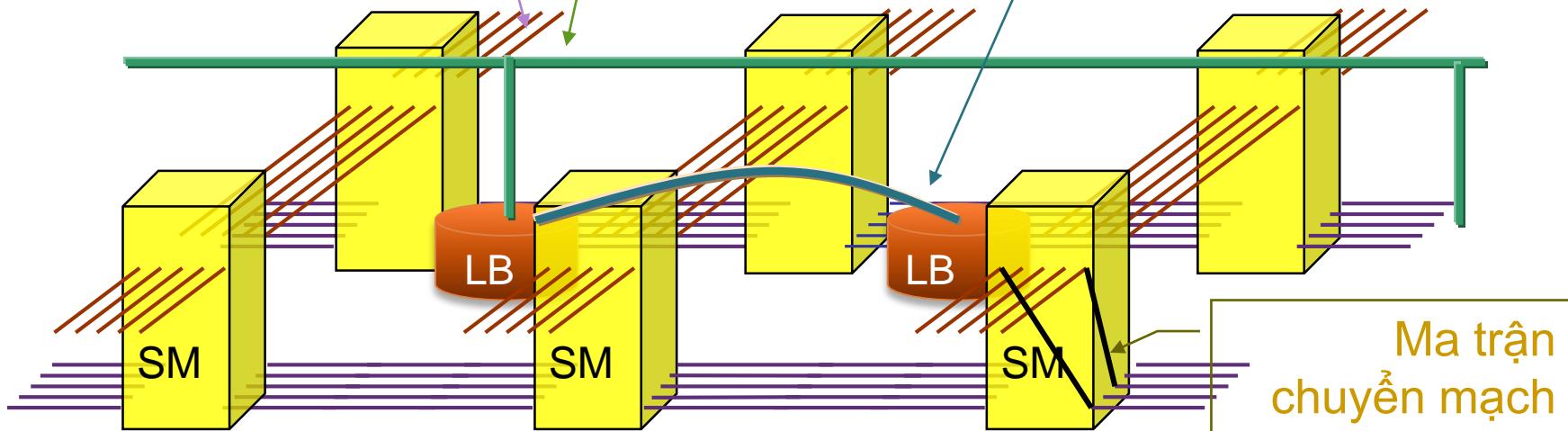
IR: Ma trận đối xứng

1/3

Kết nối đa năng

Kết nối dài

Kết nối trực tiếp



- Liên kết nối dài, tín hiệu clk liên thông toàn bộ dải.
- Liên kết nối trực tiếp giữa 2 khối LB.
- Liên kết nối đa năng gồm nhiều kết nối và các chuyển mạch.

IR: Ma trận đối xứng

2/3

Kết nối dài



Kết nối trực tiếp



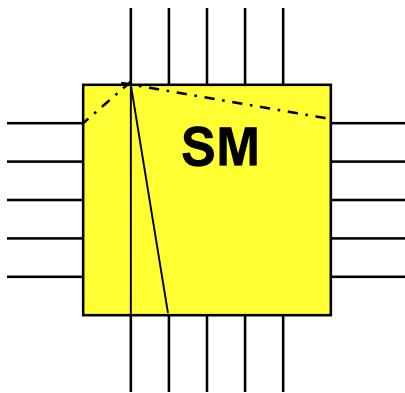
Kết nối đa năng



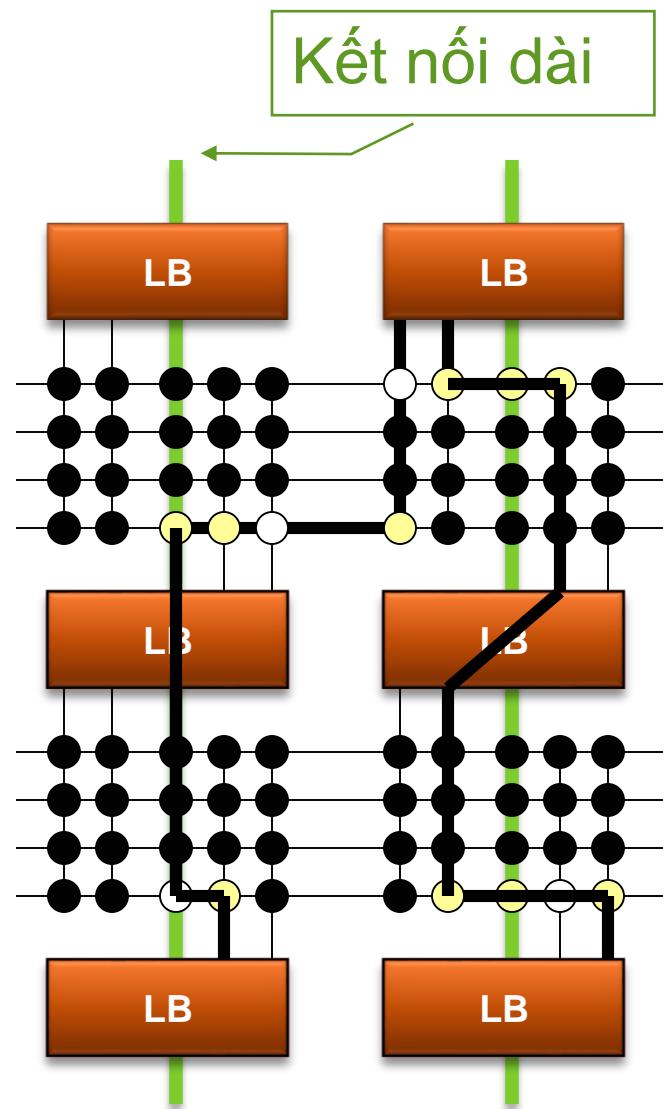
IR: Ma trận đối xứng

3/3

- Phần lớn các kết nối tạo thành các lưới kết nối theo hàng và theo cột.
- Giao cắt giữa các kết nối theo hàng và theo cột sẽ tập trung các điểm cần lập trình kết nối, tạo thành ma trận chuyển mạch (Switching Matrix), nằm phân tán trong FPGA.



IR: Cấu trúc dòng



Nhà sản xuất FPGA



Arrix
middle end



Virtex
high end



Cyclone
low end



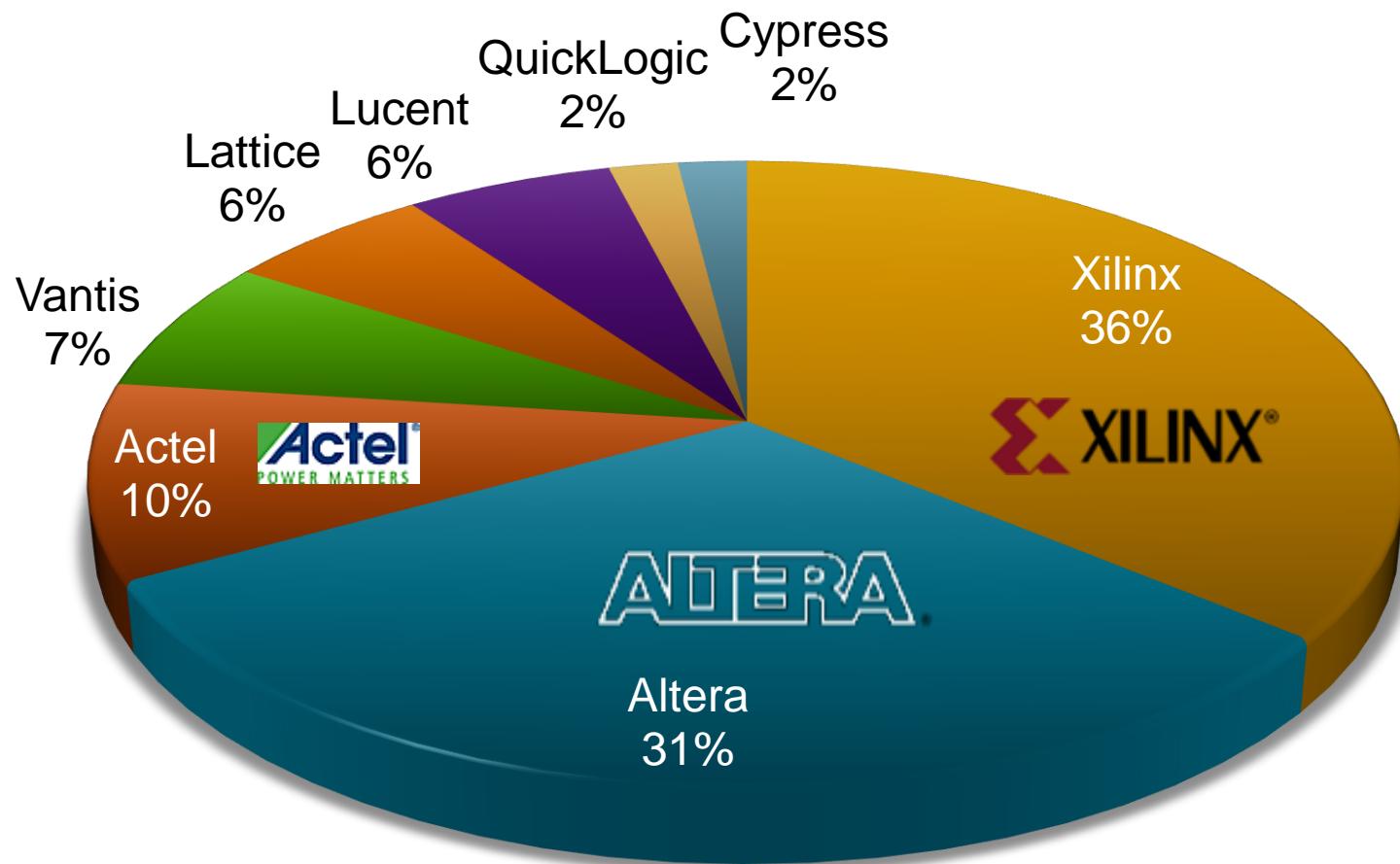
Arria
middle end



Stratix
high end



Thị phần FGPA

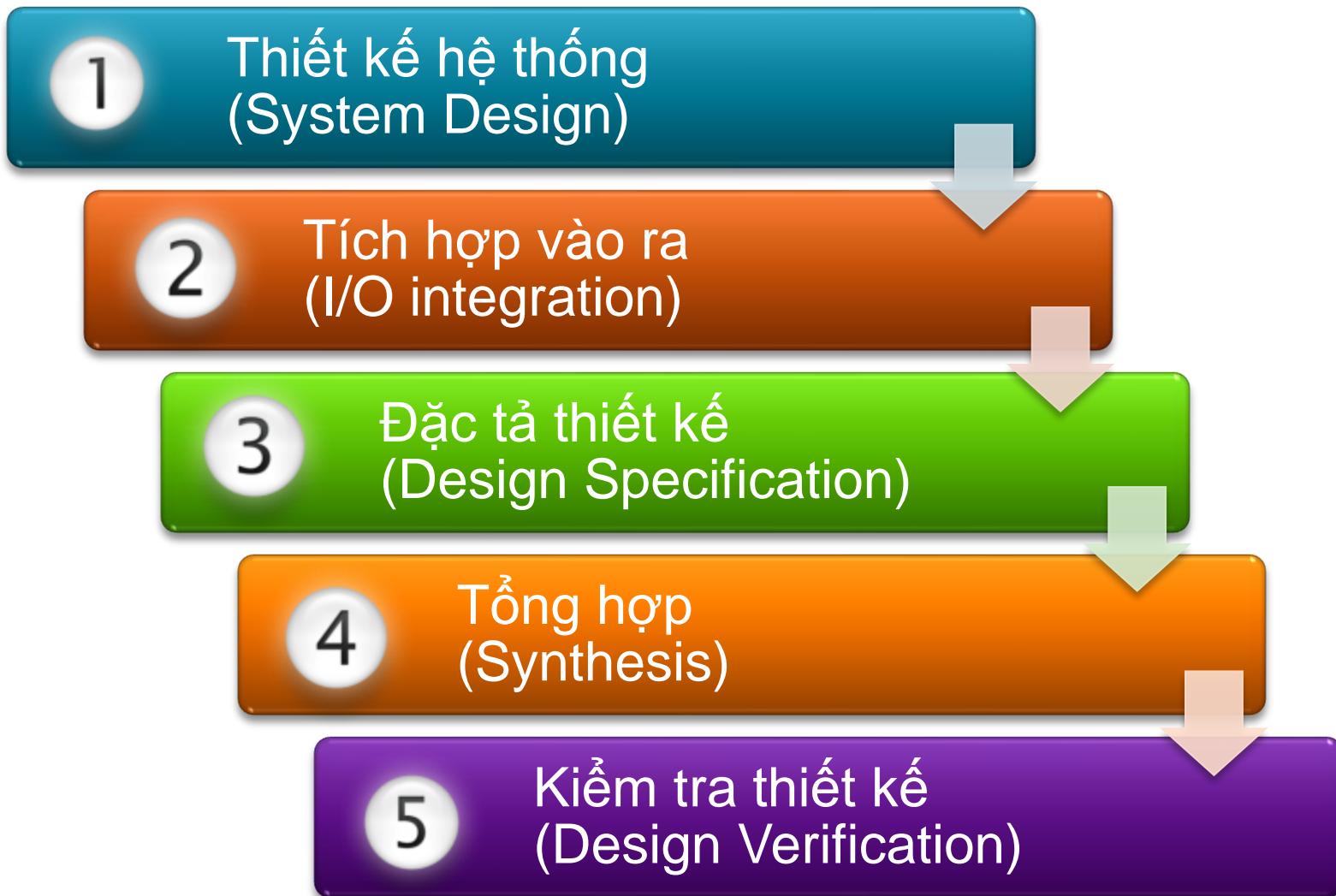


Lựa chọn FPGA

- Ngôn ngữ HDL phù hợp.
- Bộ công cụ CAD, EDA phù hợp.
- Ước lượng được số lượng các CLB cần thiết
- Dự kiến số lượng các chân I/O cần thiết.
- Điện áp hoạt động. Các FPGA mới sử dụng mức điện áp thấp LVTTL, LVCMOS, đòi hỏi phải chuyển đổi điện áp để tương thích với điện áp TTL, cung cấp một hoặc nhiều vùng sử dụng đồng thời đa mức điện áp.
- Tốc độ FPGA.
- Khả năng tài chính.

Quy trình thiết kế ASIC dựa trên FPGA

1/4



- Bước 1 - Thiết kế hệ thống
 - Phần chức năng thực hiện trên FPGA
 - Phần chức năng này tích hợp (kết hợp) với phần còn lại của hệ thống như thế nào
- Bước 2 - Tích hợp vào ra với phần còn lại của hệ thống

• Bước 3 - ĐẶC TẢ THIẾT KẾ

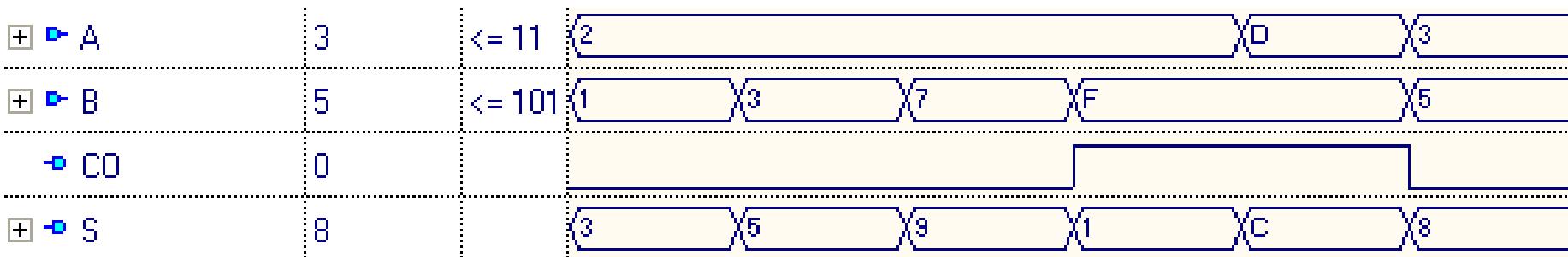
- Mô tả chức năng của thiết kế bằng:
 - Các trình soạn sơ đồ logic
 - Các ngôn ngữ đặc tả phần cứng
- Kết hợp mô phỏng

• Bước 4 - TỔNG HỢP LOGIC

- Giống bước Tổng hợp logic trong quy trình đầy đủ
- Thực hiện tối ưu hóa
 - trễ
 - năng lượng hao phí

• Bước 5 - Kiểm tra thiết kế

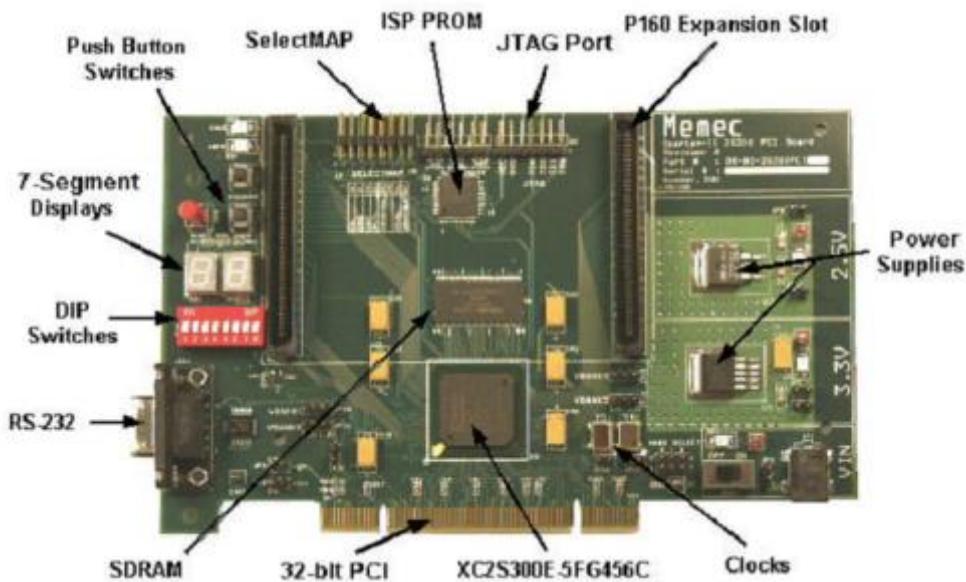
- Thực hiện các mô phỏng, phân tích cuối cùng (RTL, thời gian...)
- Xác định các thông số của ASIC đã thiết kế (tần số xung nhịp...)



• Nạp chip và chạy thử trên hệ thống!

Kit phát triển FPGA

1/3

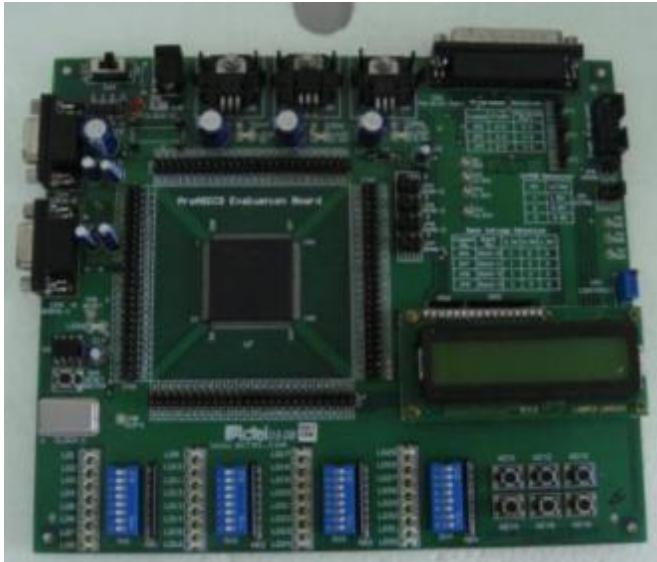


- Spactan II 200
 - Spactan II 300
 - PCI 32bits
 - RS232

[Userguide](#)

Kit phát triển FPGA

2/3

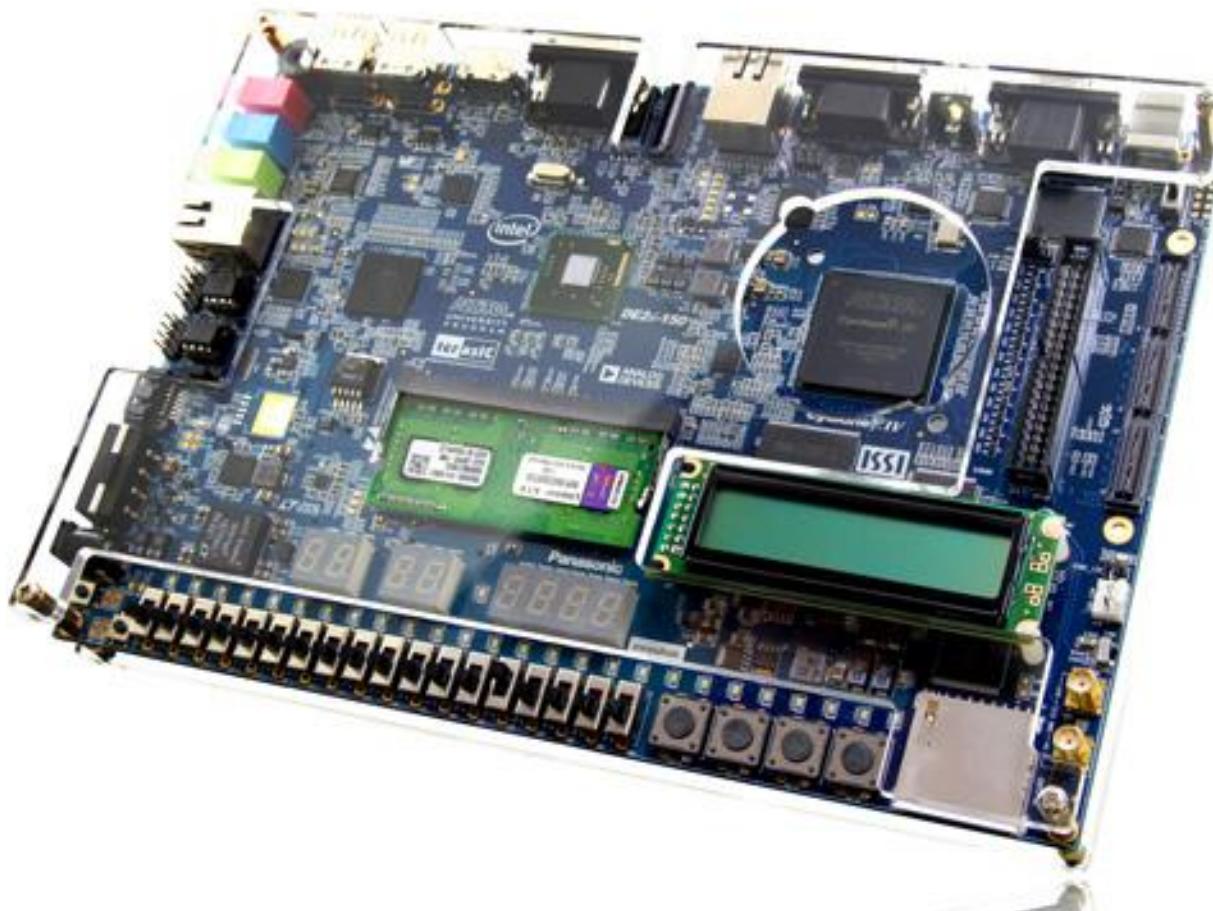


- ProASIC3 Evaluation Board

[Userguide](#)

Kit phát triển FPGA

3/3



- DE2i-150 Intel Atom & Altera Cyclone IV

Công cụ thiết kế trên FPGA

1/2

- IDE của nhà sản xuất FPGA.



- Chỉ có nhà sản xuất mới thấu hiểu nguyên tắc hoạt động của FPGA của họ.
→ chỉ có các IDE của nhà sản xuất mới routing, timing, cấu hình được cho FPGA.
- EDA của bên thứ 3 chỉ xử lý mức logic, rồi gọi IDE của nhà sản xuất để đảm nhiệm mức vật lý.

Công cụ thiết kế trên FPGA

2/2

- Một số gói chương trình của bên thứ 3:
 - Leonardo Spectrum, CT tổng hợp của Mentor Graphics
 - Synplify, CT tổng hợp của Synplicity
 - ModelSim , CT mô phỏng của Mentor Graphics.
 - Active-HDL, CT thiết kế và mô phỏng của Aldec Active

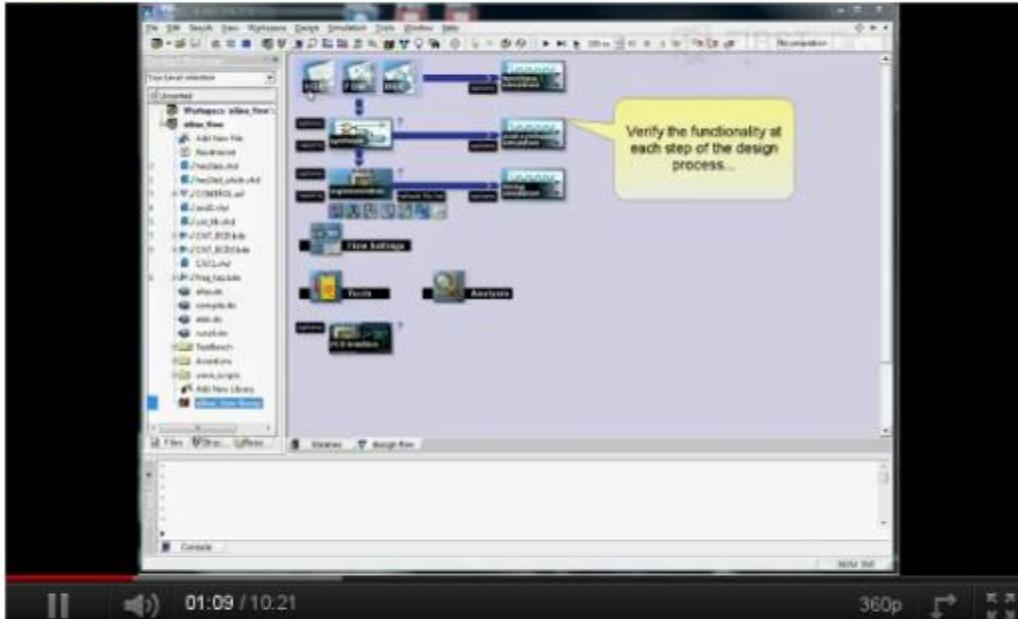


Sử dụng Active-HDL

YouTube Search Browse Upload

Aldec Active-HDL Demo

fedasupport 3 videos Subscribe



01:09 / 10:21 360p 1,159

Like Add to Share Download

Link to this video:
<http://youtu.be/RPTaOz-mqVI>

show options

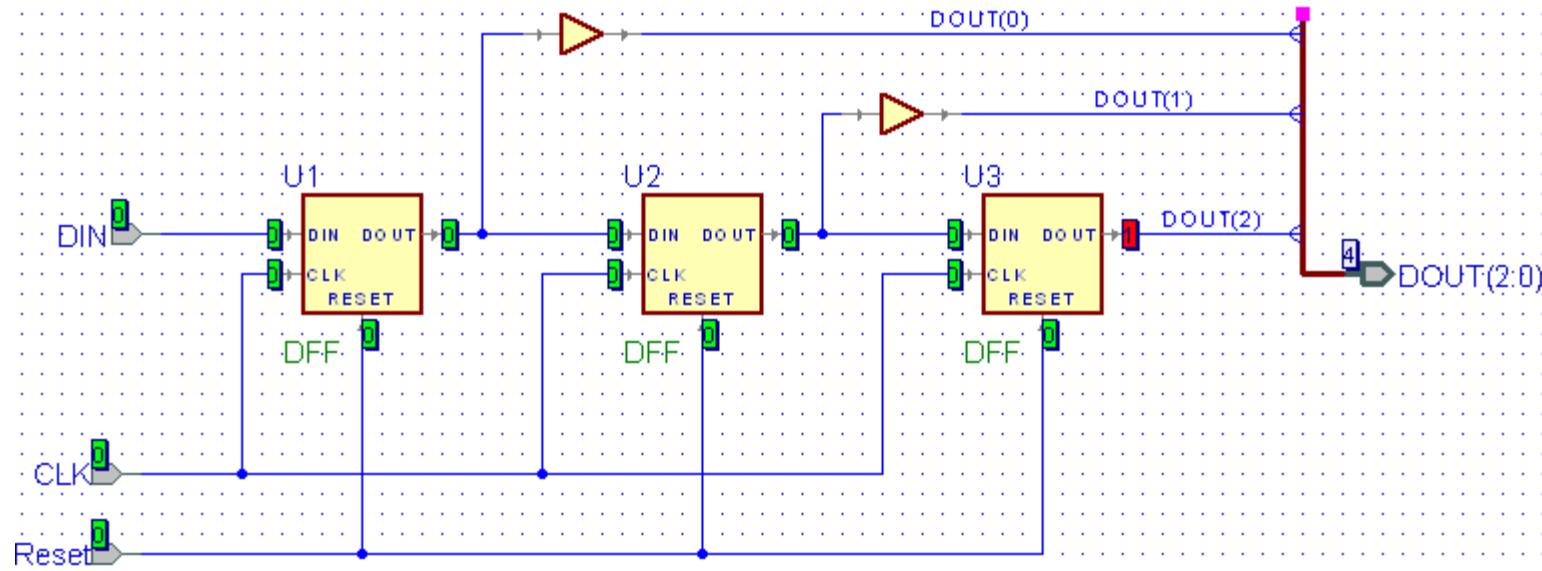
Suggestions

- Active HDL Tutorial - Part 1 by kyle1352000 136 views 14:56
- Active HDL Tutorial - Part 2 by kyle1352000 68 views 14:03
- VHDL-FPGA - Ejemplo #3 Simulación de un Multip... by DigitalesI/TESI 1,513 views 4:03
- programar en vhdl con activ-vhdl.avi by Tricangry0 140 views 4:36
- ModelSim w/ VHDL top module by ahocc 15,092 views 7:10
- What's New with VHDL by ajnsleij 1,046 views 14:47



Active-HDL: Tạo thanh ghi dịch

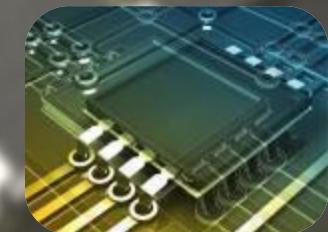
- Tạo thực thể thiết kế bằng code mẫu, lấy từ Language Assistant.
- Tạo thực thể thiết kế, bằng giao diện Block Diagram.
- Tra cứu lỗi và xử lý “port mapping” trong quá trình thiết kế bằng Block Diagram.
- Chạy giả lập mức logic,
- Cách đưa giá trị vào các tín hiệu để giả lập logic.





Phần II:

Điện tử số





KIẾN THỨC CƠ BẢN

- Các hệ cơ số
- Transistor
- Các phần tử trong mạch số
- Công suất tiêu thụ

Tín hiệu, tương tự, số

- **Tín hiệu** là một đại lượng vật lý **chứa đựng** thông tin hay dữ liệu và có thể **truyền đi** được
- Trong tự nhiên, mọi tín hiệu đều liên tục theo thời gian và biên độ,
- Trong lý thuyết thông tin, tín hiệu có 2 dạng

Tín hiệu tương tự

- Liên tục theo thời gian
- Liên tục về biên độ



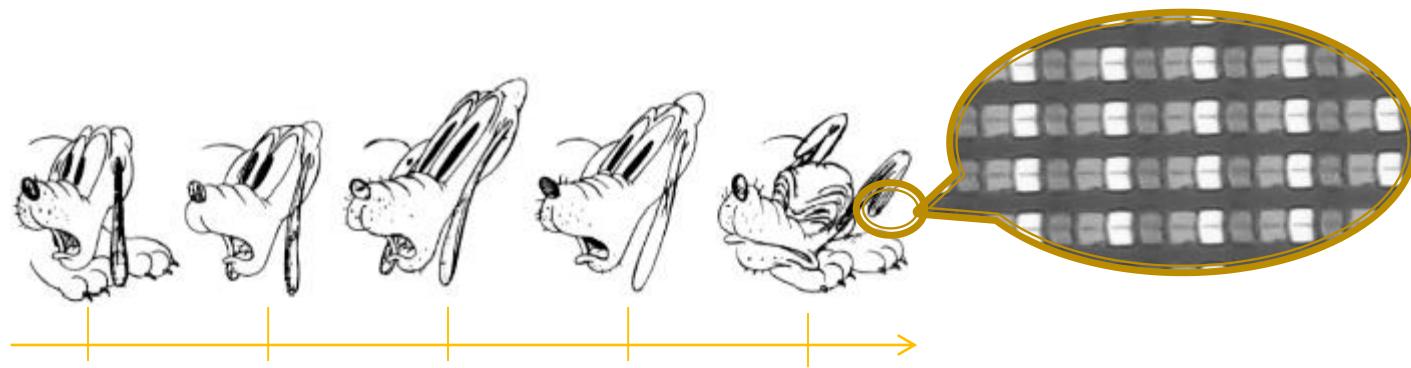
Tín hiệu số

- Rời rạc theo thời gian
- Rời rạc về biên độ



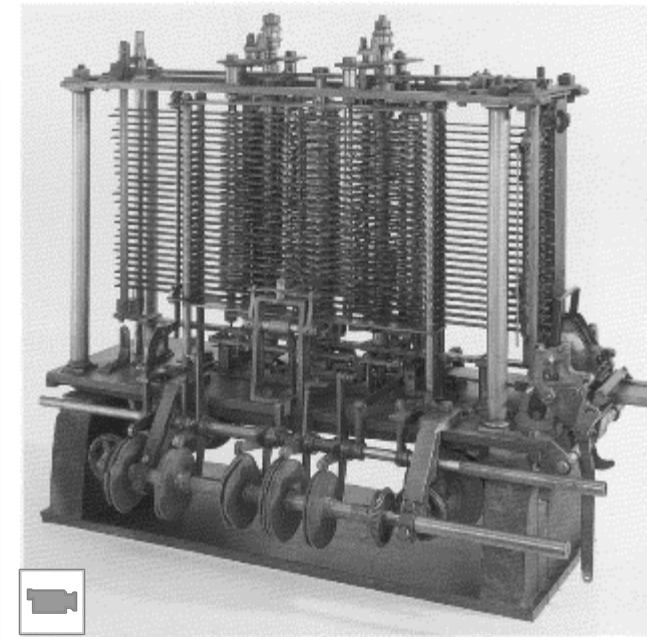
Quá trình số hóa

- Quá trình số hóa sẽ:
 - Rời rạc hóa về thời gian → **thời điểm hữu hạn**
 - Lượng tử hóa giá trị → **bộ giá trị hữu hạn**



Cỗ máy phân tích dữ liệu

- Thiết kế bởi Charles Babbage 1834 – 1871
- Được coi như máy tính số đầu tiên
- Được lắp ráp từ các bánh răng cơ khí, trong đó, mỗi bánh răng đại diện cho một giá trị rời rạc (0-9)
- Babbage mất trước máy được hoàn thành



Điện tử số: Các giá trị nhị phân

- **Các giá trị rời rạc:**
 - 1's và 0's
 - 1, TRUE, HIGH
 - 0, FALSE, LOW
- Mạch số sử dụng **2 mức điện áp**, đại diện cho 2 giá trị 1 và 0.
- **Bit**: viết tắt của **Binary digit**

Hệ đếm

Số thập phân

cột 1, x1, đơn vị
cột 2, x10, chục
cột 3, x100, trăm
cột 4, x1000, nghìn

$$5374_{10} = 5 \times 10^3 + 3 \times 10^2 + 7 \times 10^1 + 4 \times 10^0$$

năm nghìn ba trăm bảy mươi bốn

Số nhị phân

cột 1, x1,
cột 2, x2,
cột 3, x4,
cột 4, x8,

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10}$$

Lũy thừa của 2

$$2^0 = 1$$

$$2^1 = 2$$

$$2^2 = 4$$

$$2^3 = 8$$

$$2^4 = 16$$

$$2^5 = 32$$

$$2^6 = 64$$

$$2^7 = 128$$

$$2^8 = 256$$

$$2^9 = 512$$

$$2^{10} = 1024$$

$$2^{11} = 2048$$

$$2^{12} = 4096$$

$$2^{13} = 8192$$

$$2^{14} = 16384$$

$$2^{15} = 32768$$

Nên thuộc lòng tới 2^{12}

Chuyển đổi hệ 2 \leftrightarrow hệ 10

- Chuyển đổi từ hệ nhị phân \rightarrow hệ thập phân:
 - Biểu diễn số 10011_2 sang hệ thập phân
 - $1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 19_{10}$
- Chuyển đổi từ hệ thập phân \rightarrow hệ nhị phân:
 - Biểu diễn số 47_{10} sang hệ nhị phân
 - $1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 101111_2$

Phạm vi biểu diễn

• Với N chữ số thập phân

- Có thể có bao nhiêu giá trị? 10^N
- Phạm vi biểu diễn? $[0, 10^N - 1]$
- Ví dụ: với 3 chữ số thập phân:
 - $10^3 = 1000$ giá trị có thể có
 - Phạm vi: $[0, 999]$

• Với N chữ số nhị phân

- Có thể có bao nhiêu giá trị? 2^N
- Phạm vi biểu diễn? $[0, 2^N - 1]$
- Ví dụ: với 3 chữ số thập phân:
 - $2^3 = 8$ giá trị có thể có
 - Phạm vi: $[0, 7] = [000_2 \text{ to } 111_2]$

Hệ mươi sáu (Hexa)

Hệ mươi
sáu dùng
để viết gọn
số nhị phân

Số Hexa	Số thập phân	Số nhị phân
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Chuyển đổi hệ 16 \leftrightarrow hệ 10

- Chuyển đổi từ hệ mười sáu \rightarrow hệ thập phân:
 - Biểu diễn số **4AF** sang hệ thập phân
 - $4 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 = 19_{10}$
- Chuyển đổi từ hệ mười sáu \rightarrow hệ nhị phân :
 - Biểu diễn số **4AF** sang hệ nhị phân
 - **0100 1010 1111**₂

Bits, Bytes, Nibbles...

- Bits

10010110

most significant bit least significant bit

- Bytes & Nibbles

byte
10010110
nibble

- Bytes

CEBF9AD7

most significant byte least significant byte

Các lũy thừa 2

- $2^{10} = 1 \text{ kilo} \quad \approx 1000 \text{ (1024)}$
- $2^{20} = 1 \text{ mega} \quad \approx 1 \text{ million (1,048,576)}$
- $2^{30} = 1 \text{ giga} \quad \approx 1 \text{ billion (1,073,741,824)}$

Phép cộng

- Số thập phân

$$\begin{array}{r} & \text{11} \leftarrow \text{carries} \\ 3734 & \\ + 5168 & \\ \hline 8902 & \end{array}$$

- Số nhị phân

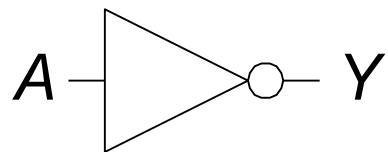
$$\begin{array}{r} & \text{11} \leftarrow \text{carries} \\ 1011 & \\ + 0011 & \\ \hline 1110 & \end{array}$$

Cổng Logic

- **Các cổng logic cơ bản:**
 - đảo (NOT), AND, OR, NAND, NOR, etc.
- **Một đầu vào:**
 - cổng NOT, đệm
- **Hai đầu vào:**
 - AND, OR, XOR, NAND, NOR, XNOR
- **Nhiều đầu vào:**
 - Thông số fan-in: số lượng đầu vào của một cổng logic.
 - Ví dụ: cổng AND có 3 đầu vào → fan-in bằng 3

Các cỗng logic 1 đầu vào

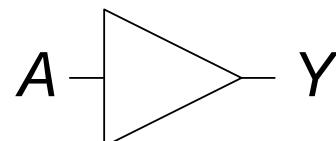
NOT



$$Y = \overline{A}$$

A	Y
0	1
1	0

BUF (Đệm)

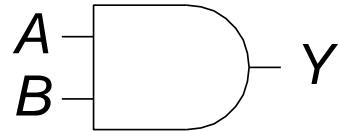


$$Y = A$$

A	Y
0	0
1	1

Các cỗng logic 2 đầu vào

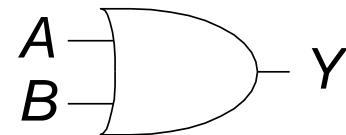
AND



$$Y = AB$$

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

OR

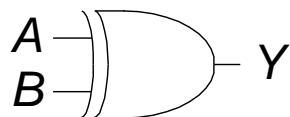


$$Y = A + B$$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Các cỗng logic 2 đầu vào

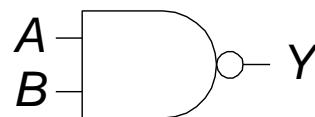
XOR



$$Y = A \oplus B$$

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

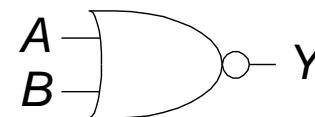
NAND



$$Y = \overline{AB}$$

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

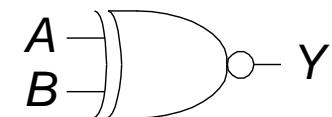
NOR



$$Y = \overline{A + B}$$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

XNOR



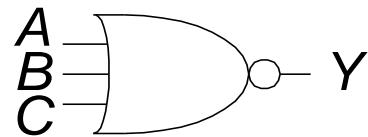
$$Y = \overline{A \oplus B}$$

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1



Các cổng logic nhiều đầu vào

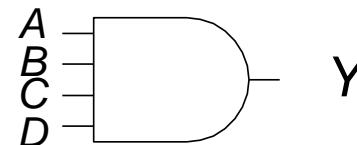
NOR3



$$Y = \overline{A+B+C}$$

A	B	C	Y
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

AND4



$$Y = ABCD$$

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Cổng XOR nhiều đầu vào: dùng để tính parity

Mức logic

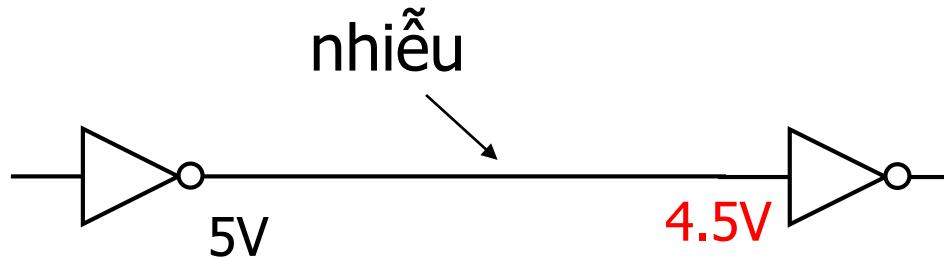
- Các mức điện áp rời rạc khác nhau, đại diện cho logic 0 và logic 1.
- Ví dụ:
 - 0 = *ground* (GND, đất) hoặc 0 volts
 - 1 = V_{DD} hoặc 5 volts
- Vậy còn điện áp 4.99 volts? Ứng với 0 hay 1?
- Điện áp 3.2 volts thì sao?

Mức logic

- Dải điện áp của logic 0 và dải điện áp của logic 1
- Điện áp nằm ngoài dải điện áp của 0 và 1 được coi là do *noise* (*nhiễu*) gây ra.
- *Nhiễu là gì?*

Nhiễu (noise) là gì

- **Là tất cả những gì làm suy hao tín hiệu**
 - Chẳng hạn trở kháng, nhiễu nguồn cấp, nhiễu xuyên kenh giữa các sợi dây cùng cắp...
- **Ví dụ:** một cổng (driver) có điện áp đầu ra là 5 V nhưng sau khi truyền tới một cổng khác (receiver) thì điện áp chỉ còn 4.5V do điện trở dây dẫn.



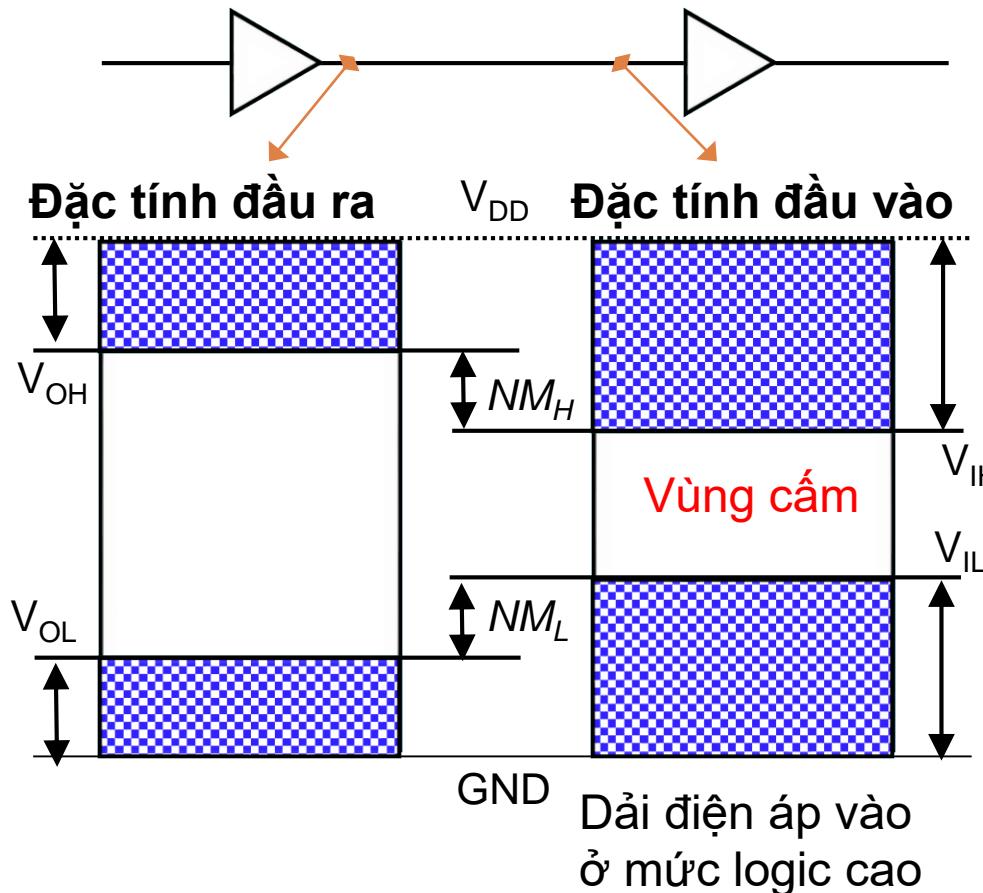
Yêu cầu bắt buộc

- Với các đầu vào có mức logic hợp lệ, mọi phần tử, cấu trúc trong mạch xử lý phải đưa kết quả hợp lệ ở đầu ra.
- Mỗi mức logic 0 hoặc 1 tương ứng với một dải điện áp **có giới hạn**

Các mức logic

Dải điện áp ra
ở mức logic cao

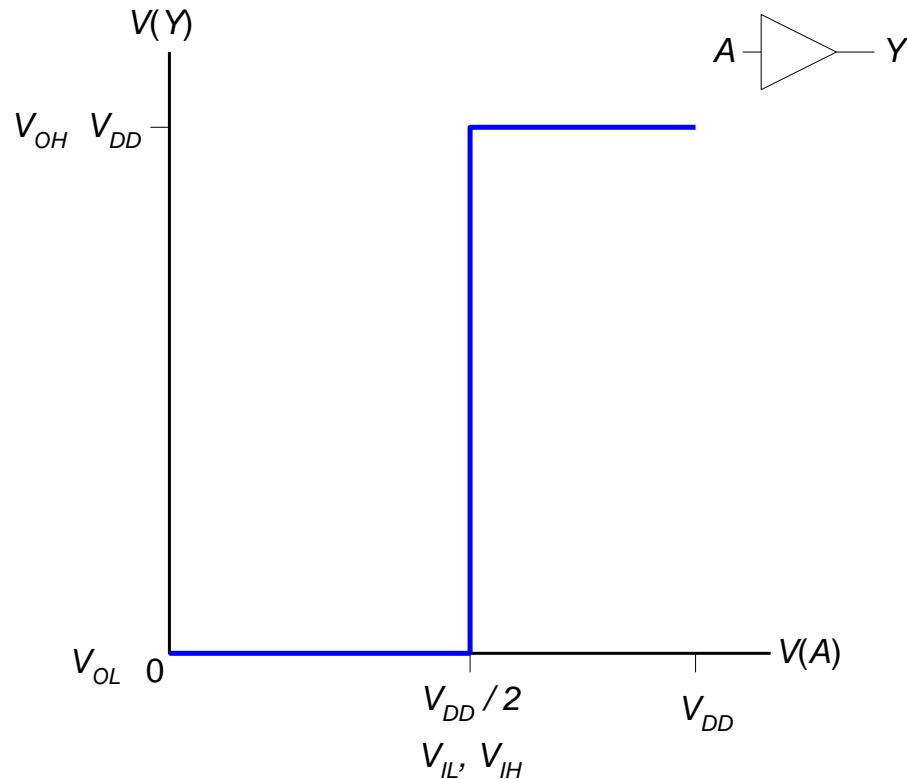
Dải điện áp ra
ở mức logic thấp



Biên chống nhiễu
(Noise Margin)

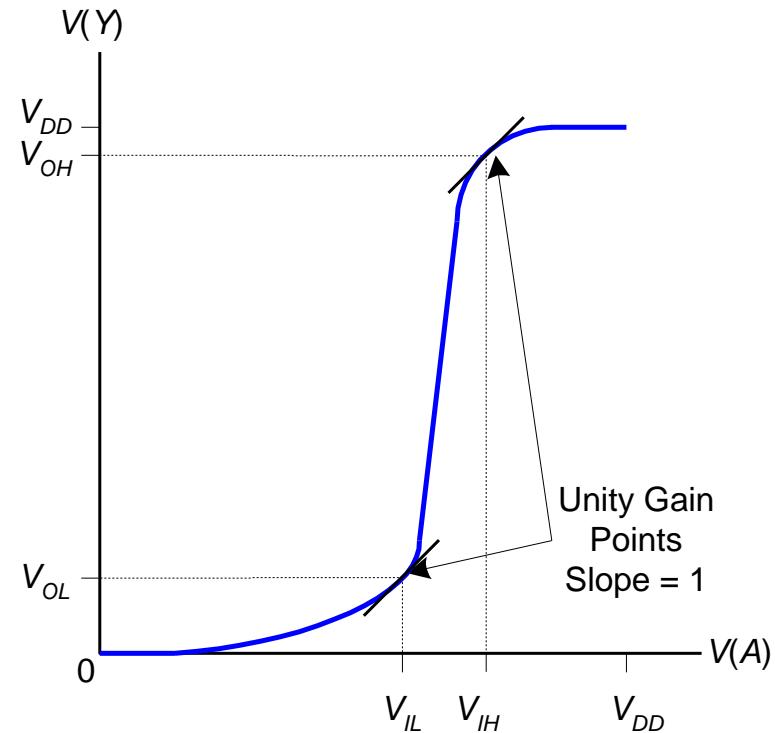
Đặc tính biến đổi DC

Bộ đệm lý tưởng:



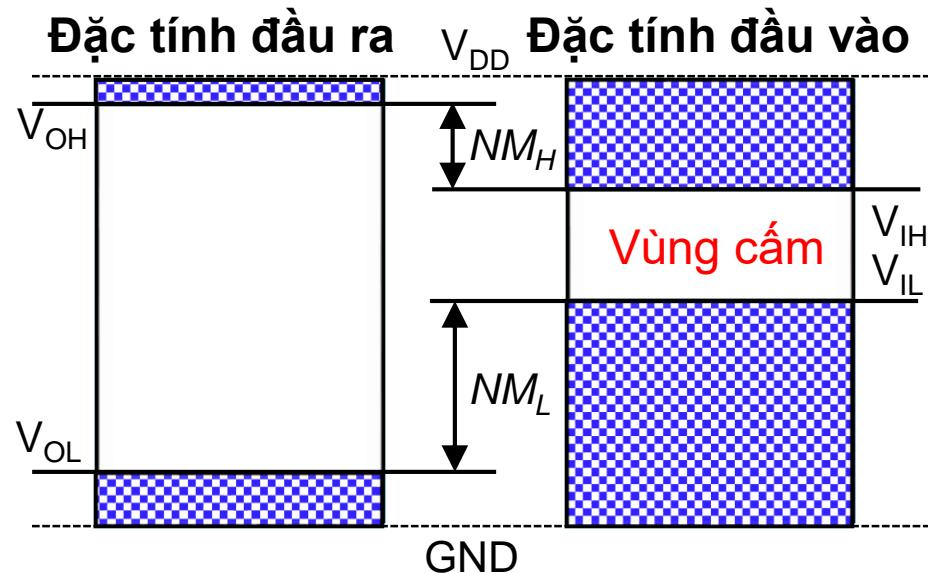
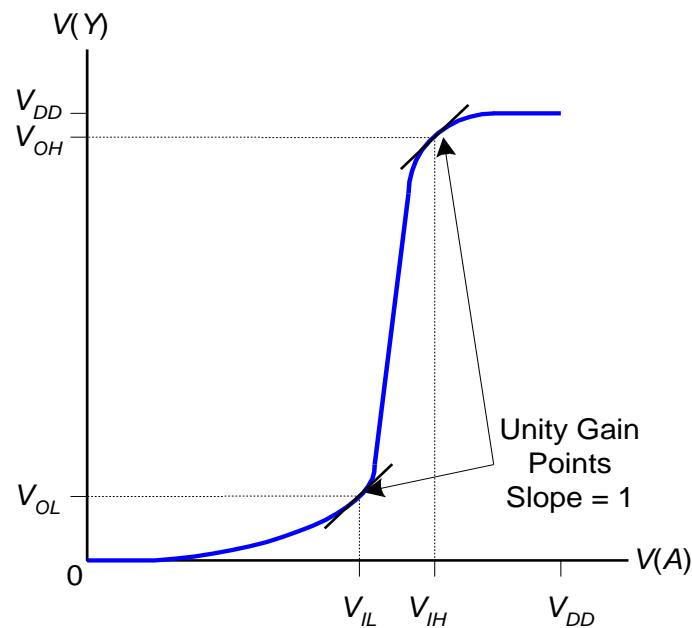
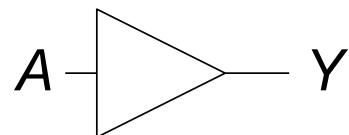
$$NM_H = NM_L = V_{DD}/2$$

Bộ đệm thực tế:



$$NM_H, NM_L < V_{DD}/2$$

Đặc tính biến đổi DC



Phạm vi giá trị của điện áp V_{DD}

- Năm 1970's và 1980's, $V_{DD} = 5$ V
- V_{DD} có xu hướng giảm dần
 - Transistors sẽ ít nóng hơn, tốc độ cao hơn
 - Tiết kiệm năng lượng
- 3.3 V, 2.5 V, 1.8 V, 1.5 V, 1.2 V, 1.0 V, ..
- Hết sức tránh đưa các nguồn điện áp khác nhau vào chip.

Chip hoạt động được nhờ chứa 1 loại khói thần kỳ.

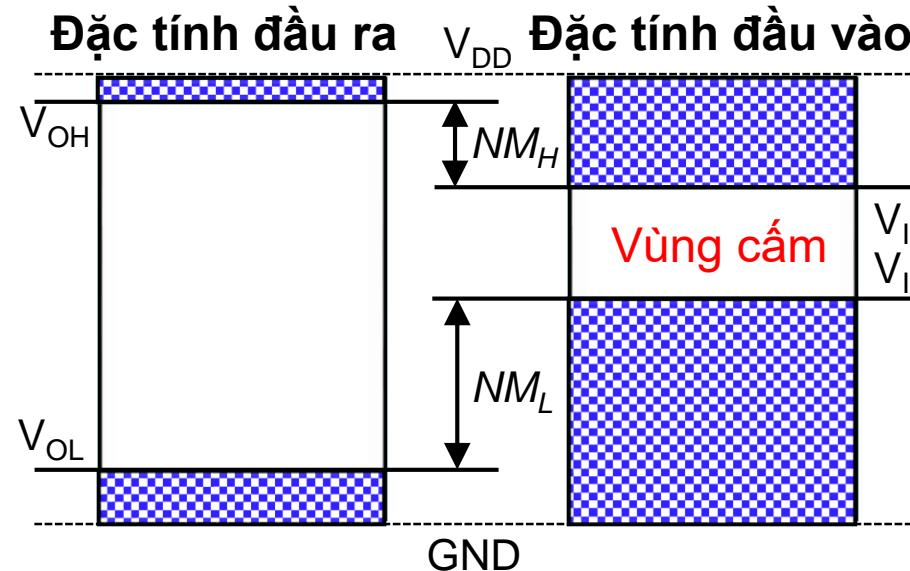
Chứng minh:

- *nếu nhìn thấy khói bay ra khỏi chip, chip sẽ không hoạt động được nữa*



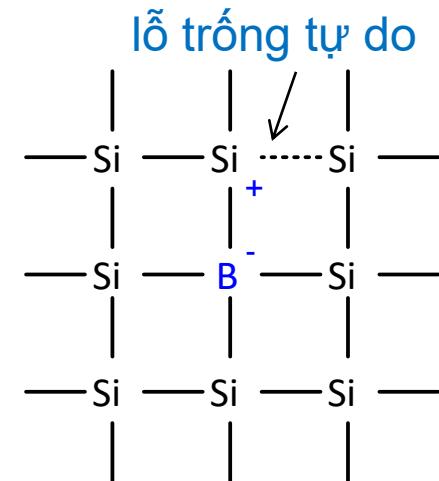
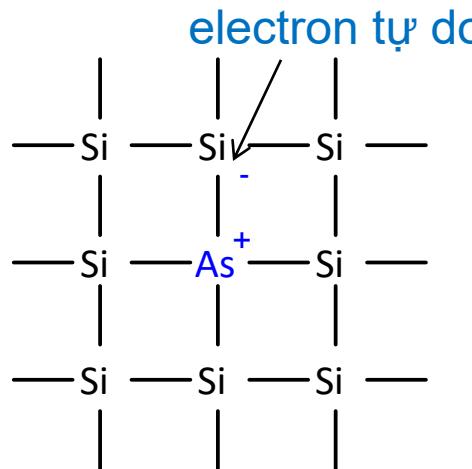
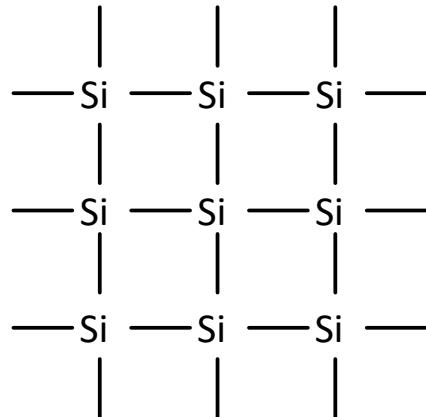
Ví dụ về các mức điện áp

Logic Family	V_{DD}	V_{OL}	V_{OH}	V_{IL}	V_{IH}
TTL	5 (4.75 - 5.25)	0.4	2.4	0.8	2.0
CMOS	5 (4.5 - 6)	0.33	3.84	1.35	3.15
LV TTL	3.3 (3 - 3.6)	0.4	2.4	0.8	2.0
LVC MOS	3.3 (3 - 3.6)	0.36	2.7	0.9	1.8



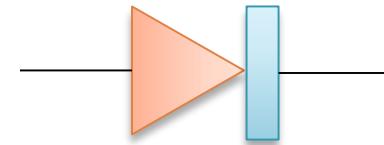
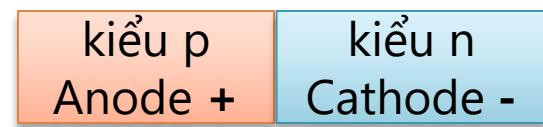
Silicon

- Silicon là một chất bán dẫn
- Silicon nguyên chất dẫn điện kém (không có phần tử tự do)
- Silicon có tạp chất, dẫn điện tốt (có phần tử tự do)
 - kiểu n: chứa phần tử mang điện âm tự do, electron
 - kiểu p: chứa phần tử mang điện dương tự do, lỗ trống



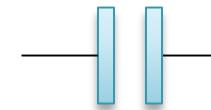
Diode

- Diode có 2 cực, làm từ 2 lớp bán dẫn n và p
 - Cực theo kiểu p: gọi là anode, dương cực +
 - Cực theo kiểu n: gọi là cathode, âm cực -
- $V_{anode} > V_{cathode}$ (điện áp thuận):
 - có dòng điện từ anode → cathode
- $V_{anode} < V_{cathode}$ (điện áp nghịch):
 - không có dòng điện
- Diode chỉ cho dòng điện chạy theo một chiều



Tụ điện

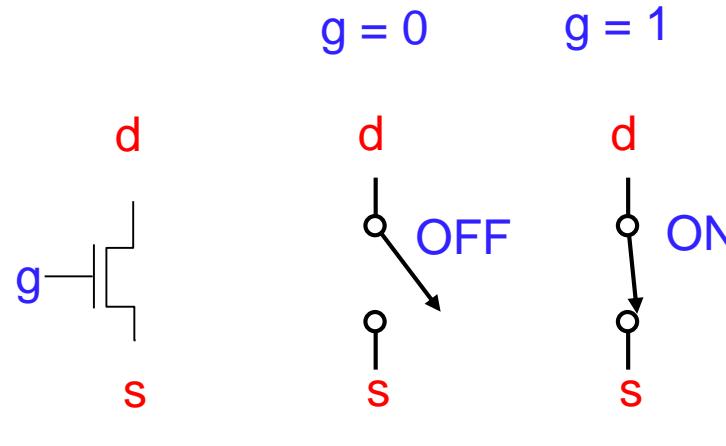
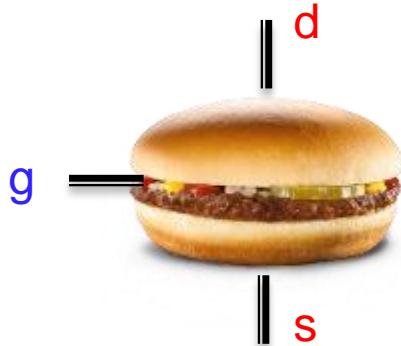
- Tụ, capacitor, gồm 2 bề mặt dẫn điện, ngăn cách bởi điện môi cách điện ở giữa.
- Khi một bề mặt có điện áp V , bề mặt đó sẽ được nạp điện tích Q , và bề mặt còn lại có điện tích $-Q$



- Điện dung $C = Q / V$
 - Tỷ lệ thuận với kích thước bề mặt dẫn điện
 - Tỷ lệ nghịch với khoảng cách giữa 2 bề mặt dẫn điện
- Chức năng: quá trình nạp/xả điện trên các bề mặt sẽ tiêu tốn điện năng và làm chậm.
 - Điện dung càng lớn thì mạch càng chậm, tiêu thụ điện năng càng nhiều

Transistor

- Transistor có 3 cực, làm từ 3 lớp bán dẫn n và p
- Giống các chuyển mạch bằng điện áp có 3 cực
 - 2 cực được nối với nhau hay không, phụ thuộc vào điện áp trên cực thứ 3.
 - cực d và s sẽ nối thông nhau (ON) khi g bằng 1
- Các cổng logic được xây dựng từ các transistor

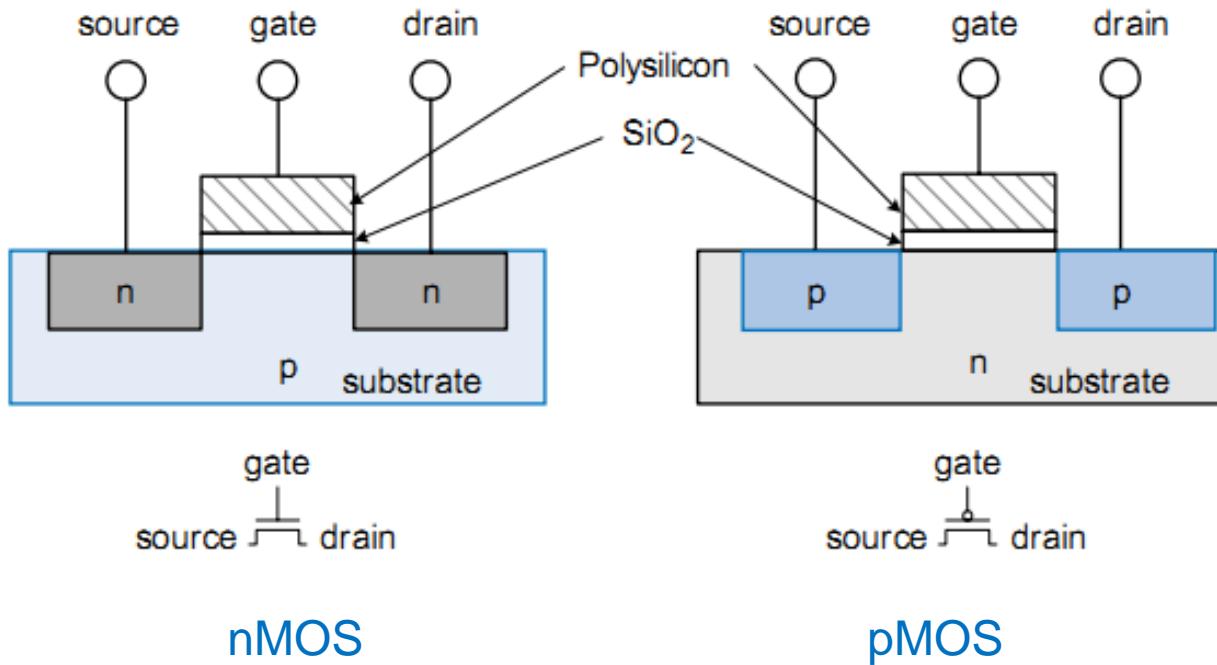


d: drain, máng
g: gate, cổng
s: source, nguồn

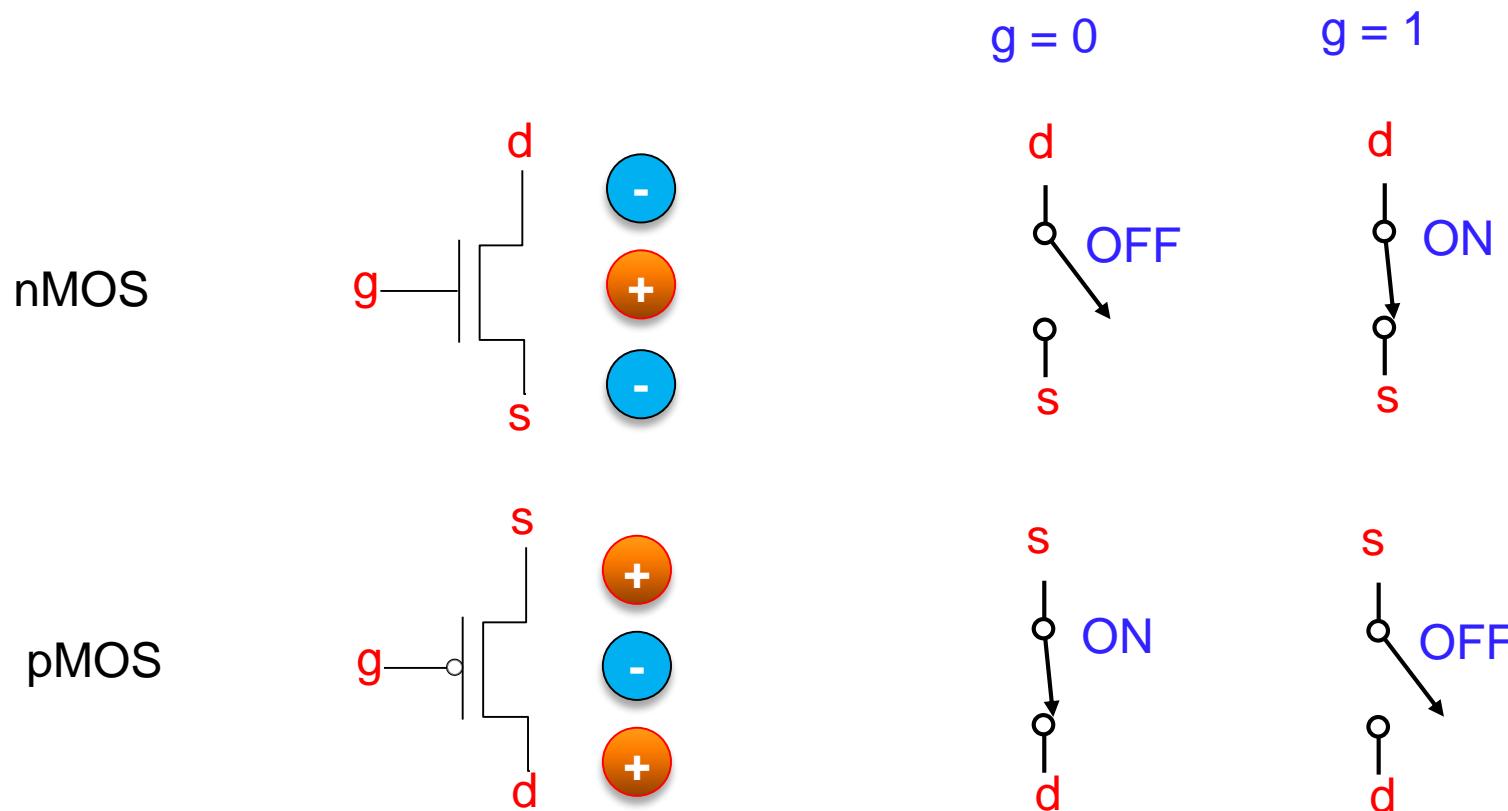
Transistor MOS

- Transistor có cực oxide kim loại (MOS)

- Cực Polysilicon (sử dụng kim loại)
- Cách ly bằng Oxide Silicon
- Silicon tệp chất



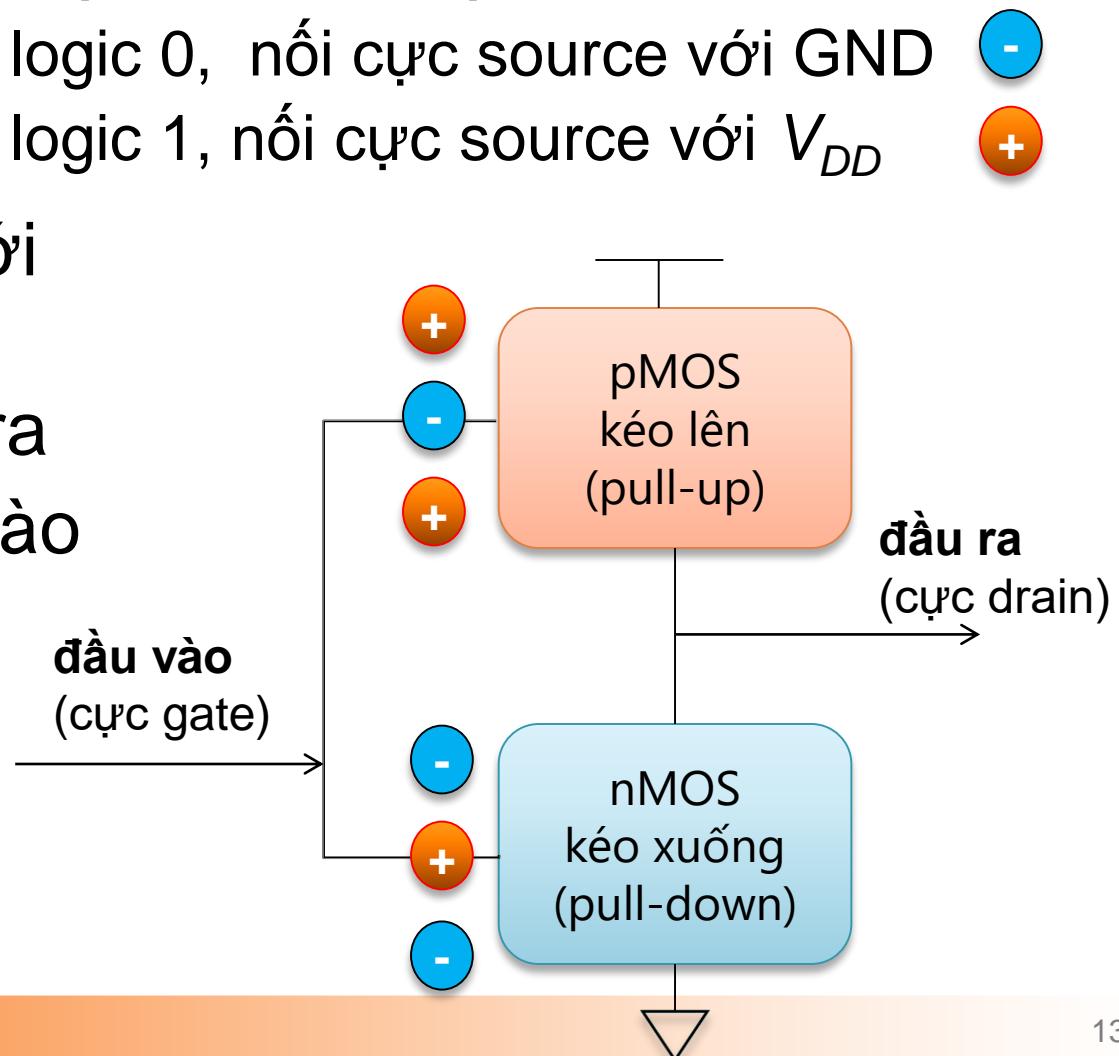
Hoạt động của transistor MOS



Transistor chỉ có 2 trạng thái, ON/OFF. Vậy logic 0/1 ở đâu?

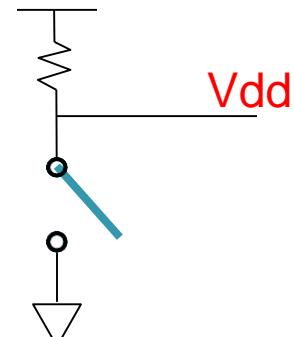
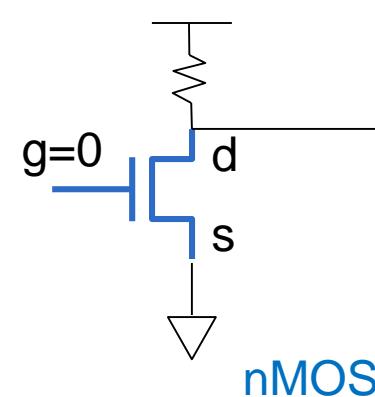
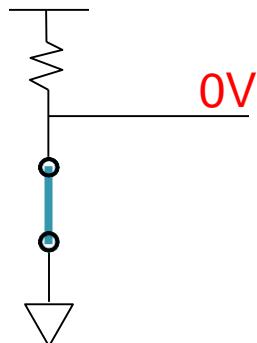
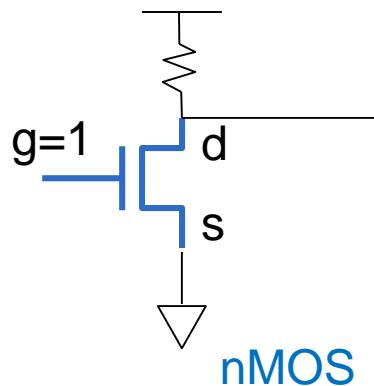
Hoạt động của transistor MOS

- Nguồn điện luôn được nối với cực source
 - **nMOS**: để tạo ra logic 0, nối cực source với GND
 - **pMOS**: để tạo ra logic 1, nối cực source với V_{DD}
- Cực source nối với nguồn cấp
- Cực drain là đầu ra
- Cực gate là đầu vào



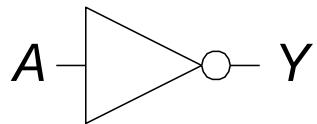
Hoạt động của transistor

- Cách khác để tạo logic 0/1 bằng transistor như hình dưới. **So sánh?**
- Một logic được tạo ra khi transistor ở trạng thái ON
→ drain nối thông với source
- Logic còn lại được tạo ra khi transistor ở trạng thái OFF
→ drain không thông với source



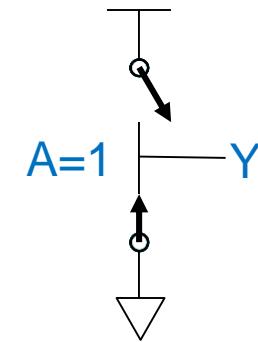
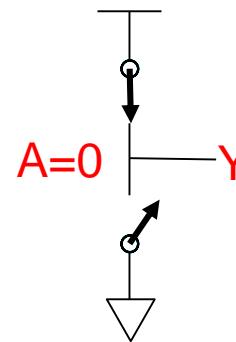
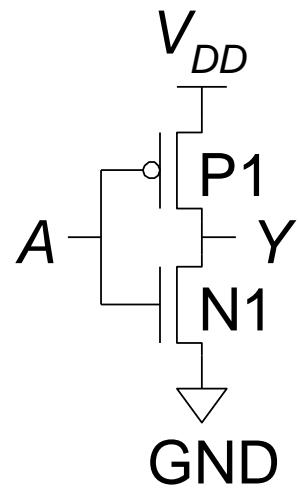
CMOS: Cổng NOT

NOT

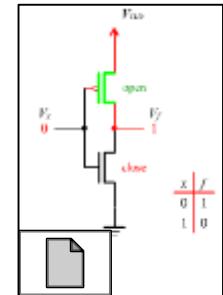


$$Y = \overline{A}$$

A	Y
0	1
1	0

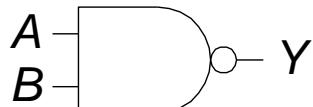


A	P1	N1	Y
0	ON	OFF	1
1	OFF	ON	0



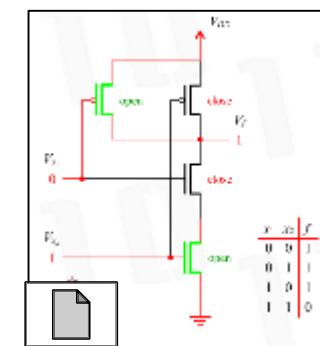
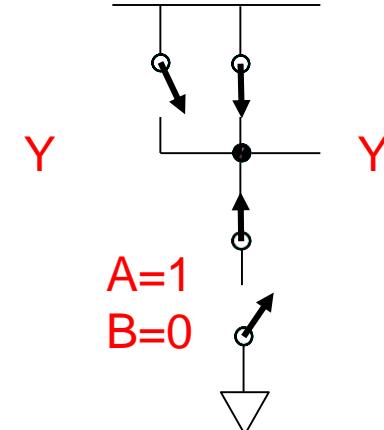
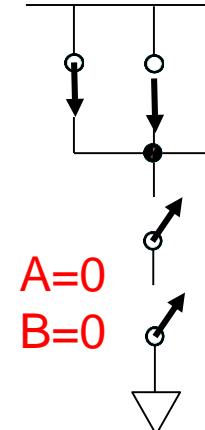
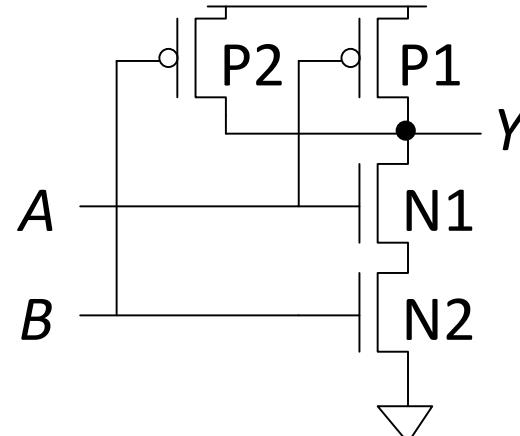
CMOS: Cổng NAND

NAND



$$Y = \overline{AB}$$

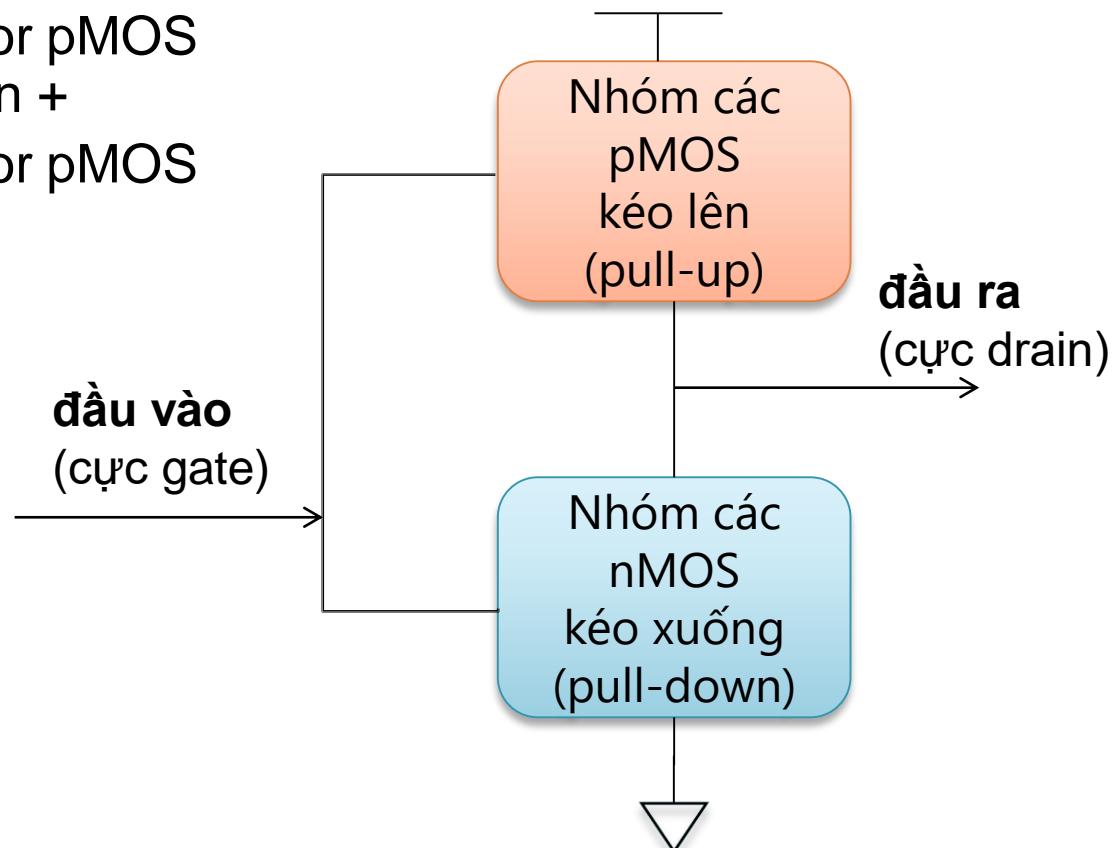
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0



Cấu trúc chung của cổng CMOS

- Cấu trúc chung của các cổng CMOS gồm 2 nhóm transistor

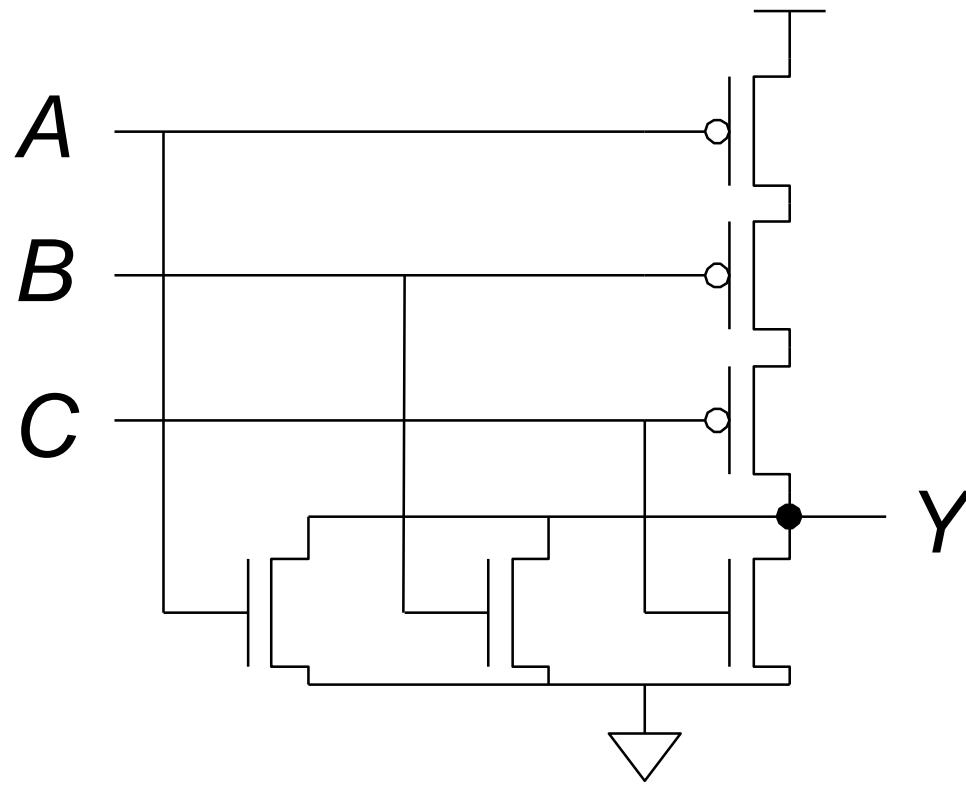
- Nhóm các transistor pMOS được nối với nguồn +
- Nhóm các transistor nMOS được nối với GND



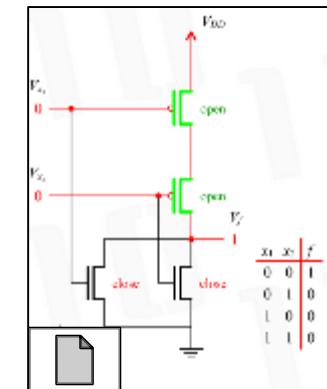
CMOS: Cổng NOR

- Cổng NOR 2 đầu vào được thiết kế bởi 4 transistor.
→ Cổng NOR 3 đầu vào được thiết kế bởi mấy transistor?

CMOS: Cổng NOR3

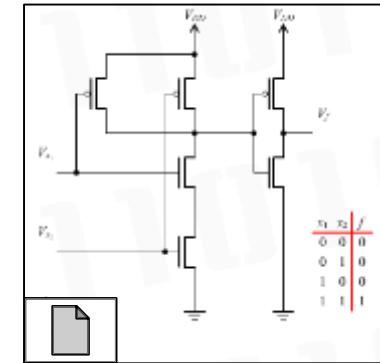
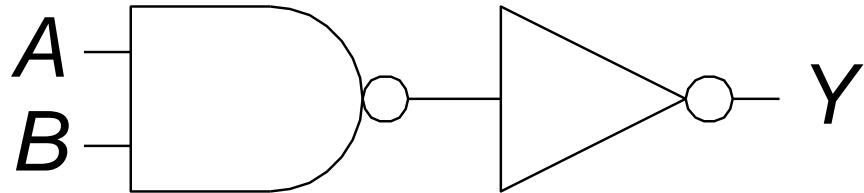


- Cổng NOR N-đầu vào sử dụng:
 - N nMOS mắc song song
 - N pMOS mắc nối tiếp



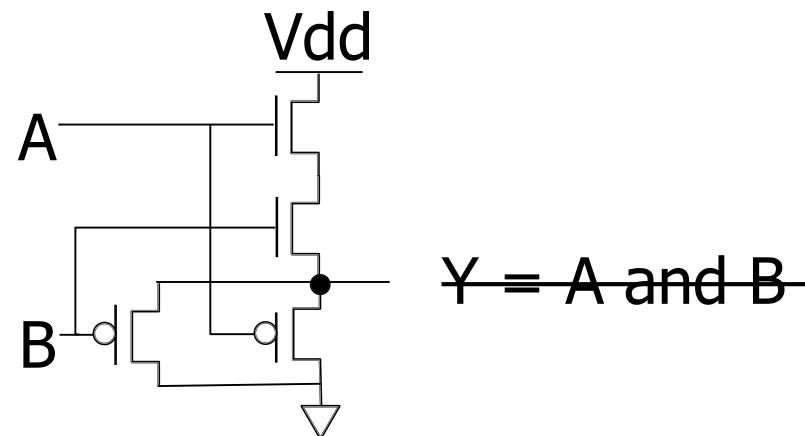
CMOS: Cổng AND2

- Cổng AND được thiết kế từ NAND



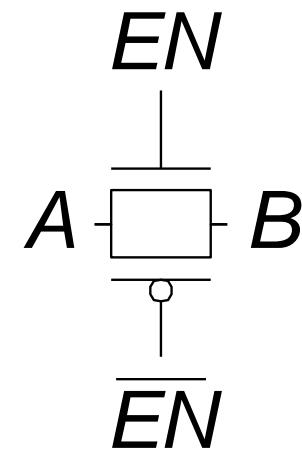
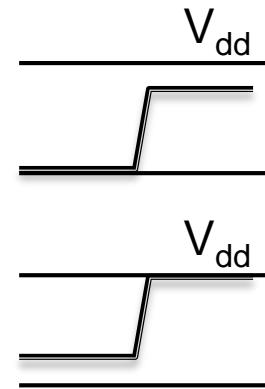
- Tại sao không thiết kế AND như hình vẽ

Giải thích:



Cổng truyền

- nMOS có logic 1 yếu (1's poorly), tức là điện áp ở drain ở mức $0 \sim V_{dd} - V_t$
- pMOS có logic 0 yếu (0's poorly), tức là điện áp ở drain ở mức $V_t \sim V_{dd}$
- Sử dụng cổng truyền sẽ tốt hơn so với việc đóng/ngắt mạch bằng 1 transistor.
 - truyền đi mức logic 0 và 1 khỏe
- Khi $EN = 1$, switch ở trạng thái ON:
 - $EN = 0$ và A kết nối thông với B
- Khi $EN = 0$, switch ở trạng thái OFF:
 - A không nối với B. B ở trạng thái HiZ



Pseudo-nMOS Logic

- Cổng NOR N-đầu vào sử dụng
 - N nMOS transistor mắc song song
 - N pMOS transistor mắc nối tiếp (**critical time**)
 - Đặc tính thời gian
 - Thời gian truyền trên các transistor mắc nối tiếp lâu hơn so với song song, vì trỏ kháng nối tiếp lớn hơn trỏ kháng song song.
 - pMOS có tốc độ chậm hơn nMOS, vì thời gian để lõi trống di chuyển lâu hơn so với electron.
- cần giảm số lượng pMOS

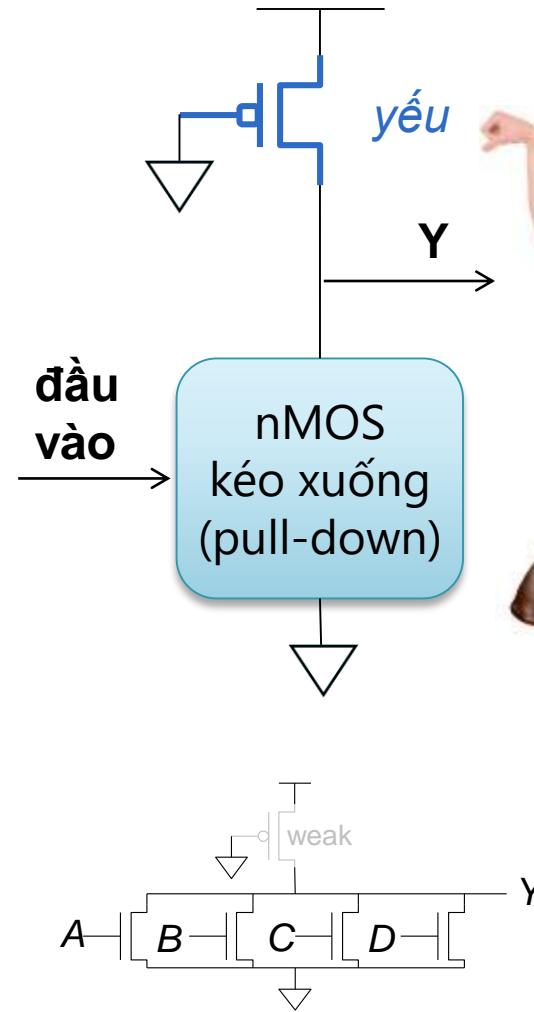
Pseudo-nMOS Logic với cỗng NOR

Nguyên lý

- Thay tất cả các pMOS mắc nối tiếp bằng một pMOS luôn ở trạng thái ON → tạo ra logic 1 yếu
- pMOS này được gọi là kéo-lên yếu (weak pull-up)

Giải thích

- pMOS sẽ tạo ra logic 1 yếu trên Y.
- Chỉ cần một trong số các nMOS nối ở trạng thái ON, công suất lớn hơn sẽ kéo điện áp Y về logic 0



Công suất tiêu thụ

- Công suất = năng lượng trên mỗi đơn vị thời gian
 - Công suất tiêu thụ động: để nạp điện dung khi tín hiệu thay đổi 0 \leftrightarrow 1
 - Công suất tiêu thụ tĩnh: năng lượng sử dụng ngay cả khi tín hiệu không có thay đổi và hệ thống ở trạng thái idle.

$$P = P_{\text{động}} + P_{\text{tĩnh}}$$

Công suất tiêu thụ động

- Công suất để nạp điện dung cho cổng logic
 - Năng lượng cần thiết để nạp cho điện dung C, có điện áp V_{DD} là CV_{DD}^2
 - Giả định mạch làm việc với tần số f → transistor sẽ đảo giá trị (0 \leftrightarrow 1) cũng theo tần số đó
 - Từ sẽ được nạp $f/2$ lần mỗi giây (việc xả điện để từ 1 về 0 không tiêu thụ năng lượng)
- Công suất tiêu thụ động:

$$P_{động} = \frac{1}{2} \cdot C \cdot V_{DD}^2 \cdot f$$

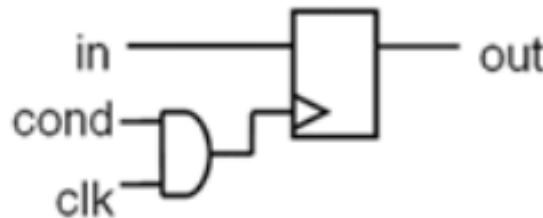
Công suất tiêu thụ tĩnh

- Là công suất tiêu thụ khi cổng logic không hoạt động
- Gây ra bởi dòng điện rò rỉ, I_{DD}
- Công suất tiêu thụ tĩnh:

$$P_{tĩnh} = I_{DD}V_{DD}$$

Giảm tiêu thụ điện trong Intel Core i7

- Khi CPU ở các trạng thái Idle trong chu kì bus, xung nhịp CLK bị ngắt để giảm $P_{động}$
- Giải pháp: sử dụng mạch AND với tín hiệu CLK



Bài tập công suất tiêu thụ 1

- Xác định công suất tiêu thụ của một máy tính cầm tay không dây có thông số như sau:
 - $V_{DD} = 1.2 \text{ V}$
 - $C = 20 \text{ nF}$
 - $f = 1 \text{ GHz}$
 - $I_{DD} = 20 \text{ mA}$

Bài tập công suất tiêu thụ 2

Một điện thoại di động có pin 6Watt-giờ (Wh) và hoạt động với điện áp 1.2V. Giả định rằng điện thoại hoạt động ở tần số 300MHz, và tổng điện dung trung bình của các chuyển mạch trong mọi thời điểm là $10nF$ (10^{-8} Farad). Khi vận hành, điện thoại phát một nguồn năng lượng 3W qua ăng ten. Khi điện thoại không sử dụng, công suất tiêu thụ động gần như bằng 0. Ngoài ra, còn có dòng rò rì 40mA bất kể điện thoại có sử dụng hay không.

Hãy cho biết thời gian hoạt động mà pin có thể cung cấp cho điện thoại trong 2 tình huống sau

- a) Điện thoại ở trạng thái không hoạt động.
- b) Điện thoại ở trạng thái phát sóng liên tục.

Bài tập công suất tiêu thụ 2

- Theo đề bài $V_{DD} = 1.2 \text{ V}$; $f = 300 \cdot 10^6 \text{ Hz}$; $C = 10^{-8} \text{ F}$; $P_{\text{động}} = 3 \text{ W}$; $I_{DD} = 0.04 \text{ A}$
- Thay vào công thức ta có
 - $P_{\text{không hoạt động}} = P_{\text{tĩnh}} = I_{DD} \cdot V_{DD} = 0.04 \times 1.2 = 0.048 \text{ (W)}$
 - $P_{\text{phát sóng}} = 3 + P_{\text{động}} + P_{\text{tĩnh}}$
 $= 3 + \frac{1}{2} \times C \cdot V_{DD}^2 \cdot f + I_{DD} \cdot V_{DD}$
 $= 3 + \frac{1}{2} \times 10^{-8} \times 1.2^2 \times 300 \times 10^6 + 0.048$
 $= 3 + 2.16 + 0.048 = 5.208 \text{ (W)}$
- Vậy:
 - Thời gian pin để điện thoại không hoạt động = $6 / 0.048 = 125 \text{ (giờ)}$
 - Thời gian pin để điện thoại phát sóng liên tục = $6 / 5.208 = 1.15 \text{ (giờ)}$



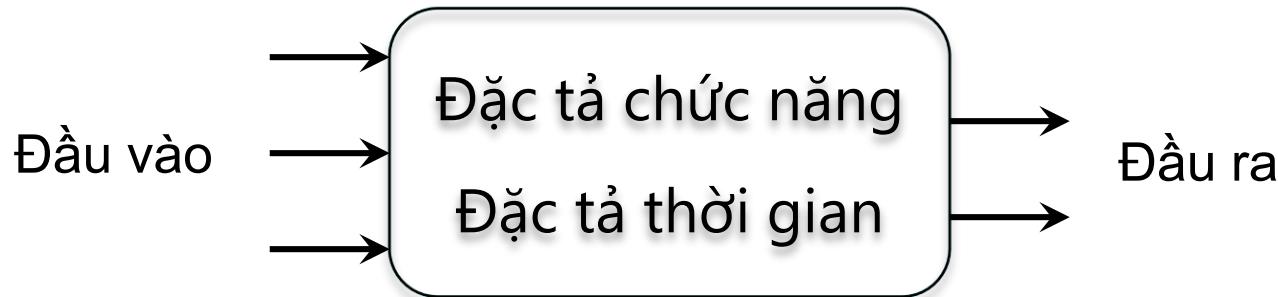
LOGIC SỐ

- Đại số Bool
- Từ logic tới cổng
- Các mệnh đề
- Hazard

Khái niệm mạch logic

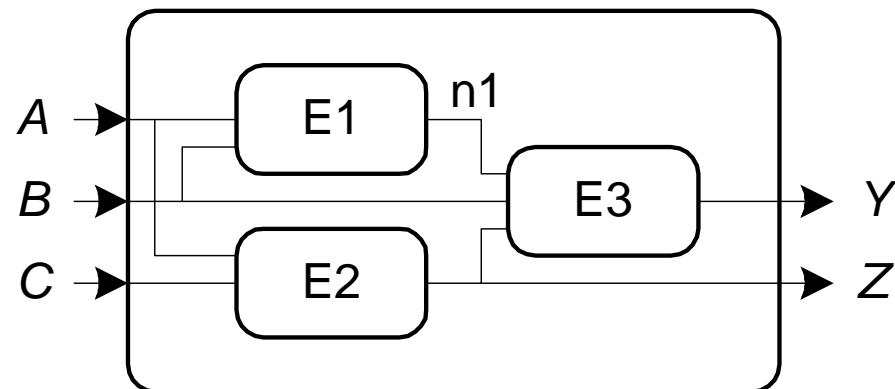
- Mạch logic bao gồm

- Một hoặc nhiều đầu vào logic 0 và 1
- Một hoặc nhiều đầu ra logic 0 và 1
- Đặc tả chức năng: mối quan hệ giữa đầu vào và ra
- Đặc tả thời gian: độ trễ giữa đầu vào và ra



Ví dụ về mạch logic

- Node (dây nối)
 - Đầu vào: A, B, C
 - Đầu ra: Y, Z
 - Bên trong: $n1$
- Các phần tử của mạch
 - $E1, E2, E3$
 - Có mỗi phần tử lại có thể là một mạch nào đó



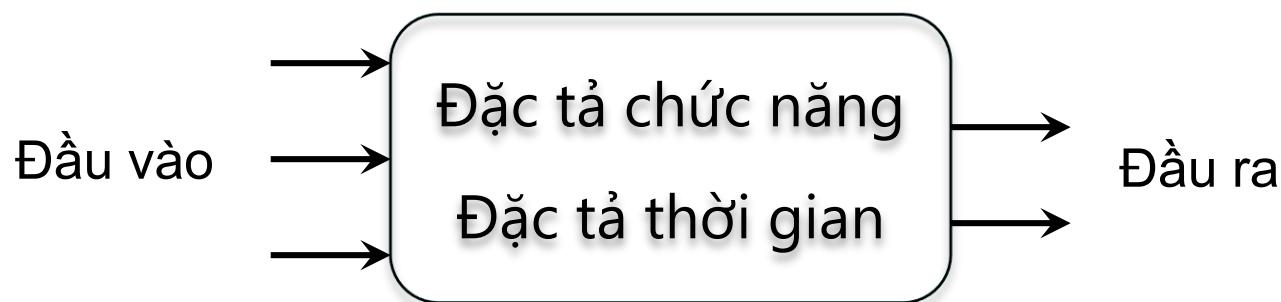
Phân loại mạch logic

Mạch tổ hợp (Combinational Logic)

- Không nhớ
- Đầu ra phụ thuộc vào đầu vào tại thời điểm hiện tại

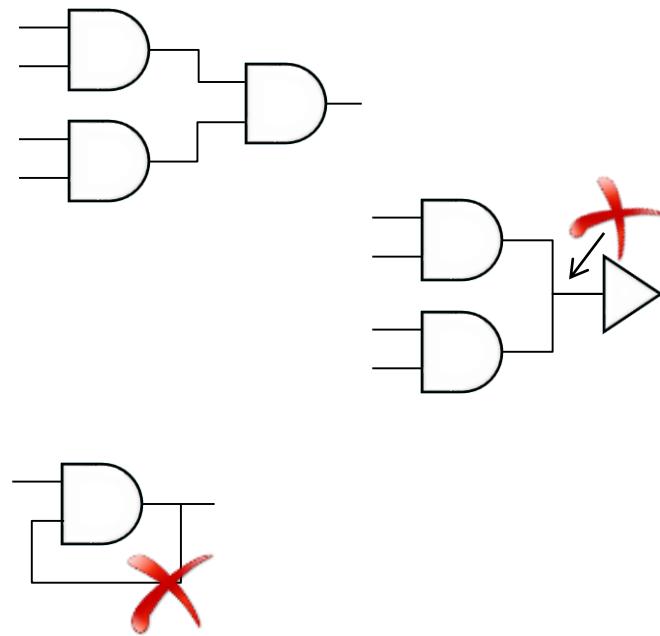
Mạch dây (Sequential Logic)

- Có nhớ
- Đầu ra phụ thuộc vào đầu vào tại thời điểm hiện tại và quá khứ



Các luật của mạch tổ hợp

1. Mỗi phần tử cũng là mạch tổ hợp
2. Mỗi node có thể là 1 đầu vào hoặc nối chính xác với 1 đầu ra.
3. Không chứa đường dây dẫn vòng, truy hồi
Lưu ý: Các qui định mang tính tương đối, không chặt
4. Tín hiệu ra không phục thuộc vào tín hiệu vào trong quá khứ, không nhớ (bắt buộc)

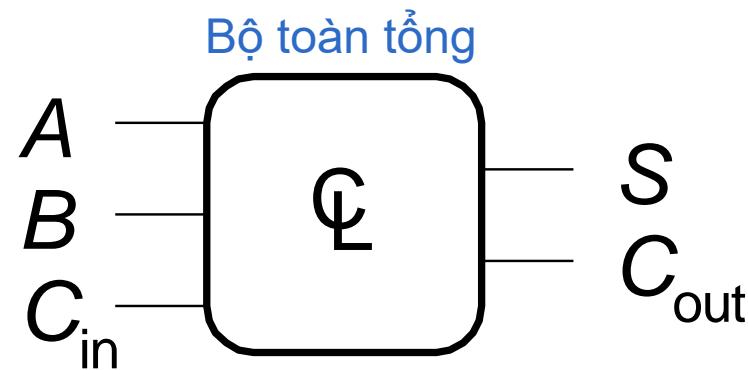


Một số định nghĩa

- Đảo (Complement): tên biến có gạch ngang ở trên
 - Ví dụ: \bar{A} , \bar{B}
- Biến logic (literal): là biến, hoặc đảo của biến
 - Ví dụ: \bar{A} , \bar{B} , B
- Dạng tuyễn (minterm): tích của tất cả các biến
 - Ví dụ: $\bar{A}BC$, $A\bar{B}C$, ABC
- Dạng hội (maxterm): tổng của tất cả các biến
 - Ví dụ: $(\bar{A}+B+C)$,
 $(A+\bar{B}+C)$, $(A+B+C)$
- Tuyễn chính qui (SOP): tổng của các tuyễn
 - Ví dụ: $\bar{A}BC + A\bar{B}C + ABC$
- Hội chính qui (POS): tích của các hội
 - Ví dụ: $(\bar{A}+B+C)$.
 $(A+\bar{B}+C)$. $(A+B+C)$

Biểu thức đại số bool

- Các biến logic đầu ra là hàm số của các biến đầu vào
- Ví dụ:
 - $S = F(A, B, C_{in})$
 - $C_{out} = F(A, B, Cin)$



$$\begin{aligned}S &= A \oplus B \oplus C_{in} \\C_{out} &= AB + AC_{in} + BC_{in}\end{aligned}$$

Ví dụ biểu thức bool

- SV đi ra cửa hàng để mua nem chua
 - Kết luận rằng SV có ăn nem chua (E)
 - Nếu cửa hàng mở cửa (A)
 - Nếu cửa hàng đã hết nem chua (C)
- Viết bảng sự thật để xác định bạn có ăn được nem chua hay không?



A	C	E
0	0	0
0	1	0
1	0	1
1	1	0

Ví dụ biểu thức bool dạng chuẩn

- Tuyễn chính qui (SOP): tổng các tích

A	C	E	tuyễn
0	0	0	$\bar{A} \cdot \bar{C}$
0	1	0	$\bar{A} \cdot C$
1	0	1	$A \cdot \bar{C}$
1	1	0	$A \cdot C$

$$\begin{aligned}Y &= A\bar{C} \\&= \Sigma(2)\end{aligned}$$

- Hội chính qui (POS): tích các tổng

A	C	E	hội
0	0	0	$A + C$
0	1	0	$A + \bar{C}$
1	0	1	$\bar{A} + C$
1	1	0	$\bar{A} + \bar{C}$

$$\begin{aligned}Y &= (A + C) \cdot (A + \bar{C}) \cdot (\bar{A} + C) \\&= \prod(0,1,3)\end{aligned}$$

Đại số Bool

- Các mệnh đề và định lý về tối thiểu hóa biểu thức
- Giống như biểu thức đại số thông thường, nhưng đơn giản hơn: các biến chỉ có 2 giá trị (0 và 1).
- Tính đối ngẫu trong các mệnh đề và định lý
 - AND và OR
 - 0 và 1
 - Ví dụ: $(A + B).C = A.C + B.C \Leftrightarrow (A.B) + C = (A + C).(B + C)$

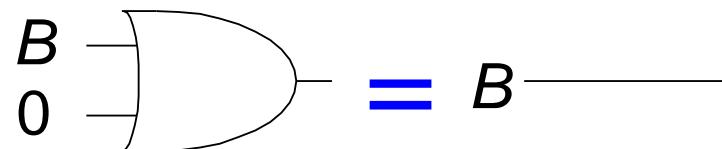
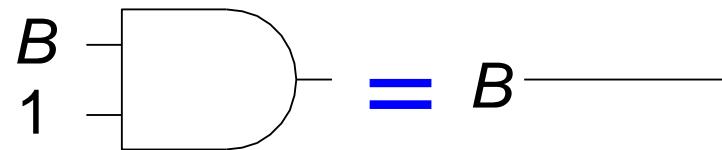
Các mệnh đề cơ bản

	Axiom		Dual	Name
A1	$B = 0 \text{ if } B \neq 1$	A1'	$B = 1 \text{ if } B \neq 0$	Binary field
A2	$\bar{0} = 1$	A2'	$\bar{1} = 0$	NOT
A3	$0 \bullet 0 = 0$	A3'	$1 + 1 = 1$	AND/OR
A4	$1 \bullet 1 = 1$	A4'	$0 + 0 = 0$	AND/OR
A5	$0 \bullet 1 = 1 \bullet 0 = 0$	A5'	$1 + 0 = 0 + 1 = 1$	AND/OR

	Theorem		Dual	Name
T1	$B \bullet 1 = B$	T1'	$B + 0 = B$	Identity
T2	$B \bullet 0 = 0$	T2'	$B + 1 = 1$	Null Element
T3	$B \bullet B = B$	T3'	$B + B = B$	Idempotency
T4		$\bar{\bar{B}} = B$		Involution
T5	$B \bullet \bar{B} = 0$	T5'	$B + \bar{B} = 1$	Complements

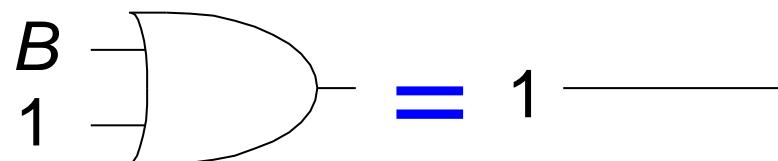
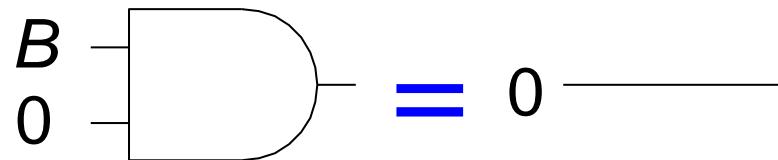
T1: Phần tử trung tính

- $B \bullet 1 = B$
- $B + 0 = B$



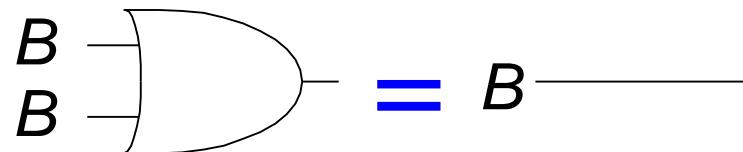
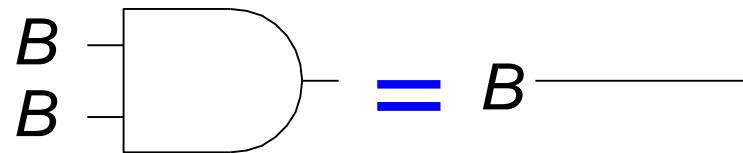
T2: Phần tử xóa/thiết lập

- $B \bullet 0 = 0$
- $B + 1 = 1$



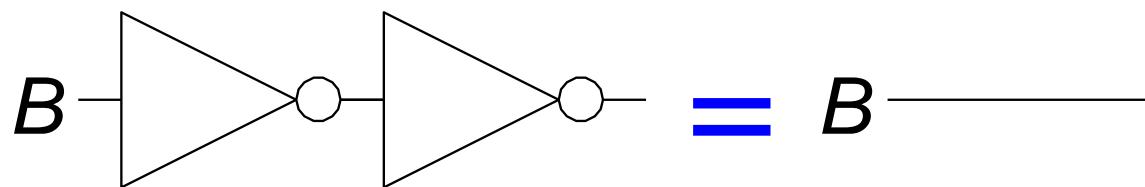
T3: Trùng lắp

- $B \bullet B = B$
- $B + B = B$



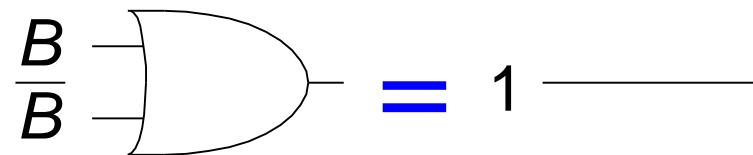
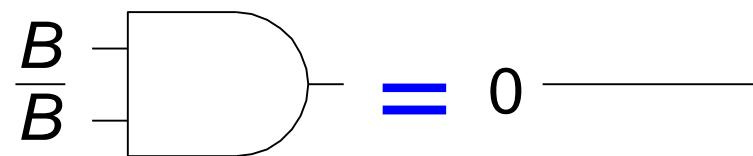
T4: Đồng nhất

- $\overline{\overline{B}} = B$



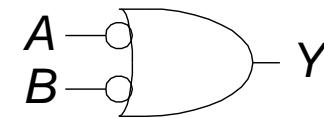
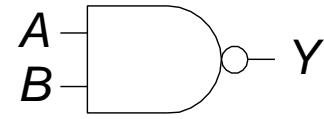
T5: Phần bù

- $B \bullet \bar{B} = 0$
- $B + \bar{B} = 1$

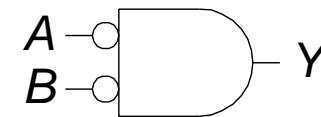
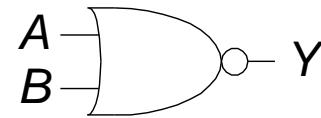


Định lý DeMorgan

- $Y = \overline{A \cdot B} = \overline{A} + \overline{B}$



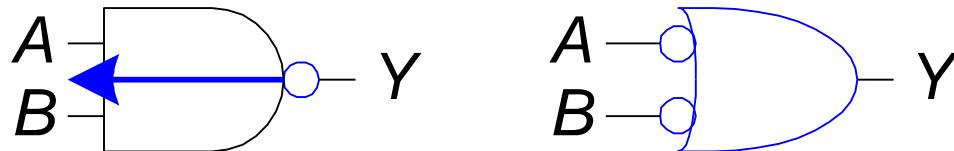
- $Y = \overline{A + B} = \overline{A} \cdot \overline{B}$



Quy tắc “đẩy bóng”

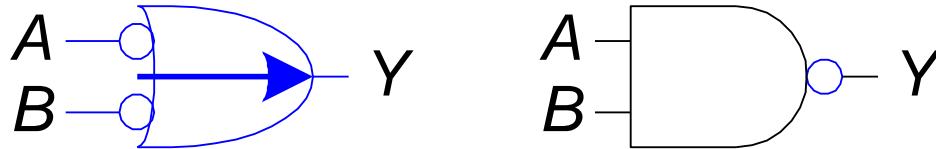
Đẩy bóng về sau:

- Thay đổi cổng \rightarrow cổng đối ngẫu
- Thêm bóng vào tất cả đầu vào



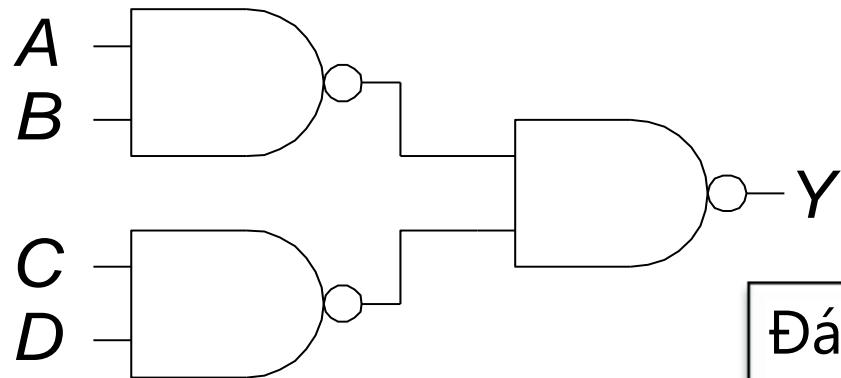
Đẩy bóng lên trước:

- Thay đổi cổng \rightarrow cổng đối ngẫu
- Thêm bóng vào đầu ra



Quy tắc “đẩy bóng”

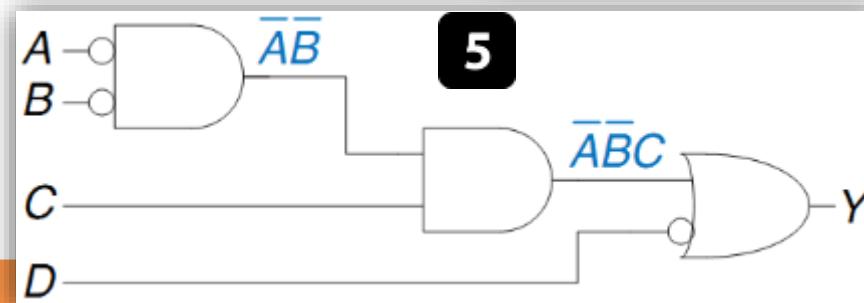
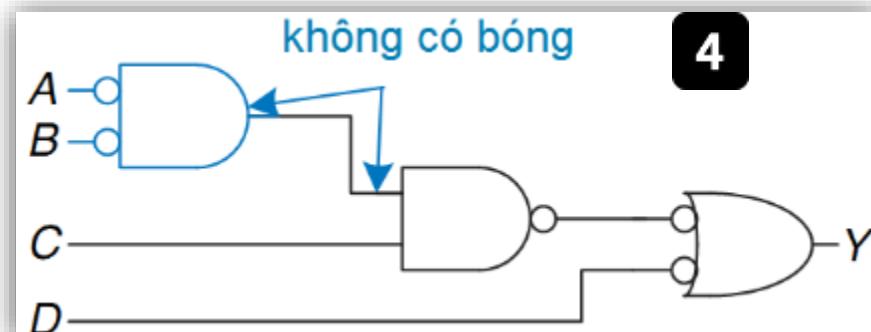
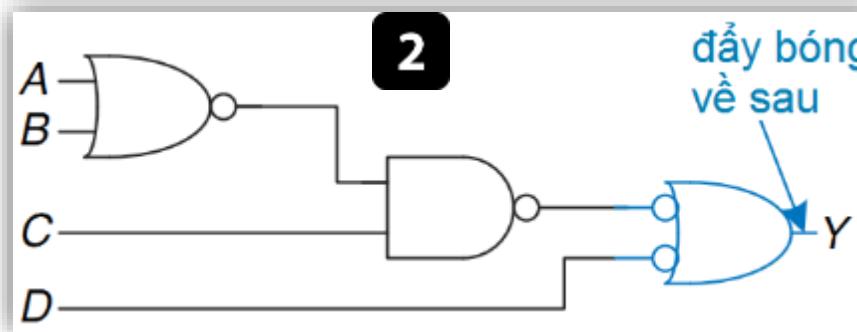
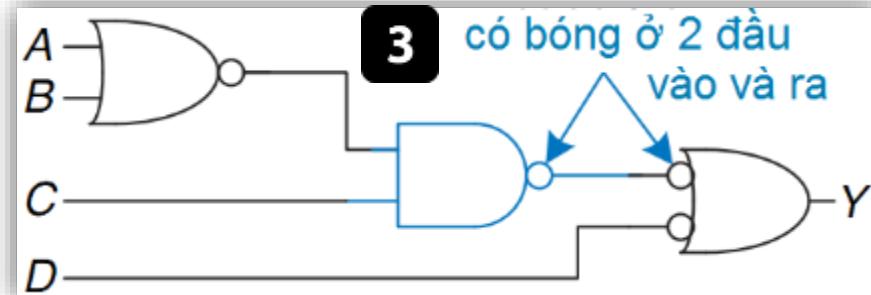
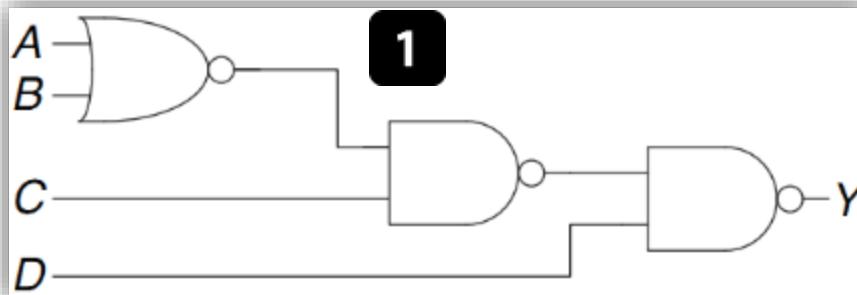
- Biểu thức đại số Boolean ứng với mạch dưới đây là gì?



Đáp án

Quy tắc “đẩy bóng”

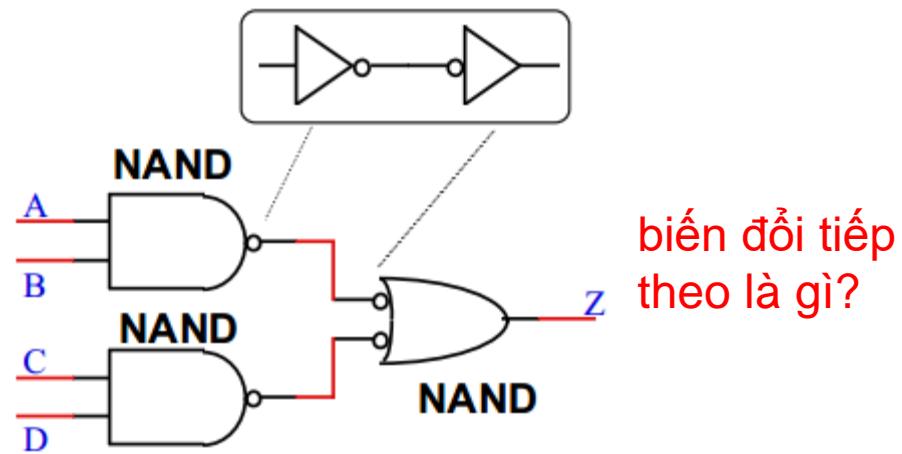
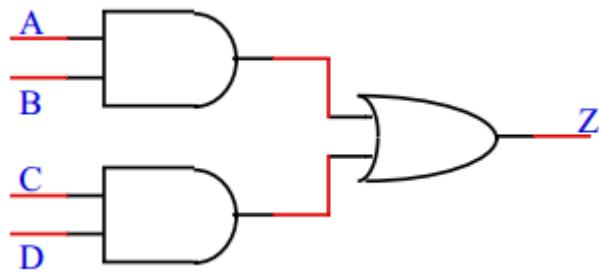
- Áp dụng quy tắc để biến đổi mạch



Mạch tổ hợp làm từ NAND/NOR

- Mặc dù các hàm logic ở dạng chính tắc: chuẩn tuyễn, chuẩn hội, sử dụng cỗng AND, OR → thực tế triển khai, hàm xây dựng từ cỗng NAND, NOR
 - Cỗng AND: xây dựng từ NAND và NOT
 - Cỗng OR: xây dựng từ NOR và NOT
 - NAND (hoặc NOR) là tập đầy đủ, có thể tạo nên mọi hàm logic.
- Để triển khai, các hàm dựa trên AND, OR được biến đổi thành NAND, NOR, dựa trên quy tắc “đẩy bóng”

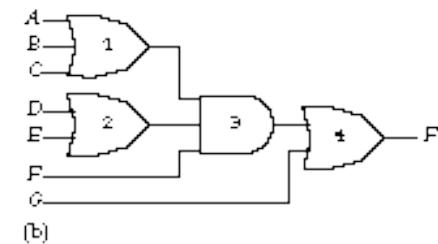
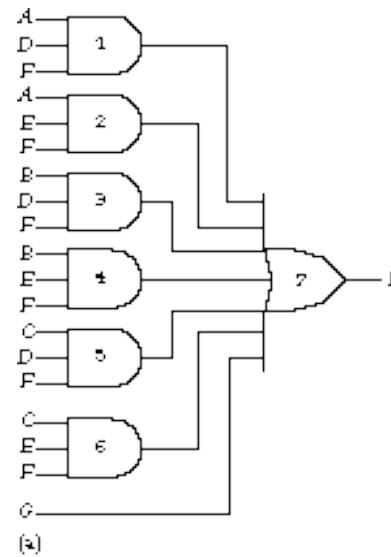
Mạch tổ hợp làm từ NAND/NOR



biến đổi tiếp
theo là gì?

Mạch tổ hợp đa tầng

- Biểu thức Bool ở dạng SOP gọi là *logic 2 tầng*, vì gồm 1 tầng các cổng AND, và 1 tầng cổng OR
 - Ví dụ: $ADF + AEF + BDF + BEF + CDF + CEF + G$
→ Tốn nhiều phần cứng
- Người ta thường thiết kế ở dạng không chính tắc, đa tầng
 - Ví dụ trên tương đương với $(A + B + C)(D + E) F + G$
→ Càng nhiều tầng thì trễ lớn
→ Khó xử lý hazard



Các phát sinh với mạch đa tầng

• Hazard/Không đều: hiện tượng nhảy kết quả

- Xảy ra khi đường dẫn tín hiệu có độ trễ khác nhau
- Nhiều
- Rủi ro nếu sử dụng giá trị khi đầu ra chưa đạt trạng thái ổn định.
- Rủi ro nếu mạch không đồng bộ

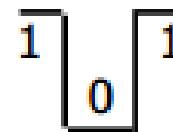
• Giải pháp:

- Thiết kế mạch không có hazard → Rất khó khả thi với mạch đa tầng
- Đợi cho tới khi tín hiệu đầu ra đạt trạng thái ổn định
- Sử dụng mạch đồng bộ

Các kiểu hazard

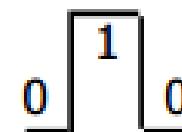
- Hazard tĩnh – logic 1

- Đầu ra nhận giá trị 1
- Trễ gây ra nhảy nhanh về 0



- Hazard tĩnh – logic 0

- Đầu ra nhận giá trị 0
- Trễ gây ra nhảy nhanh về 1



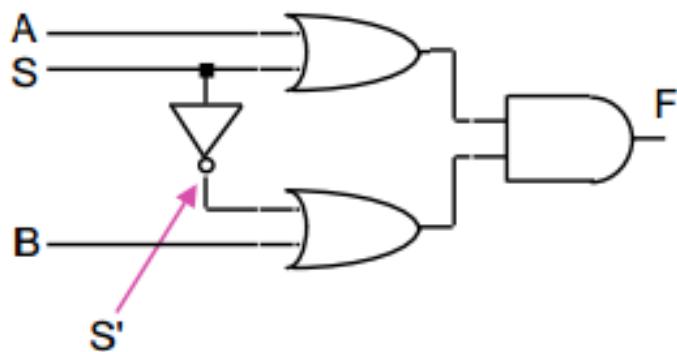
- Hazard động

- Trễ gây ra nhảy giá trị liên tục ở đầu ra

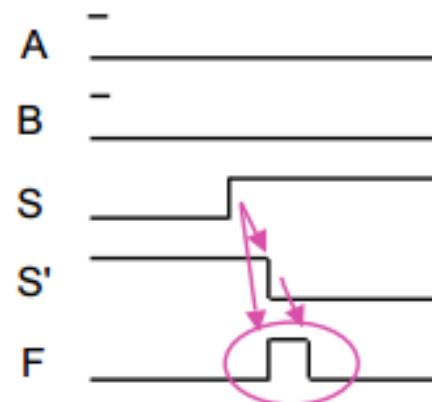


Hazard tĩnh

- Nguyên nhân: khi một biến và đảo của nó tạm thời có cùng một giá trị



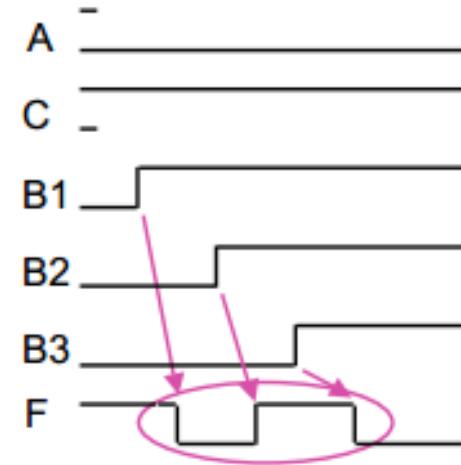
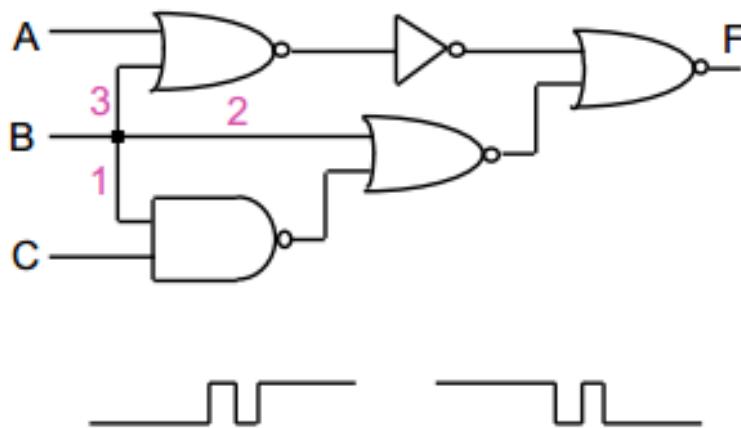
Bộ ghép kênh



Hazard tĩnh – logic 0

Hazard động

- Nguyên nhân: khi một biến có nhiều giá trị tại một thời điểm tạm thời nào đó

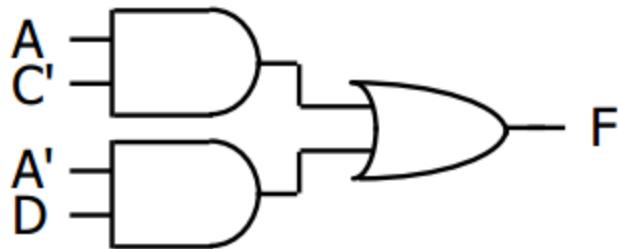


Hazard động

Xem thêm slide “Không đồng đều, glitch”

Loại bỏ Hazard

- Xét trường hợp đơn giản, mạch 2 tầng với 1 bit thay đổi
 - → Hazard xảy ra tại **điểm nối giữa 2 nhóm rìa rạc** trong bảng k-map
 - Ví dụ: mạch dưới đây sẽ có hazard khi giá trị chuyển **1101 -> 0101**



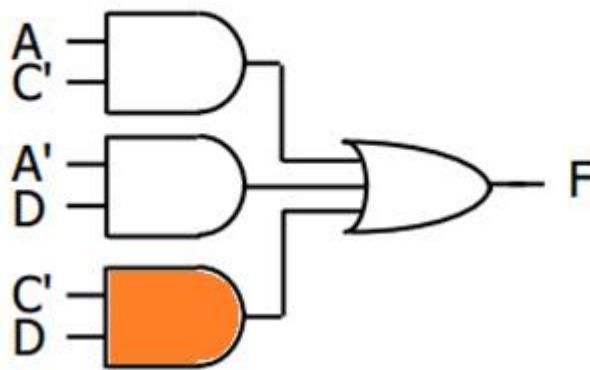
		AB		A	
		00	01	11	10
CD	00	0	0	1	1
	01	1	1	1	1
	11	1	1	0	0
	10	0	0	0	0

The K-map shows four minterms: m0, m1, m3, and m5. The minterms m0 and m1 are grouped by a pink rectangle, while m3 and m5 are grouped by another pink rectangle. This indicates a hazard at the connection between the two groups of minterms.

Loại bỏ Hazard

- Giải pháp: bổ sung thêm nhóm để “che” vào vị trí nối

$$F = AC' + A'D + C'D$$



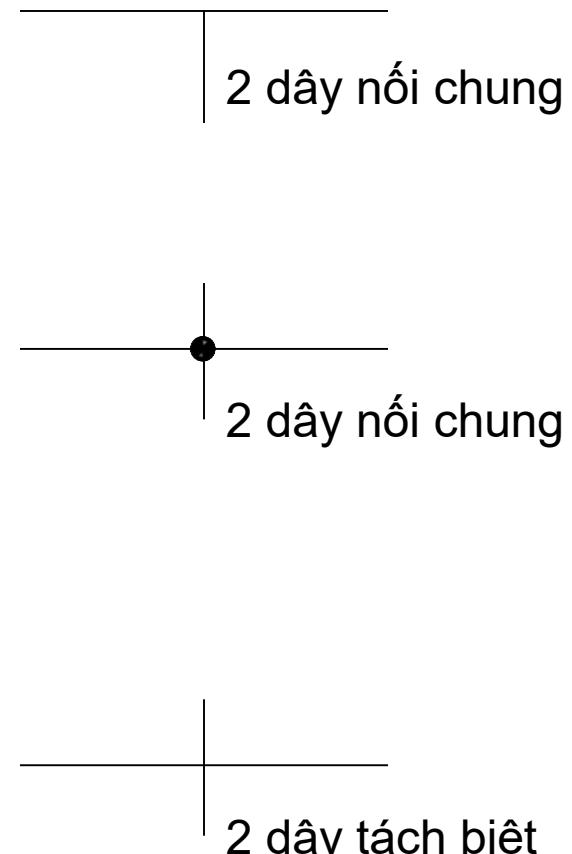
		AB		A	
		00	01	11	10
CD	00	0	0	1	1
	01	1	1	1	1
C	11	1	1	0	0
	10	0	0	0	0

The Karnaugh map shows the function $F = AC' + A'D + C'D$. The variables are labeled A (top), B (bottom), C (left), and D (right). The minterms are: (00,00) = 0, (00,01) = 0, (01,00) = 1, (01,01) = 1, (11,00) = 1, (11,01) = 1, (11,11) = 0, (10,10) = 0. The terms AC' , $A'D$, and $C'D$ are highlighted with red boxes.

- Các công cụ CAD và giả lập là hết sức cần thiết để tính toán và ước lượng hazard

Qui tắc vẽ sơ đồ nguyên lý

- Các dây dẫn luôn nối với nhau tại nút giao chữ T
- Chấm đen ở vị trí giao nhau giữa 2 dây dẫn thể hiện điểm nối chúng với nhau
- Các dây dẫn giao nhau, nhưng không có chấm đen thể hiện chúng tách rời nhau, không nối.



Các trạng thái logic đặc biệt

- Ngoài logic 0, logic 1, còn có một số trạng thái logic khác.
 - Trạng thái X (không quan tâm, don't care)
 - Trạng thái X (tương tranh, contention)
 - Trạng thái Z

Trạng thái X

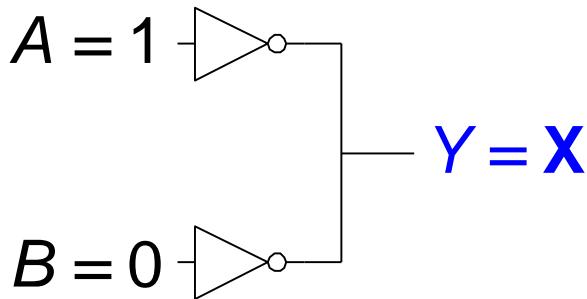
- Trạng thái X (không quan tâm, don't care)
- là trạng thái **đầu vào** dù bằng 0, hay 1 cũng không ảnh hưởng tới kết quả đầu ra

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	0
0	1	0	0	0	1	0	0
0	1	0	1	0	1	0	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	0
1	0	0	0	1	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	1	0	0	0
1	0	1	1	1	0	0	0
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	0
1	1	1	0	1	0	0	0
1	1	1	1	1	0	0	0

A_3	A_2	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	X	0	0	1	0
0	1	X	X	0	1	0	0
1	X	X	X	1	0	0	0

Trạng thái X

- Trạng thái X (tương tranh, contention)
 - là trạng thái **đầu ra** bị ép buộc về logic 0 và 1 đồng thời
 - có thể tạo ra mức logic 0, hoặc logic 1, hoặc rơi vào vùng điện áp cấm
 - thường gây ra quá tải
 - cho thấy có một **Iỗi thiết kế**



- Lưu ý: có 2 loại trạng thái X

Trạng thái trở kháng cao Z

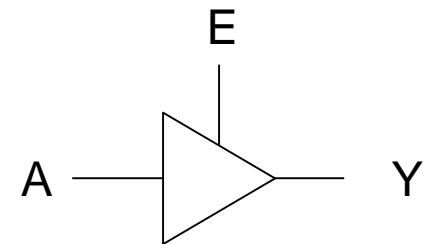
- Các tên gọi tương đương:

- Trôi
- Trở kháng cao
- Mở
- High Z (HiZ)

- Trạng thái Z có thể là bất cứ giá trị nào

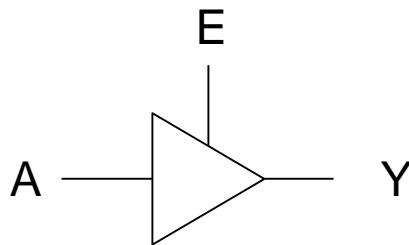
- Đồng hồ đo điện áp sẽ không thay đổi giá trị gì khi node ở trạng thái Z

Bộ đệm 3-trạng thái

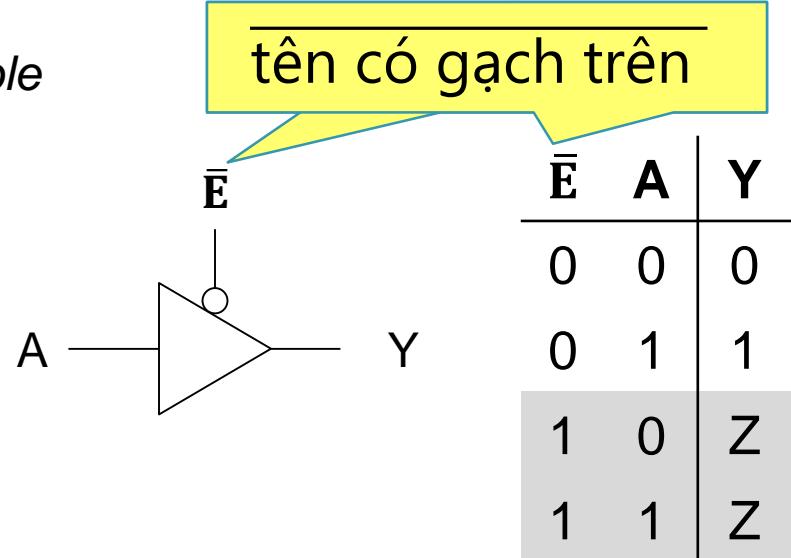


E	A	Y	
0	0	Z	
0	1	Z	
1	0	0	
1	1	1	

Tích cực mức cao/mức thấp



E = Enable		Y
E	A	
0	0	Z
0	1	Z
1	0	0
1	1	1



Bộ đệm 3 trạng thái **tích cực mức cao**

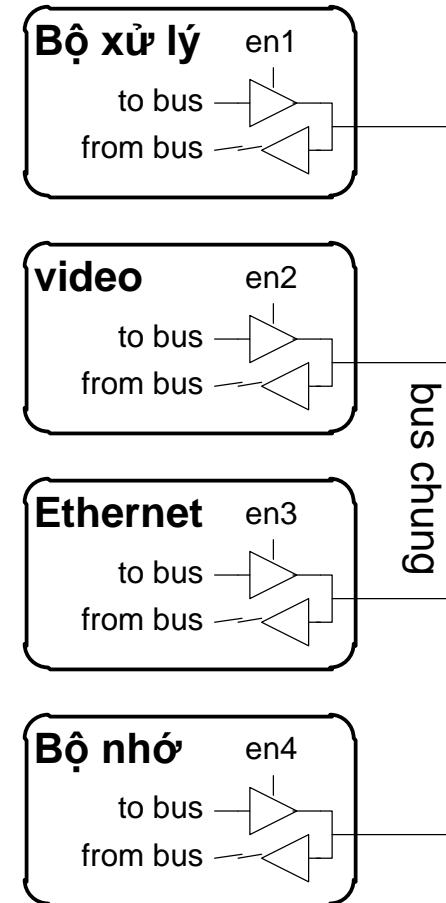
- Khi giá trị ở **E** là **TRUE**, bộ đệm 3-trạng thái hoạt động giống bộ đệm thường, có giá trị truyền từ đầu vào tới đầu ra.
- Khi giá trị ở **E** là **FALSE**, đầu ra luôn ở trạng thái Z

Bộ đệm 3 trạng thái **tích cực mức thấp**

- Khi giá trị ở **E** là **FALSE**, bộ đệm 3-trạng thái hoạt động giống bộ đệm thường, có giá trị truyền từ đầu vào tới đầu ra.
- Khi giá trị ở **E** là **TRUE**, đầu ra luôn ở trạng thái Z

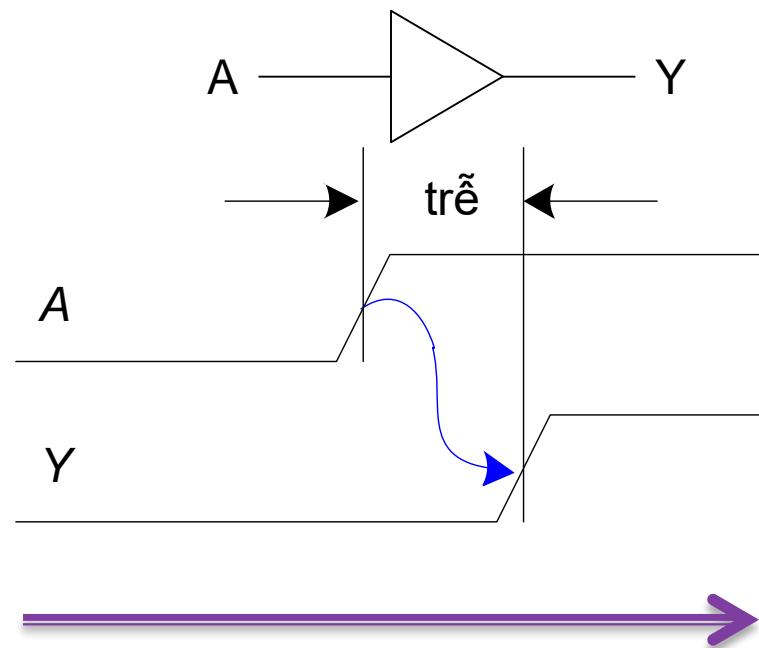
Các đường bus 3-trạng thái

- Ứng dụng trạng thái trở kháng cao Z để tạo ra các đường bus với 3 trạng thái
 - Có nhiều module cùng truyền/nhận dữ liệu từ đường bus
 - Chỉ có một** module được phép gửi dữ liệu lên bus tại 1 thời điểm
 - Có thể có **một hoặc nhiều** module cùng **nhận** dữ liệu từ bus



Timing

- Có độ trễ giữa tín hiệu ở đầu vào và tín hiệu kết quả ở đầu ra.
- Trễ truyền lan tăng khi nhiệt độ môi trường hoạt động, điện áp nguồn cấp, kéo tải đầu ra tăng.
- Làm sao để thiết mạch có độ trễ giảm → tốc độ xử lý tăng lên?



thời gian

Tác động của độ trễ

15,2cm, trễ ~1 ns

- Dây dẫn dài 6 inch, có trễ truyền lan khoảng 1ns.
- Cổng logic, tùy công nghệ chế tạo, trễ khoảng 1ns ~ 1000 ps.A standard logic gate symbol, specifically an AND gate, represented by a gray rounded rectangle with two input lines on the left and one output line on the right, all ending in small arrows.
- Độ trễ lớn nhất của một IC chính bằng chu kì hoạt động của IC đó → quyết định tần số làm việc tối đa F_{max} .

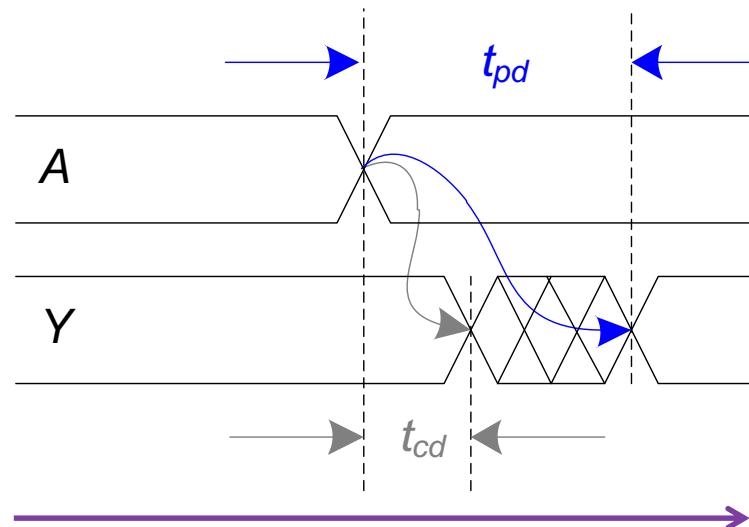
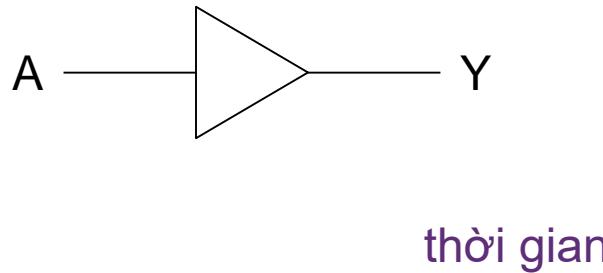
Trễ truyền lan và trễ lây nhiễm

Trễ truyền lan (Propagation Delay)

- t_{pd} = thời gian tối thiểu kể từ thời điểm bắt đầu xảy ra sự thay đổi từ đầu vào X cho tới khi sự thay đổi này tạo ra sự thay đổi xác định tại đầu ra Y, hay nói cách khác cho tới khi đầu ra Y ổn định giá trị

Trễ lây nhiễm (Contamination delay)

- t_{cd} = khoảng thời gian kể từ thời điểm xuất hiện sự thay đổi của đầu vào X cho tới khi đầu ra Y bắt đầu xảy ra sự mất ổn định.

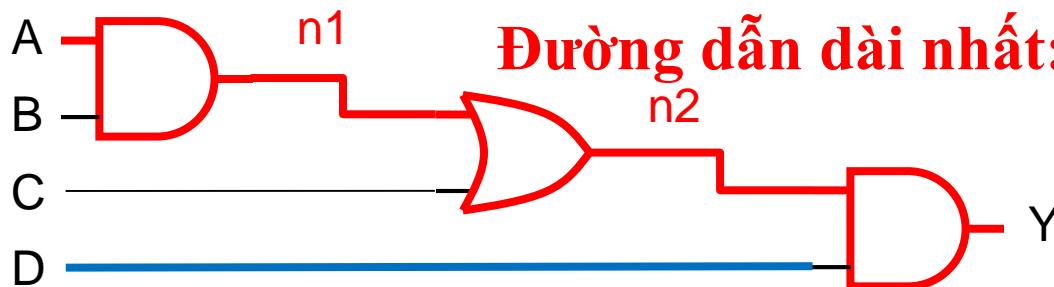


Trễ truyền lan và trễ lây nhiễm

- Trễ có nguyên nhân từ:
 - Thời gian nạp điện dung và trở kháng của mạch
 - Tốc độ của sóng điện từ (vận tốc ánh sáng) có hạn
- Tại sao t_{pd} và t_{cd} khác nhau?
 - Có sự khác biệt trong trễ ở sườn lên và sườn xuống
 - Mạch có nhiều đầu vào và đầu ra, một số đầu vào/ra có tốc độ nhanh hơn so với cái khác.
 - Tốc độ của mạch tăng lên ở nhiệt độ thấp, và chậm hơn ở nhiệt độ cao.

Đường dẫn tín hiệu dài/ngắn

- Thời gian trễ t_{pd} và t_{cd} được xác định thông qua trễ trên các đường dẫn tín hiệu
- Trễ truyền lan của mạch bằng tổng trễ truyền lan thành phần trên đường dẫn dài nhất
- Trễ lây nhiễm của mạch bằng tổng trễ lây nhiễm thành phần trên đường dẫn ngắn nhất



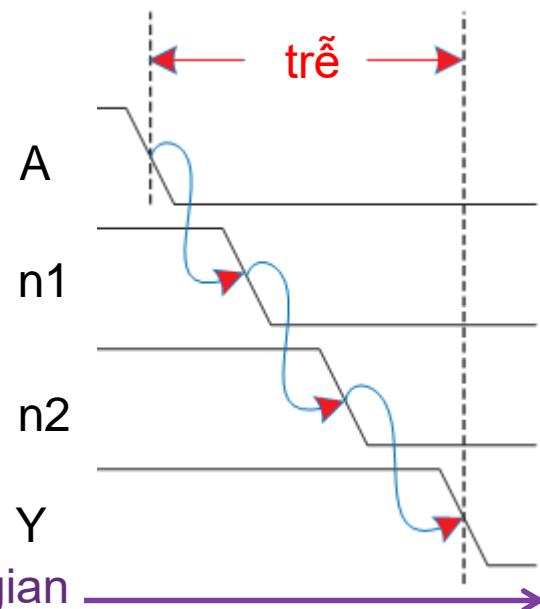
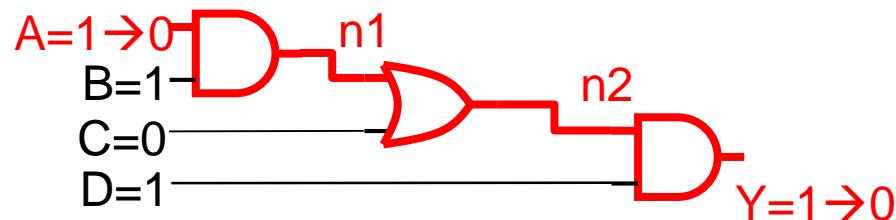
Đường dẫn dài nhất:

$$t_{pd} = 2t_{pd_AND} + t_{pd_OR}$$

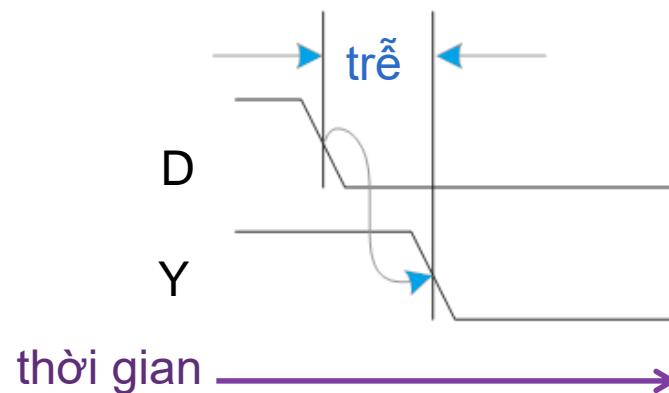
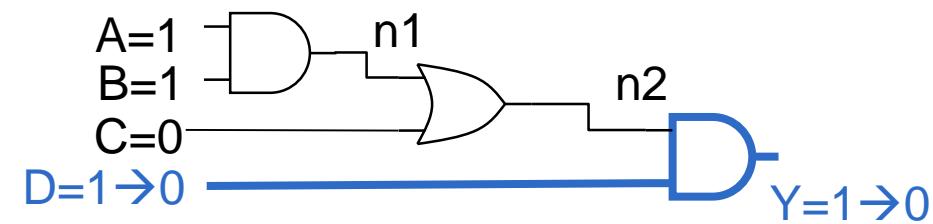
Đường dẫn ngắn nhất: $t_{cd} = t_{cd_AND}$

Dạng sóng tín hiệu trên đường dẫn

- Đường dẫn dài nhất
(Critical Path)

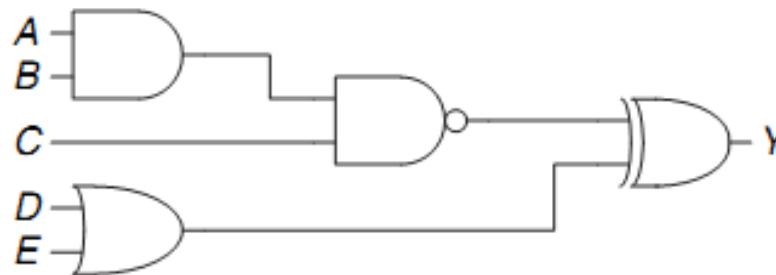


- Đường dẫn ngắn

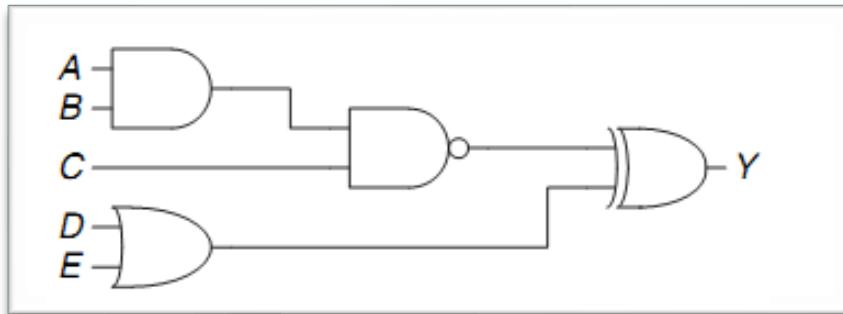


Bài tập về thời gian trễ

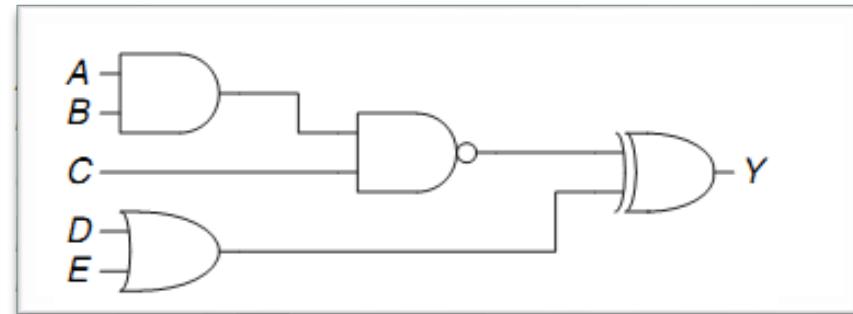
- Sinh viên Hoa cần tìm thời gian trễ truyền lan và lây nhiễm cho mạch số của mình dưới đây?
Theo [tra cứu datasheet](#), mỗi cổng logic đều có trễ truyền lan là 27ns và trễ lây nhiễm là 18ns. Bỏ qua trễ dây dẫn.



Đường dẫn dài

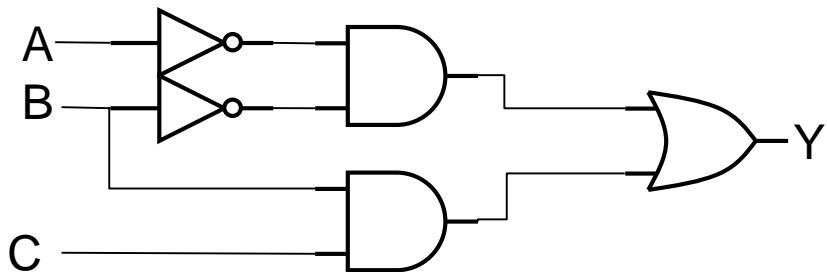


Đường dẫn ngắn



Không đồng đều

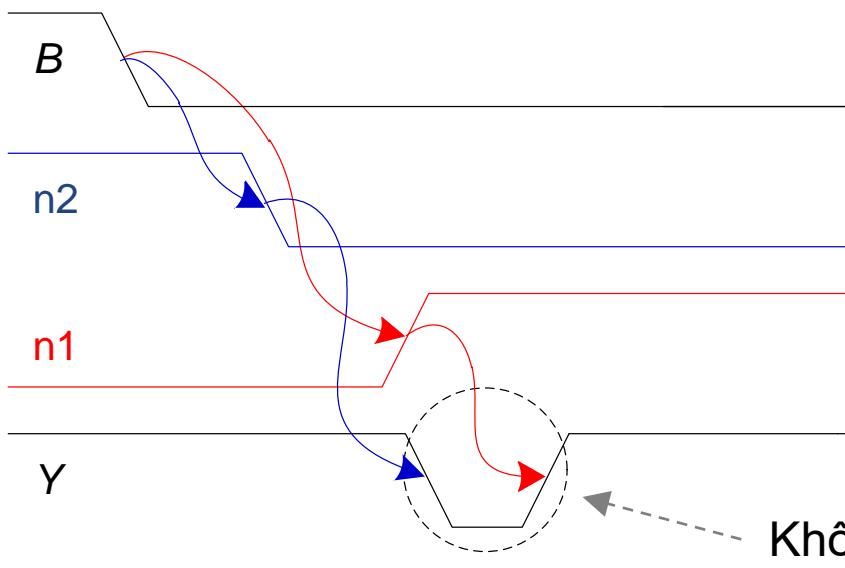
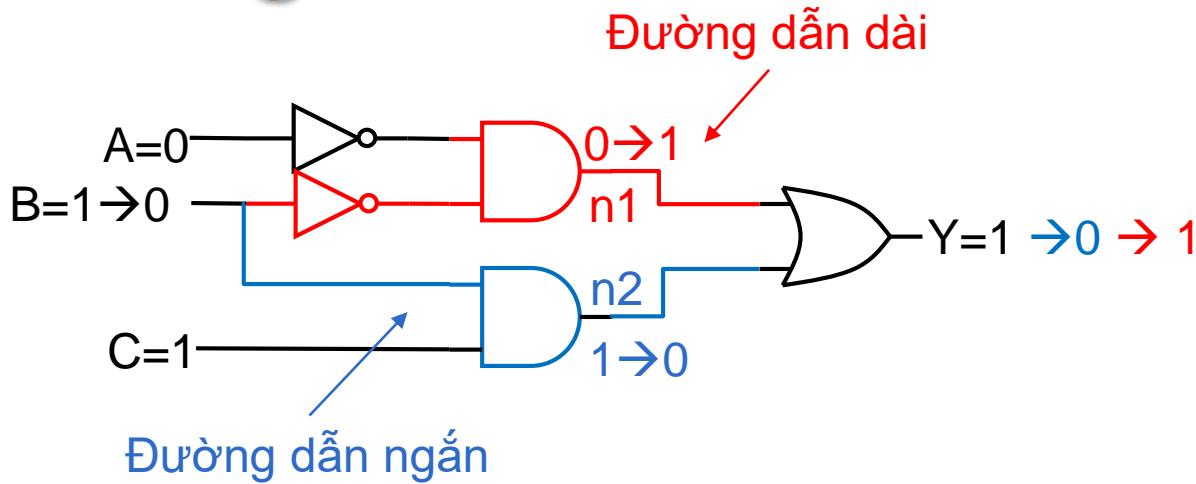
- Không đồng đều, glitch, là trạng thái khi 1 đầu vào có thay đổi, thì đầu ra bị thay đổi nhiều lần.
- Trong mạch bên, nếu $A=0, C=1, B=1 \rightarrow 0$ thì ?



		AB	00	01	11	10	
		C	0	1	0	0	0
Y	AB	0	1	0	0	0	
		1	1	1	1	0	

$$Y = \bar{A}\bar{B} + BC$$

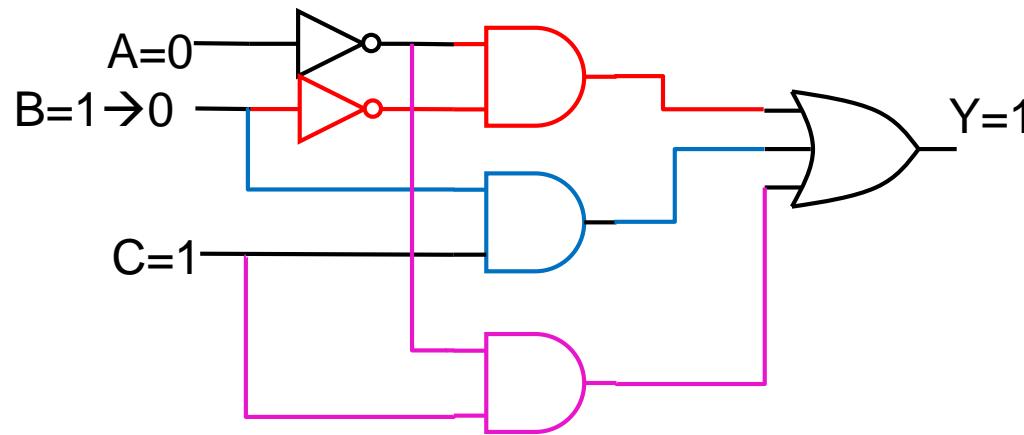
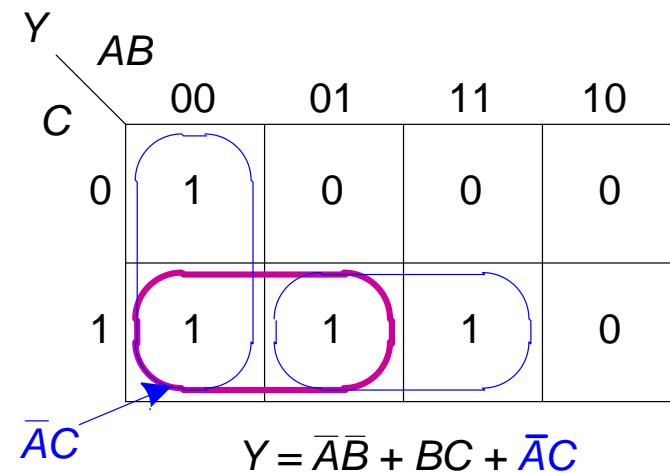
Không đồng đều



Xem lại slide
“Hazard động”

Không đồng đều

Khắc phục “không đồng đều”



Hiểu rõ hiện tượng “không đồng đều”

- “Không đồng đều” có thể được giải quyết bằng các thiết kế theo kiểu **đồng bộ**.
- Nắm rõ về “không đồng đều” để hiểu tại sao dạng sóng tín hiệu lại bị nhấp nhô khi xem trên máy hiện sóng, hoặc giả lập.
- Không thể loại bỏ tình trạng không đồng đều, gần như xảy ra với mọi phần tử có nhiều đầu vào.



MẠCH DÃY

Sequential Logic

- Mạch chốt Latch và mạch lật Flip-Flop
- Thiết kế logic đồng bộ
- Sơ đồ máy trạng thái hữu hạn
- Thời gian của mạch dãy
- Song song hóa

Các định nghĩa

- Mạch dây có tín hiệu đầu ra phụ thuộc vào đầu vào tại thời điểm hiện tại và quá khứ
→ mạch **có nhớ**.
- **Trạng thái - State:** là tất cả các thông tin cần thiết về mạch để có thể xác định hành vi của mạch đó trong tương lai. Thông tin đó nằm trong một tập các bit gọi là **các biến trạng thái**.
- **Chốt và Lật - latches và flip-flop:** là trạng thái lưu trữ một bit
- **Mạch dây đồng bộ:** là mạch dây được kiểm soát bởi một dãy các Flip-flop

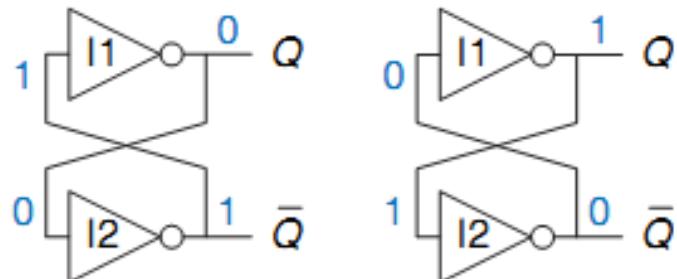
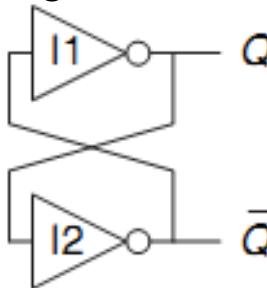
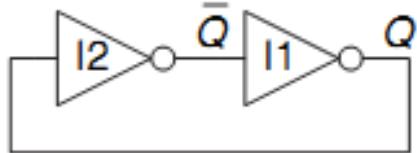
Phần tử trạng thái

- Trạng thái của mạch ảnh hưởng tới hành vi của chính nó trong tương lai
- Phần tử trạng thái chứa trạng thái
 - Phần tử Bistable
 - SR Latch
 - D Latch
 - D Flip-flop

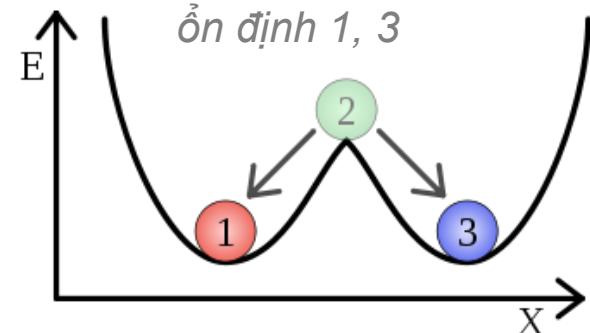
Phần tử bi-stable

- Phần tử Bi-stable: là phần tử có hai trạng thái ổn định, là cốt lõi để xây dựng chức năng nhớ

- Ví dụ: 2 cỗng đảo mắc chéo; 0 đầu vào; 2 đầu ra Q , \bar{Q}
 - 2 trạng thái ổn định $Q=0$ và $Q=1$
 - Khi mới cấp nguồn, mạch ở trạng thái không xác định.



- Một mạch có N trạng thái tương ứng với $\log_2 N$ bit thông tin, trong đó mỗi bit chứa trong một phần tử bistable.



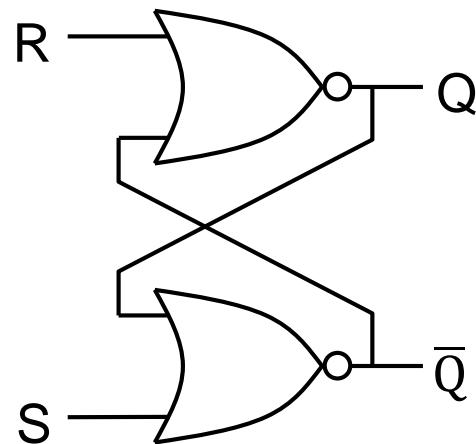
bóng ở vị trí 2 sẽ
trôi về trạng thái
ổn định 1, 3

SR (Set/Reset) Latch

- SR Latch
Mạch chốt SR

- Cần xem xét 4 trường hợp sau

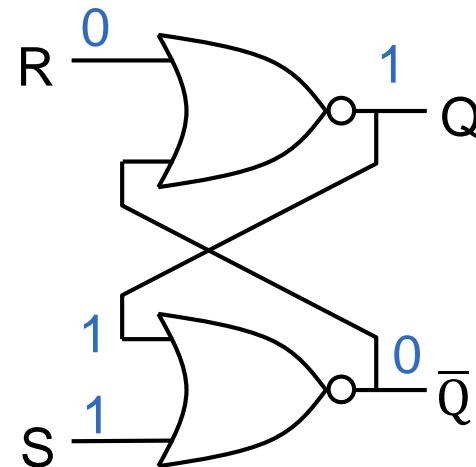
- $S = 1, R = 0$
- $S = 0, R = 1$
- $S = 0, R = 0$
- $S = 1, R = 1$



SR Latch: phân tích hoạt động

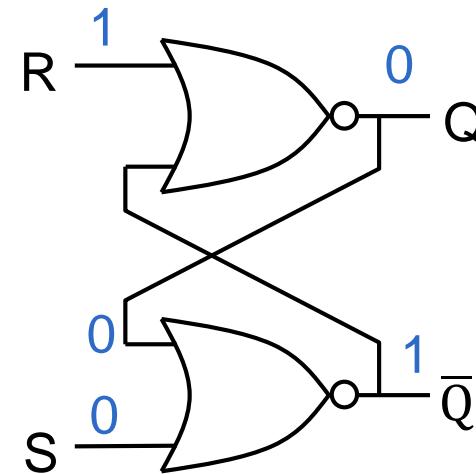
- $S = 1, R = 0$:

thì $Q = 1$ và $\bar{Q} = 0$



- $S = 0, R = 1$:

thì $\bar{Q} = 1$ và $Q = 0$

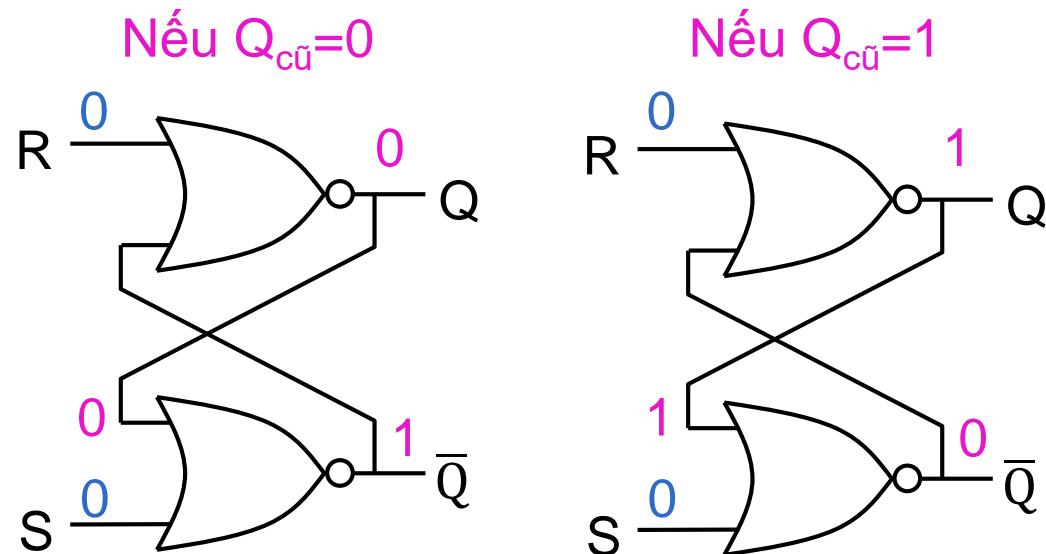


SR Latch: phân tích hoạt động

- $S = 0, R = 0$:

thì $Q = Q_{cũ}$

→ Trạng thái NHỚ

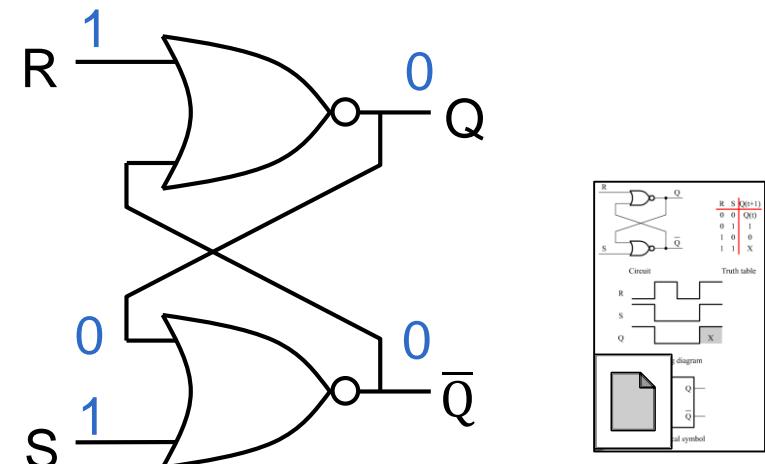


- $S = 1, R = 1$:

thì $Q = \bar{Q} = 0$

→ Không hợp lệ vì

$\bar{Q} \neq \text{not } Q$



SR Latch: kí hiệu

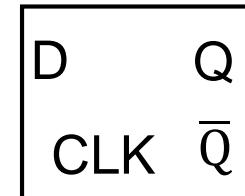
- Kí hiệu SR nghĩa là Set Reset Latch
 - Lưu trữ một bit trạng thái Q
- Đầu vào S, R sẽ làm thay đổi giá trị được lưu trữ
 - **Set:** thiết lập đầu ra bằng 1
(S = 1, R = 0, Q = 1)
 - **Reset:** thiết lập đầu ra bằng 0
(S = 0, R = 1, Q = 0)

Biểu tượng
SR Latch

R	Q
S	\bar{Q}

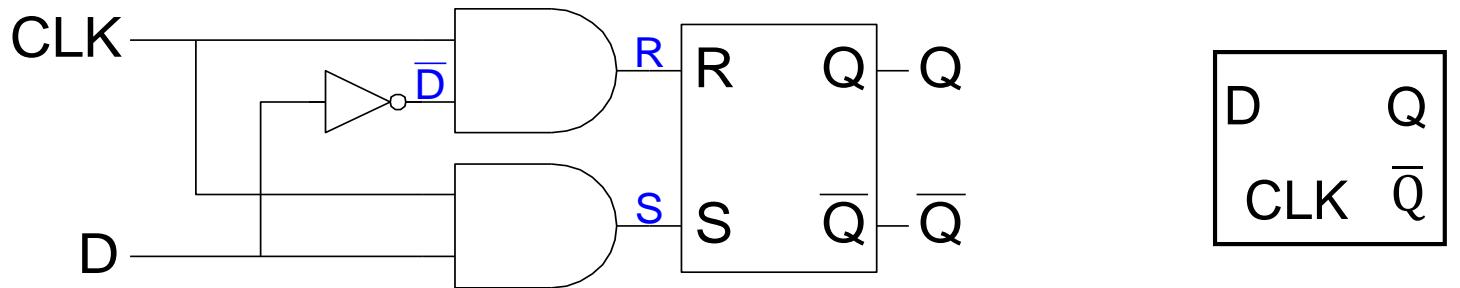
D Latch

Biểu tượng
D Latch



- D Latch: Mạch chốt D
 - Hai đầu vào CLK, D
 - CLK cho biết **thời điểm** đầu ra thay đổi giá trị
 - D (dữ liệu vào) cho biết **giá trị** mà đầu ra sẽ trở thành
- Hoạt động:
 - Khi $CLK = 1$, giá trị ở D sẽ truyền tới Q (transparent, thông suốt)
 - Khi $CLK = 0$, giá trị ở Q sẽ giữ nguyên như cũ (opaque, cản trở D không cho truyền tới Q)
→ Tránh được tình trạng không hợp lệ $\bar{Q} \neq \text{not } Q$

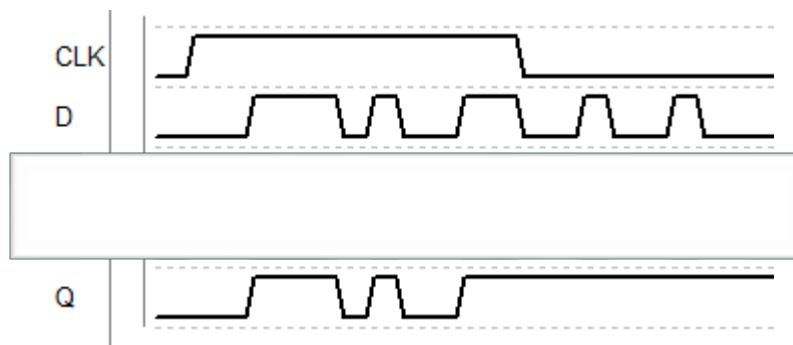
D Latch: phân tích hoạt động



CLK	D	\bar{D}	S	R	Q	\bar{Q}
0	X	\bar{X}	0	0	$Q_{\text{cũ}}$	$\bar{Q}_{\text{cũ}}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

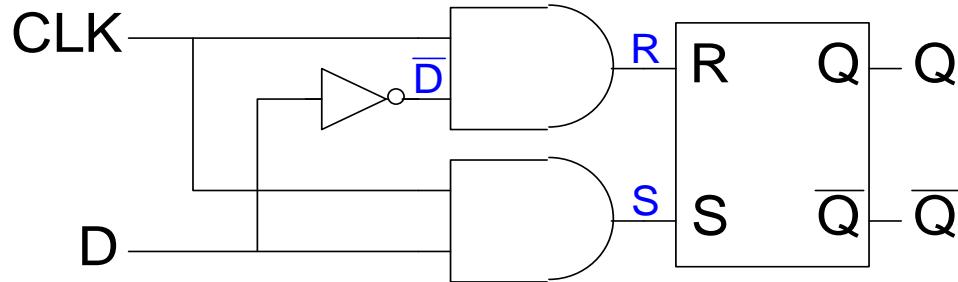
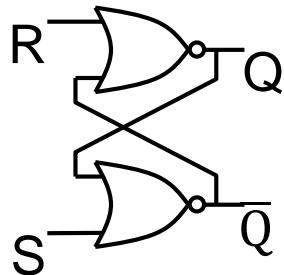
Bảng sự thật

Ví dụ với waveform



Bài tập

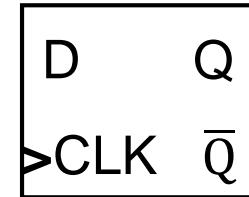
- Cho biết số transistor cần thiết để chế tạo D-FF theo sơ đồ dưới



Giải:

D Flip-flop

- D Flip-flop (D FF): Mạch lật D



- Hai đầu vào CLK, D

- Đầu vào D được lấy mẫu tại sườn lên của CLK
 - Khi CLK ở sườn lên \uparrow , D truyền giá trị tới Q
 - Trong mọi trường hợp còn lại, Q giữ nguyên giá trị cũ
 - Q chỉ thay đổi giá trị tại sườn lên của CLK

- Tích cực sườn lên, hoặc còn gọi là sườn dương.
- Lưu ý: khoảng thời gian CLK ở sườn lên cực ngắn.

D Flip-flop: phân tích hoạt động

- D Flip-flop gồm 2 D latch có tín hiệu CLK là đảo của nhau

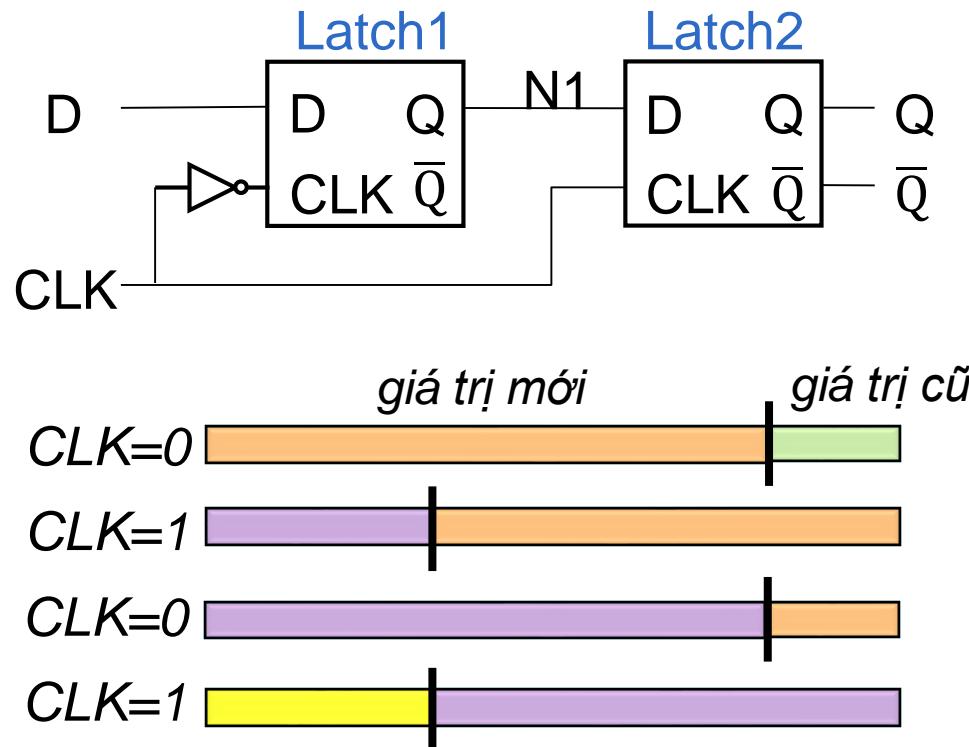
- **Hoạt động**

Khi $CLK=0$

- Latch1 thông suốt
- Latch2 cản trở
- Giá trị ở D truyền tới N1

Khi $CLK=1$

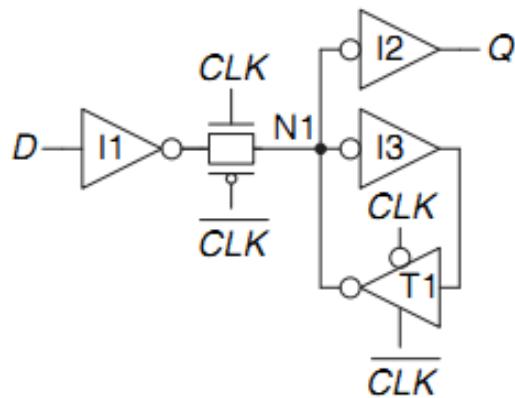
- Latch1 cản trở
- Latch2 thông suốt
- Giá trị ở D truyền tới Q



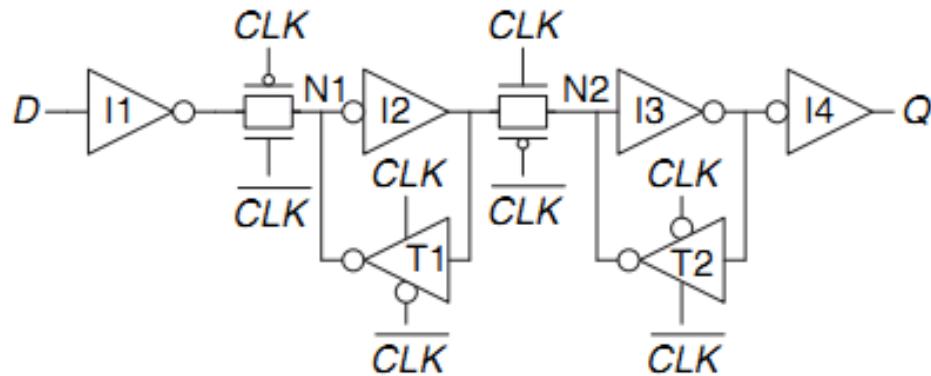
- Do đó, chỉ tại sườn lên của CLK (tức là CLK từ 0 → 1) thì D mới truyền giá trị tới Q

Một số cách thiết kế D-FF

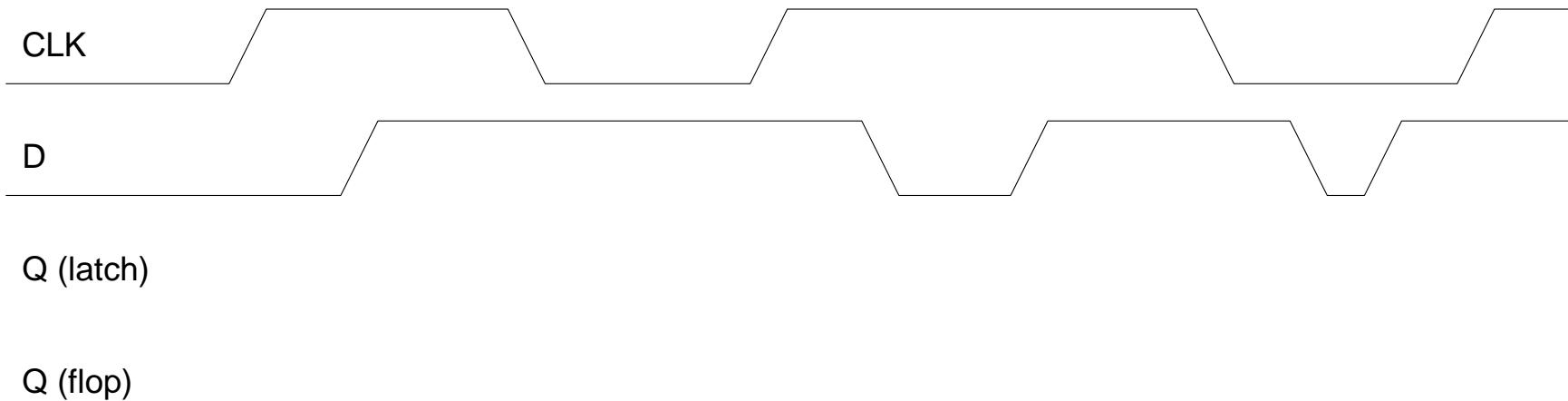
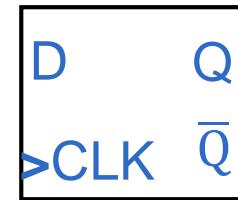
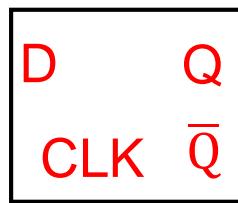
- 12 transistor



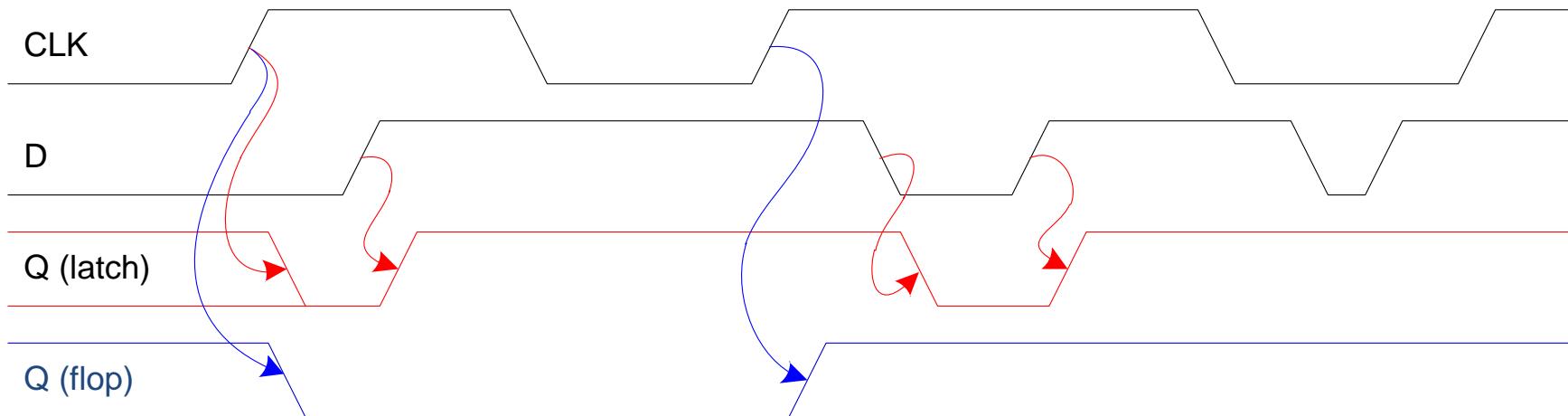
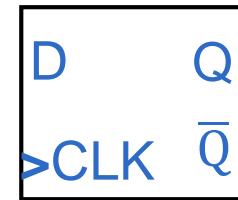
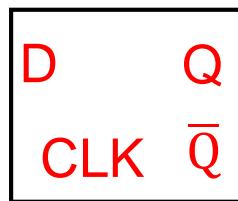
- 20 transistor



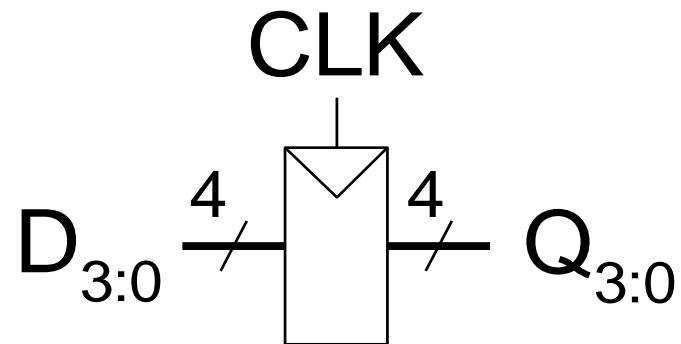
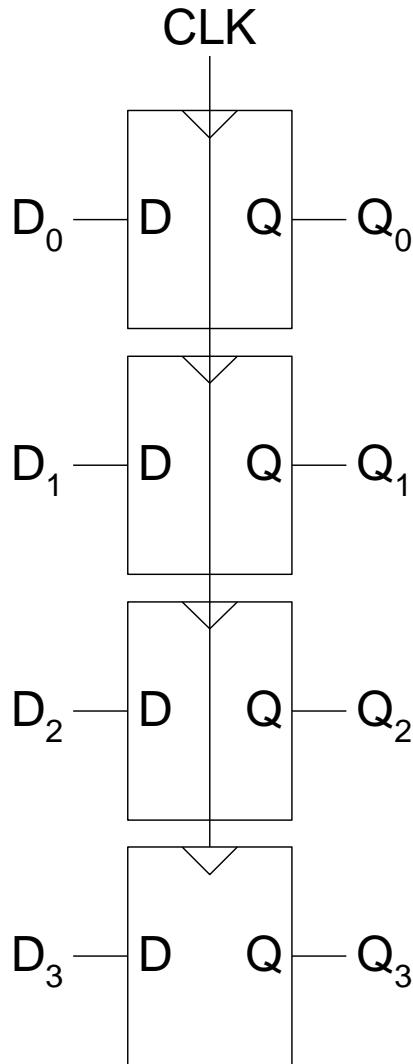
So sánh D Latch vs D Flip-flop



So sánh D Latch vs D Flip-flop



Thanh ghi (Register)



D Flip-flop có tín hiệu cho phép

- Đầu vào: CLK, D, EN

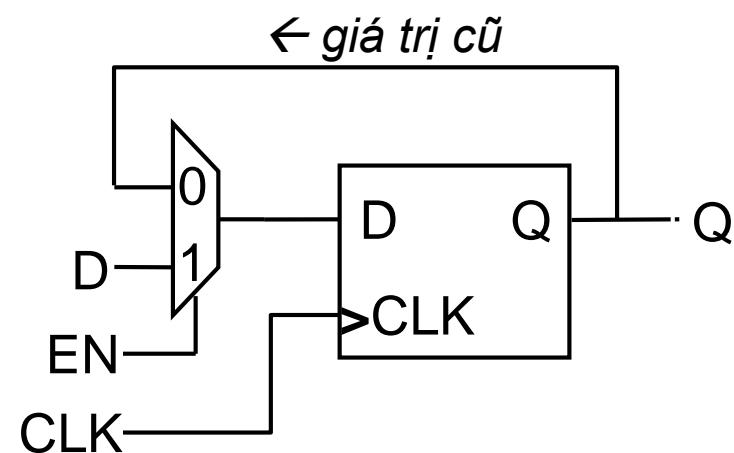
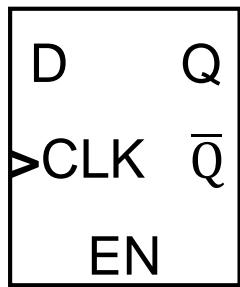
- Tín hiệu cho phép EN (Enable) cho biết thời điểm dữ liệu mới D được lưu lại.

- Hoạt động

- Khi $EN = 1$, D truyền giá trị tới Q tại sườn lên của CLK
- Khi $EN = 0$, Q giữ nguyên giá trị cũ

Biểu tượng

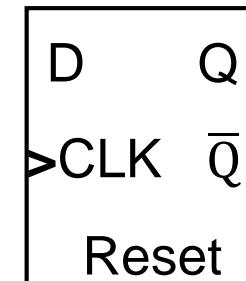
D Flip-flop có EN



D Flip-flop có tín hiệu Reset

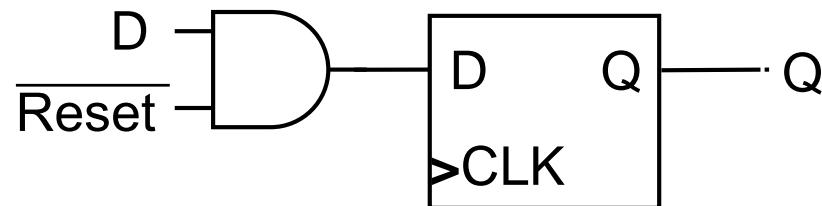
- Đầu vào: CLK, D, **Reset**
 - Tín hiệu Reset sẽ xóa dữ liệu đầu ra về 0
- Hoạt động
 - Khi **Reset = 1**, Q có giá trị bằng 0
 - Khi **Reset = 0**, hoạt động giống như D-FF thường

Biểu tượng
D Flip-flop có Reset



D Flip-flop có tín hiệu Reset

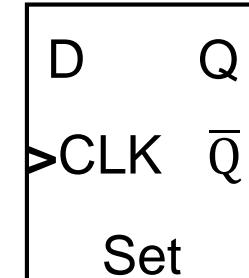
- Có 2 kiểu, tùy theo thời điểm tín hiệu Reset có tác dụng
 - Đồng bộ (Synchronous): tín hiệu Reset chỉ có **tác dụng tại sườn** của CLK, làm cho đầu ra Q=0.
 - Không đồng bộ (Asynchronous): tín hiệu Reset =1 **có tác dụng ngay lập tức**, làm cho đầu ra Q = 0.
- Thiết kế D-FF với Reset KHÔNG ĐỒNG BỘ đòi hỏi phải xây dựng lại mạch bên trong FF.
- Thiết kế D-FF với Reset ĐỒNG BỘ đơn giản hơn



D Flip-flop có tín hiệu Set

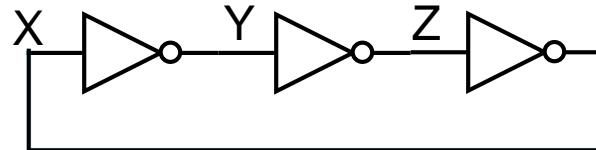
- Đầu vào: CLK, D, **Set**
 - Tín hiệu Set sẽ thiết lập dữ liệu đầu ra về 1
- Hoạt động
 - Khi **Set = 1**, Q có giá trị bằng 1
 - Khi **Set = 0**, hoạt động giống như D-FF thường
- Tín hiệu Set cũng có 2 kiểu:
 - Đồng bộ
 - Không đồng bộ

Biểu tượng
D Flip-flop có Set



Bài tập: mạch dao động vòng

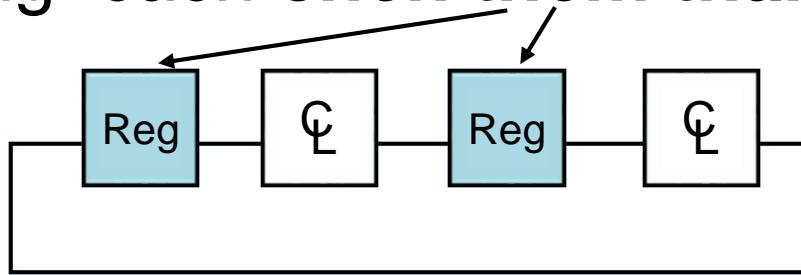
- SV Lê Văn Long mắc mạch như hình bên dưới. Biết rằng trễ truyền lan trên mỗi cổng đảo là 1ns. Hãy cho biết dạng sóng của tín hiệu tại X, Y, Z?



Giải:

Thiết kế mạch dây

- Chia nhỏ đường dẫn tín hiệu vòng thành các đoạn bằng cách **chèn thêm thanh ghi**.



- Các thanh ghi chứa **trạng thái** của mạch.
- Việc chuyển trạng thái xảy ra tại sườn của tín hiệu đồng hồ: mạch được **đồng bộ** bởi tín hiệu đồng hồ.

Thiết kế mạch dây

4. Một số luật thiết kế cần tuân thủ

- Mỗi phần tử của mạch dây là một thanh ghi, hoặc một mạch tổ hợp
- Có tối thiểu 1 thanh ghi
- Tất cả các thanh ghi đều chung nhau tín hiệu đồng hồ.
- Tất cả đường dẫn vòng đều chứa ít nhất 1 thanh ghi

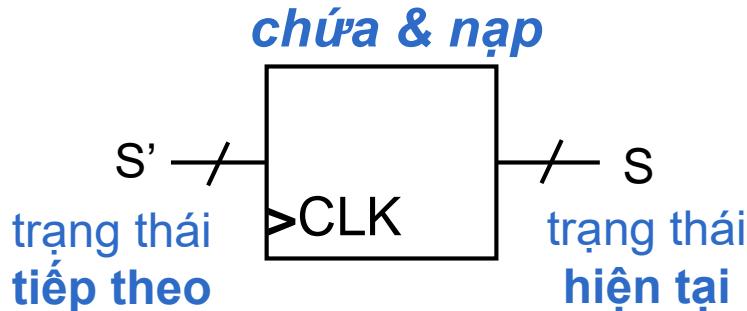
5. Có 2 dạng mạch dây đồng bộ thường gặp

- Máy trạng thái hữu hạn (Finite State Machines - FSM)
- Đường ống (Pipeline)

Finite State Machine

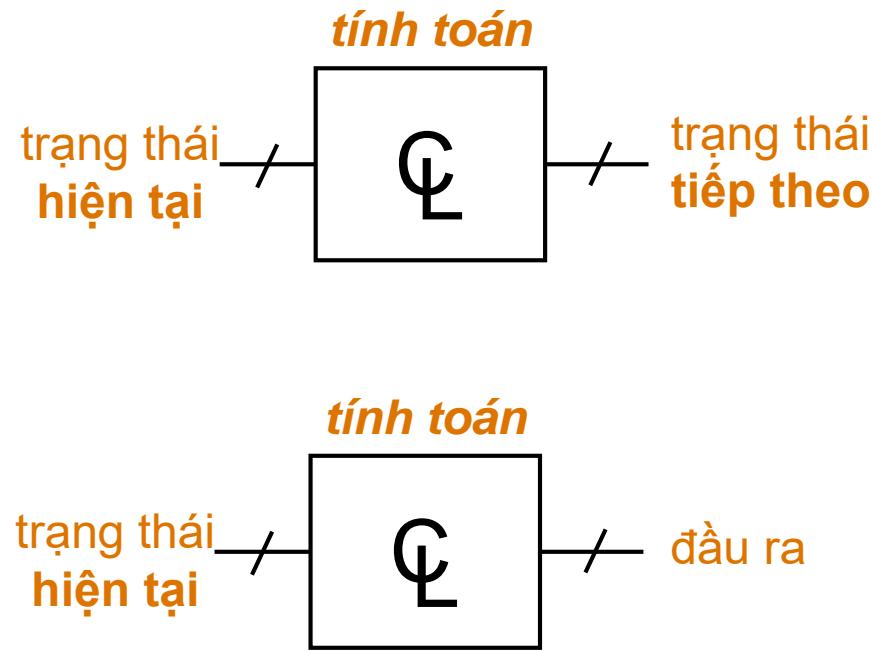
- Bao gồm:

- Thanh ghi trạng thái
 - **Chứa** trạng thái hiện tại
 - Sẽ được **nạp** trạng thái tiếp theo tại sườn lên của tín hiệu đồng hồ



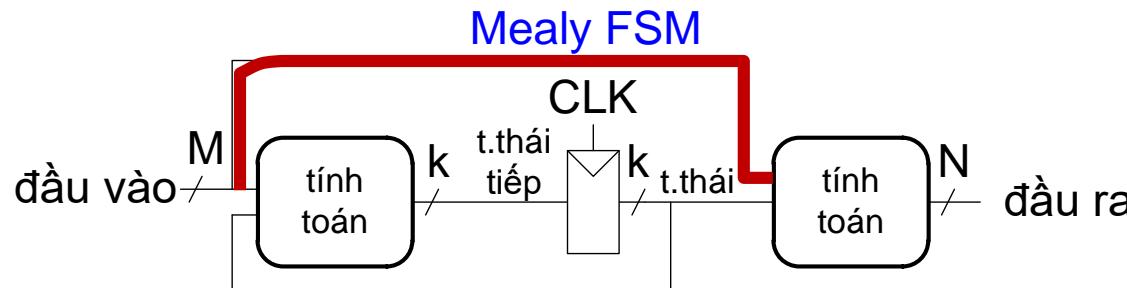
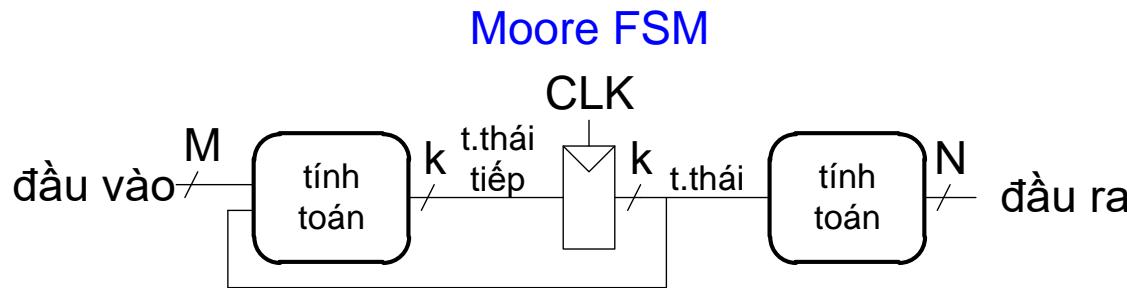
- Mạch tổ hợp

- **Tính toán** trạng thái tiếp theo
- **Tính toán** giá trị đầu ra



Finite State Machine

- Trạng thái tiếp theo được xác định thông qua trạng thái hiện tại và đầu vào.
- Có 2 kiểu FSM
 - Moore FSM: đầu ra phụ thuộc vào trạng thái hiện tại
 - Mealy FSM: đầu ra phụ thuộc vào trạng thái hiện tại và đầu vào.



Ví dụ FSM (1)



Mạch tổ hợp:
tính toán thực hiện (hái táo)

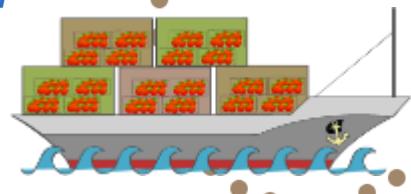


Thanh ghi:
chở lô hàng tiếp theo



Mạch tổ hợp:
tính toán thực hiện (chế biến)

Thanh ghi:
chở lô hàng tiếp theo



Mạch tổ hợp:
tính toán thực hiện (bán táo)

Ví dụ FSM đèn giao thông

Điều khiển đèn giao thông

- Cảm biến lưu lượng giao thông: T_A , T_B (TRUE nếu có giao thông)
- Đèn giao thông: L_A , L_B

Qui tắc

If $T_A == 1$ then

$L_A = \text{green}$

$L_B = \text{red}$

else

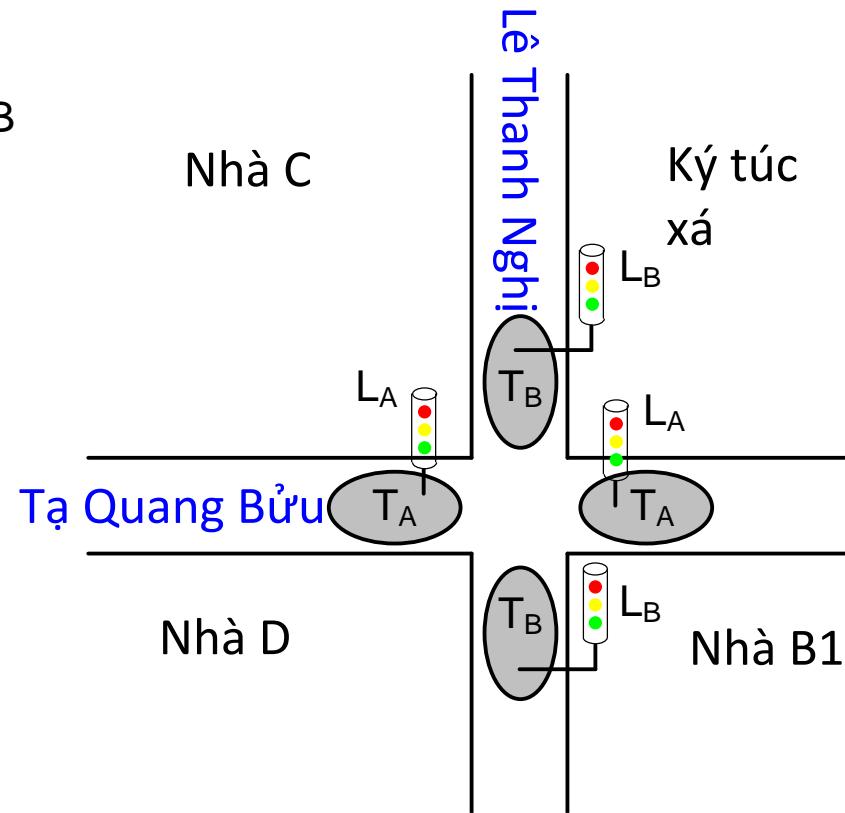
$L_A = \text{yellow}$

$L_B = \text{red}$

Đợi 1 nhịp

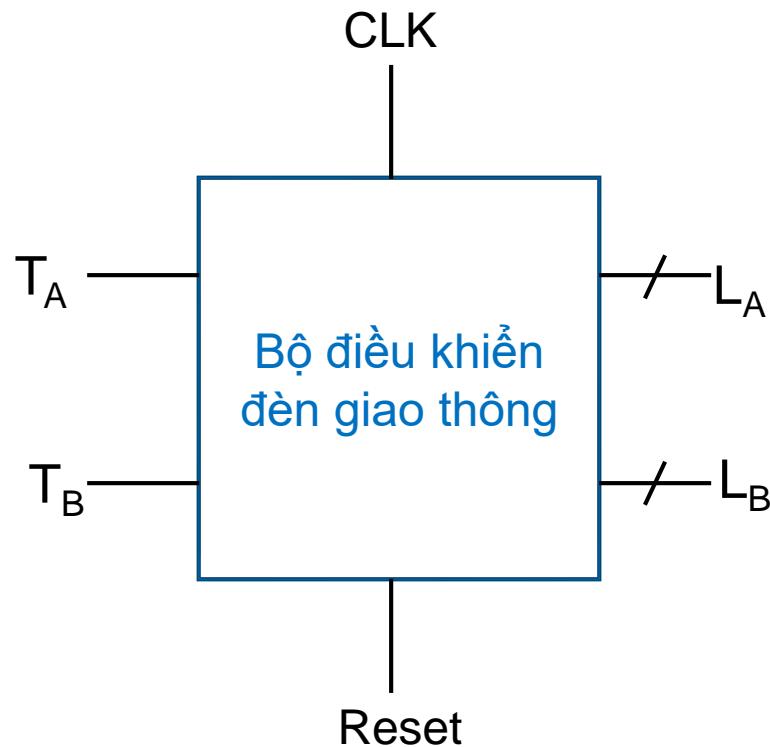
$L_A = \text{red}$

$L_B = \text{green}$



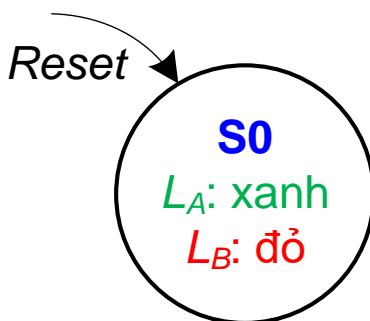
Ví dụ thiết kế hộp đèn FSM

- Đầu vào: CLK, Reset, T_A , T_B
- Đầu ra: L_A , L_B



Mô hình biến đổi trạng thái FSM

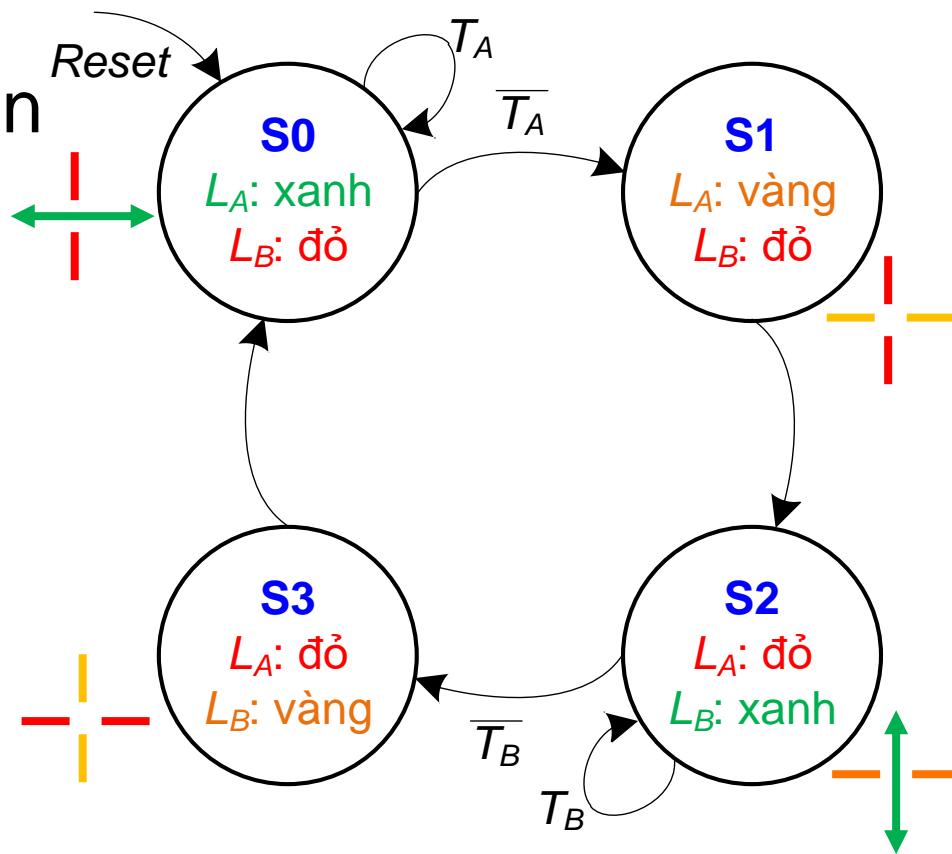
- Moore FSM: đầu ra được tính bên trong một trạng thái nào đó.
- Trạng thái: biểu diễn bằng vòng tròn
- Biến đổi trạng thái: đường nối



Mô hình biến đổi trạng thái FSM

- Moore FSM: đầu ra được tính bên trong một trạng thái nào đó.
- Trạng thái: biểu diễn bằng vòng tròn
- Biến đổi trạng thái: đường nối

If $T_A == 1$ then
 $L_A = \text{green}$
 $L_B = \text{red}$
else
 $L_A = \text{yellow}$
 $L_B = \text{red}$
 Đợi 1 nhịp
 $L_A = \text{green}$
 $L_B = \text{red}$



Bảng biến đổi trạng thái FSM

Trạng thái hiện tại S	Đầu vào		Trạng thái tiếp theo S'
	T_A	T_B	
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0

Ý nghĩa: chuyển đổi FSM ở dạng đồ thị thành dạng văn bản (script hóa)

Bảng biến đổi trạng thái có mã hóa

Ý nghĩa: biến đổi tên trạng thái → mã nhị phân. Số hóa trạng thái nội bộ.
Câu hỏi: nếu bảng mã hóa thay đổi thì sao?

Trạng thái	Mã hóa
S0	00
S1	01
S2	10
S3	11

Trạng thái hiện tại		Đầu vào		Trạng thái tiếp theo	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

$$S'_1 = S_1 \oplus S_0$$

$$S'_0 = \overline{S}_1 \cdot \overline{S}_0 \cdot \overline{T_A} + S_1 \cdot \overline{S}_0 \cdot \overline{T_B}$$

Bảng đầu ra FSM

Ý nghĩa: biên dịch đầu ra → mã nhị phân. Số hóa kết quả đầu ra.

Trạng thái hiện tại		Đầu ra			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

Đầu ra	Mã hóa $L_1 L_0$
xanh	00
vàng	01
đỏ	10

L_{A1}, L_{A2} là cặp bit tín hiệu đầu ra của IC, tương ứng với màu của 1 đèn giao thông, sẽ được truyền dẫn tới đèn giao thông để làm đèn sáng theo 3 mức

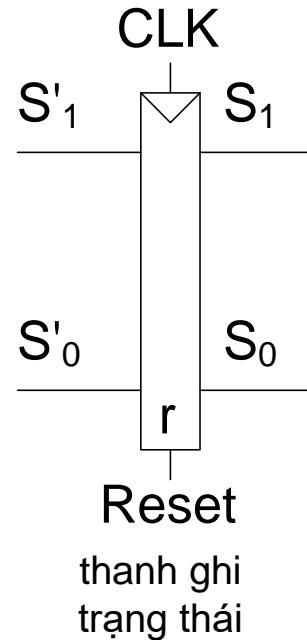
$$L_{A1} = S_1$$

$$L_{A0} = \bar{S}_1 \cdot S_0$$

$$L_{B0} = \bar{S}_1$$

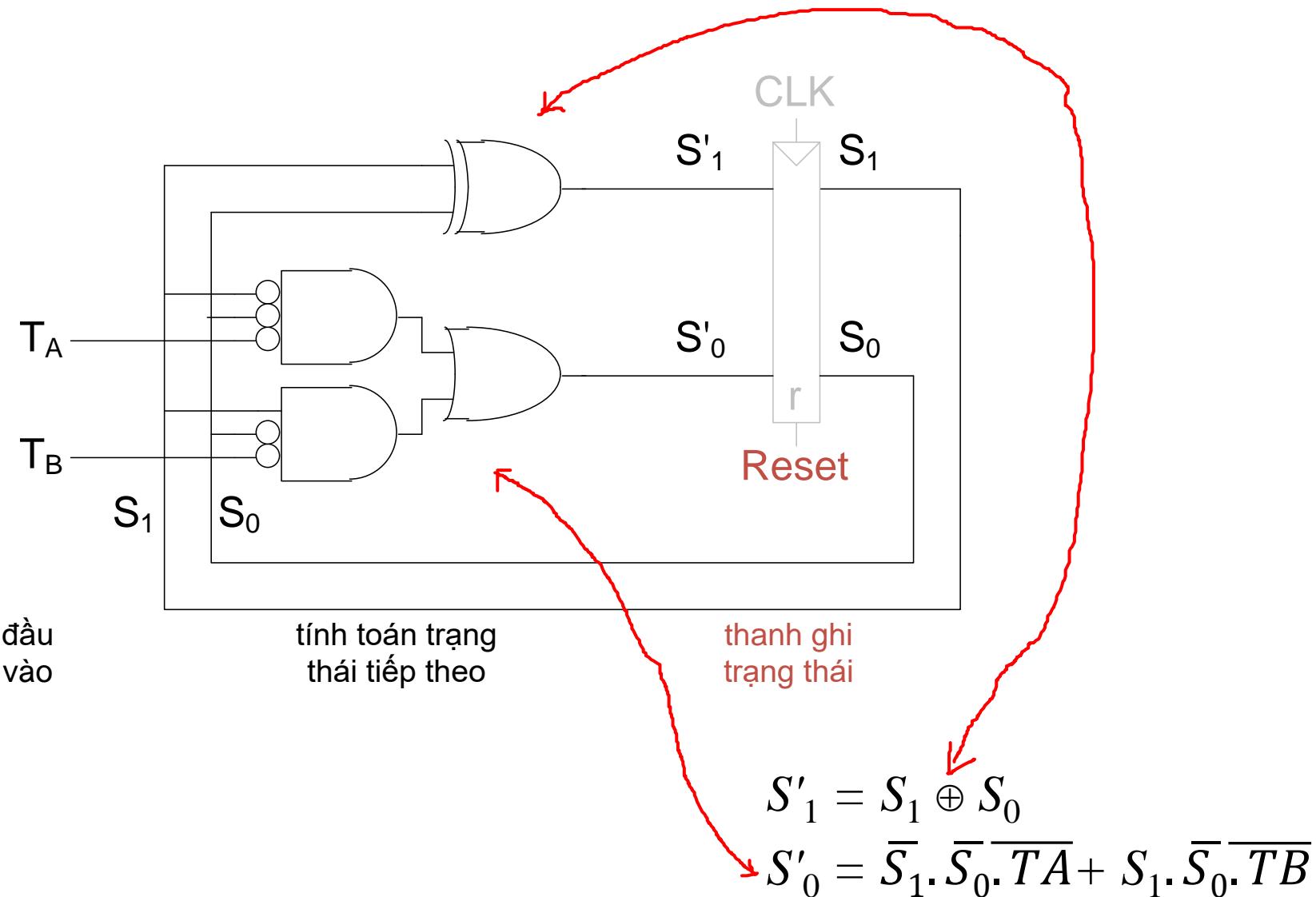
$$L_{B1} = S_1 \cdot S_0$$

FSM Schematic : thanh ghi t.thái

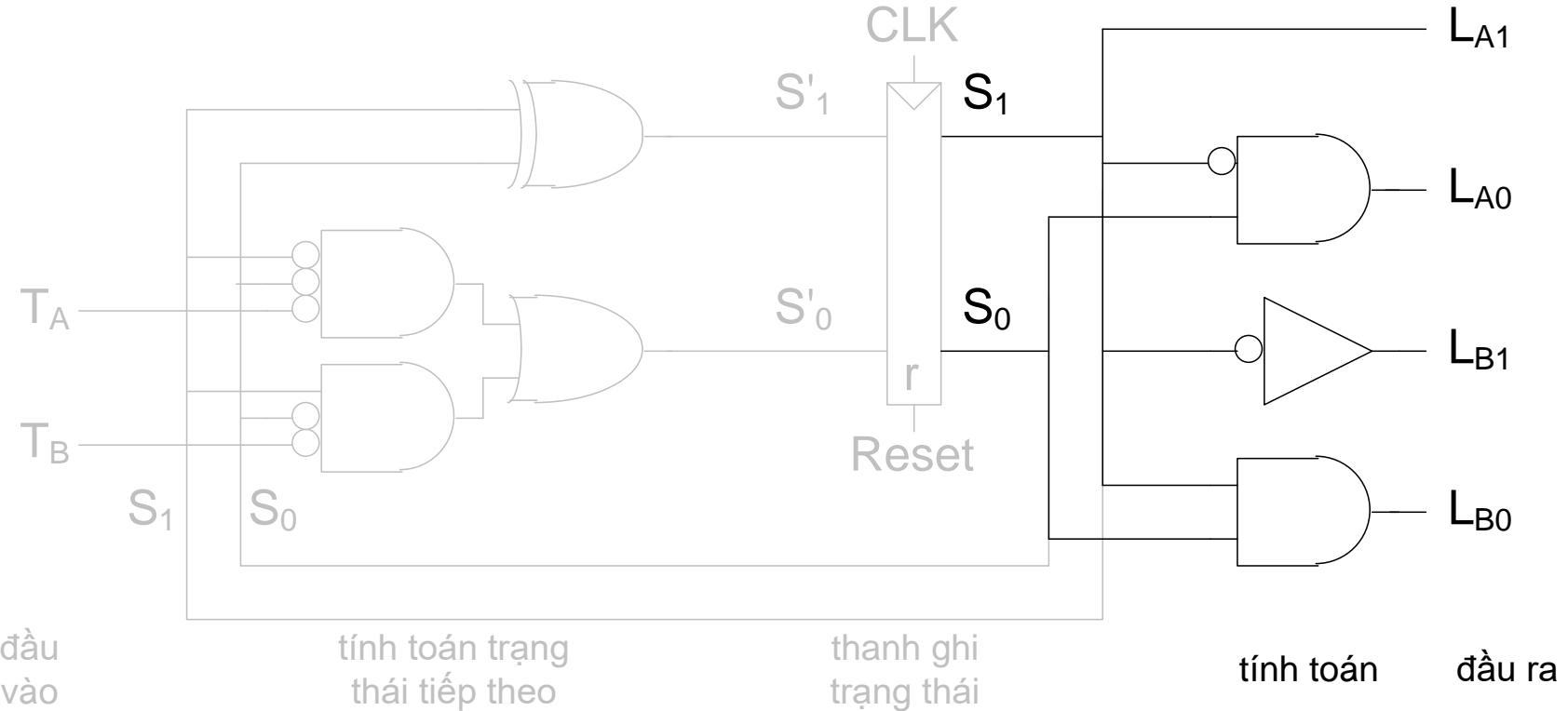


thanhs
trạng thái

FSM Schematic : tính t.thái tiếp theo



FSM Schematic : tính giá trị đầu ra



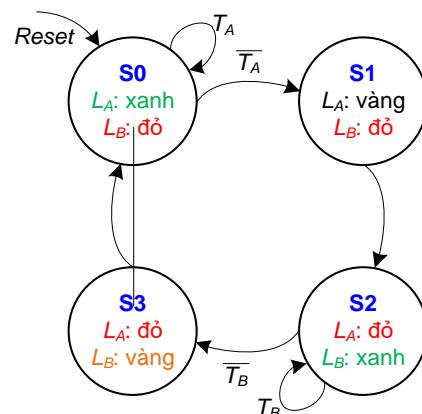
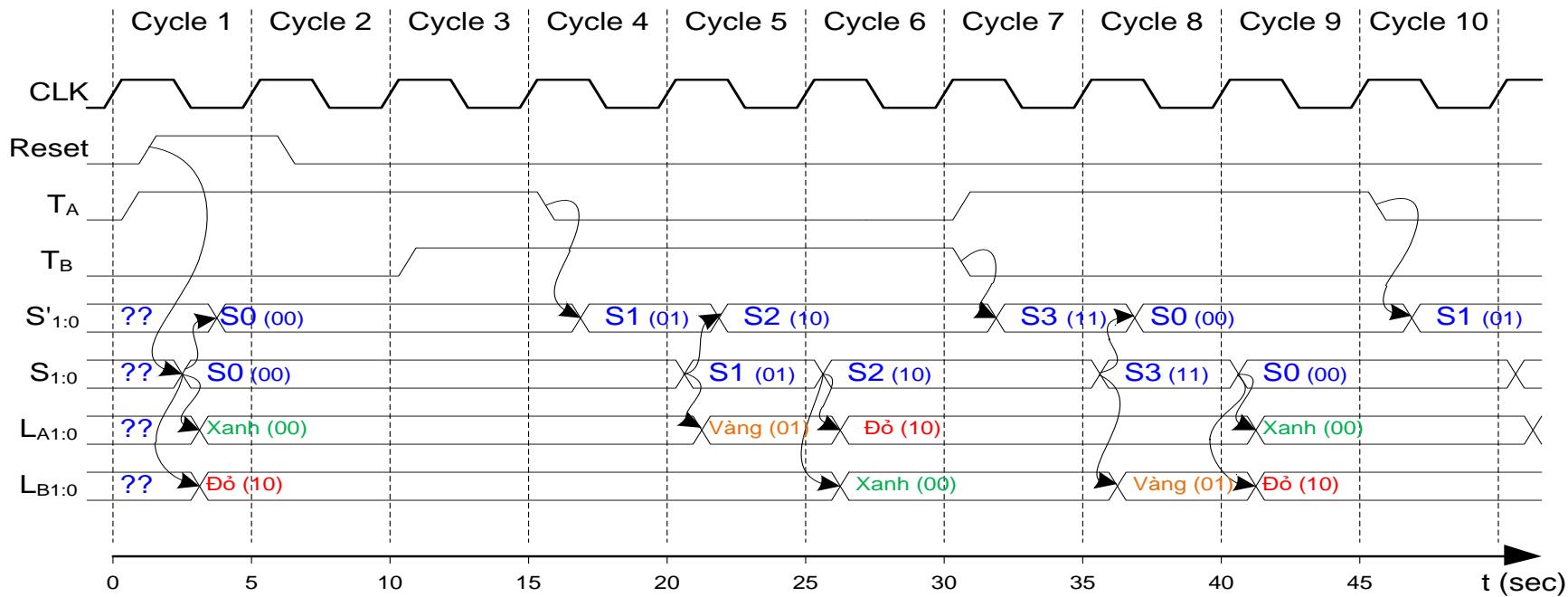
$$L_{A1} = S_1$$

$$L_{A0} = \overline{S_1} \cdot S_0$$

$$L_{B0} = \overline{S_1}$$

$$L_{B1} = S_1 \cdot S_0$$

Giản đồ thời gian FSM



Các bước trong thiết kế với FSM

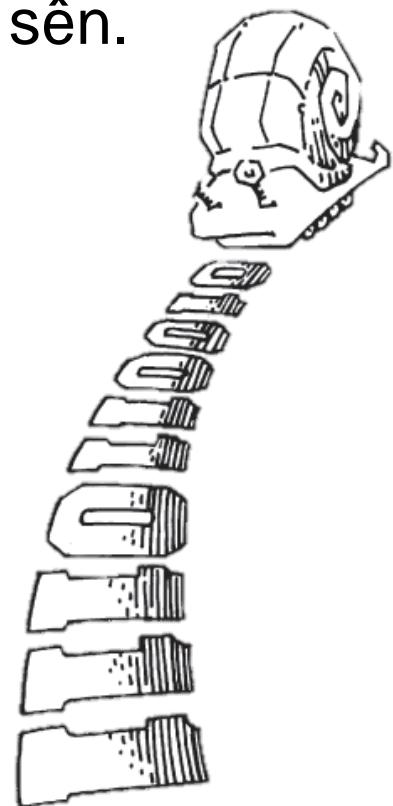
1. Xác định đầu vào, đầu ra
2. Vẽ sơ đồ chuyển đổi trạng thái
3. Lập bảng chuyển đổi trạng thái
4. Mã hóa trạng thái
5. Đối với mô hình Moore
 1. Viết lại bảng trạng thái với trạng thái sau mã hóa
 2. Viết bảng đầu ra
6. Đối với mô hình Mealy
 1. Viết lại bảng kết hợp trạng thái & đầu ra với trạng thái sau mã hóa.
7. Viết biểu thức Boolean cho quá trình biến đổi trạng thái và đầu ra
8. Vẽ sơ đồ nguyên lý schematic

Mã hóa trạng thái trong FSM

- Mã hóa nhị phân
 - Ví dụ, với 4 trạng thái, mã hóa thành 00, 01, 10, 11
- Mã hóa **One-hot**
 - Mỗi trạng thái ứng với 1 bit
 - Chỉ duy nhất 1 bit ở logic 1
 - Ví dụ, với 4 trạng thái, mã hóa thành 0001, 0010, 0100, 1000
 - Cần nhiều Flip-flop để chế tạo
 - Tính trạng thái tiếp theo và tính giá trị đầu ra đơn giản hơn.
- Mã hóa **One-cold** tương tự nhưng ngược lại với One-hot

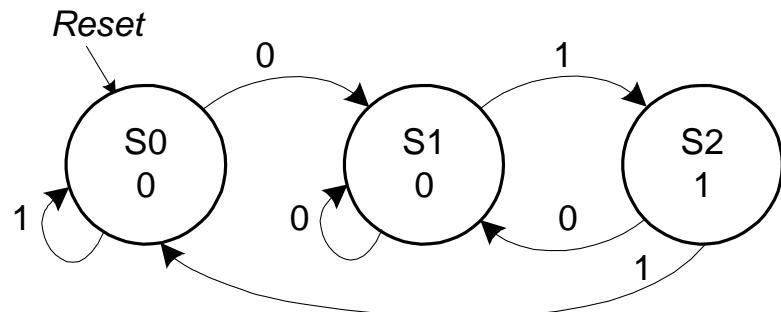
Ex: Moore vs. Mealy FSM

- Mạnh Tuấn nuôi một con sên. Nó đang bò trên một dải băng từ có ghi các bit 0 và 1. Con sên sẽ mỉm cười 😊 nếu 2 bit gần nhất mà nó đang bò lên là **01**. Hãy vẽ sơ đồ Mealy và Moore thể hiện cách nghĩ của con sên.



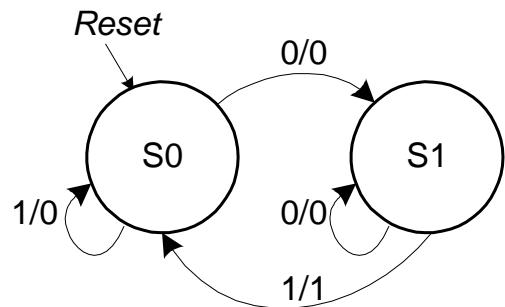
Ex: Sơ đồ chuyển trạng thái

Moore FSM



Tên	Ý nghĩa
S0	Sên đang ở bit 1, trước đó là bit 1
S1	Sên đang ở bit 0
S2	Sên đang ở bit 1, trước đó là bit 0 (cười)

Mealy FSM



Tên	Ý nghĩa
S0	Đợi sên bò lên bit 0
S1	Đợi sên bò lên bit 1

Ở sơ đồ Mealy, mũi tên có ghi giá trị theo thứ tự *đầu vào/đầu ra*

Ex: Thiết kế mạch

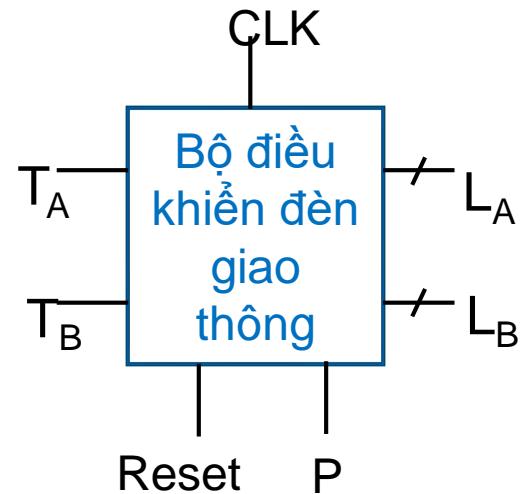
- Từ sơ đồ mealy/moore thiết kế điều khiển cho ốc sên bằng cách:
 - Vẽ trực tiếp sơ đồ Mealy/Moore trên file thiết kế dạng Block Diagram trong các IDE lập trình → biên dịch ra HDL → biên dịch file cấu hình và nạp FPGA
 - Tiếp tục biến đổi sơ đồ chuyển trạng thái thành sơ đồ schematic

Phân chia FSM

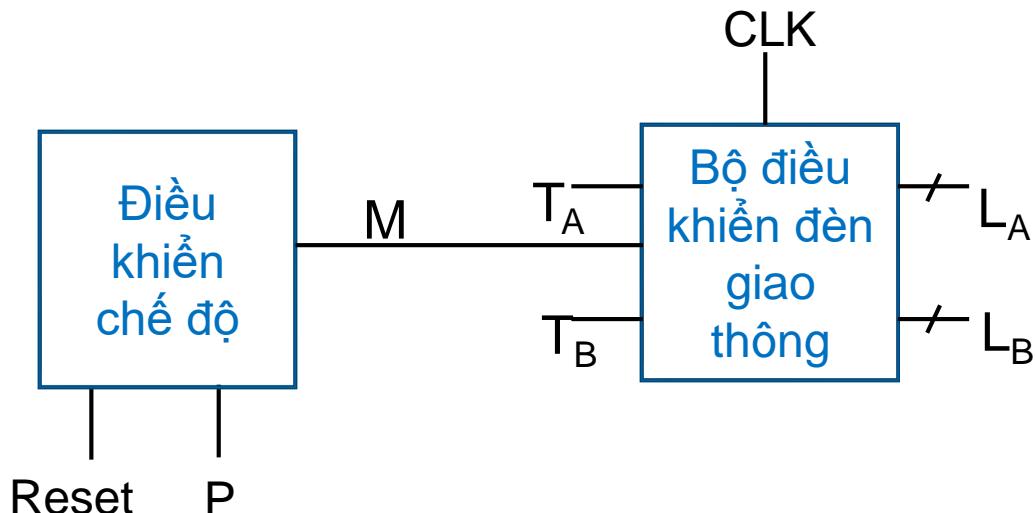
- Phân chia FSM cho phép tách một FSM phức tạp thành nhiều mô hình FSM nhỏ hơn
- Ví dụ: thay đổi hệ thống đèn giao thông để có chế độ Cấm
 - Có thêm hai đầu vào P, R
 - Khi $P = 1$, đèn giao thông chuyển sang trạng thái Cấm
 - Khi $R = 1$, trở về trạng thái bình thường

Phân chia FSM

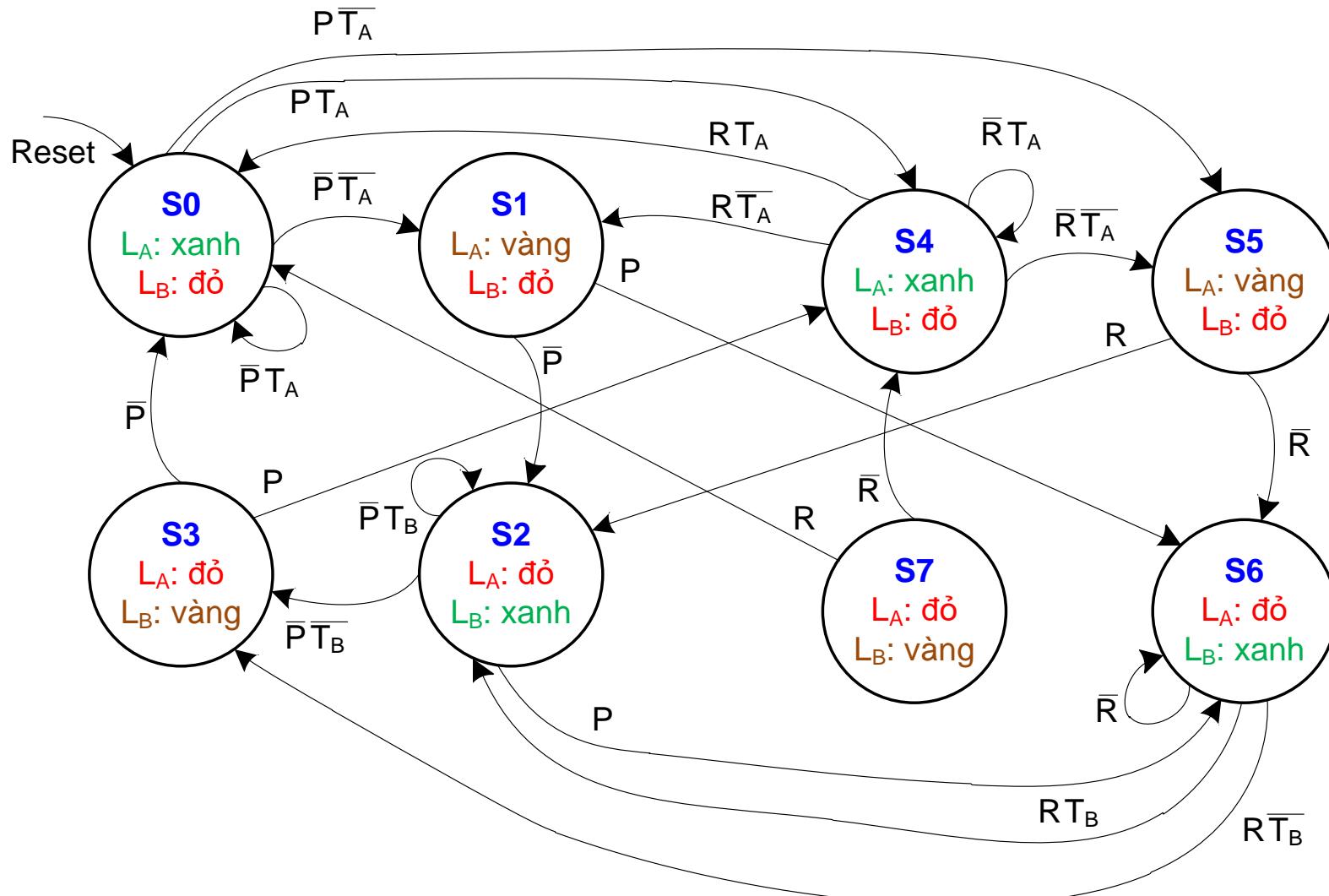
- FSM chưa phân chia



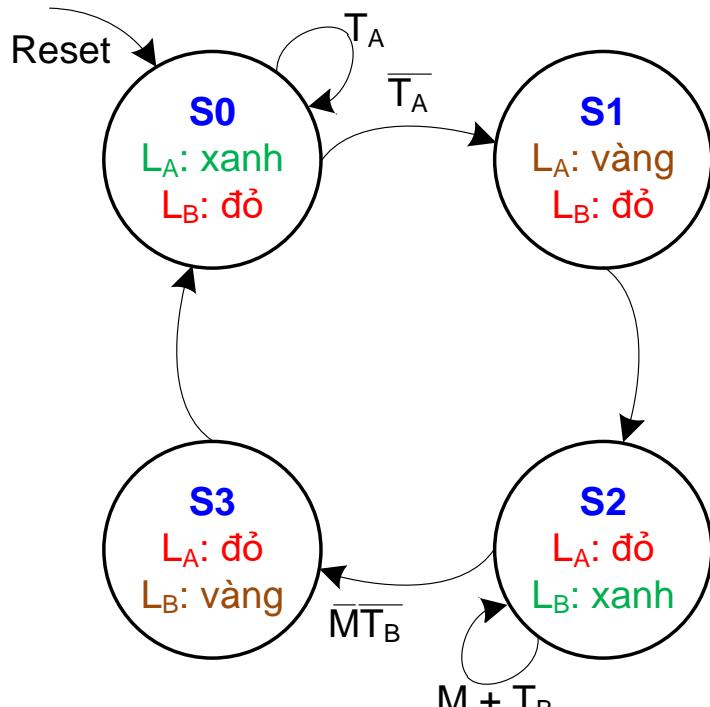
- FSM đã phân chia



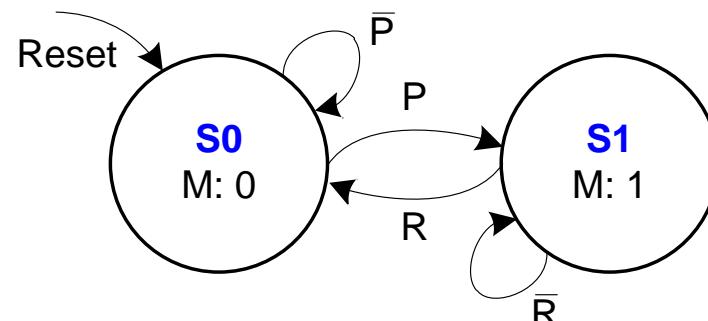
FSM chưa phân chia



FSM đã phân chia



FSM điều khiển đèn



FSM điều khiển chế độ

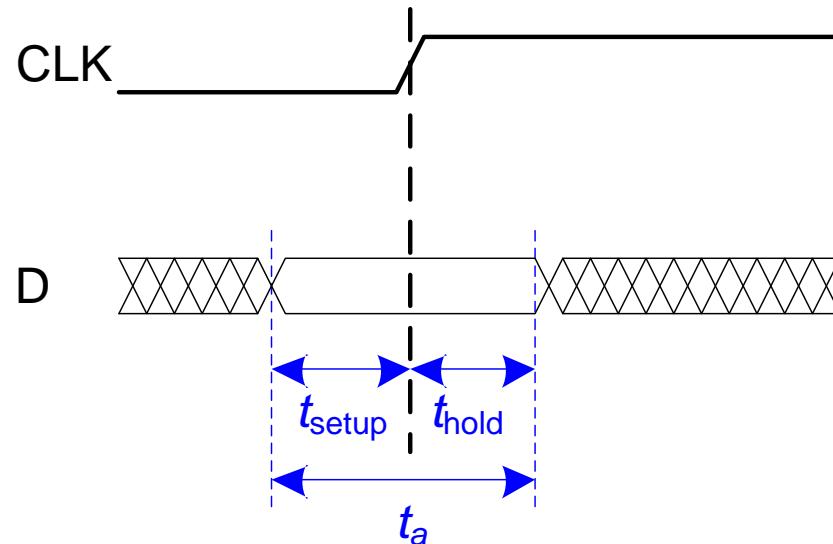
Timing (Định thời)

- Các flip-flop lấy mẫu giá trị đầu vào D tại sườn của tín hiệu đồng hồ
- Tín hiệu D phải ổn định tại thời điểm lấy mẫu
- Giống như chụp ảnh, tín hiệu D phải ổn định trong lân cận của thời điểm lấy mẫu.
- Nếu không, dữ liệu sẽ lưu trữ sai/không hợp lệ.

Ràng buộc thời gian với đầu vào

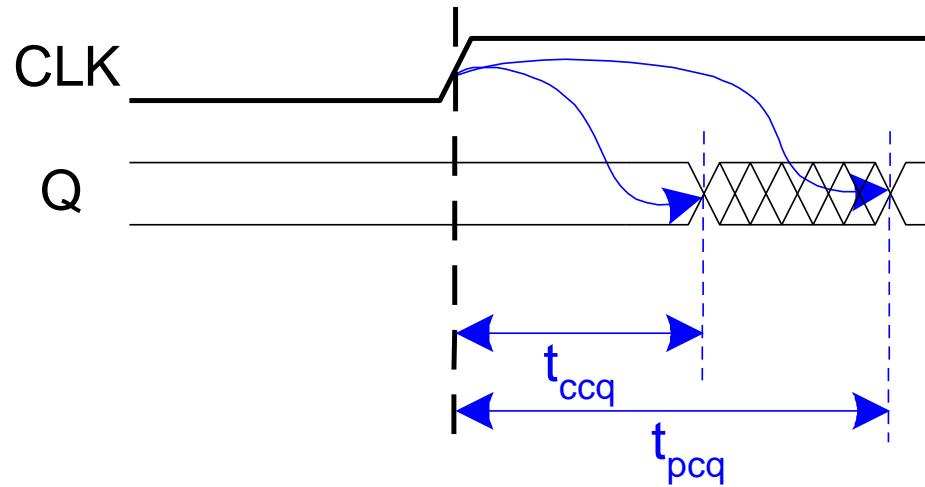
- Thời gian thiết lập: t_{setup} = khoảng thời gian dữ liệu phải ổn định (bất biến) trước khi có sườn đồng hồ
- Thời gian duy trì: t_{hold} = khoảng thời gian dữ liệu phải ổn định sau khi có sườn đồng hồ
- Thời gian mở (aperture time): t_a = khoảng thời gian dữ liệu phải ổn định quanh sườn đồng hồ

$$t_a = t_{\text{setup}} + t_{\text{hold}}$$



Ràng buộc thời gian với đầu ra

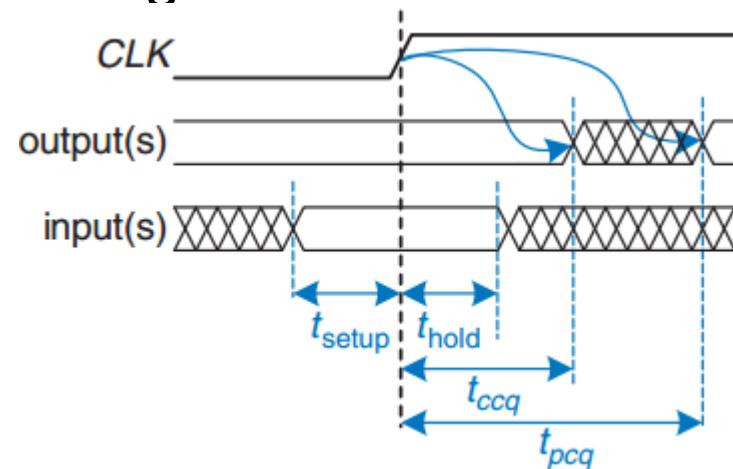
- Trễ truyền lan: t_{pcq} = khoảng thời gian từ khi có sườn đồng hồ tới khi đầu ra Q ở trạng thái ổn định
- Trễ lấy nhiệm: t_{ccq} = khoảng thời gian từ khi có sườn đồng hồ tới khi đầu ra Q bắt đầu thay đổi



Qui tắc định thời

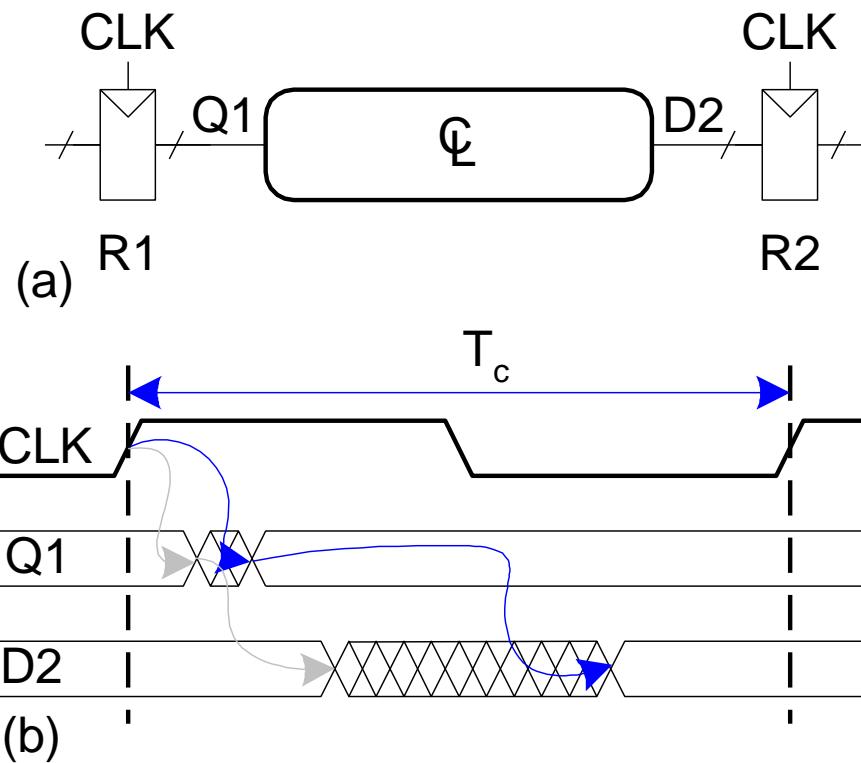
- Các đầu vào đồng bộ của mạch dãy phải ổn định trong toàn bộ khoảng thời gian mở (thiết lập và duy trì) của tín hiệu đồng hồ.
- Cụ thể hơn, tín hiệu đầu vào đồng bộ phải ổn định trong:
 - ít nhất t_{setup} trước khi có sườn đồng hồ, và
 - ít nhất t_{hold} sau khi có sườn đồng hồ

Thông số t_{pcq} , t_{setup} được cung cấp bởi nhà sản xuất



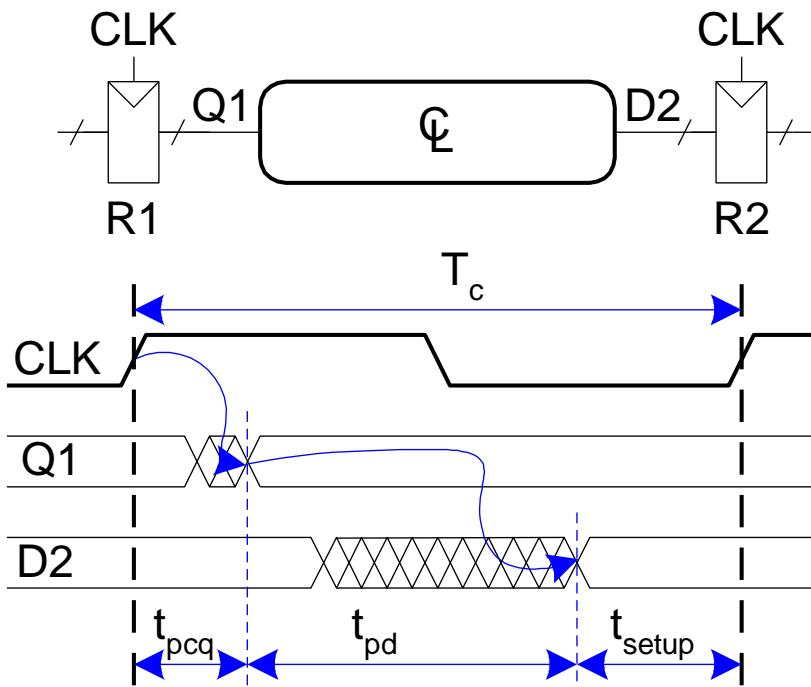
Qui tắc định thời

- Khoảng thời gian trễ giữa 2 thanh ghi có giá trị **tối thiểu** và **tối đa**, được tính theo trễ trên từng mạch thành phần.



Ràng buộc về thời gian thiết lập

- Phụ thuộc vào **độ trễ tối đa** giữa 2 thanh ghi R1, R2 và phần mạch tổ hợp ở giữa.
- Đầu vào của thanh ghi R2 (tín hiệu D2) phải đạt trạng thái ổn định ít nhất t_{setup} **trước** khi có sườn đồng hồ



$$T_c \geq \dots$$

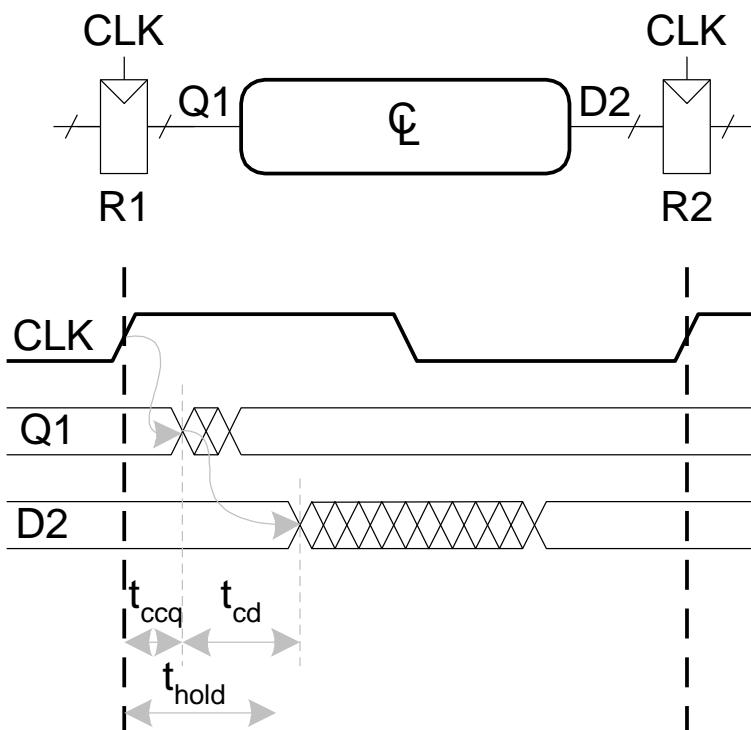
↗

$$\dots$$



Ràng buộc về thời gian duy trì

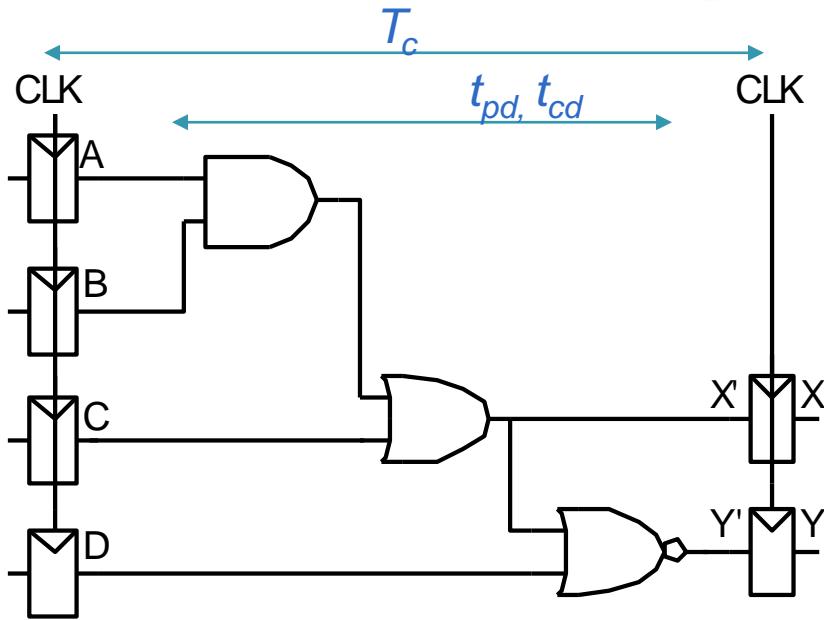
- Phụ thuộc vào **độ trễ tối thiểu** giữa 2 thanh ghi R1, R2 và phần mạch tổ hợp ở giữa.
- Đầu vào của thanh ghi R2 (tín hiệu D2) phải đạt trạng thái ổn định ít nhất t_{hold} sau khi có sườn đồng hồ



$t_{hold} <$



Phân tích thời gian



$$t_{pd} =$$

$$t_{cd} =$$

Ràng buộc thời gian thiết lập (setup)

$$T_c \geq$$

$$f_c =$$

Đặc tính thời gian

$$t_{ccq} = 30 \text{ ps}$$

$$t_{pcq} = 50 \text{ ps}$$

$$t_{\text{setup}} = 60 \text{ ps}$$

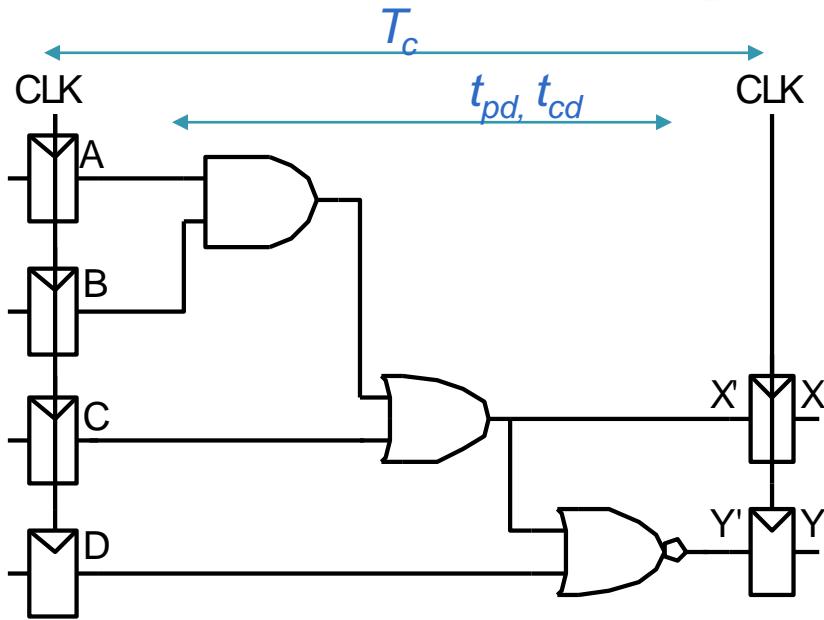
$$t_{\text{hold}} = 70 \text{ ps}$$

$$\begin{aligned} \text{mỗi gate} & [t_{pd} = 35 \text{ ps} \\ & t_{cd} = 25 \text{ ps} \end{aligned}$$

Ràng buộc thời gian duy trì (hold)

$$t_{ccq} + t_{cd} > t_{\text{hold}} ?$$

Phân tích thời gian



t_{pd} = <trễ truyền lan của mạch tổ hợp>

t_{cd} = <trễ lây nhiễm của mạch tổ hợp>

Ràng buộc thời gian thiết lập (setup)

T_c = <chu kỳ xung nhịp CLK> ≥

f_c = <tần số xung nhịp>

Đặc tính thời gian

$$t_{ccq} = 30 \text{ ps}$$

$$t_{pcq} = 50 \text{ ps}$$

$$t_{\text{setup}} = 60 \text{ ps}$$

$$t_{\text{hold}} = 70 \text{ ps}$$

mỗi gate

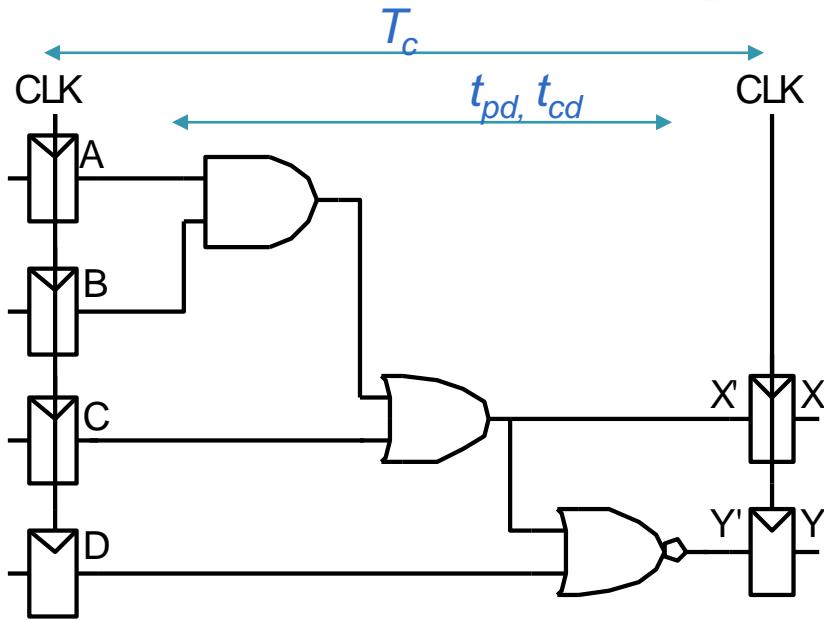
$$\begin{cases} t_{pd} = 35 \text{ ps} \\ t_{cd} = 25 \text{ ps} \end{cases}$$

Ràng buộc thời gian duy trì (hold)

$$t_{ccq} + t_{cd} > t_{\text{hold}}$$

<trễ lây nhiễm ở ABCD> + <trễ lây nhiễm của mạch tổ hợp> > <thời gian duy trì>

Phân tích thời gian



$$t_{pd} = 3 \times 35 = 105 \text{ ps}$$

$$t_{cd} = 25 \text{ ps}$$

Ràng buộc thời gian thiết lập (setup)

$$T_c \geq (50 + 105 + 60) \text{ ps} = 215 \text{ ps}$$

$$f_c = 1 / T_c = 4.65 \text{ GHz}$$

Đặc tính thời gian

$$t_{ccq} = 30 \text{ ps}$$

$$t_{pcq} = 50 \text{ ps}$$

$$t_{\text{setup}} = 60 \text{ ps}$$

$$t_{\text{hold}} = 70 \text{ ps}$$

mỗi gate

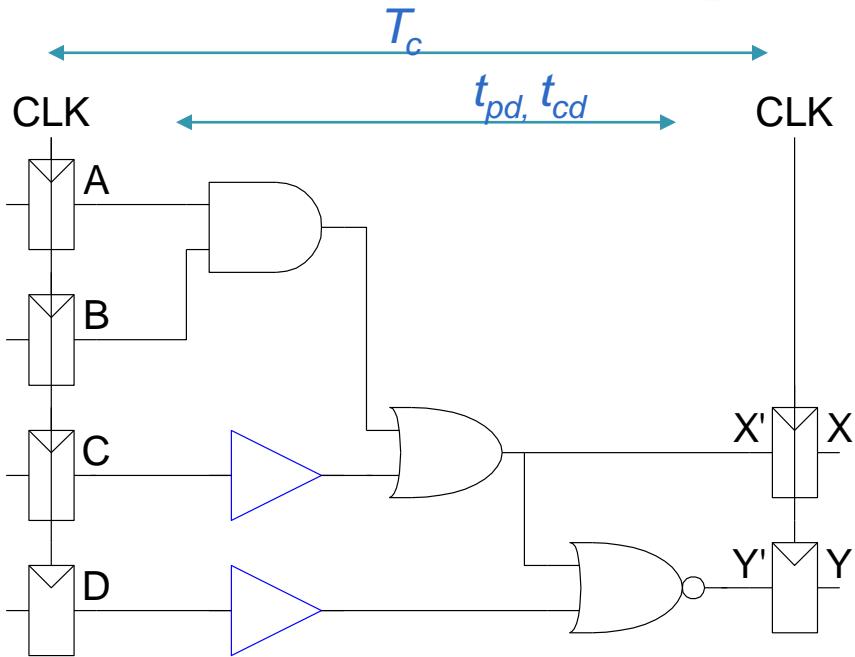
$$\begin{cases} t_{pd} & = 35 \text{ ps} \\ t_{cd} & = 25 \text{ ps} \end{cases}$$

Ràng buộc thời gian duy trì (hold)

$$t_{ccq} + t_{cd} > t_{\text{hold}} ?$$

$$(30 + 25) \text{ ps} > 70 \text{ ps} ? \text{ No!}$$

Phân tích thời gian



$$t_{pd} =$$

$$t_{cd} =$$

Ràng buộc thời gian thiết lập (setup)

$$T_c \geq$$

$$f_c =$$

Đặc tính thời gian

$$t_{ccq} = 30 \text{ ps}$$

$$t_{pcq} = 50 \text{ ps}$$

$$t_{\text{setup}} = 60 \text{ ps}$$

$$t_{\text{hold}} = 70 \text{ ps}$$

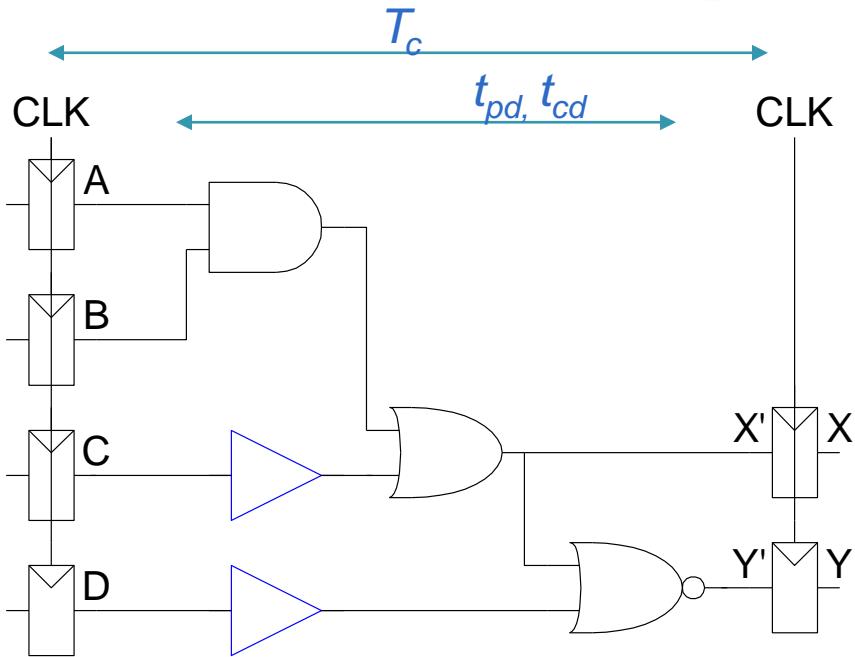
mỗi gate

$$\begin{cases} t_{pd} & = 35 \text{ ps} \\ t_{cd} & = 25 \text{ ps} \end{cases}$$

Ràng buộc thời gian duy trì (hold)

$$t_{ccq} + t_{cd} > t_{\text{hold}} ?$$

Phân tích thời gian



$$t_{pd} = 3 \times 35 = 105 \text{ ps}$$

$$t_{cd} = (25 + 25) = 50 \text{ ps}$$

Ràng buộc thời gian thiết lập (setup)

$$T_c \geq (50 + 105 + 60) \text{ ps} = 215 \text{ ps}$$

$$f_c = 1 / T_c = 4.65 \text{ GHz}$$

Đặc tính thời gian

$$t_{ccq} = 30 \text{ ps}$$

$$t_{pcq} = 50 \text{ ps}$$

$$t_{\text{setup}} = 60 \text{ ps}$$

$$t_{\text{hold}} = 70 \text{ ps}$$

mỗi gate

$$\begin{cases} t_{pd} & = 35 \text{ ps} \\ t_{cd} & = 25 \text{ ps} \end{cases}$$

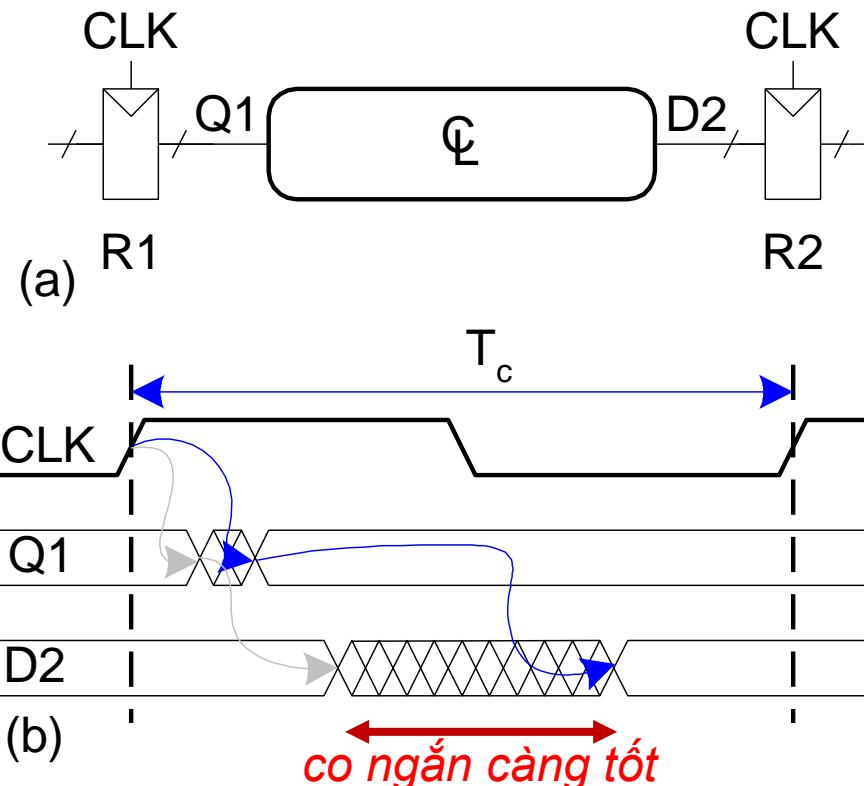
Ràng buộc thời gian duy trì (hold)

$$t_{ccq} + t_{cd} > t_{\text{hold}} ?$$

$$(30 + 50) \text{ ps} > 70 \text{ ps} ? \text{ Thỏa mãn!}$$

Phân tích thời gian

- Tính toán thold và tsetup để giảm khoảng thời gian hazard của mạch tổ hợp giữa 2 thanh ghi ($= t_{pd} - t_{cd}$)



Clock Skew – Lệch xung clock

- Bản thân xung clock được truyền đi với vận tốc hữu hạn, nên sẽ tới các thanh ghi không cùng lúc
- **Skew:** hiện tượng sai lệch giữa hai sườn clock
- Giải pháp: phải tính toán trong tình huống xấu nhất để bảo đảm các qui tắc định thời vẫn đúng với mọi thanh ghi.



Phần III:

Ngôn ngữ VHDL

Very high speed integrated circuit Hardware Description Language

- Giới thiệu các ngôn ngữ HDL (2)
- Cấu trúc mã nguồn VHDL (2)
- Mô hình lập trình VHDL
- Các phương pháp thiết kế

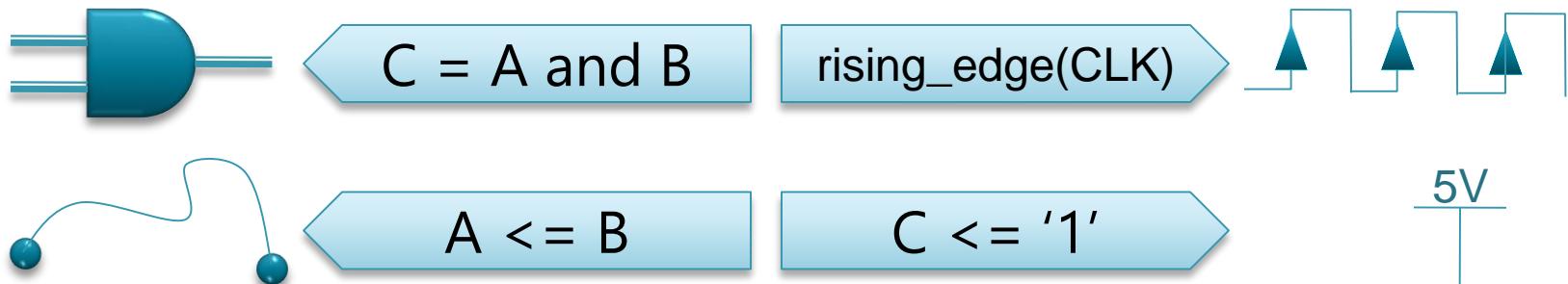


Các ngôn ngữ **HDL**

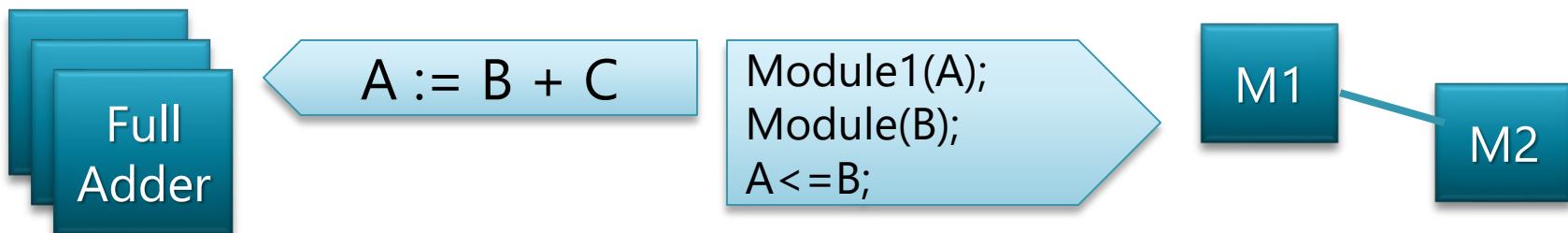
Về ngôn ngữ HDL

1/3

- Hardware Description Language, ngôn ngữ mô tả phần cứng.



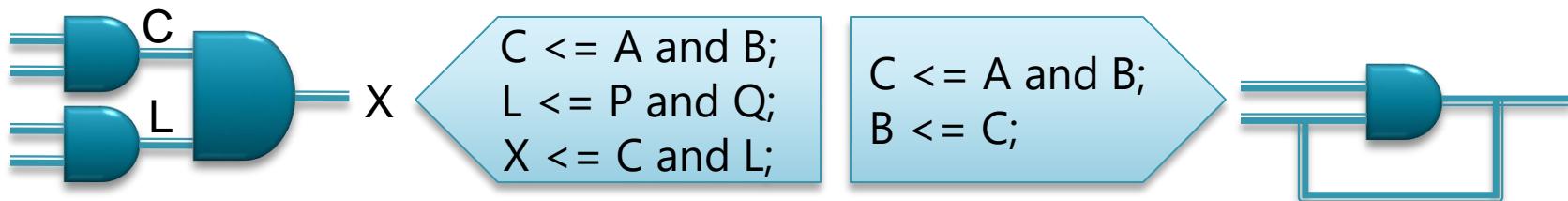
- Mọi kết cấu phần cứng đều có thể biểu diễn dưới dạng các lệnh phần mềm, các hàm và thủ tục.



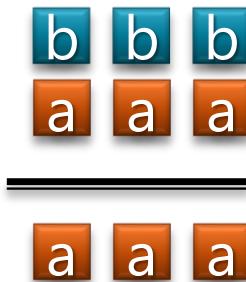
Về ngôn ngữ HDL

2/3

- Mô tả được kiến trúc xử lý tương tranh (đặc trưng riêng của phần cứng so với phần mềm).



- Sử dụng các lệnh tuần tự để mô phỏng xử lý song song của phần cứng → dễ làm quen, dễ lập trình



```
for (i=0; i<3;  
     i++)  
{  
    a[i]:= a[i] + b[i]  
}
```



thay 3
bằng J?

Xử lý tuần tự của
phần cứng biểu
diễn thế nào?



Về ngôn ngữ HDL

3/3

- Cho phép giả lập các điều kiện đầu vào để kiểm thử thiết kế phần cứng -> thực hiện test-driven, unittest.



- Khi áp dụng source HDL lên 1 chip lập trình được nào đó thì các trình giả lập có thể tính toán chính xác độ trễ truyền lan, dạng sóng... tại mỗi điểm.
- Gợi ý: nếu không chắc chắn về thiết kế mạch số từ các phần tử logic, có thể viết bằng VHDL rồi tham khảo phân tích RTL.

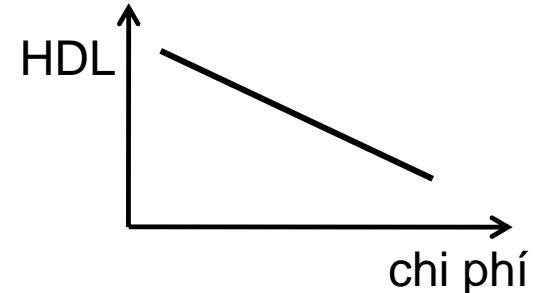
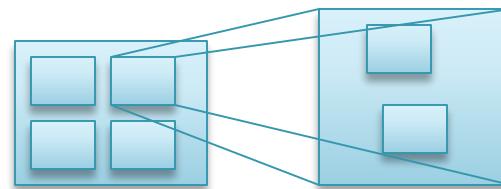
Các ngôn ngữ HDL

- Có 2 ngôn ngữ, được sử dụng rộng rãi và công nhận bởi IEEE:
 - VHDL
 - Verilog
- Các ngôn ngữ khác như AHDL, JHDL, RHDL...
- Ngoài HDL, còn có HVL – *Hardware Verification Language* như OpenVera, Superlog

Đặc trưng của HDL

- Là ngôn ngữ lập trình trừu tượng, đối lập với thiết kế mức logic, mức transistor.
- Phù hợp với phương pháp thiết kế top-down.
- Giảm thời gian thiết kế, kiểm thử, sản xuất...
- Giảm chi phí kỹ thuật không lặp lại **N**one **R**ecurring **E**ngineering.
- Tái sử dụng thiết kế.
- Dễ debug, tính tài nguyên.

$$C \leq A + B$$



? cell
? trigger

? tốc độ
? phần tử logic

Các bước sử dụng HDL

- 
- 1 Mô tả phần cứng bằng HDL.
 - 2 Kiểm thử nguyên lý bằng giả lập trên HDL.
 - 3 Kiểm tra chi tiết bằng phân tích RTL.
 - 4 Chọn IC lập trình được.
 - 5 Gắn kết các chân IO của phần cứng HDL với IC.
 - 6 Biên dịch phần cứng HDL theo IC.
 - 7 Nạp phần cứng HDL lên IC.

VHDL: Lịch sử

1/3

- *Very high speed integrated circuit Hardware Description Language*
- 1980, ngôn ngữ HDL đầu tiên, VHDL được ra đời theo mệnh lệnh của bộ quốc phòng Mỹ.
- VHDL dùng để ghi nhận cách hoạt động của các ASIC mà các công ty cung cấp sử dụng trong thiết bị quân sự
→ giống đặc tả cú pháp tài liệu.
- Bộ QP yêu cầu VHDL phải kế thừa cú pháp và các định nghĩa của ngôn ngữ Ada (mở rộng của Pascal) để tái sử dụng.



VHDL: Lịch sử

2/3

- 1983,  ,  TEXAS INSTRUMENTS, Intermetrics, góp kinh nghiệm về ngôn ngữ bậc cao và thiết kế top-down để phát triển VHDL.
- 1987, bộ QP tuyên bố: “**Tất cả mạch điện tử số đều được mô tả trong VHDL**”.
VHDL cũng được công nhận là chuẩn IEEE 1076.
Mọi hệ thống con điện tử của máy bay F-22 đều đặc tả bởi VHDL.
- Lúc này, VHDL đã là một chuẩn công nghiệp, nhưng ít công cụ hỗ trợ.



VHDL: Lịch sử

3/3

- 1996, các công cụ giả lập logic, có thể hiểu cú pháp VHDL xuất hiện nhanh chóng.
- Sau đó, các công cụ tổng hợp, với đầu vào là VHDL, đầu ra là mô tả mạch phần cứng thực thi.
- Chuẩn [IEEE 1164](#) được bổ sung, với nhiều kiểu logic đa giá trị khác nhau (U, Z, X...)
(Sẽ được thường xuyên sử dụng trong code VHDL)
- 01/2009, VHDL 4.0, còn gọi là VHDL 2008, là chuẩn IEEE

VHDL
library IEEE;
IEEE.std_logic_1164.all;

[VHDL wiki](#)

Tính chất của VHDL

1/2

- Có tính định kiểu mạnh.
- Không phân biệt chữ hoa, thường.
- Cho phép tạo mảng với index tăng dần hoặc giảm dần. Ví dụ: bienx : out STD_LOGIC_VECTOR(2 downto 1)
- Có thể đọc/ghi file, thường dùng để giả lập, tạo số liệu vào, và thẩm định kết quả.
- Một người ít kinh nghiệm cũng có thể viết VHDL và giả lập thành công, nhưng không chắc đã tổng hợp được lên một thiết bị vật lý, hoặc vượt quá khả năng thực tế.
- Người dùng có thể dùng các VHDL IDE như Altera Quatus, Xilinx ISE... để tạo sơ đồ RTL.

Tính chất của VHDL

2/2

- Có thể chạy giả lập để xem dạng sóng tín hiệu với các testbench tương ứng.
- Khi source VHDL được dịch dưới dạng “cổng và kết nối”, tức là đã được ánh xạ lên một thiết bị vật lý như CPLD, FPGA, thì đó chính là một phần cứng thực sự.
- Phù hợp với thiết kế mức hệ thống, vì cho phép kiểm tra hành vi của hệ thống, giả lập mà không cần thông qua công cụ tổng hợp lên phần cứng.
- Mô tả được tính tương tranh của phần cứng - *concurrent system*.
- 1 mã nguồn VHDL phù hợp với nhiều phần cứng.

Verilog: Lịch sử

1/3

- 1983~1984, Phil Moorby và Prabhu Goel đã phát triển xong một ngôn ngữ HDL, và được Gateway Design Automation công bố với tên **Verilog**, cùng trình mô phỏng Verilog.
- 1985, ra mắt bản nâng cấp của ngôn ngữ và trình mô phỏng Verilog XL, có thể mô phỏng thiết kế trên 1 triệu cổng, dễ dàng debug, mô phỏng phần cứng và cả tác nhân kích thích.
- 1987, Verilog thỏa mãn mong đợi của các kỹ sư thiết kế và ngày càng phổ biến.
- 1989, cadence™ mua lại Gateway.

Verilog: Lịch sử

2/3

- 1990, Cadence chia ngôn ngữ Verilog và trình mô phỏng Verilog-XL thành các sản phẩm riêng biệt.
→ công khai Verilog thành chuẩn mở, cạnh tranh với VHDL.
→ Hầu hết các xưởng chế tạo ASIC đều hỗ trợ Verilog và sử dụng Verilog-XL làm trình mô phỏng.
- 1993, 85% thiết kế và thẩm tra ASIC dùng Verilog.
- 1995, trở thành chuẩn IEEE 1364, hay còn gọi là Verilog-95
Cùng lúc, Cadence phát triển Verilog-A dành cho mạch tương tự, và là một bộ phận của Verilog-AMS (**A**nalog and **M**ixed-**S**ignal)

Verilog: Lịch sử

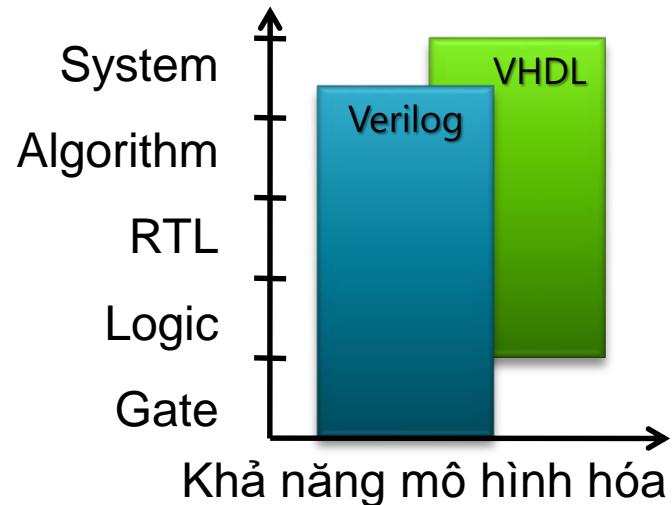
3/3

- 2005, là chuẩn IEEE 1634-2005, còn gọi là Verilog 2005.
- SystemVerilog là tập bao quát hơn, gồm cả Verilog 2005 bên trong, với nhiều tính năng mới và khả năng hỗ trợ thẩm định thiết kế, mô hình hóa thiết kế (nhờ tích hợp **Hardware Verification Language**).
- 2009, SystemVerilog và Verilog đều được hợp nhất lại trong IEEE 1800-2009, còn gọi là SystemVerilog 2009.

VHDL vs Verilog

1/4

- Khả năng trừu tượng



- Kiểu dữ liệu

- VHDL: có nhiều kiểu, hàm chuyển kiểu, người dùng tự định nghĩa kiểu.
- Verilog: kiểu đơn giản, gần với phần cứng, người dùng không tự định nghĩa kiểu được.

- Tái sử dụng thiết kế

- VHDL: các procedure và function có thể đặt trong một package và có thể được dùng lại trong thiết kế khác.
- Verilog: các procedure và function phải đặt trong một module, trong các file riêng rẽ và được bao gồm bằng cách sử dụng chỉ dẫn biên dịch include.

- Tính dễ học

- VHDL: do tính định kiểu mạnh nên có lợi cho người nhiều kinh nghiệm, và có nhiều cách để mô hình cùng một mạch số.
- Verilog: tương đối dễ học với người mới bắt đầu. Tuy nhiên, các chỉ định biên dịch và ngôn ngữ PLI đính kèm lại không đơn giản.

Định thời

- Verilog cho khả năng liên kết thông tin giữa công cụ layout với công cụ tổng hợp/mô phỏng tốt hơn VHDL, nên khả năng biểu diễn định thời chính xác hơn.

Thư viện

- VHDL: thư viện là miền lưu trữ trong môi trường máy chủ các thực thể (entity), kiến trúc (architecture), gói (package) và cấu hình (configuration) biên dịch được.
- Verilog: không tồn tại.

Tính dễ đọc

- VHDL: giống ngôn ngữ Ada, Pascal.
- Verilog: giống ngôn ngữ C (50%C + 50%Ada)

VHDL vs Verilog

4/4

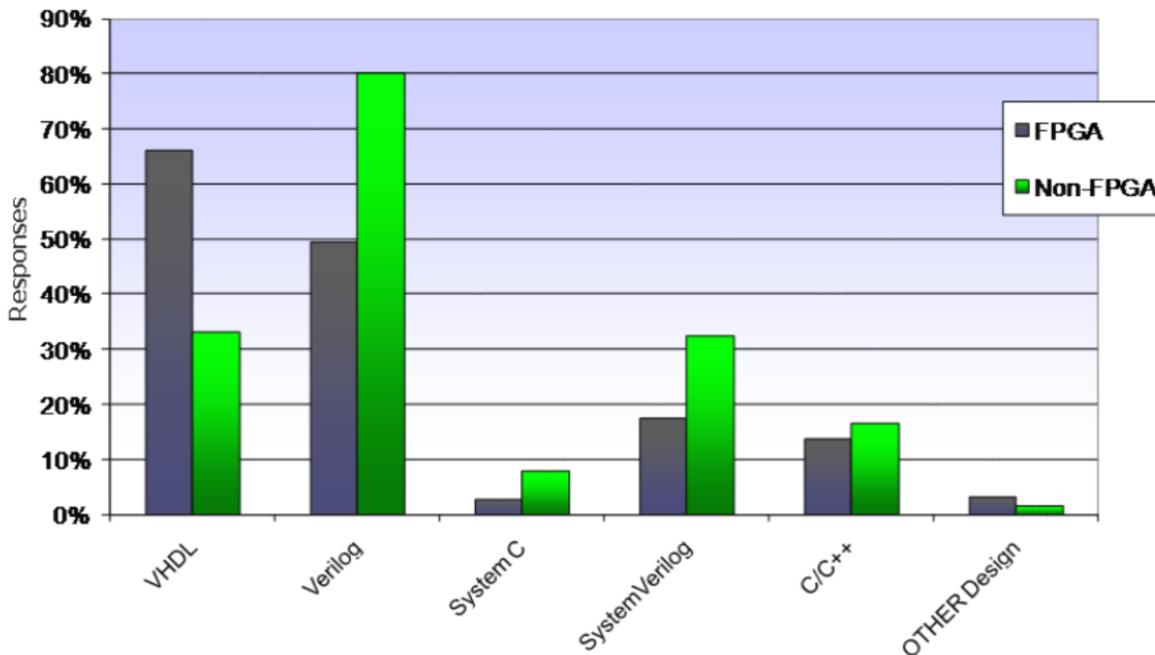
- Mô hình thông số hóa
 - VHDL, Verilog đều cho phép. Ví dụ, một đối tượng tổng quát n-bit, rồi được cụ thể hóa với n = 8.
- Sao chép cấu trúc
 - VHDL: phát biểu generate sẽ tạo ra bản sao của các instance của cùng một đơn vị thiết kế hoặc một phần của thiết kế và kết nối một cách thích hợp.
 - Verilog: không có phát biểu tương ứng.

Các xu hướng

1/4

Languages and Libraries

Languages used for design



Combined FPGA & Non-FPGA Designs

1 HF - Compilation and Analysis performed in January 2011

Wilson Research Group and Mentor Graphics
2010 Functional Verification Study, Used with permission

© 2011 Mentor Graphics Corp. Company
www.mentor.com

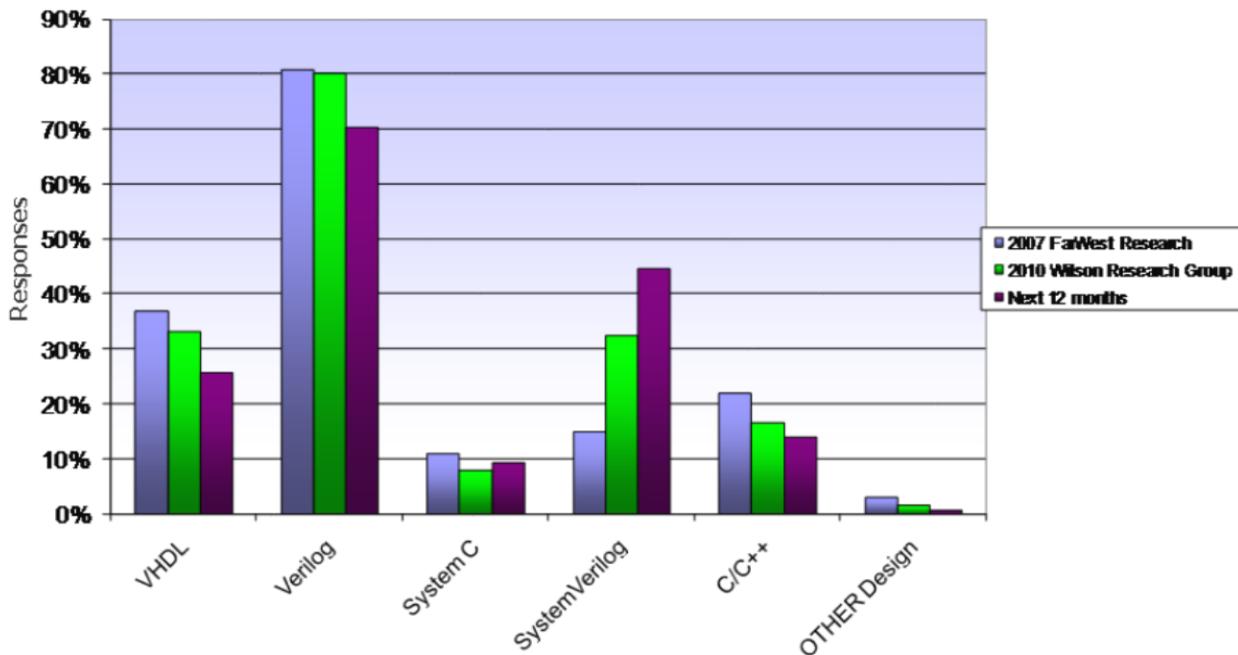
<http://blogs.mentor.com/verificationhorizons/blog/tag/vhdl/>

Các xu hướng

2/4

Languages and Libraries

Trends: Languages used for *design*



Non-FPGA Designs

2 HF - Compilation and Analysis performed in January 2011

Wilson Research Group and Mentor Graphics
2010 Functional Verification Study, Used with permission

© 2011 Mentor Graphics Corp. Company
www.mentor.com

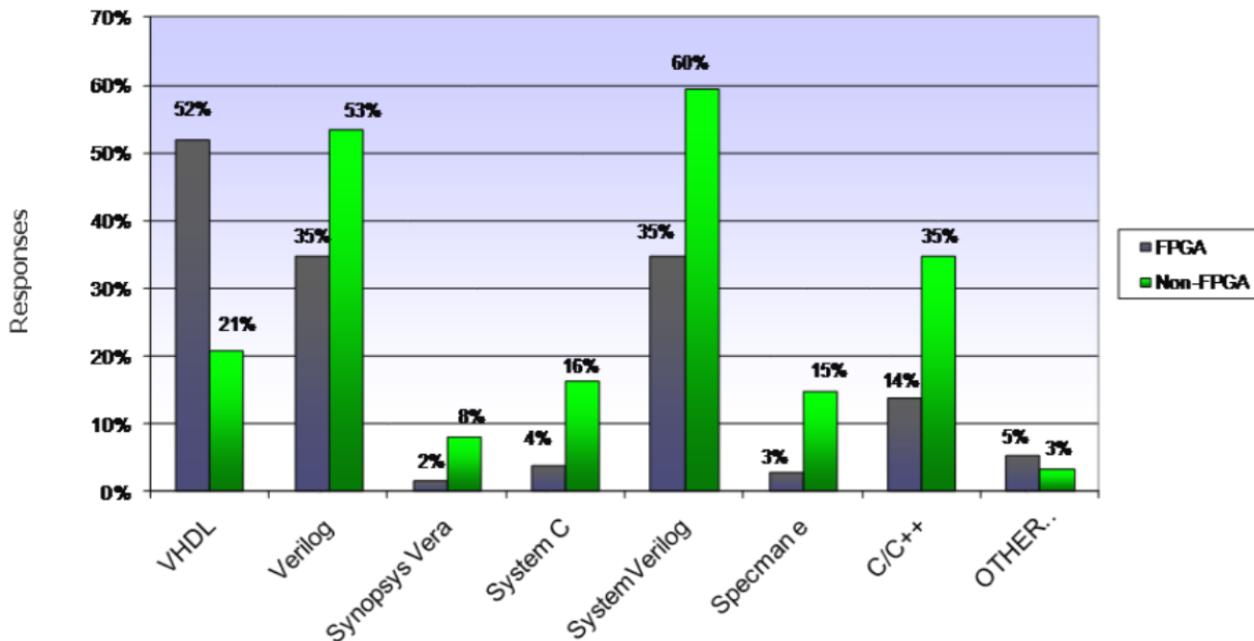
<http://blogs.mentor.com/verificationhorizons/blog/tag/vhdl/>

Các xu hướng

3/4

Languages and Libraries

Languages used for verification (testbenches)



Combined FPGA & Non-FPGA Designs

3 HF - Compilation and Analysis performed in January 2011

Wilson Research Group and Mentor Graphics
2010 Functional Verification Study, Used with permission

© 2011 Mentor Graphics Corp. Company
www.mentor.com

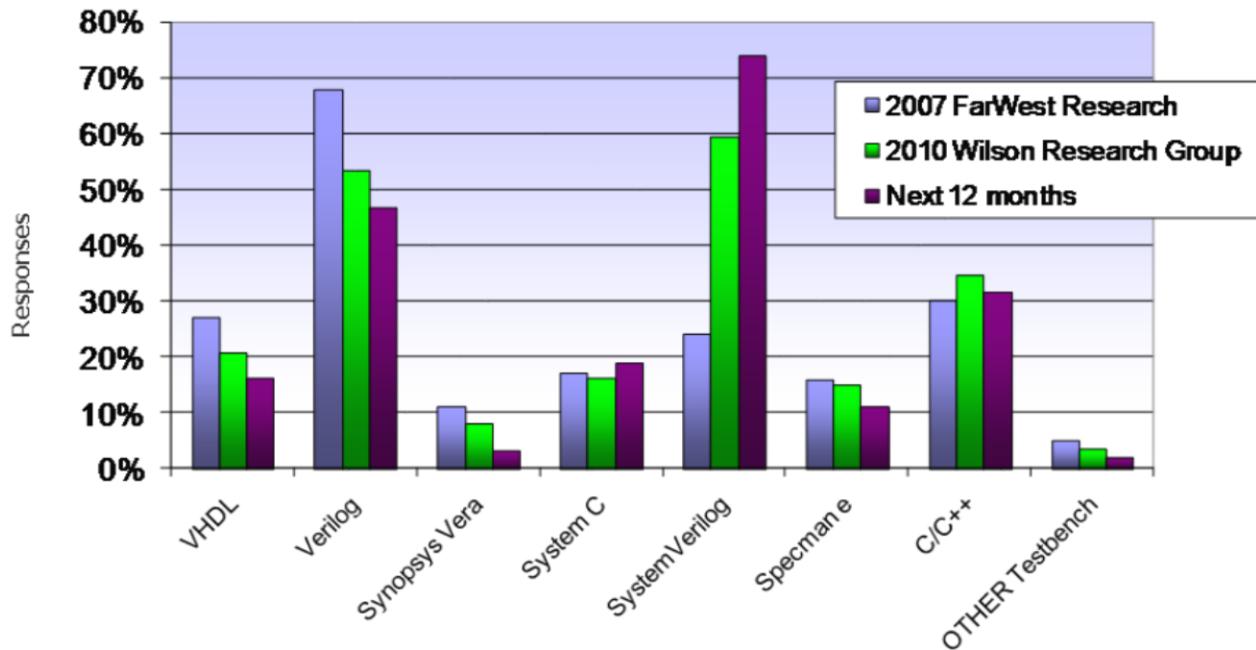
<http://blogs.mentor.com/verificationhorizons/blog/tag/vhdl/>

Các xu hướng

4/4

Languages and Libraries

Trends: Languages used for *verification* (testbenches)



Non-FPGA Designs

4 HF - Compilation and Analysis performed in January 2011

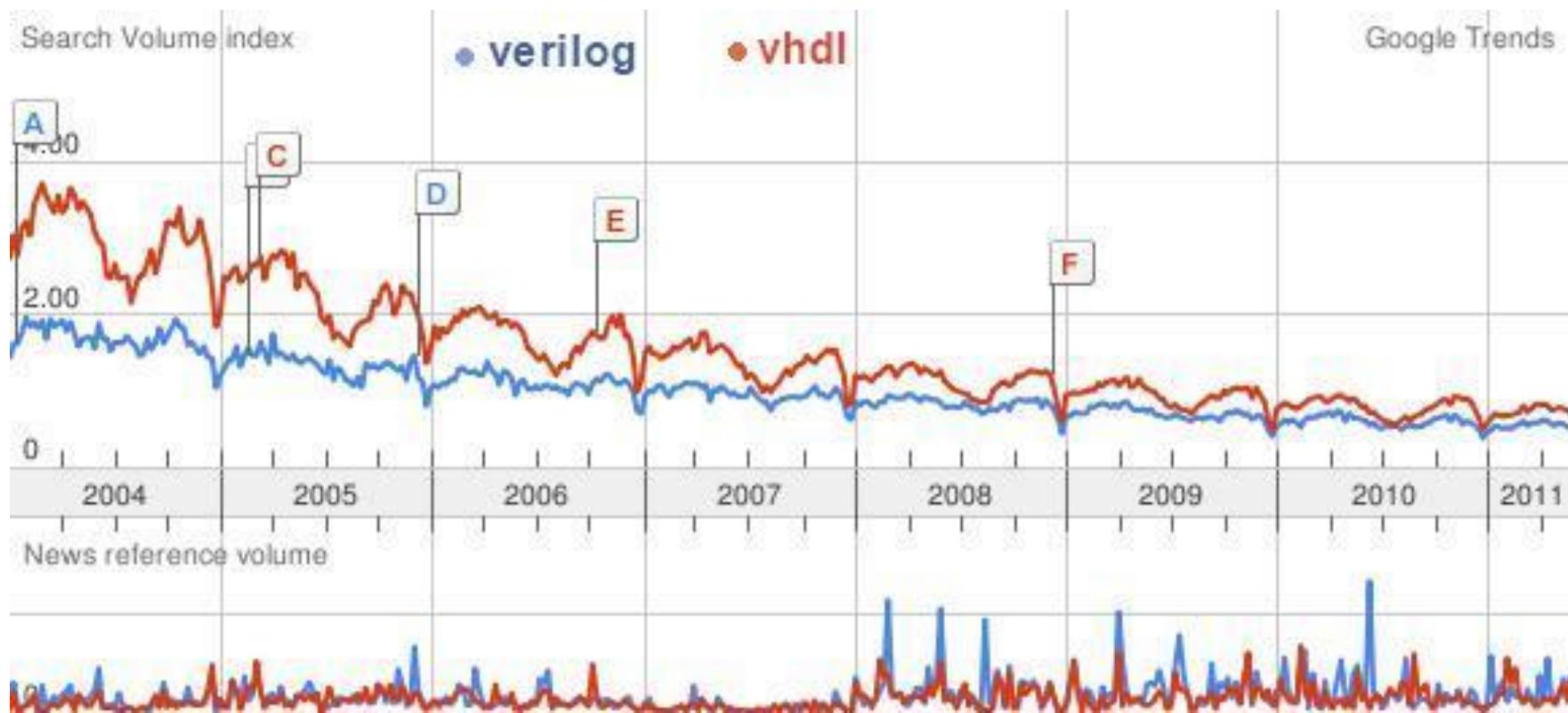
Wilson Research Group and Mentor Graphics
2010 Functional Verification Study, Used with permission

© 2011 Mentor Graphics Corp. Company
www.mentor.com

<http://blogs.mentor.com/verificationhorizons/blog/tag/vhdl/>

VHDL vs Verilog

- Số lượng tìm kiếm về VHDL nhiều hơn Verilog, nhưng sự khác biệt đang giảm dần.





VHDL: Cấu trúc mã lệnh

Thành phần cơ bản của VHDL

- Mã lệnh VHDL gồm 3 phần cơ bản sau:

LIBRARY

- thư viện
- gồm danh sách các thư viện sử dụng trong thiết kế
(VD: ieee, std, work, ...)

ENTITY

- thực thể
- mô tả các chân vào-ra của mạch
- **component** là các thể hiện vật lý

ARCHITECTURE

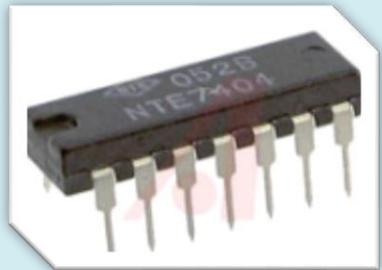
- kiến trúc
- mô tả hoạt động của mạch.
- một Entity có thể có 1 hoặc nhiều Architecture

Một đoạn VHDL đơn giản

Khai báo thư viện

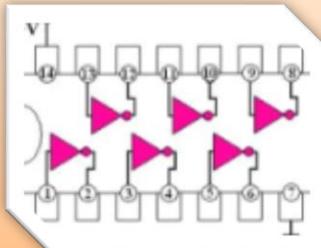
```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;
```

Phần vỏ bên ngoài



```
entity Mach_AND is  
port(  
    a : in STD_LOGIC;  
    b : in STD_LOGIC;  
    o : out STD_LOGIC  
);  
end Mach_AND;
```

Hoạt
động
xử lý



```
architecture Cach_Hoat_Dong_01 of Mach_AND is  
begin  
    o <= a and b;  
end Cach_Hoat_Dong_01;
```

Đoạn code VHDL và code C

Khai báo thư viện
#include <stdio.h>

```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;
```

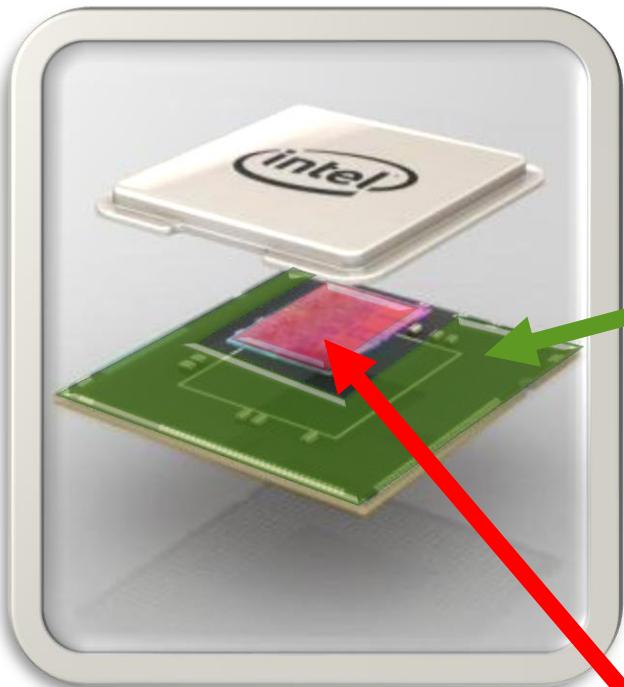
Định nghĩa giao diện
void Mach_AND(
 bool a,
 bool b,
 bool & o
);

```
entity Mach_AND is  
    port(  
        a : in STD_LOGIC;  
        b : in STD_LOGIC;  
        o : out STD_LOGIC  
    );  
end Mach_AND;
```

Hoạt động xử lý
void Mach_AND(.....)
{ *o = a & b;*
}

```
architecture Cach_Hoat_Dong_01 of Mach_AND is  
begin  
    o <= a and b;  
end Cach_Hoat_Dong_01;
```

Đoạn code VHDL,die&leadframe



```
library IEEE;  
use IEEE.STD_LOGIC_1164.all;
```

```
entity Mach_AND is  
port(  
    a : in STD_LOGIC;  
    b : in STD_LOGIC;  
    o : out STD_LOGIC  
);  
end Mach_AND;
```

```
architecture Cach_Hoat_Dong_01 of Mach_AND is  
begin  
    o <= a and b;  
end Cach_Hoat_Dong_01;
```

Đoạn code VHDL và đa kiến trúc



Nem chua (**entity**):

- Thanh Hóa (**architecture**)
- Hà Tây (**architecture**)
- Lai Vung (**architecture**)

Kiến trúc hoạt động thứ 1,
do công ty A phát triển.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all
entity Mach_AND is
    port(
        a  in STD_LOGIC;
        b  in STD_LOGIC;
        o  out STD_LOGIC
    );
end Mach_AND;

architecture Cach_Hoat_Dong_01 of Mach_AND is
begin
    o <= a and b;
end Cach_Hoat_Dong_01;
```

Kiến trúc hoạt động thứ 2,
do công ty B phát triển.

```
architecture Cach_Hoat_Dong_02 of Mach_AND is
begin
    o <= not ( (not a) or (not b) );
end Cach_Hoat_Dong_02;
```



Qui định cách đặt tên đối tượng

- Nên khai báo tên gợi nhớ
- VHDL không phân biệt chữ hoa, thường
- Đặt tên của thực thể phải tuân theo các quy tắc:
 - Khai báo trên một dòng duy nhất
 - Phải bắt đầu bằng chữ cái
 - Chỉ có thể bao gồm chữ cái, chữ số và dấu gạch dưới
 - Dấu gạch dưới (underscores) không được sử dụng ở đầu, ở cuối hoặc sử dụng hai dấu liên tiếp nhau
 - Không cho phép sử dụng khoảng trắng
 - Không được sử dụng các từ khóa

Chú thích trong VHDL

- Chú thích theo dòng:
 - Bắt đầu với 2 dấu trừ: --
- Chú thích theo khối
 - Không có

```
/*
 * @brief      Chon kenh 2x1
 * @details    Khong co tin hieu CE
 */
entity MUX2x1 is
    port (
        I1  : in std_logic;      --! Kenh vao I0
        I0  : in std_logic;      --! Kenh vao I1
        SEL : in std_logic;      --! Tin hieu chon kenh
        O   : out std_logic;     --! Kenh dau ra
    );
end entity MUX2x1;
```



VHDL: Library, thư viện

Library là gì?

- **Library**, thư viện là kho chứa các định nghĩa hoặc các thiết kế đã có sẵn, có thể sử dụng ngay trong thiết kế mới.
- Trong một library, có nhiều package. Mỗi **package** là một nhóm các định nghĩa hoặc thiết kế có liên quan tới nhau.

- Cú pháp:

```
LIBRARY library_name;  
USE     library_name.package_name.package_parts;
```

- Gần như mọi project đều sử dụng thư viện IEEE:

```
LIBRARY ieee;  
USE     ieee.std_logic_1164.all;  
USE     ieee.std_logic_arith.all;  
USE     ieee.std_logic_unsigned.all;  
--USE   ieee.std_logic_signed.all;
```

Các thư viện thường dùng

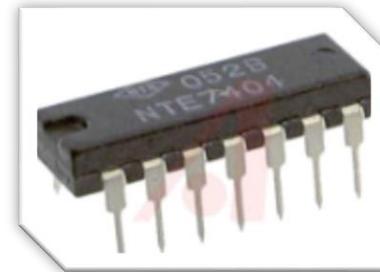
- Sử dụng thư viện **ieee** khi dùng các kiểu dữ liệu STD_LOGIC, STD_ULOGIC, ...
- Thư viện **ieee** gồm các gói (package):
 - *std_logic_1164*: các mức logic STD_LOGIC và STD_ULOGIC
 - *std_logic_arith*:
 - Các kiểu dữ liệu SIGNED và UNSIGNED
 - Các thuật toán xử lý dữ liệu và các phép toán so sánh
 - Các hàm chuyển đổi dữ liệu
 - *std_logic_signed*: cho phép xử lý dữ liệu STD_LOGIC_VECTOR như là dữ liệu kiểu SIGNED
 - *std_logic_unsigned*: cho phép xử lý dữ liệu STD_LOGIC_VECTOR như là dữ liệu kiểu UNSIGNED



VHDL: Entity

Entity là gì?

- Entity, thực thể là một khai báo, **đặc tả giao diện** của đối tượng thiết kế, qua đó, thực thể này trao đổi với thực thể khác trong cùng môi trường.
- Entity cho biết IC có các chân giao tiếp gì, nhưng không cho biết cách thực hiện
- Thông qua entity, các thiết kế có thể triệu gọi lẫn nhau



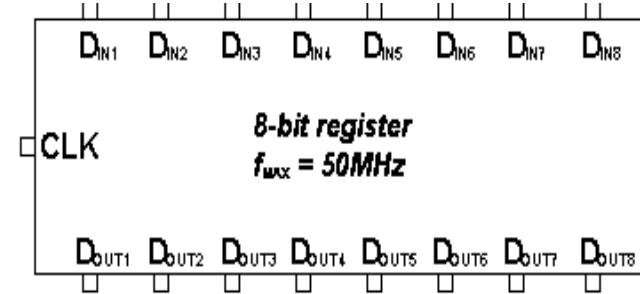
Giống như khai báo class trong C

```
class Mach_And
{
    int a;
    int CachHoatDong(int);
};
```

Hai thành phần bên trong Entity

- Mỗi Entity bao gồm hai thành phần

1. Parameter: các thuộc tính, thông số cấu hình.
Không bắt buộc.
2. In/Out: các đường kết nối vào, ra với bên ngoài.
Bắt buộc.



```
entity Eight_bit_register is
    LENGTH=8
    fmax= 50 MHz
    parameters
        D_IN    vào, độ rộng LENGTH bit
        D_OUT   ra, độ rộng LENGTH bit
        CLK     vào, độ rộng 1 bit
    connections
end entity Eight_bit_register;
```

Cú pháp khai báo Entity

- Cú pháp:

```
ENTITY entity_name IS
    GENERIC ( -- phần parameter
        generic_name : generic_type := <Giá trị mặc định>;
        ... );
    PORT ( -- phần in/out, đặc tả vào ra
        port_name : signal_mode signal_type;
        port_name : signal_mode signal_type;
        ... );
END ENTITY entity_name;
```

- Trong đó:

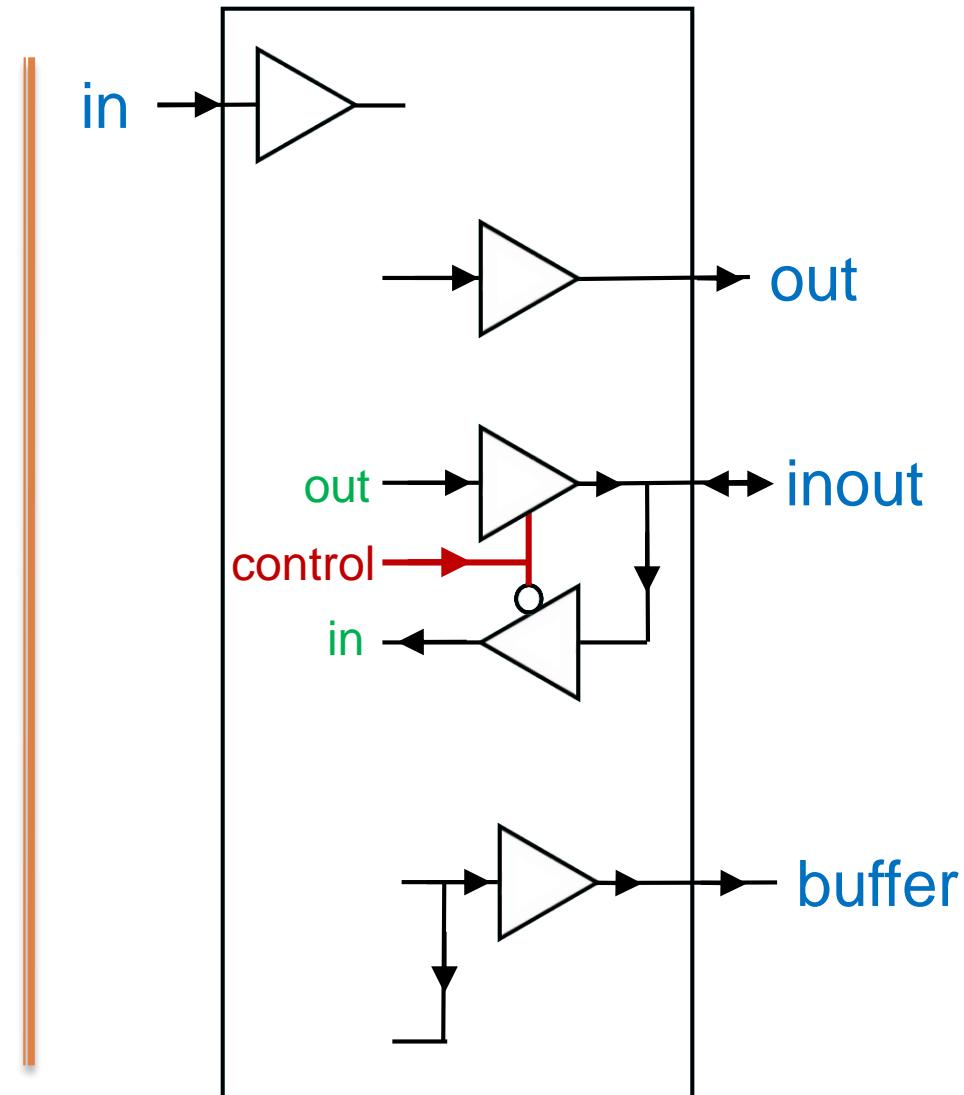
- **signal mode**: IN, OUT, INOUT, BUFFER
- **signal type**: STD_LOGIC, STD_LOGIC_VECTOR...
- **generic type**: integer, time...

Khai báo port trong Entity

- Danh sách các port được khai báo trong cấu trúc **port** (.....);
- Mỗi port được khai báo như sau:
port_name : **signal_mode** **signal_type**;
- Tên cổng, theo sau là dấu hai chấm “:”
- **signal_mode**: chế độ hoạt động:
 - IN, OUT, INOUT, BUFFER
- **signal_type**: loại cổng, kiểu dữ liệu:
 - STD_LOGIC, STD_LOGIC_VECTOR...
- Không có dấu ; ở sau khai báo port cuối cùng

Signal mode: hướng vào/ra của tín hiệu

- in: tín hiệu vào
- out: tín hiệu ra
- inout: tín hiệu vào ra 2 chiều bán song công, có tín hiệu điều khiển chọn chiều.
- buffer: tín hiệu ra, có thêm đường dẫn hồi tiếp bên trong.



Khai báo generic trong Entity

- Là hằng số, được sử dụng để mô tả một thông số của hệ thống
 - Kích thước của hệ thống (độ rộng bus, ...)
 - Tắt bật chức năng nào đó
 - Số phiên bản, version
- Mỗi parameter khai báo trong generic như sau:
`generic_name : generic_type := <Giá trị mặc định>;`
 - Tên của thông số, theo sau là dấu hai chấm ":"
 - **generic type**: kiểu dữ liệu:
 - integer, time...
 - Giá trị mặc định: có thể có hoặc không, viết sau dấu gán :=
 - Liên quan tới việc khởi tạo component
 - Ví dụ: := 5
- Không có dấu ; ở sau khai báo parameter cuối cùng

Cách kết thúc khai báo Entity

- Sử dụng câu lệnh **end**

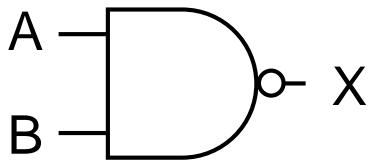
- Cách 1: **end entity;**
- Cách 2: **end entity_name; (end ExampleEnt;)**
- Cách 3: **end entity entity_name; -> Nên dùng**
- Cách 4: **end; -> Không nên dùng**

```
entity ExampleEnt is
    generic( ... );
    port ( ... );
```

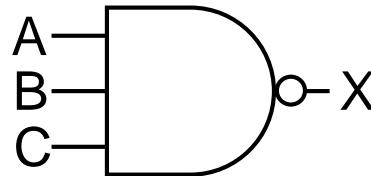
► ????|

Ví dụ 1: Entity của cổng AND

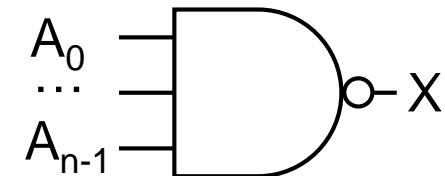
cổng NAND 2 đầu vào



cổng NAND 3 đầu vào



cổng NAND n đầu vào



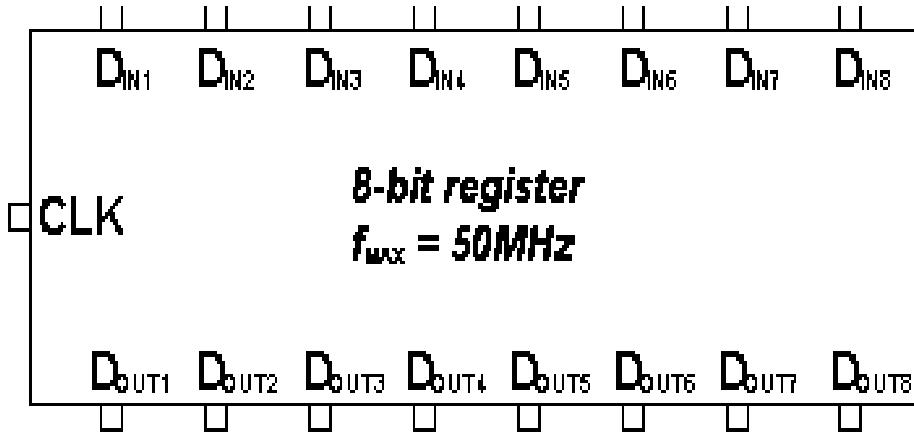
```
entity nand_gate is
port (
    A, B : in std_logic;
    X    : out std_logic
);
end entity nand_gate;
```

```
entity nand_gate is
port (
    A, B, C : in std_logic;
    X       : out std_logic
);
end entity nand_gate;
```

```
entity nand_gate is
generic (n: integer);
port (
    A : in std_logic_vector
        (n-1 downto 0);
    X : out std_logic
);
end entity nand_gate;
```

parameter giúp cho thiết kế có tính linh hoạt, dễ mở rộng

Ví dụ 2: Entity của thanh ghi



```
entity eight_bit_register is
    generic (
        LENGTH : integer := 8 -- Giá trị mặc định = 8
        Fmax: integer := 50      -- Đơn vị MHz
    );
    port(
        CLK   : in STD_LOGIC;
        DIN   : in STD_LOGIC_VECTOR(LENGTH downto 1);
        DOUT : out STD_LOGIC_VECTOR(LENGTH downto 1)
    );
end entity eight_bit_register;
```



VHDL: Architecture

Architecture là gì?

- **Architecture**, kiến trúc của một **Entity**, cho biết cách hoạt động của **Entity** đó.
- Giống phần thân của hàm trong class của C++.

```
void Mach_And.CachHoatDong(bool a, bool b, bool &o)
{
    o = a & b;
}
```
- Mỗi **Entity** phải đi kèm với ít nhất 1 **Architecture**.
- **Đa kiến trúc**: mỗi **Entity** có thể có nhiều **Architecture**, giống như kỹ thuật chồng hàm của C++. Việc kiến trúc nào được dùng, sẽ tùy vào việc tạo **Component**.

[Đa kiến trúc trong VHDL](#)

Cú pháp Architecture

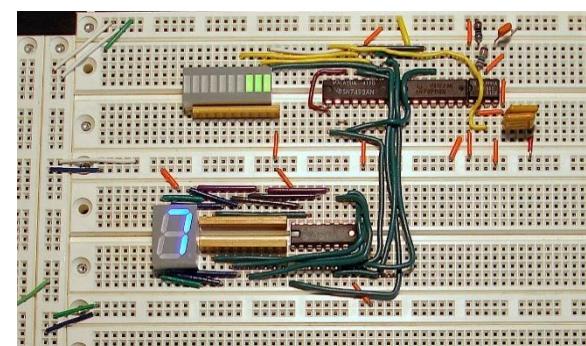
- Cú pháp:

```
ARCHITECTURE architecture_name OF entity_name IS
    -- [khai báo tín hiệu, component (IC con)]
BEGIN
    -- [code]
END architecture_name;
```

- Ví dụ:

```
ARCHITECTURE myarch1 OF and_gate IS
BEGIN
    x <= a AND b;
END myarch1;

ARCHITECTURE myarch2 OF and_gate IS
    SIGNAL a_ : STD_LOGIC;
BEGIN
    x    <= NOT ( a_ OR (NOT b) );
    a_   <= NOT a;
END myarch2;
```



ARCHITECTURE

Cú pháp Architecture

```
ARCHITECTURE architecture_name OF entity_name IS
    -- [khai báo tín hiệu, component (IC con)]
BEGIN
    -- [code]
END architecture_name;
```



Khai báo: giống như hộp toolbox, chuẩn bị sẵn sàng các linh kiện



● Khai báo **tín hiệu**: mỗi tín hiệu giống như một sợi dây dẫn, để truyền tín hiệu bên trong thiết kế

- Toán tử gán tín hiệu: <=

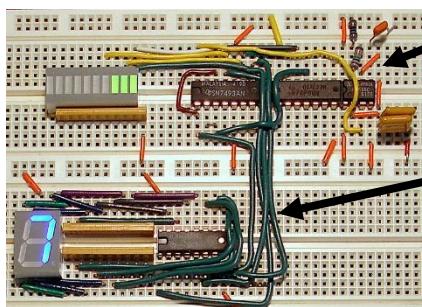


● Khai báo **component**: cho biết các IC con, đã thiết kế trước đó, sẽ được sử dụng trong thiết kế

Cú pháp Architecture

```
ARCHITECTURE architecture_name OF entity_name IS
    -- [khai báo tín hiệu, biến, component(IC con)]
BEGIN
    -- [code]
END architecture_name;
```

- **Code:** thể hiện cách lắp ráp, gắn các linh kiện



Sử dụng các component hợp lý

- Gắn các IC đã thiết kế lên mạch

Viết các lệnh gán tín hiệu: $x \leq y$;

- Đấu dây, cắm dây nối giữa các thành phần

Lập trình các process

- Đây là phần xử lý riêng của lập trình viên, mà trong đó các mã lệnh được thực thi tuần tự.

Tất cả các lệnh, process, component.. trong thân của Architecture đều được thực thi đồng thời.

- Tất cả các dây dẫn, linh kiện đều hoạt động đồng thời

Các kiểu mô tả Architecture

1/3

- Mô hình cấu trúc (Structural Style): gồm các thành phần (component) được liên kết với nhau.
- Mô hình luồng dữ liệu (Dataflow Style): gồm các lệnh gán được thực hiện đồng thời.
- Mô hình hoạt động (Behavioral Style): gồm các lệnh gán được thực hiện tuần tự.
- Dạng kết hợp của 3 kiểu mô hình trên.

→ Mô hình cấu trúc và luồng dữ liệu gần với hoạt động thực tế của phần cứng hơn. Mô hình hoạt động dễ học, dễ quen hơn vì giống phần mềm.

Các kiểu mô tả Architecture

2/3

Dataflow

```
• architecture myArch of  
  myEntity is  
  • signal tmp: STD_LOGIC;  
  
  • begin  
    • x <= a;  
    • y <= tmp;  
    • z <= tmp or c;  
    • tmp <= a and b;  
  • end myArrch;
```

Behavior

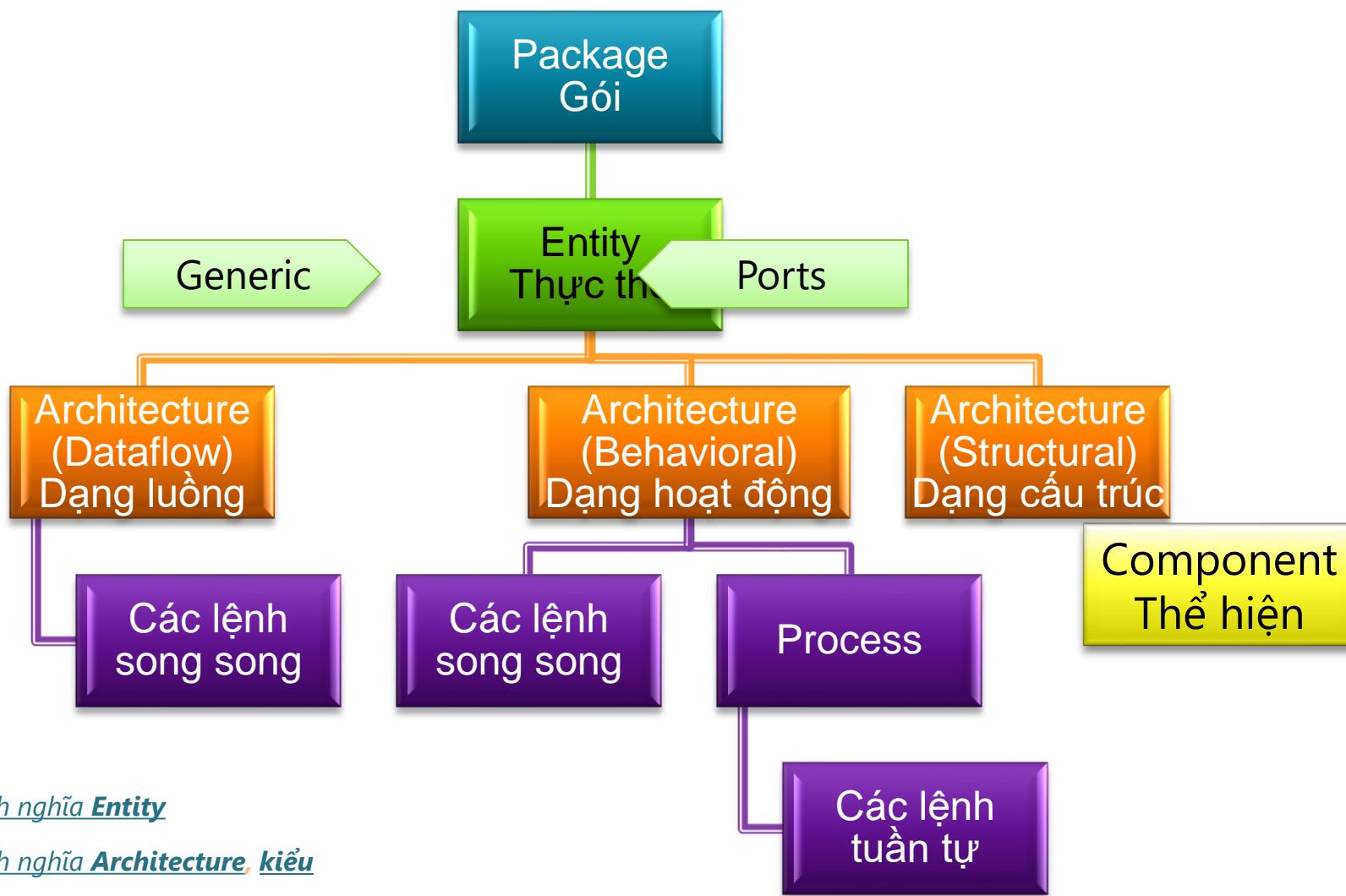
```
• architecture myArch of  
  myEntity is  
  • signal tmp: STD_LOGIC;  
  
  • begin  
    • x <= a;  
    • tmp <= a and b;  
    • process (tmp,b)  
      • begin  
        • y := tmp;  
        • z := tmp or c;  
      • end process;  
  • end myArrch;
```

Structural

```
• architecture myArch of  
  myEntity is  
  • signal tmp: STD_LOGIC;  
  • component OR2  
    • port (  
      • i1 : in STD_LOGIC;  
      • i2 : in STD_LOGIC;  
      • o : out STD_LOGIC  
    • );  
  • end component;  
  
  • begin  
    • U1 : OR2  
    • port map(  
      • i1 => tmp,  
      • i2 => c,  
      • o => z  
    • );  
    • x <= a;  
    • tmp <= a and b;  
    • y <= tmp;  
  • end myArrch;
```

Các kiểu mô tả Architecture

3/3





VHDL: Process

Process là gì?

- **Process** là một chuỗi các câu lệnh tuần tự, mô tả hành vi của một phần mạch nào đó trong thiết kế.
- Trong process, sử dụng được các cấu trúc lập trình như if then, for, do while...
- Vị trí: Process nằm trong thân của một Architecture.
- Cú pháp

```
Process_name: process(danh sách tín hiệu tích cực)
              -- [khai báo biến]
              BEGIN
              -- [code]
END PROCESS Process_name;
```

Cú pháp Process

- Process_name: tên gợi nhớ của process, có thể bỏ.
- Danh sách tín hiệu tích cực: khi các tín hiệu này có sự thay đổi, phần thân của process sẽ bắt đầu hoạt động. Cụ thể là toàn bộ phần code trong process sẽ được chạy lại từ đầu, để cập nhật giá trị mới cho đầu ra.
- Ngược lại, nếu một tín hiệu không có trong danh sách tín hiệu tích cực thì khi tín hiệu đó thay đổi, process không chạy.
 - Gợi ý 1: mọi tín hiệu vào của process đều nằm trong danh sách
 - Gợi ý 2: mọi tín hiệu vào của entity có thể nằm trong danh sách
 - Gợi ý 3: khai báo danh sách bị thừa thì không gây lỗi.

Cú pháp Process

- Khai báo biến: kiểu integer, real..

- Cú pháp:

- ```
variable var_name : type := default_value;
```

- type: kiểu dữ liệu: integer, real, bản ghi...

- default\_value: giá trị mặc định. Có thể bỏ.

- Biến sẽ được biên dịch thành các tín hiệu, thanh ghi... nhưng cũng có thể không thể biên dịch thành đối tượng vật lý được (chỉ chạy giả lập).

- Biến chỉ có phạm vi bên trong process.

- Khai báo tín hiệu:

- Không được phép khai báo tín hiệu mới trong process

- Nhưng vẫn được sử dụng các tín hiệu trong Entity và Architecture

# Cú pháp Process

## Code:

- Cách lập trình giống ngôn ngữ Pascal.
- Phép gán:
  - Gán tín hiệu:      <=
  - Gán biến:            :=
- Xác định tín hiệu X đang ở trạng thái nào
  - Sườn lên:                 rising\_edge(X)  
Ví dụ:                    if rising\_edge(X) then
  - Sườn xuống:               falling\_edge(X)  
Ví dụ:                    if falling\_edge(X) then
  - Logic 0:                    ' 0'  
Ví dụ:                    if X = ' 0' then
  - Logic 1:                    ' 1'  
Ví dụ:                    if X = ' 1' then

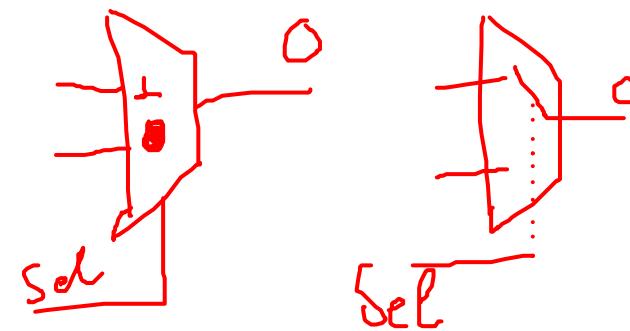
# Ví dụ: bộ chọn kênh 2x1

```
entity MUX2x1 is
 port (
 I1 : in std_logic;
 I0 : in std_logic;
 SEL : in std_logic;
 O : out std_logic
);
end entity MUX2x1;

architecture behavior of MUX2x1 is
begin
 process (I0, I1, SEL)
 begin
 if SEL = '1' then
 O <= I1;
 else
 O <= I0;
 end if;
 end process;
end behavior;
```

```
entity MUX2x1 is
 port (
 I1 : in std_logic;
 I0 : in std_logic;
 SEL : in std_logic;
 O : out std_logic
);
end entity MUX2x1;

architecture dataflow of MUX2x1 is
begin
 O <= (I1 and SEL) or (I0 and (not SEL));
end dataflow;
```



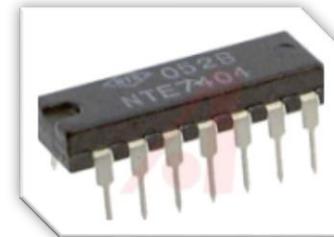
Mux2x1.vhd



# VHDL: Component

# Component là gì?

- **Component**, thể hiện, là một thực thể vật lý được đem vào sử dụng, được nhúng vào **Architecture** của một **Entity** khác.  
*(sẽ đề cập tới Architecture ở phần sau)*



Giống như khái niệm instance của một class

```
class Mach_And { }
```

```
bool MyFunction {
 Mach_And c1; --instance
 Mach_And c2; --instance
 return c1(true,false) &
 c2(false,true)
}
```

- Entity chỉ là định nghĩa về IC. Component là một thể hiện được triệu gọi ra.

# Cú pháp khai báo Component

- Component chỉ xuất hiện trong Architecture ở 2 bước
  - Bước 1: Khai báo Component.
    - Giống hệt khai báo Entity, chỉ khác ở chỗ:
      - thay từ khóa **entity** bằng **component**,
      - bỏ từ khóa **is**
    - Nằm trong phần khai báo của một Architecture. Nói cách khác là nằm giữa 2 từ khóa **architecture** và **begin**
  - Bước 2: Sử dụng các đối tượng Component
    - Gán cụ thể các chân pin vào/ra của component với các tín hiệu khác
    - Gán parameter với một hằng số cụ thể nào đó. *Bỏ qua nếu entity tương ứng không có parameter nào.*

# Cú pháp khai báo Component

## • Bước 1: Khai báo component

```
ARCHITECTURE MyArch OF MyEntity IS
 -- Bước 1, khai báo Component ở đây
 COMPONENT entity_name
 GENERIC (
 generic_name : generic_type := <Giá trị mặc định>;
 . . .);
 PORT (-- phần in/out, đặc tả vào ra
 port_name : signal_mode signal_type;
 port_name : signal_mode signal_type;
 . . .);
 END COMPONENT;

 BEGIN
 -- Bước 2, gán tham số và tín hiệu vào ra ở chỗ này
 END ARCHITECTURE MyArch;
```

# Cú pháp khai báo Component

- Bước 2: Sử dụng các đối tượng Component

```
ARCHITECTURE MyArch OF MyEntity IS
```

```
-- Bước 1, khai báo Component ở chỗ này
```

```
BEGIN
```

```
-- Bước 2, gán tham số và tín hiệu vào ra
component_name: entity_name
```

```
 GENERIC MAP (generic_name => hằng_số)
```

```
 PORT MAP (
```

```
 port_name => SignalA,
```

```
 port_name => SignalB
```

```
);
```

```
END ARCHITECTURE MyArch;
```

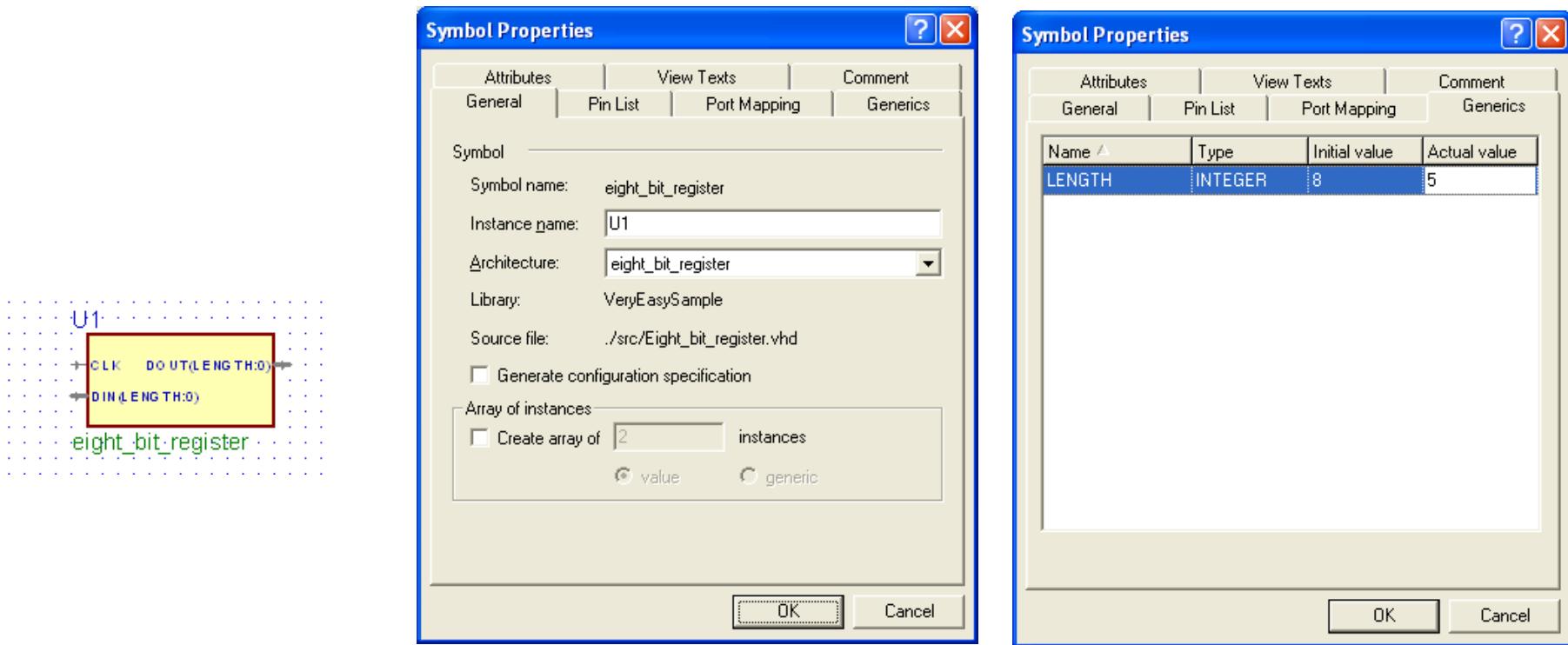
# Ví dụ: Component của thanh ghi

```
ARCHITECTURE MyArch OF MyEntity IS
 -- Bước 1, khai báo Component ở chỗ này
 component eight_bit_register
 generic (LENGTH : integer := 8 -- Giá trị mặc định = 8
 Fmax: integer := 50 -- Đơn vị MHz
);
 port(CLK : in STD_LOGIC;
 DIN : in STD_LOGIC_VECTOR(LENGTH downto 1);
 DOUT : out STD_LOGIC_VECTOR(LENGTH downto 1)
);
 end component;
BEGIN
 -- Bước 2, tạo component với độ rộng bus mặc định = 8 bit.
 U1 : eight_bit_register port map(CLK => NET2342);

 -- Bước 2, tạo component với độ rộng bus mặc định = 5 bit.
 U2 : eight_bit_register generic map (LENGTH =>5) port map(CLK => N121);

END ARCHITECTURE MyArch;
```

# Ví dụ: Component trong phần mềm Active-HDL



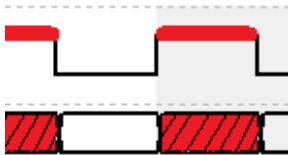


# VHDL: Các ví dụ

# Ví dụ: Nhớ lại một số thuật ngữ

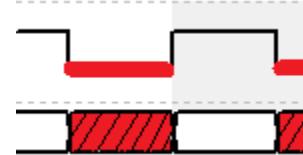
## Tín hiệu tích cực mức cao/thấp/... là gì?

- Trả lời: là trạng thái mà ở đó tín hiệu **sẽ gây ra sự thay đổi** nào đó
- Khi xem giản đồ sóng tín hiệu, chỉ cần chú ý tới vị trí mà tín hiệu điều khiển ở trạng thái tích cực.
- Thường áp dụng cho tín hiệu điều khiển, ít khi áp dụng cho dữ liệu



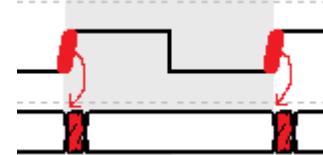
Tích cực  
mức cao

- Gây ra sự thay đổi khi ở logic 1.



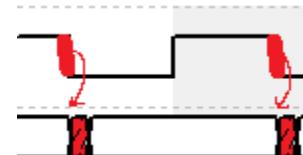
Tích cực  
mức thấp

- Gây ra sự thay đổi khi ở logic 0.



Tích cực  
sườn lên

- Gây ra sự thay đổi khi chuyển giá trị từ 0 → 1
- Hoặc tích cực sườn dương



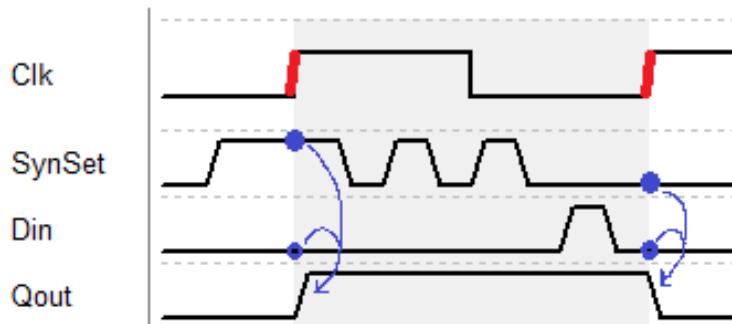
Tích cực  
sườn xuống

- Gây ra sự thay đổi khi chuyển giá trị từ 1 → 0
- Hoặc tích cực sườn âm

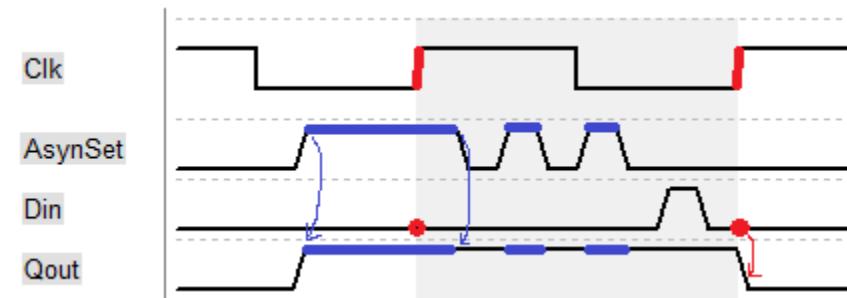
# Ví dụ: Nhớ lại một số thuật ngữ

## Tín hiệu đồng bộ/không đồng bộ là gì?

- Trả lời: xem xét yếu tố **tích cực** khi có mặt của tín hiệu CLK
  - tín hiệu đồng bộ, synchronous**, chỉ tích cực **tại sườn** của tín hiệu CLK.
  - tín hiệu không đồng bộ, asynchronous**, tích cực mà **không phụ thuộc** vào tín hiệu CLK.
  - Không có tín hiệu phụ thuộc vào cả 2 sườn, vì vậy*
    - có tín hiệu đồng bộ tích cực theo mức If (rising\_edge(X)) th
    - không có tín hiệu đồng bộ tích cực theo sườn If (rising\_edge(Y)



SynSet: thiết lập Qout=1, đồng bộ mức cao  
Din: dữ liệu vào, đồng bộ

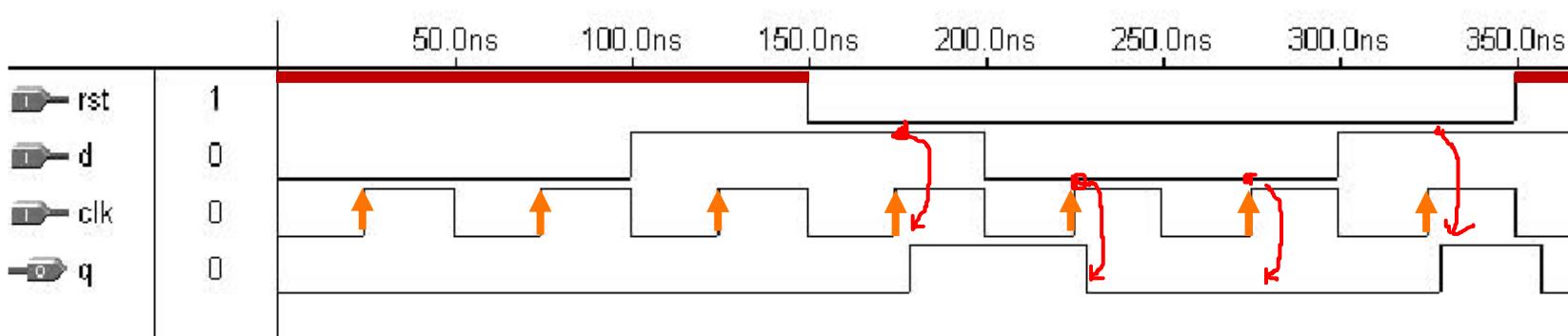
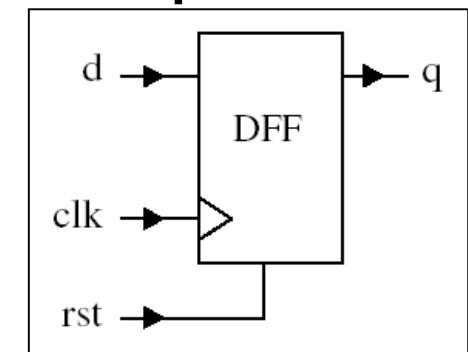


AsynSet: không đồng bộ mức cao  
Din: dữ liệu vào, đồng bộ

# Ví dụ 1: mô tả D-FF bằng HDL

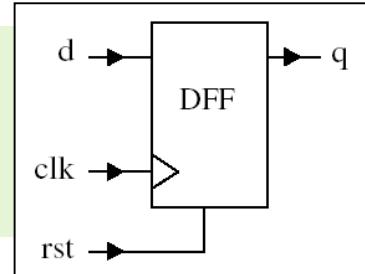
- Flip-flop D tích cực sườn dương có tín hiệu Rst không đồng bộ tích cực mức cao.

- Nếu  $rst = 1$  thì  $q = 0$
- Ngược lại:
  - Nếu  $clk$  chuyển từ 0 lên 1 thì  $q = d$
  - Còn lại thì hệ giữ nguyên trạng thái.



# Ví dụ 1: mô tả D-FF bằng HDL

```
1 -----
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -----
5 ENTITY dff IS
6 PORT (d, clk, rst: IN STD_LOGIC;
7 q: OUT STD_LOGIC);
8 END dff;
9 -----
10 ARCHITECTURE behavior OF dff IS
11 BEGIN
12 PROCESS (rst, clk)
13 BEGIN
14 IF (rst='1') THEN
15 q <= '0';
16 ELSIF (clk'EVENT AND clk='1') THEN
17 q <= d;
18 END IF;
19 END PROCESS;
20 -----
21 END behavior;
```



Nếu  $rst = 1$  thì  $q = 0$

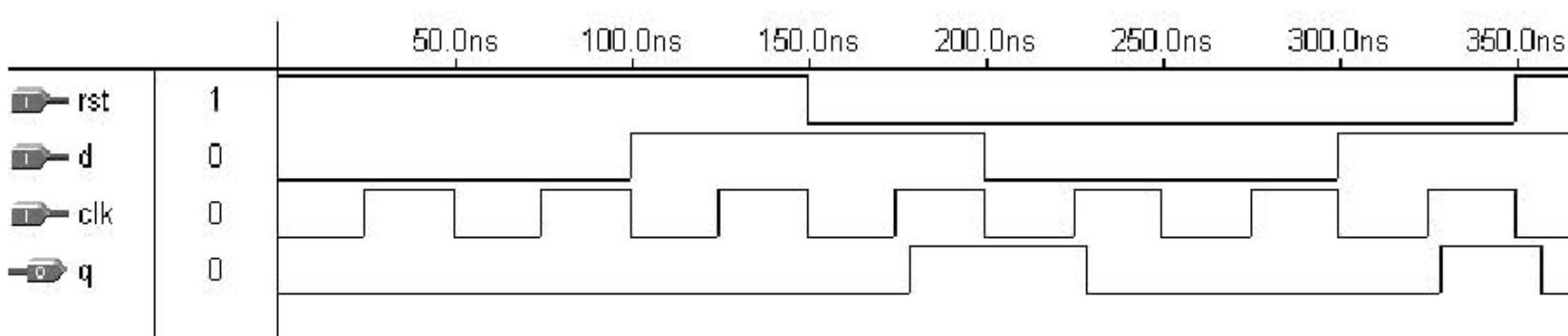
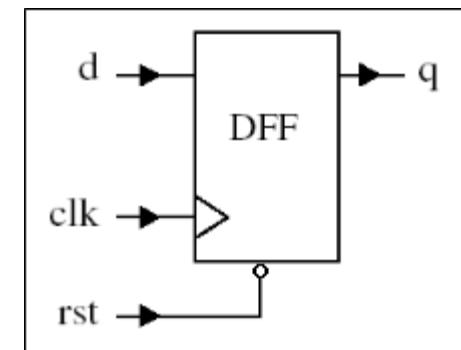
$clk$  chuyển từ 0 lên 1 thì  $q = d$

Không làm gì cả, tức là  $q$  không đổi, tức là hệ giữ nguyên trạng thái

# Ví dụ 2: mô tả D-FF bằng HDL

Flip-flop D tích cực sườn dương có tín hiệu  $\overline{Rst}$  đồng bộ tích cực mức thấp.

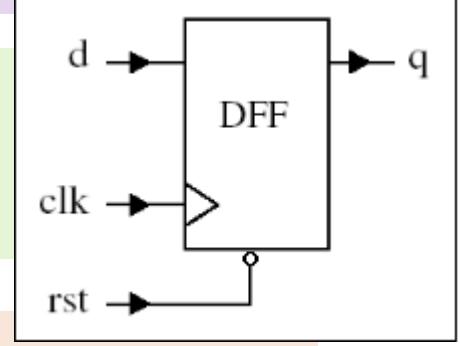
- Nếu clk chuyển từ 0 lên 1
  - Nếu  $\overline{Rst} = 0$  thì  $q = 0$
  - Ngược lại: thì  $q = d$
- Còn lại thì hệ giữ nguyên trạng thái.



# Ví dụ 2: mô tả D-FF bằng HDL

```
1 -----
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -----
5 ENTITY dff IS
6 PORT (d, clk, rst: IN STD_LOGIC;
7 q: OUT STD_LOGIC);
8 END dff;
9 -----
10 ARCHITECTURE behavior OF dff IS
11 BEGIN
12 PROCESS (rst, clk)
13 BEGIN
14 IF (clk'EVENT AND clk='1') THEN
15 IF (rst='0') THEN
16 q <= '0';
17 ELSE
18 q <= d;
19 END IF;
20 END IF;
21 END PROCESS;
22 END behavior;
```

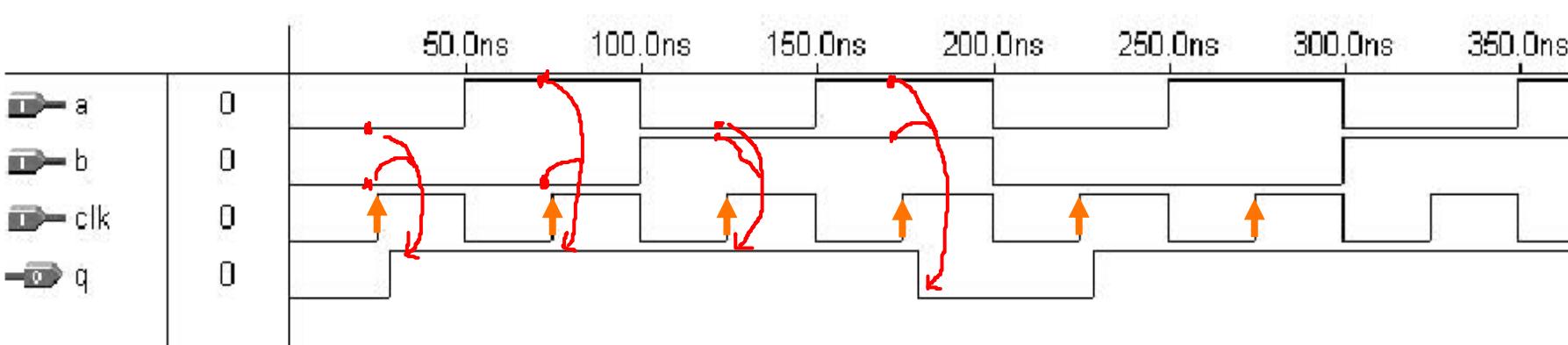
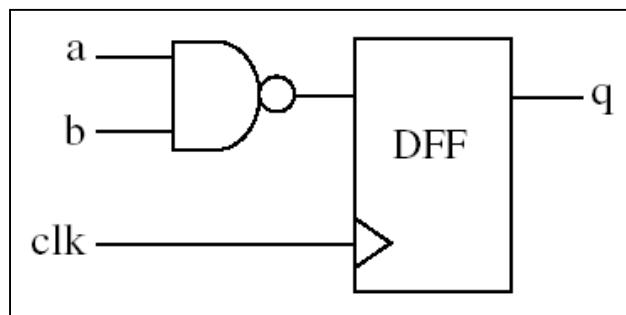
clk chuyển  
từ 0 lên 1  
thì mới hoạt  
động



Nếu  $rst = 0$   
thì  $q = 0$   
Nếu  $rst = 1$   
thì  $q = d$   
Không làm gì cả, tức là  $q$  không đổi,  
tức là hệ giữ nguyên trạng thái

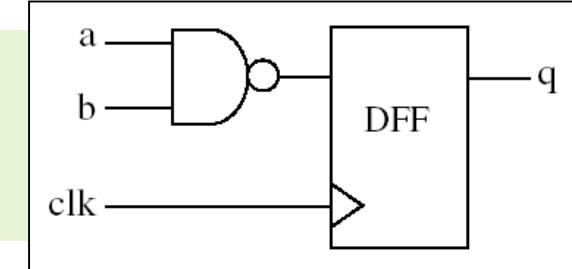
# Ví dụ 3: D-FF và mạch tổ hợp

- Flip-flop D tích cực sườn dương có đầu vào D là đầu ra của NAND2.



# Ví dụ 3: D-FF và mạch tổ hợp

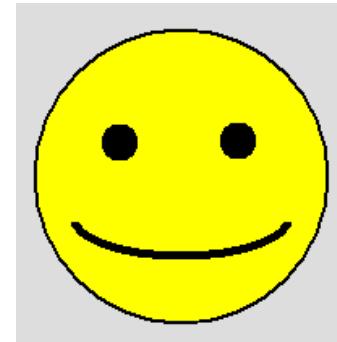
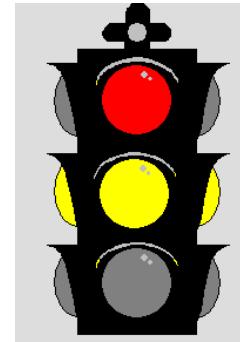
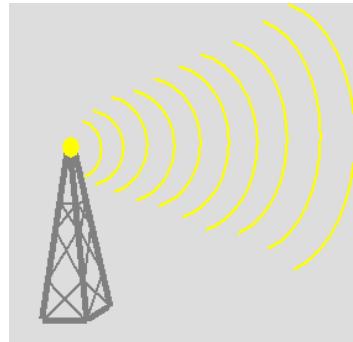
```
1 -----
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -----
5 ENTITY MyIC IS
6 PORT (a, b, clkt: IN STD_LOGIC;
7 q: OUT STD_LOGIC);
8 END MyIC ;
9 -----
10 -----
11 ARCHITECTURE behavior OF MyIC IS
12 SIGNAL temp : STD_LOGIC;
13 BEGIN
14 temp <= a nand b;
15 DFF1: PROCESS (temp, clk)
16 BEGIN
17 IF (clk'EVENT AND clk='1') THEN
18 q <= temp;
19 END IF;
20 END PROCESS DFF1;
21 END behavior;
22 -----
```



# Tín hiệu là gì?

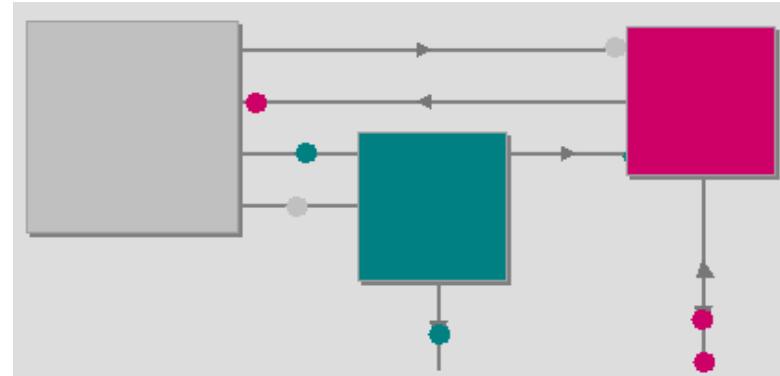
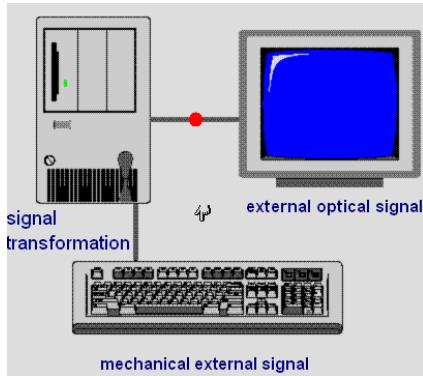
- Truyền thông là quá trình truyền thông tin từ đầu phát đến đầu thu
- Có nhiều phương thức để truyền thông tin nhưng đều có một thành phần chung là

tín hiệu (**Signal**)



# Tín hiệu

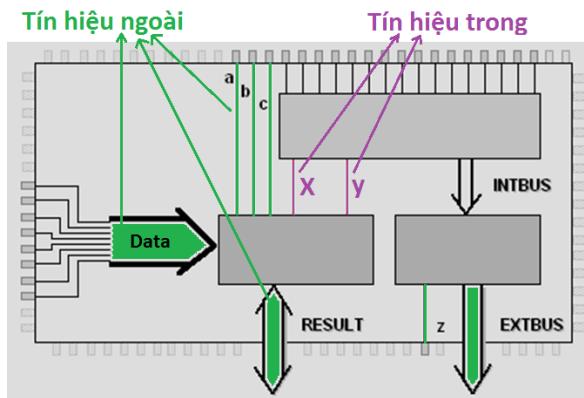
- Tín hiệu trong các thiết bị điện tử là tín hiệu điện.
- Tín hiệu điện là khái niệm ở mức vật lý. Tín hiệu điện kết hợp với các qui tắc tạo nên các chuẩn và giao thức truyền thông như I<sup>2</sup>C, USB, qui tắc đầu to/đầu nhỏ...
- Về không gian mà trong đó tín hiệu truyền đi:
  - Nếu không gian rộng, chịu sự tác động của môi trường thì tín hiệu chịu nhiều, suy hao → cần các chuẩn, và giao thức để bảo đảm chính xác.
  - Nếu không gian rất hẹp, bên trong die của IC (FPGA), thì môi trường ít tác động → tín hiệu truyền với tốc độ cao và đơn giản.



# Phân loại tín hiệu

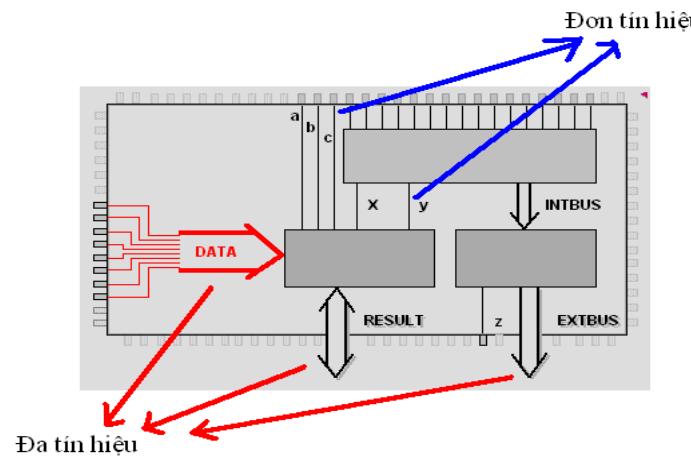
## Phân loại theo chức năng của tín hiệu

- Tín hiệu ghép nối với bên ngoài (External signal)
  - Khai báo trong Entity
- Tín hiệu bên trong thiết bị (Internal signal)
  - Khai báo trong Architecture



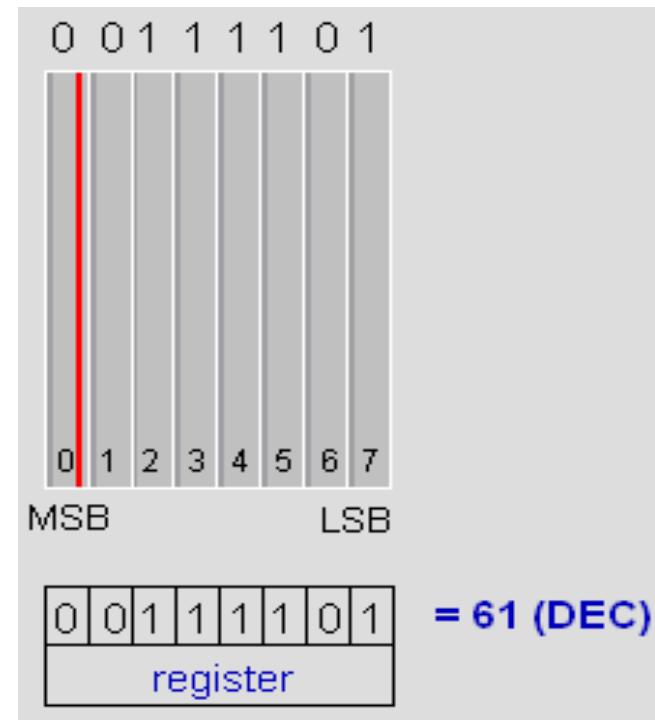
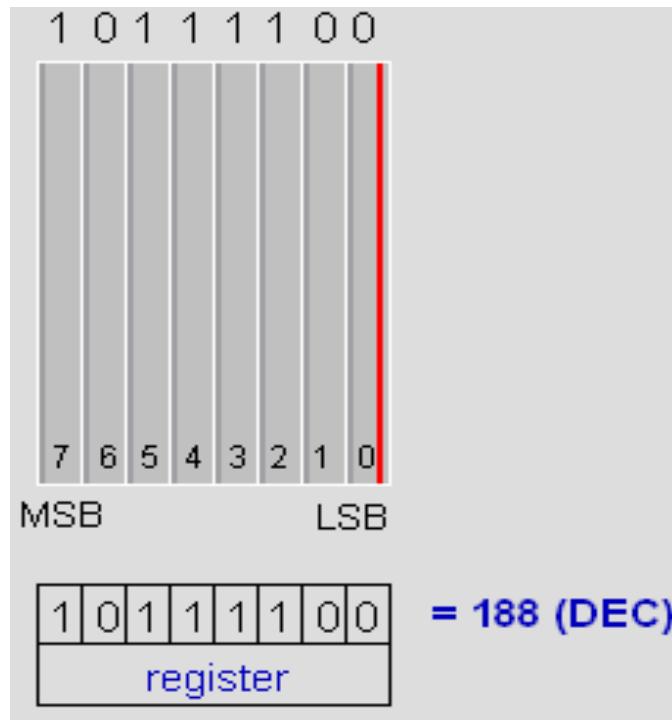
## Phân loại theo số đường tín hiệu

- **Đơn tín hiệu** (single signal): 1 đường kết nối truyền 1 bit
  - x: STD\_LOGIC
- **Đa tín hiệu** (multiple signal): nhiều đường kết nối, truyền nhiều bit đồng thời
  - intbus: STD\_LOGIC\_VECTOR(1 to 8)



# Độ rộng bus và trạng thái các bit

- VHDL cho phép đặt chỉ số bất kỳ và theo cả 2 chiều.  
Ví dụ 9 **downto** 5; 12 **to** 17;

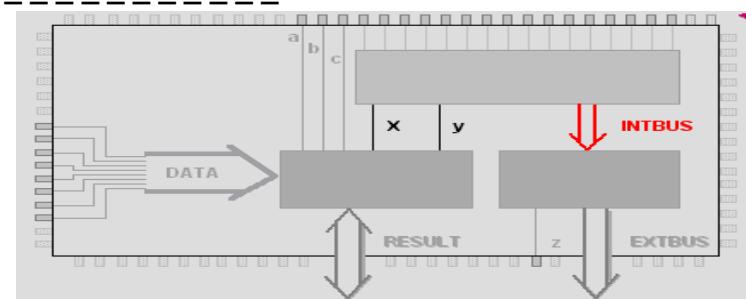
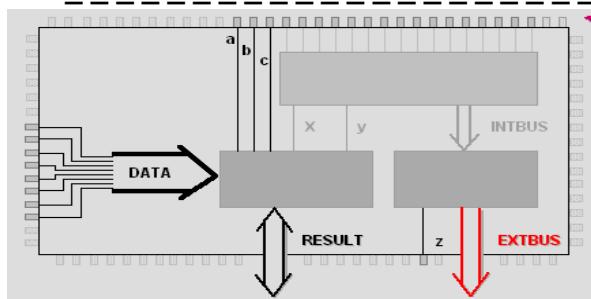


STD\_LOGIC\_VECTOR(7 **downto** 0)  
*nên sử dụng*

STD\_LOGIC\_VECTOR(0 **to** 7)  
*khó debug*

# Khai báo tín hiệu

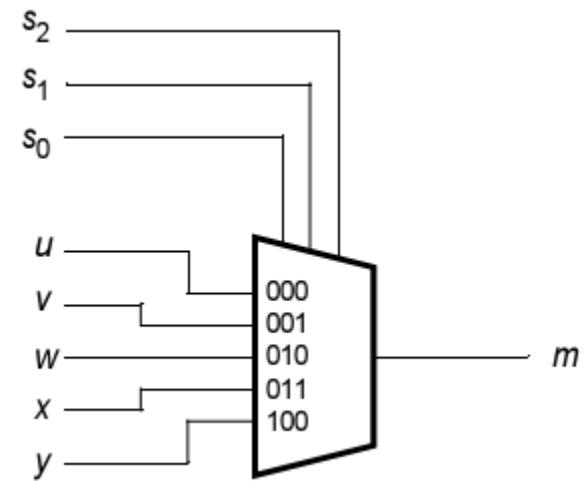
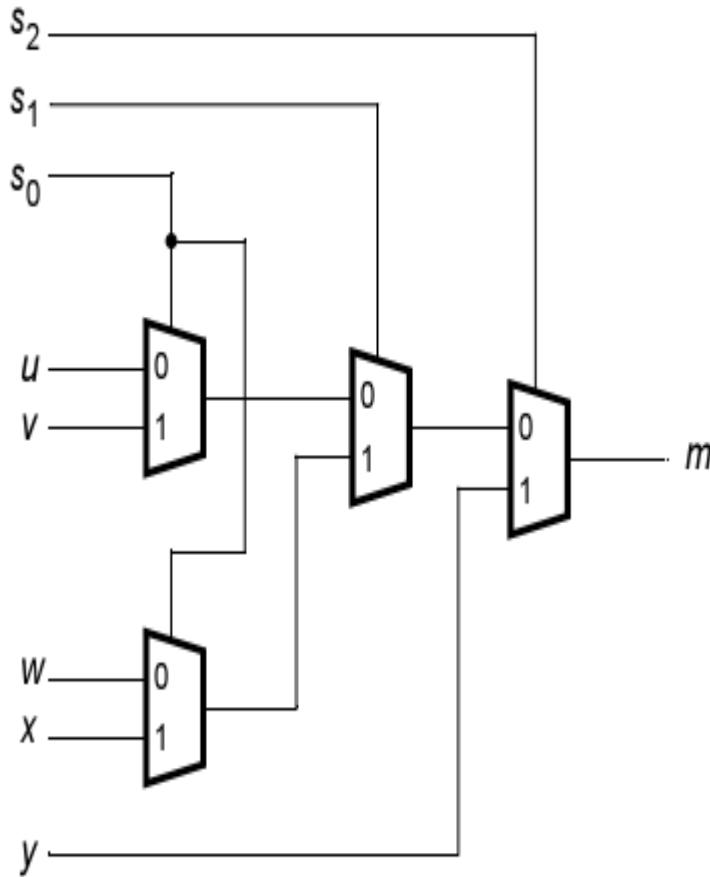
```
1 -----
2 LIBRARY ieee;
3 USE ieee.std_logic_1164.all;
4 -----
5 ENTITY myIC IS
6 PORT (EXTBUS : OUT STD_LOGIC_VECTOR(4 downto 0);
7 a,b,c : IN STD_LOGIC;.....);
8 END myIC;
9 -----
10 -----
11 ARCHITECTURE myArch OF myIC IS
12 SIGNAL INTBUS: STD_LOGIC_VECTOR(4 downto 0);
13 SIGNAL x, y : STD_LOGIC;
14 BEGIN
15 END myArch;
```



# Phạm vi sử dụng tín hiệu

- Các tín hiệu khai báo trong một Package có thể sử dụng trong tất cả các đối tượng sử dụng Package đó
- Các tín hiệu của Entity (external signals) có thể sử dụng trong tất cả các Architecture của Entity đó
- Các tín hiệu khai báo bên trong một Architecture thì chỉ có thể sử dụng trong Architecture đó, bao gồm cả các Component, Process nếu có.

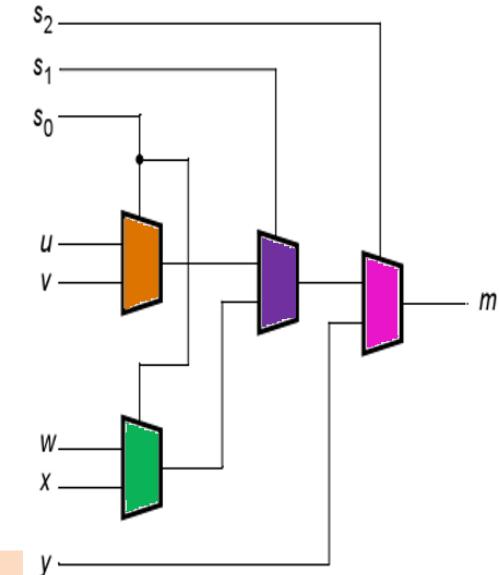
# Ví dụ: Xây dựng Mux5x1 từ Mux2x1



MUX5-1

# Ví dụ: Xây dựng Mux5x1 từ Mux2x1

```
entity mux5_1 is
 port(u, v, w, x, y: IN std_logic_vector(2 downto 0);
 s: IN std_logic_vector(2 downto 0);
 m: OUT std_logic_vector(2 downto 0));
end entity mux5_1;
architecture structure1 of mux5_1 is
signal out1, out2, out3: std_logic_vector(2 downto 0);
begin
 --Khai báo component và nối dây vào component
 mux21_1: work.mux2_1(behavior)
 port map(u, v, out1, s(0));
 mux21_2: work.mux2_1(behavior)
 port map(w, x, out2, s(0));
 mux21_3: work.mux2_1(behavior)
 port map(out1, out2, out3, s(1));
 mux21_4: work.mux2_1(behavior)
 port map(out3, y, m, s(2));
end structure1;
```





# VHDL: Kiểu dữ liệu

# Các kiểu dữ liệu có trong VHDL

- Các kiểu dữ liệu cơ sở (kiểu dữ liệu vô hướng – scalar)
  - **Kiểu logic**, các đại lượng vật lý, **kiểu số**, **kiểu liệt kê**
  - Có rất nhiều kiểu dữ liệu cơ sở chuẩn đã được định nghĩa trước.
  - Ví dụ: enumeration, integer, physical, floating point.
- Các kiểu dữ liệu tổng hợp (composite)
  - **Kiểu mảng** (array)
  - Kiểu bản ghi (record)
- Ngoài ra VHDL còn cung cấp cả kiểu access (pointer) và kiểu file
- Trong phạm vi học phần, chỉ quan tâm tới **kiểu logic**, **kiểu số**, **kiểu liệt kê**, **kiểu mảng**

# Các kiểu dữ liệu cơ sở chuẩn

- **LIBRARY std; USE std.standard.all;**
  - Cung cấp kiểu BIT, BOOLEAN, INTEGER và REAL
  - Mặc định được sử dụng mà không cần khai báo
- **LIBRARY IEEE; USE IEEE.std\_logic\_1164.all;**
  - Cung cấp kiểu STD\_LOGIC và STD\_ULOGIC
  - Sử dụng nhiều nhất, cơ bản nhất. Luôn phải có.

# Các kiểu dữ liệu cơ sở chuẩn

- **LIBRARY IEEE; USE IEEE.std\_logic\_arith.all;**
  - Cung cấp kiểu SIGNED và UNSIGNED,
  - Và các hàm chuyển đổi dữ liệu, VD: *conv\_integer(p)*,  
*conv\_unsigned(p, b)*, *conv\_signed(p, b)*, *conv\_std\_logic\_vector(p, b)* ...
- **LIBRARY IEEE; USE IEEE.std\_logic\_signed.all;**  
**LIBRARY IEEE; USE IEEE.std\_logic\_unsigned.all;**
  - chứa các hàm cho phép hoạt động với dữ liệu STD\_LOGIC\_VECTOR được thực hiện khi mà kiểu dữ liệu là SIGNED hoặc UNSIGNED

# Kiểu logic: BIT, BIT\_VECTOR

- BIT là kiểu tín hiệu đơn giản chỉ với 2 giá trị logic 0,1
- BIT\_VECTOR là kiểu đa tín hiệu mà mỗi tín hiệu bên trong là kiểu BIT.
- Kiểu BIT không mô tả được hết các trạng thái vật lý của tín hiệu, vì vậy hiện nay ít được sử dụng.
- Không sử dụng kiểu BIT trong học phần này.

# Kiểu logic: STD\_LOGIC

- Kiểu STD\_LOGIC gồm 8 giá trị sau:
  - 'X' : không xác định, xảy ra khi tương tranh giữa logic '0' và '1'.
  - 'W' : không xác định yếu, xảy ra khi tương tranh giữa logic 'L' và 'H'.
  - '0' : mức thấp khỏe
  - '1' : mức cao khỏe
  - 'Z' : trễ kháng cao
  - 'L' : mức thấp yếu
  - 'H' : mức cao yếu
  - ‘–’ : không quan tâm, tùy ý (don't care)
- STD\_LOGIC\_VECTOR là kiểu đa tín hiệu của STD\_LOGIC

# Kiểu logic: STD\_ULOGIC

- Hệ thống logic 9 mức được giới thiệu trong chuẩn IEEE 1164 ('U', 'X', '0', '1', 'Z', 'W', 'L', 'H', '-'), gọi là STD\_ULOGIC.
- STD\_LOGIC là subtype của STD\_ULOGIC (có thêm giá trị logic 'U' – Uninitialized, chưa khởi tạo)
  - Ví dụ: nếu IC mạch dây, nhưng không có tín hiệu Reset thì các tín hiệu bên trong sẽ rơi vào logic 'U'.

# Kiểu số

- Kiểu INTEGER:
  - số nguyên 32 bit (từ -2.147.483.647 đến +2.147.483.647)
- Kiểu NATURAL:
  - số tự nhiên (từ 0 đến +2.147.483.647)
- Kiểu REAL:
  - số thực (từ -1.0E38 đến +1.0E38)
- Các biến kiểu số có sử dụng trực tiếp các phép toán số học như + - x /. Trình biên dịch sẽ tự chuyển đổi thành mạch logic phù hợp.
- Thường kết hợp với việc định nghĩa lại các subtype để thu nhỏ phạm vi kiểu số, giúp giới hạn năng lực xử lý mạch logic cho phù hợp

# Kiểu vật lý

## ● Kiểu dữ liệu vật lý:

- Biểu thị một đại lượng vật lý nào đó, chẳng hạn như là khối lượng, độ dài, thời gian hoặc điện áp.
- Cú pháp khai báo một kiểu dữ liệu vật lý như sau:

```
physical_type_definition ::= range_constraint
units
 base_unit_declaration --Thứ nguyên cơ sở
 {secondary_unit_declaration} --Thứ nguyên thứ cấp
end units
base_unit_declaration ::= identifier ;
secondary_unit_declaration ::= identifier =
 physical_literal ;
physical_literal ::= [abstract_literal] unit_name
```

## ● Không sử dụng kiểu vật lý trong học phần này

# Một số kiểu dữ liệu vật lý

```
type CAPACITY is range 0 to 1E8
units
```

```
pF; -- picofarad,base_unit_declaration
nF = 1000 pF; -- nanofarad
uF = 1000 nF; -- microfarad
mF = 1000 uF; -- milifarad
F = 1000 mF; -- farad
end units CAPACITY;
```

```
type DISTANCE is range 0 to 1E5
units
```

```
um; -- micrometer, base_unit_declaration
mm = 1000 um; -- millimeter
in_a = 25400 um; -- inch
end units DISTANCE;
```

```
variable SomeVar : CAPACITY;
SomeVar := 1mF + 23pF;
```

```
variable Dis1, Dis2 : DISTANCE;
Dis1 := 28 mm;
Dis2 := 2 in_a - 1 mm;
if Dis1 < Dis2 then ...
```

# Kiểu kí tự và chuỗi

- Cặp dấu nháy đơn để biểu diễn
  - Các kí tự. Ví dụ: 'A' '\*' " "
  - Logic 0, Logic 1. Ví dụ '0', '1'
- Cặp dấu nháy kép để biểu diễn
  - Nội dung của một xâu kí tự. Ví dụ: "A string" "Hello"
  - Giá trị của một STD\_LOGIC\_VECTOR. Ví dụ: "11010101", "001010"

# Các ví dụ

**x0 <= '0';**

- -- std\_logic, std\_ulogic  
giá trị '0'

**x1 <= "00011111";**

- std\_logic\_vector,  
std\_ulogic\_vector,  
signed, or unsigned

**x3 <= "101111";**

- -- Biểu diễn dạng nhị phân

**x4 <= B"101111";**

- Biểu diễn dạng nhị phân

**x5 <= O"57";**

- Số 57 theo hệ cơ số 8.

**x6 <= X"2F";**

- Số 2F theo hệ cơ số 16.

**n <= 1200;**

- Số nguyên.

**m <= 1\_200;**

- Số nguyên, được phép có dấu gạch dưới

**y <= 1.2E-5;**

- Số thực

**q <= d after 10 ns;**

- physical

# Các ví dụ (tiếp)

```
SIGNAL c: STD_LOGIC;
SIGNAL d: STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL e: INTEGER RANGE 0 TO 255;
```

**e <= c;**

- Không hợp lệ (khác kiểu INTEGER & STD\_LOGIC)

**c <= d;**

- Không hợp lệ (khác kiểu STD\_LOGIC & STD\_LOGIC\_VECTOR)

**e <= d;**

- Không hợp lệ (khác kiểu INTEGER & STD\_LOGIC\_VECTOR)

**d <= "0101";**

- Không hợp lệ (khác độ rộng bus)

**c <= d(5);**

- Hợp lệ (cùng kiểu STD\_LOGIC)

**d(0) <= c;**

- Hợp lệ (cùng kiểu STD\_LOGIC)

**d <= "01010101"**

- Hợp lệ, độ rộng 8 bit

**d(4 downto 2) <= "111"**

- Hợp lệ, độ rộng 3 bit

# Định nghĩa kiểu dữ liệu

- Có 2 dạng dữ liệu người dùng có thể định nghĩa:
  - integer (kiểu số nguyên):
    - là tập con của kiểu số nguyên
  - enumerated (kiểu liệt kê)
    - giống khai báo enum trong ngôn ngữ C

# Định nghĩa kiểu dữ liệu số nguyên

## • Cú pháp

```
type <type_name> is range <exp> to | downto <exp>
```

- <exp> là các số nguyên

## • Ví dụ:

```
type byte_int is range 0 to 255;
```

```
type signed_word_int is range -32768 to 32767;
```

```
type bit_index is range 31 downto 0;
```

```
type my_integer is range -32 to 32;
```

```
type student_grade is range 0 to 100;
```

## • Thường kết hợp với việc định nghĩa lại các subtype để thu nhỏ phạm vi kiểu số, giúp giới hạn năng lực xử lý mạch logic cho phù hợp

- Ví dụ: thiết kế bộ đếm 0~7 thì cần định nghĩa thêm kiểu số nguyên từ 0~7.

# Định nghĩa kiểu dữ liệu liệt kê

## • Cú pháp

```
type <type_name> is (<item>, <item>)
```

- <item> là các giá trị liệt kê

## • Ví dụ:

```
type logic_level is (unknown, low, undriven, high);
type alu_function is (add, subtract, multiply);
type tiny_digit is ('0', '1', '2', '3', '4', '5', '6');
```

## • Thường áp dụng trong thiết kế các sơ đồ máy trạng thái hữu hạn FSM, trong đó mỗi trạng thái là một <item>.

- Ví dụ: type Sreg0\_type is (S2, S3, S4, S1);
- Có thể kèm theo khai báo **attribute** để tạo bảng mã hóa cho các trạng thái. Ví dụ S2="0010", S3="0100"

# Kiểu mảng

- Mảng (Array) là tập hợp các phần tử có cùng kiểu.
- Mảng có thể là mảng 1 chiều (1D), 2 chiều (2D) hoặc là mảng 1 chiều của mảng 1 chiều (1Dx1D)
- Định nghĩa mảng:  
`TYPE type_name IS ARRAY (range) OF data_type;`
- Sử dụng, khai báo mảng:  
`SIGNAL signal_name: type_name [:= initial_value];`
- Ứng dụng: để mô hình các cấu trúc tuyến tính như RAM, ROM, etc.

# Kiểu mảng: Các ví dụ

## ● Mảng 1Dx1D:

```
TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;
TYPE matrix IS ARRAY (0 TO 3) OF row;
TYPE matrix IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
SIGNAL x: matrix;
```

## ● Mảng 2D:

```
TYPE matrix2D IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;
```

## ● Khởi tạo mảng:

```
... := "0001"; -- mảng 1D
... := ('0', '0', '0', '1') -- mảng 1D
... := (('0', '1', '1', '1'), ('1', '1', '1', '0')); -- 1Dx1D / 2D
```

# Kiểu mảng: Các ví dụ

```
TYPE row IS ARRAY (7 DOWNTO 0) OF STD_LOGIC;
TYPE array1 IS ARRAY (0 TO 3) OF row;
TYPE array2 IS ARRAY (0 TO 3) OF STD_LOGIC_VECTOR(7 DOWNTO 0);
TYPE array3 IS ARRAY (0 TO 3, 7 DOWNTO 0) OF STD_LOGIC;

SIGNAL x: row;
SIGNAL y: array1;
SIGNAL v: array2;
SIGNAL w: array3;

x(0) <= y(1)(2); x(1) <= v(2)(3); x(2) <= w(2,1);
y(1)(1) <= x(6); y(2)(0) <= v(0)(0); y(0)(0) <= w(3,3);
w(1,1) <= x(7); w(3,0) <= v(0)(3);
x <= y(0); x <= v(1); x <= w(2); x <= w(2,2 DOWNTO 0);

v(0) <= w(2,2 DOWNTO 0); v(0) <= w(2);
y(1) <= v(3);
y(1)(7 DOWNTO 3) <= x(4 DOWNTO 0);
v(1)(7 DOWNTO 3) <= v(2)(4 DOWNTO 0);
w(1,5 DOWNTO 1) <= v(2)(4 DOWNTO 0);
-- Lệnh tô đậm là các phép gán sai vì khác kiểu
```

# Kiểu mảng: Gán giá trị

- Giả sử có một mảng được khai báo là:

```
type type4 is array (1 to 4) of STD_LOGIC;
signal a: type4;
```

- Các cách gán giá trị cho mảng

- a <= ('1', '1', '0', '0'); --  $a_{(1\sim 4)} = 1100$
- a <= (1 => '0', 3 => '0', 4 => '1', 2 => '1'); --  $a_{(1\sim 4)} = 0101$
- a <= ('1', 4 => '1', others => '0'); --  $a_{(1\sim 4)} = 1001$

# Các phép toán số học

- Để thực hiện các phép toán số học **có dấu** trên kiểu logic, sử dụng **library IEEE; use IEEE.std\_logic\_signed.all;**
- Để thực hiện các phép toán số học **không dấu** trên kiểu logic, sử dụng **library IEEE; use IEEE.std\_logic\_unsigned.all;**
- Không được đồng thời sử dụng 2 gói có dấu và không dấu.
- Ví dụ

```
Av : in std_logic_vector(3 downto 0);
Bv : in std_logic_vector(3 downto 0);
Cv : out std_logic_vector(3 downto 0);
.....
Cv <= Av + Bv;
Cv <= Av - Bv;
Cv <= Av * Bv;
.....
```

- Biên dịch các bit trong kiểu STD\_LOGIC\_VECTOR

**Không dấu:** "0101" biểu diễn số 5, "1101" biểu diễn số 13  
**Có dấu:** "0101" biểu diễn số 5, "1101" biểu diễn số -3

# Chuyển đổi kiểu dữ liệu

- Trong trường hợp muốn sử dụng các phép toán số học phức tạp như abs, mũ... thì phải
  - Chuyển đổi kiểu STD\_LOGIC\_VECTOR thành kiểu INTEGER
  - Thực hiện phép toán trên kiểu INTEGER
  - Chuyển đổi kiểu INTEGER thành STD\_LOGIC\_VECTOR
- conv\_integer(p):***
  - chuyển đổi STD\_LOGIC\_VECTOR → INTEGER.
  - Hàm thuộc cả 2 gói
    - `use IEEE.std_logic_signed.all;`
    - `use IEEE.std_logic_unsigned.all;`
- conv\_std\_logic\_vector(p, b):***
  - chuyển đổi INTEGER → STD\_LOGIC\_VECTOR có kích thước b bit.
  - Hàm thuộc gói
    - `use IEEE.std_logic_arith.all;`

# Ví dụ về chuyển đổi kiểu dữ liệu

```
library IEEE;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_arith.all;

...
Av : in std_logic_vector(3 downto 0);
Bv : in std_logic_vector(3 downto 0);
Cv : out std_logic_vector(3 downto 0);
.....
process (Av, Bv)
 variable varA, varB, varC: integer;
begin
 varA:= CONV_INTEGER(Av); --STD_LOGIC_VECTOR --> integer
 varB:= CONV_INTEGER(Bv); --
 varC:= varA+ varA**varB + varA mod 3 - varA rem varB + varA/varB;
 Cv <= CONV_STD_LOGIC_VECTOR(varC,BUS_WIDTH);
end process;
....
```



# VHDL: Toán tử và thuộc tính

# Toán tử gán



gán giá trị cho SIGNAL

```
SIGNAL x : STD_LOGIC;
x <= '1';
```



dùng gán giá trị cho VARIABLE, CONSTANT,  
GENERIC

```
VARIABLE x : integer;
x := 5;
```



dùng gán giá trị cho từng phần tử của kiểu vector

```
SIGNAL x : STD_LOGIC_VECTOR(3 downto 0);
x <= (2 => '0', others => '1');
```

# Toán tử logic

- VHDL định nghĩa các toán tử logic sau:  
**NOT, AND, OR, NAND, NOR, XOR, XNOR**
- Dữ liệu cho các toán tử này phải là kiểu:  
**STD\_LOGIC, STD\_LOGIC\_VECTOR,**
- Ví dụ:

```
y <= NOT (a AND b);
```

```
y <= a NAND b;
```

# Toán tử số học

- Dùng cho các kiểu dữ liệu số như là: INTEGER, SIGNED, UNSIGNED, REAL.
- Bao gồm:
  - + Toán tử cộng
  - Toán tử trừ
  - \* Toán tử nhân
  - / Toán tử chia
  - \*\* Toán tử lấy mũ
  - MOD Phép chia lấy phần nguyên ( $Y \text{ MOD } X$  có dấu của  $X$ )
  - REM Phép chia lấy phần dư ( $Y \text{ REM } X$  có dấu của  $Y$ )
  - ABS Phép lấy giá trị tuyệt đối

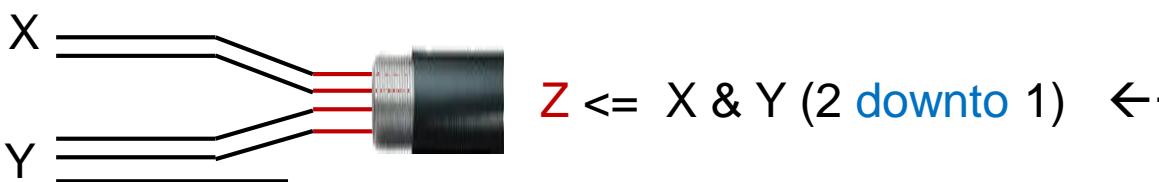
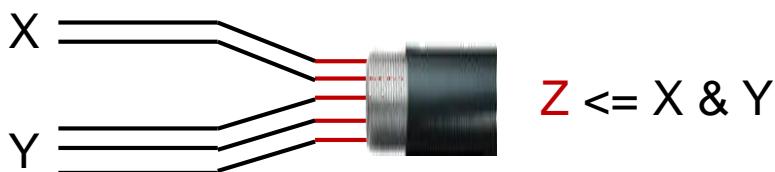
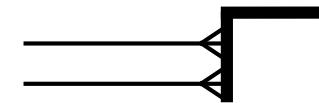
# Toán tử so sánh

- Có các toán tử so sánh sau:
  - = So sánh bằng
  - /= So sánh khác nhau
  - < So sánh nhỏ hơn
  - > So sánh lớn hơn
  - <= So sánh nhỏ hơn hoặc bằng
  - >= So sánh lớn hơn hoặc bằng

- Chỉ tác động lên 2 toán hạng có cùng kiểu
- Kết quả là một giá trị Boolean

# Toán tử ghép nối

- Toán tử ghép nối ( $\&$ ) để ghép các tín hiệu đơn/đa thành đa tín hiệu với tên mới.
- Toán tử ghép nối ( $\&$ ) giống như việc bó một nhóm dây nối thành bus



$$\begin{cases} Z(3) \leq X(1); \\ Z(2) \leq X(0); \\ Z(1) \leq Y(2); \\ Z(0) \leq Y(1); \end{cases}$$

# Toán tử dịch

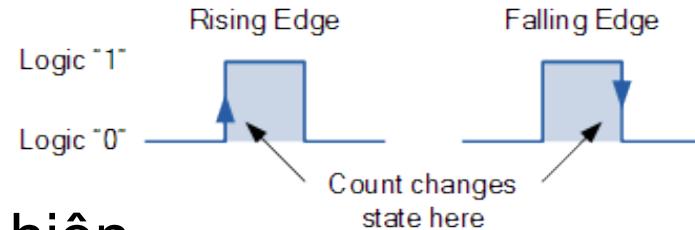
- Cú pháp sử dụng toán tử dịch là:  
`<left operand> <shift operation> <right operand>`
- Trong đó:
  - `<left operand>` : kiểu là BIT\_VECTOR
  - `<right operand>` : kiểu là INTEGER
- Có hai toán tử dịch:
  - **sll**    Toán tử dịch trái  $\leftarrow$
  - **rll**    Toán tử dịch phải  $\rightarrow$
- Ví dụ:  
`signal x: bit_vector(7 downto 0);  
x <= x sll 2; -- tương đương x<= x(5 downto 0) &"00";`

# Thuộc tính của đơn tín hiệu

- Với đơn tín hiệu `SIGNAL s : STD_LOGIC;`
- có các thuộc tính sau:

- `s'EVENT` : Trả về True khi một sự kiện xảy ra đối với s
  - Ví dụ: Khi s chuyển từ logic 0 → 1, hoặc từ 1 → 0
- `s'STABLE`: Trả về True nếu s đang giữ nguyên giá trị
  - Ví dụ Khi s đang duy trì logic 0, hoặc duy trì logic 1
  - `s'STABLE` tương đương với `not s'EVENT`, và ngược lại

- 4 câu lệnh để kiểm tra sự kiện xuất hiện sườn dương của xung clk là tương đương nhau
  - `IF (clk'EVENT and clk='1') THEN`
  - `IF (not clk'STABLE and clk='1') THEN`
  - `WAIT UNTIL (clk'EVENT and clk='1') THEN`
  - `IF rising_edge(clk) THEN`



# Thuộc tính của đa tín hiệu

1/2

- Với đa tín hiệu `SIGNAL d : STD_LOGIC_VECTOR(2 to 9)`
- có các thuộc tính sau:
  - **d'LOW** Trả về giá trị nhỏ nhất của chỉ số mảng  
d'LOW = 2;
  - **d'HIGH** Trả về chỉ số lớn nhất của chỉ số mảng  
d'HIGH = 9;
  - **d'LEFT** Trả về chỉ số bên trái nhất của mảng  
d'LEFT = 2;
  - **d'RIGHT** Trả về chỉ số bên phải nhất của mảng  
d'RIGHT = 9;
  - **d'LENGTH** Trả về kích thước của vector  
d'LENGTH = 8;
  - **d'RANGE** Trả về khoảng chỉ số của vector  
d'RANGE = (2 to 9) ; d(d'RANGE) <= "01010101";
  - **d'REVERSE\_RANGE** Trả về khoảng chỉ số theo thứ tự đảo ngược  
d' REVERSE\_RANGE = (9 downto 2) ;

# Thuộc tính của đa tín hiệu

2/2

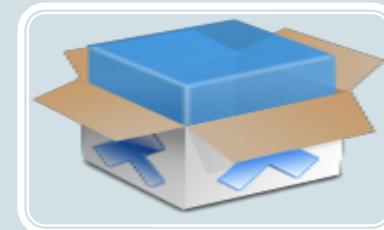
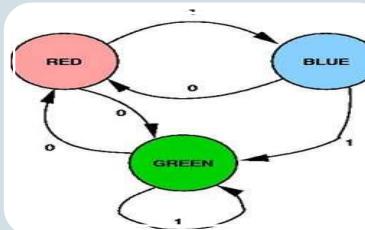
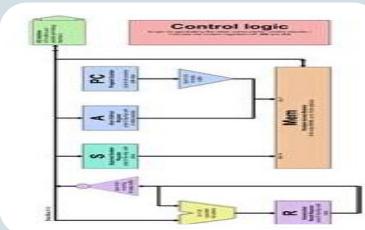
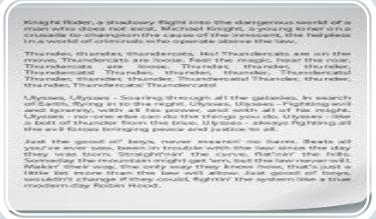
- Các thuộc tính này có thể dùng trong các vòng lặp:

- `FOR i IN RANGE (0 TO 7)  
LOOP`  
...
- `FOR i IN d'RANGE  
LOOP`  
...
- `FOR i IN RANGE (d'LOW TO d'HIGH)  
LOOP`  
...
- `FOR i IN RANGE (0 TO d'LENGTH-1)  
LOOP`  
...



# VHDL: Phương pháp thiết kế

# Các phương pháp thiết kế



# Viết trực tiếp mã nguồn bằng VHDL, Verilog.

# Sơ đồ khôi sơ đồ cấu trúc

# Sơ đồ trạng thái

# Kết hợp

- Thiết kế bằng sơ đồ sẽ được sinh tự động ra ngôn ngữ HDL, không đòi hỏi nhiều kỹ năng về HDL.
  - Thiết kế bằng sơ đồ mang tính trực quan hơn, khả năng tổng hợp được lên phần cứng cao hơn.

# Thiết kế bằng code HDL

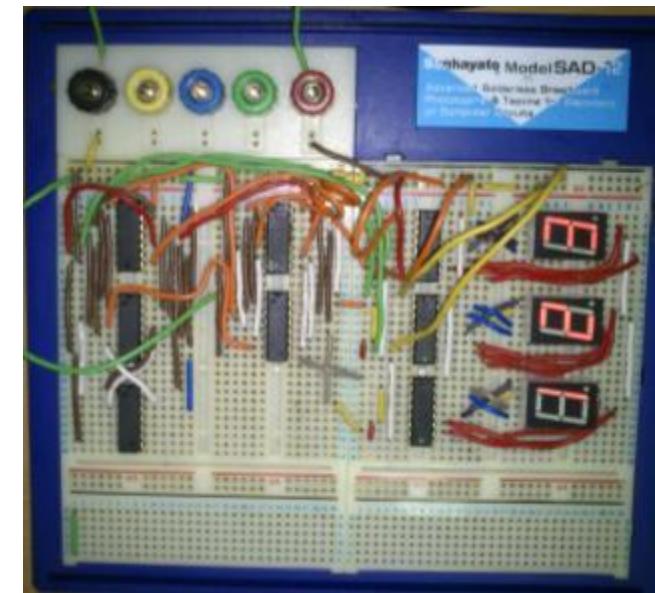
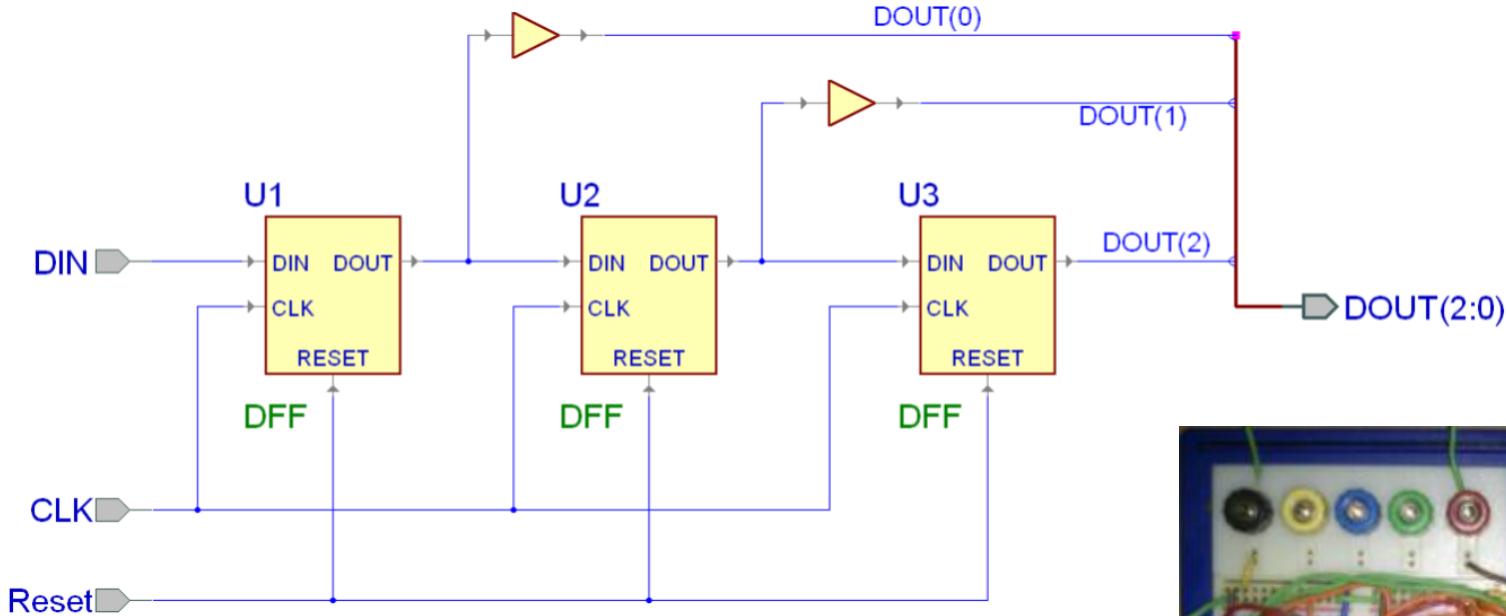
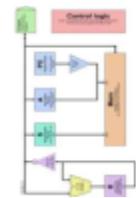


```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

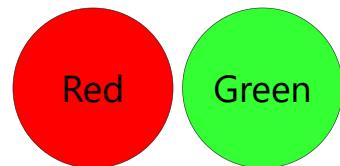
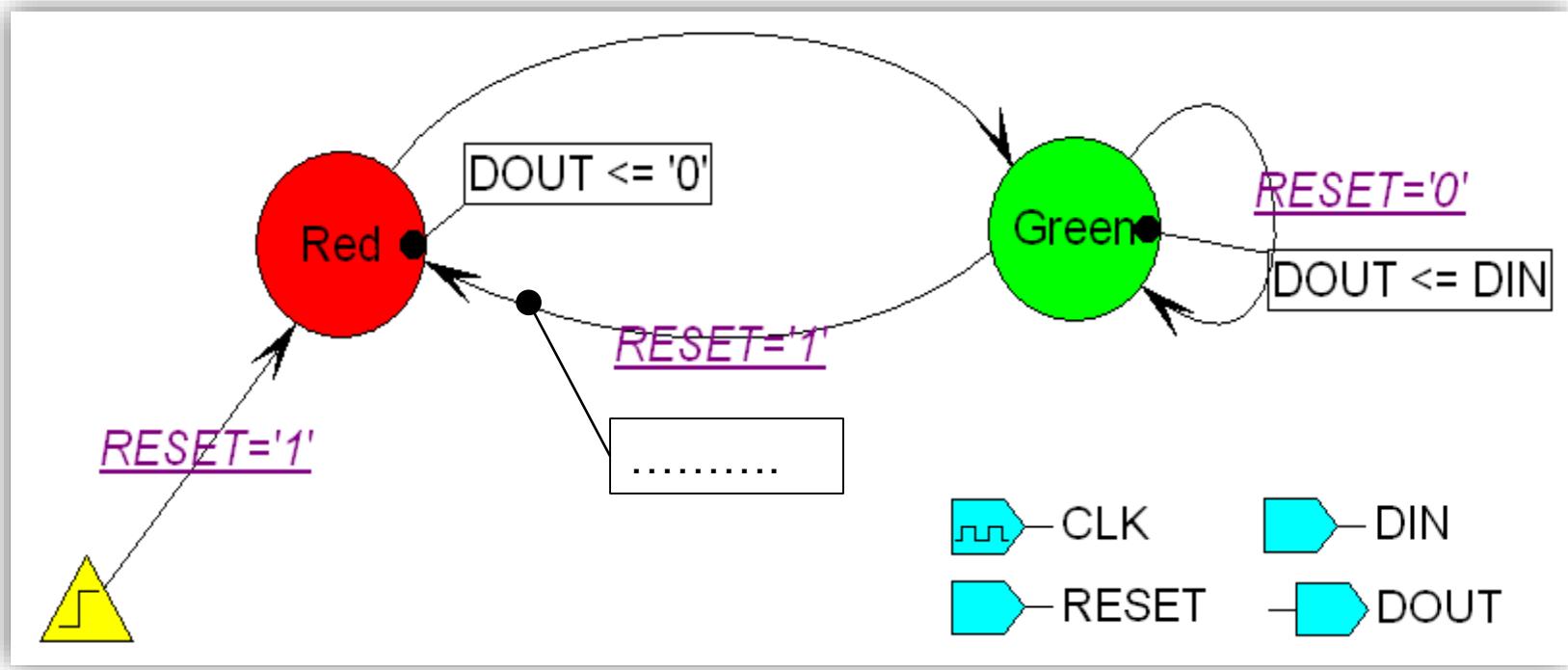
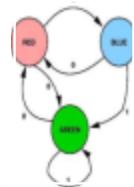
entity Mach_AND is
 port(
 a : in STD_LOGIC;
 b : in STD_LOGIC;
 o : out STD_LOGIC
);
end Mach_AND;

architecture Cach_Hoat_Dong_01 of Mach_AND is
begin
 o <= a and b;
end Cach_Hoat_Dong_01;
```

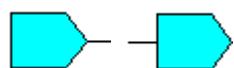
# Thiết kế bằng sơ đồ cấu trúc



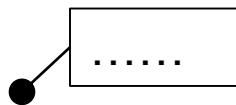
# Thiết kế bằng sơ đồ trạng thái



Các trạng thái



Chân vào ra



Hành động ứng với  
trạng thái, hoặc  
phiên chuyển t.thái  
Tín hiệu đồng hồ

# Gợi ý sử dụng các phương pháp

## Sơ đồ trạng thái

- Thiết kế để xử lý một thuật toán, ít thực thể con.
- Thiết kế được mô tả với nhiều mệnh đề “nếu.. thì..”
- Thiết kế thực thể điều khiển, thực thể trọng tài, thực thể điều phối, phân xử.
- Phù hợp với thiết kế CU, phần xử ngắt, điều phối pipeline trong CPU.

## Sơ đồ cấu trúc

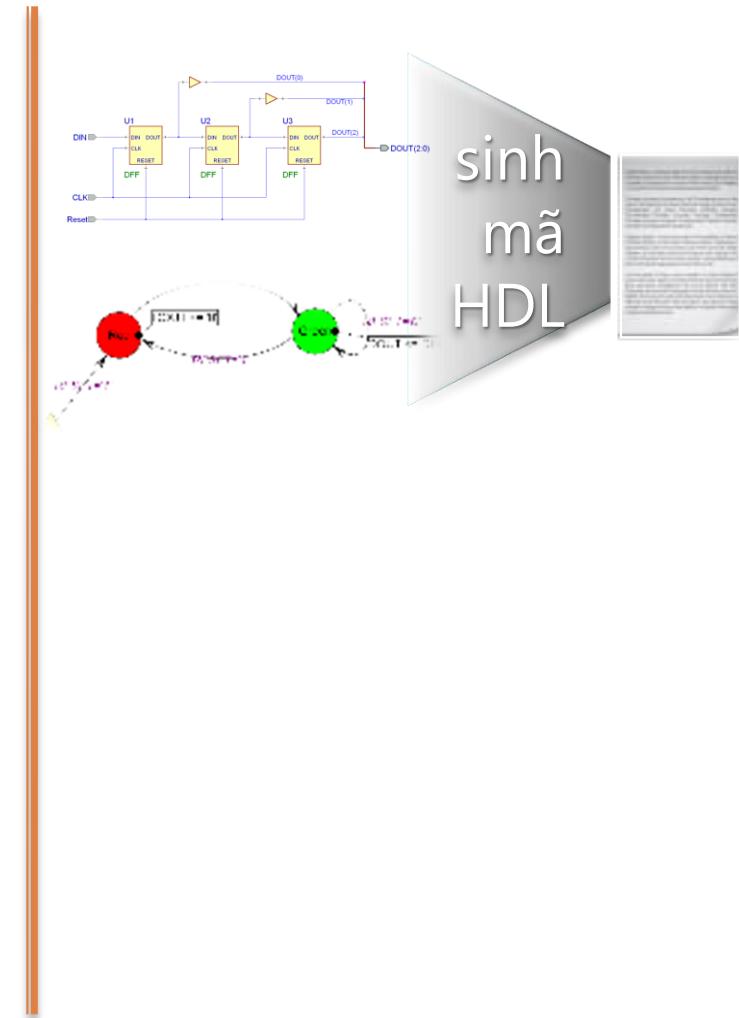
- Thiết kế không có tính thuật toán, nhiều thực thể con.
- Thiết kế mang tính “giao tiếp và phối phép” thực thể.
- Thiết kế mạch tổ hợp, thực thể xử lý tại ngõ vào/ra, schematic.
- Phù hợp với thiết kế ALU, thiết kế thanh ghi đa năng/chuyên dụng, thiết kế tổng thể toàn bộ CPU

## Viết mã nguồn HDL

- Thiết kế nhỏ, dễ kiểm soát hoạt động; hoặc thuật toán, thiết kế có tính tuần tự cao.
- Giả lập môi trường hoạt động để test.
- Thiết kế tạm mức hệ thống (không có khả năng tổng hợp), sau đó được thay thế dần bằng các thiết kế khả tổng hợp, với tiếp cận top-down.
- Thiết kế FlipFlop, Mux, ALU,

# Mối quan hệ giữa HDL và thiết kế bằng sơ đồ

- Dù thiết kế bằng bảng sơ đồ cấu trúc hay trạng thái, thiết kế đó đều được dùng để sinh ra mã HDL.
  - Thiết kế bằng sơ đồ cung cấp góc nhìn toàn diện hơn, logic hơn so với viết trực tiếp mã HDL



# Sinh mã HDL từ FSM

1/2

```
architecture FSM_ARCH of entity_name is
attribute enum_encoding: string;
-- Khai báo các trạng thái
type Sreg_type is (
 Stop, Init, Go
);

```

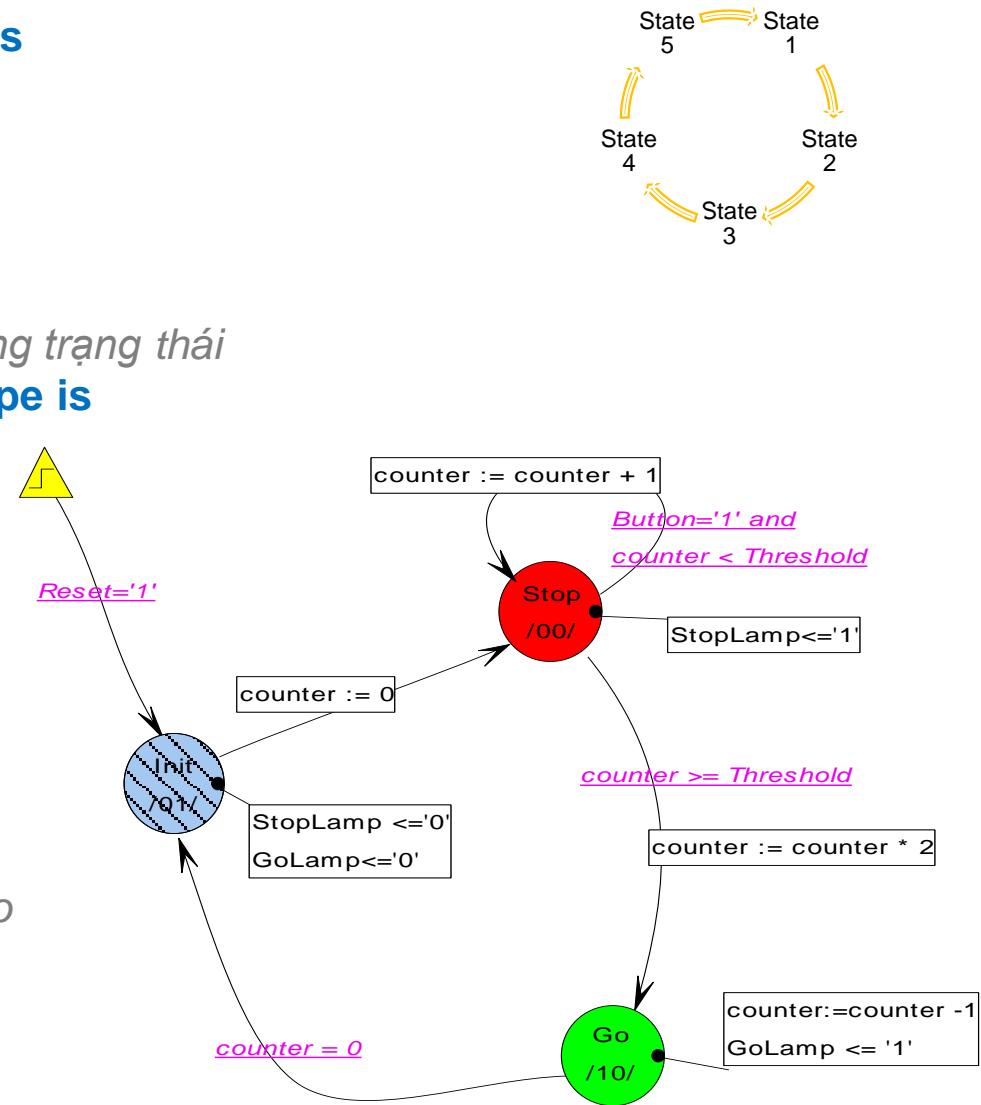
-- Nếu cần, chỉ định rõ giá trị binary của từng trạng thái  
attribute enum\_encoding of Sreg\_type type is

|        |         |
|--------|---------|
| "00" & | -- Stop |
| "01" & | -- Init |
| "10" ; | -- Go   |

-- Biến chỉ trạng thái hiện tại  
signal Sreg: Sreg\_type;

```
begin
process (
 if CLK'event and CLK = '1' then

 end if;
 end process;
end architecture FSM_ARCH
```



# Sinh mã HDL từ FSM

2/2

-- Quá trình biến đổi trạng thái  
case Sreg is

-- Chuyển từ trạng thái Go thành ??  
when Go =>

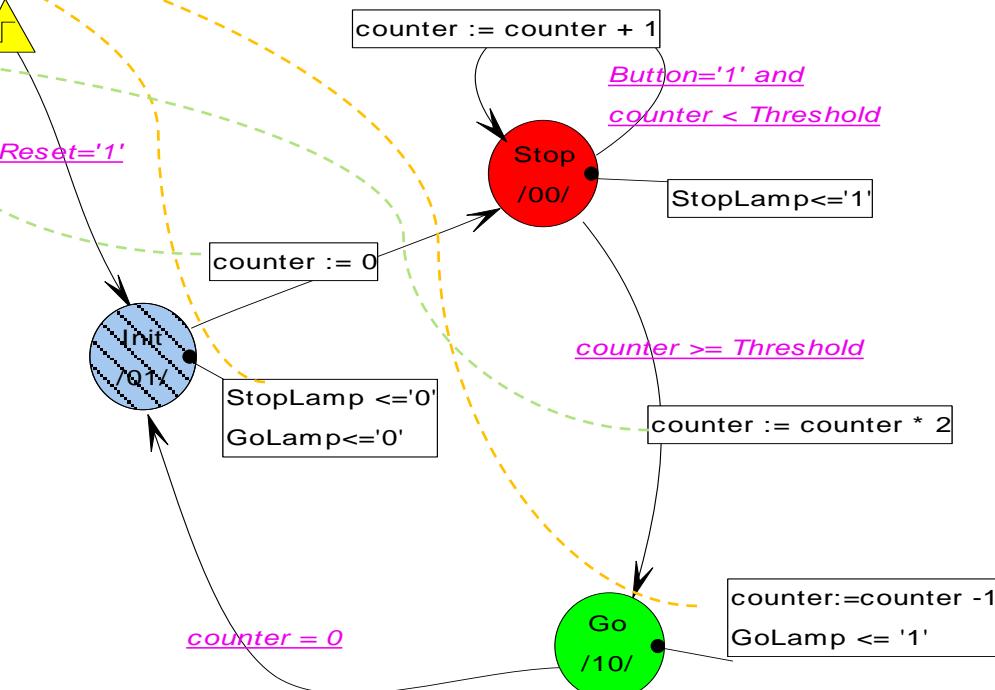
State Action *← Hành động của trạng thái*  
if condition then  
    Sreg <= State2;  
    Transition Action;  
else condition) then  
    Transition Action;  
else  
    Sreg <= Default State  
end if

Không có lệnh chuyển  
trạng thái Sreg=?  
→ giữ nguyên trạng thái

-- Biến đổi từ trạng thái Stop thành ??  
when Stop =>

-- Biến đổi từ trạng thái còn lại  
when others =>  
    Sreg <= Init State

end case



# Sinh mã HDL từ sơ đồ cấu trúc

```
entity ShiftReg_Parity is
port(
 CE : in STD_LOGIC;
 CLK : in STD_LOGIC;
 ParityType : in STD_LOGIC;
 Serial_nParallel : in STD_LOGIC;
 Din : in STD_LOGIC_VECTOR(1 downto 0);
 SerialOut : out STD_LOGIC
);
end ShiftReg_Parity;
```

```
architecture ShiftReg_Parity of ShiftReg_Parity is
```

```
---- Component declarations -----
```

```
component DFF_CE
port (
 CE : in STD_LOGIC;
 CLK : in STD_LOGIC;
 D : in STD_LOGIC;
 Q : out STD_LOGIC
);
end component;
component mux2x1
port (
 I0 : in STD_LOGIC;
 I1 : in STD_LOGIC;
 S : in STD_LOGIC;
 O : out STD_LOGIC
);
```

```
.....
U2 : mux2x1
port map(
 I0 => Din(0),
 I1 => NET472,
 O => NET165,
 S => Serial_nParallel
);
```

```
U3 : mux2x1
port map(
 I0 => Din(1),
 I1 => NET588,
 O => NET195,
 S => Serial_nParallel
);
```

```
NET2684 <= not(Serial_nParallel);
NET395 <= Din(1) xor Din(0);
NET209 <= NET2684 and NET2782;
```

```
end ShiftReg_Parity;
```



Phần IV:

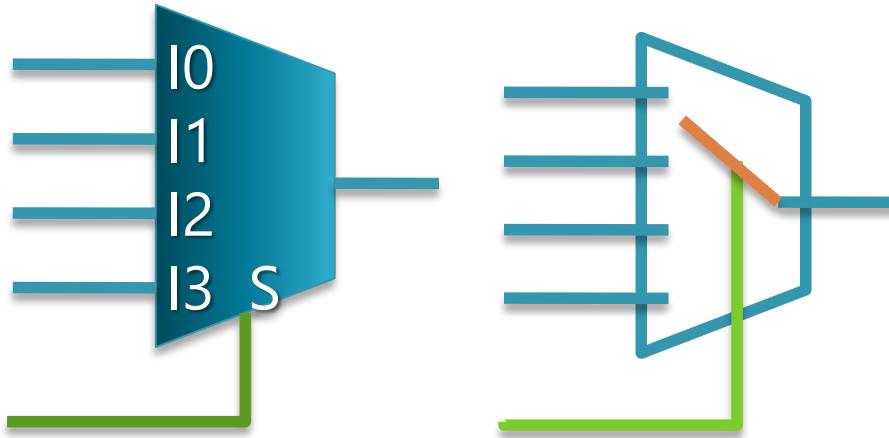
# Thiết kế mạch số cơ bản

- Bộ dồn kênh / phân kênh
- ALU
- RAM/Thanh ghi đa năng
- Bộ nhớ chương trình
- Truy cập bộ nhớ chính
- Bộ định thời
- Thanh ghi chuyên dụng
- Bộ giải mã lệnh tuần tự
- Bộ giải mã lệnh rẽ nhánh

# Bộ ghép kênh - Mux

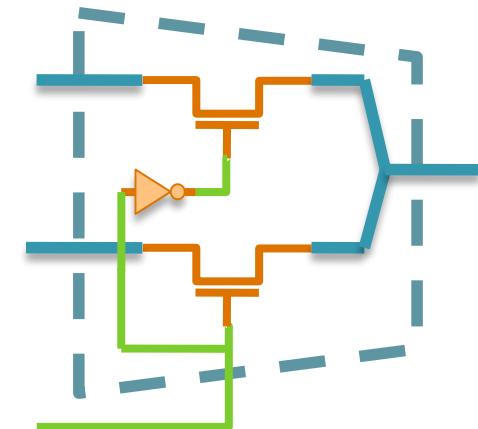


MUX4x1



- IC: Mux/Demux
- Lập trình HDL: Lệnh rẽ nhánh If ~.
- Mỗi lệnh if trong HDL, tương đương với một IC Mux/Demux, và ngược lại.

MUX2x1: Process (I,S)  
Begin  
If (S='1') Then  
    O <= I(1);  
Else  
    O <= I(0);  
End If;  
End Process;



Mô tả VHDL và sơ đồ mạch của MUX2x1

# Bộ ghép kênh – Mux (bài tập)

- Bài tập 1: Sinh viên Thuật toán muốn thực hiện hàm  $Y = A\bar{B} + \bar{B}\bar{C} + \bar{A}\bar{B}C$ , nhưng phòng lab hiện giờ chỉ còn lại 1 bộ ghép kênh 8x1. Hỏi cách thực hiện Y?

• Gợi ý

• Đáp án

Nguyên lý cần ghi nhớ

# Bộ ghép kênh – Mux (bài tập)

- Bài tập 2: Thuật đưa bạn gái mượn bộ ghép kênh 8x1, nên giờ chỉ còn 1 bộ MUX 4x1 và 1 bộ NOT để thực hiện hàm  $Y = AB + \bar{B}\bar{C} + \bar{A}BC$ . Hỏi cách thực hiện Y?



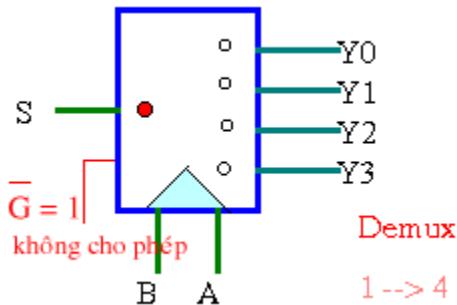
Gợi ý



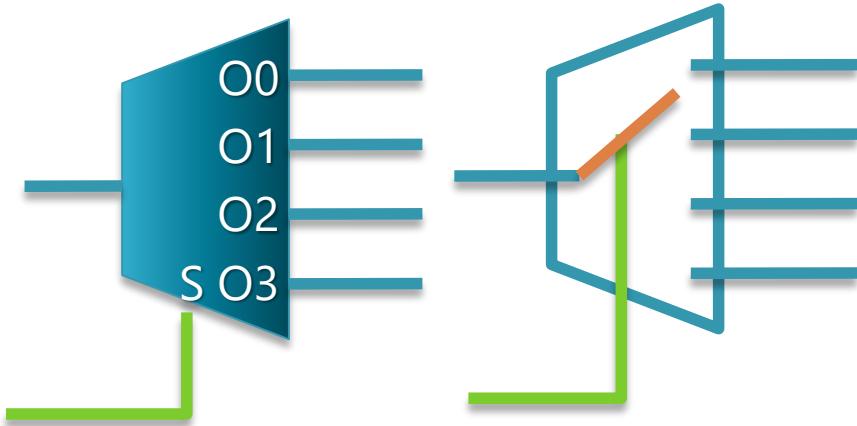
Đáp án

# Bộ phân kênh - Demux

DEMUX4x1



| Ngõ vào chọn |   | S ra ở ngõ |
|--------------|---|------------|
| B            | A |            |
| 0            | 0 | Y0         |
| 0            | 1 | Y1         |
| 1            | 0 | Y2         |
| 1            | 1 | Y3         |



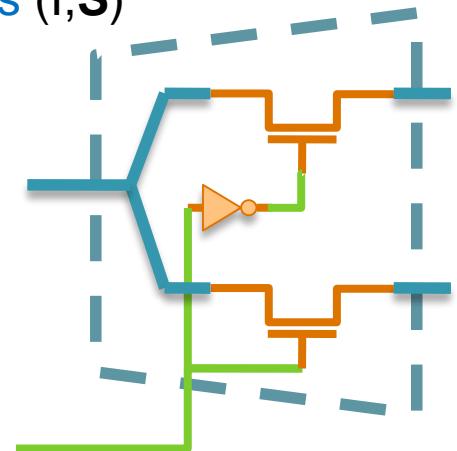
- IC: Mux/Demux
- Lập trình HDL: Lệnh rẽ nhánh If ~



2 bên truyền, nhận sử dụng Mux/Demux  
để giảm số dây dẫn truyền tín hiệu

DEMUX2x1: Process (I,S)

```
Begin
 If (S='1') Then
 O(0) <= I;
 Else
 O(1) <= I;
 End If;
End Process;
```

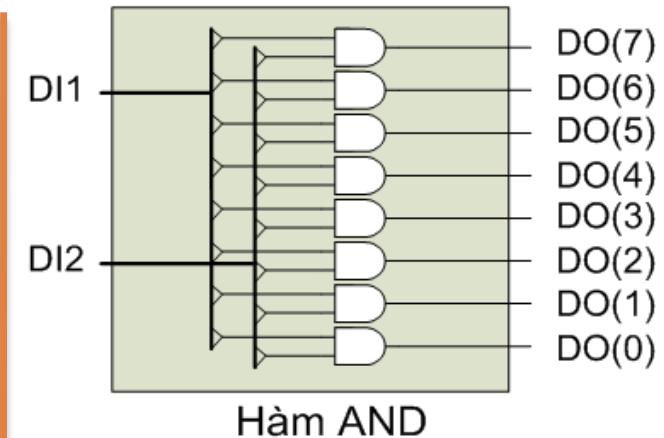


Mô tả VHDL và sơ đồ mạch của DEMUX2x1

# Các phép logic AND/OR, MOV

- DO = DI1 **and** DI2

```
architecture and_arch of ander is
signal tmp: std_logic_vector(7 downto 0);
begin
 tmp <= DI1 and DI2; -- Trung gian để tính cờ Zero
 DO <= tmp;
 ZF <= '1' when tmp = 0 else '0'; -- Tính cờ Zero
end architecture;
```



- DO = DI (phép gán)

```
architecture move_arch of mover is
begin
 DO <= DI;
end architecture;
```

# Các phép logic dịch bit

- DO = DI1 **shr** 1

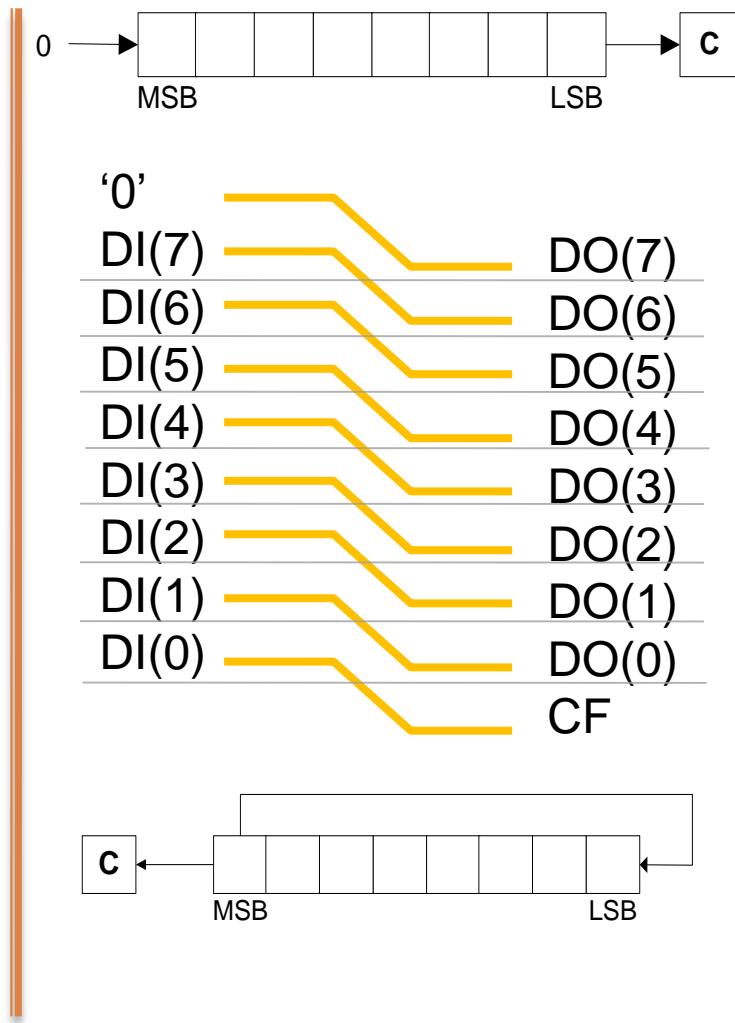
```
architecture shr_arch of shr is
begin
```

```
DO <= '0' & DI(7 downto 1);
CF <= DI(0);
end architecture;
```

- DO = DI1 **rr** 1

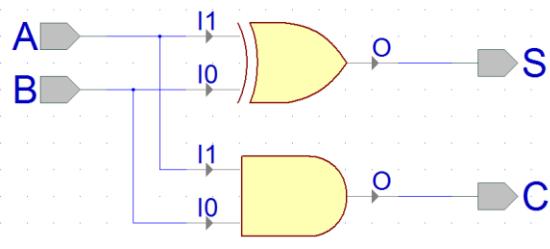
```
architecture rr_arch of rr is
begin
```

```
DO <= DI(6 downto 0) & D(7);
CF <= DI(7);
end architecture;
```



# Bộ bán tổng – Half Adder

- Bộ bán tổng thực hiện phép cộng 2 bit đầu vào
- Thiết kế bằng Block Diagram



- VHDL

```
library IEEE;
use IEEE.std_logic_1164.all;
entity HalfAdder is
port(A : in STD_LOGIC;
 B : in STD_LOGIC;
 C : out STD_LOGIC;
 S : out STD_LOGIC);
end HalfAdder;
architecture df of HalfAdder is
begin
 S <= B xor A;
 C <= B and A;
end df;
```

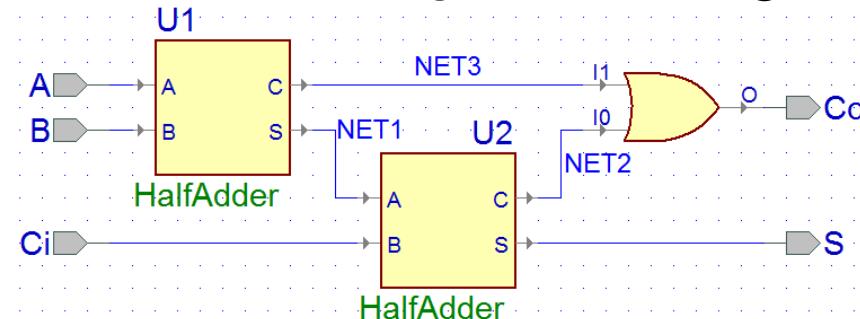
# Bộ toàn tổng – Full Adder

- Bộ toàn tổng thực hiện phép cộng 3 bit đầu vào

## VHDL

```
architecture structure of FullAdder is
component HalfAdder
 port (A : in STD_LOGIC;
 B : in STD_LOGIC;
 C : out STD_LOGIC;
 S : out STD_LOGIC);
end component;
signal NET1 : STD_LOGIC;
signal NET2 : STD_LOGIC;
signal NET3 : STD_LOGIC;
begin
U1 : HalfAdder port map(A => A, B => B, C => NET3, S => NET1);
U2 : HalfAdder port map(A => NET1, B => Ci, C => NET2, S => S);
Co <= NET2 or NET3;
end structure;
```

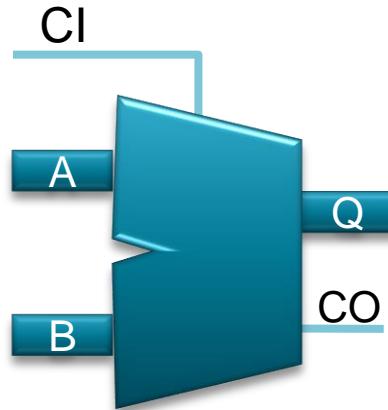
- Thiết kế bằng Block Diagram



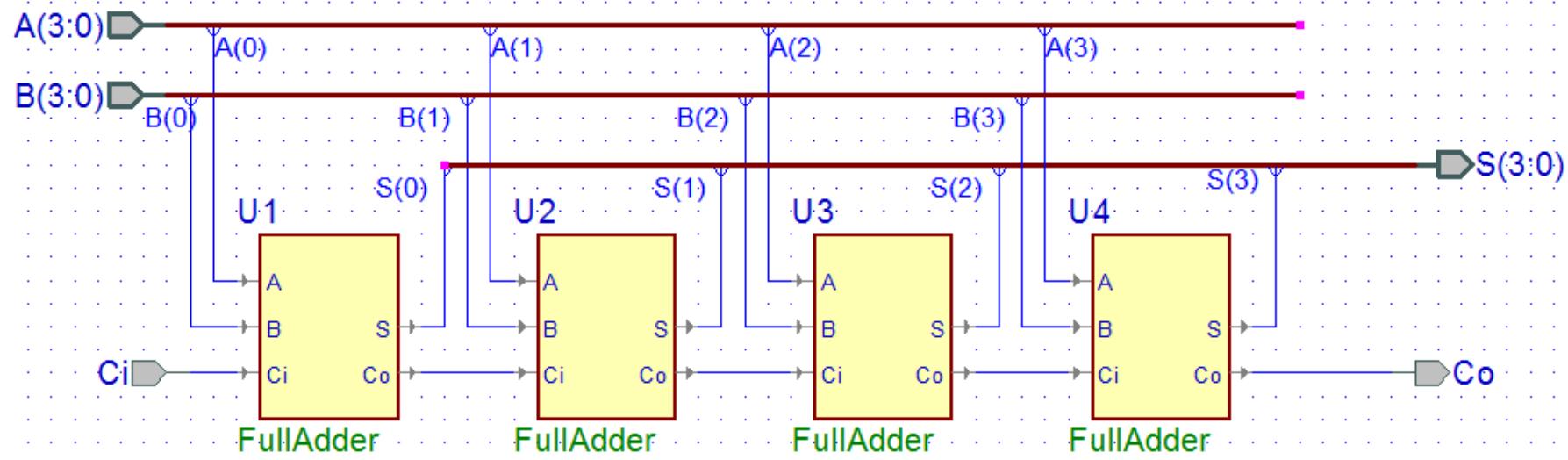
```
entity FullAdder is
 port(A : in STD_LOGIC;
 B : in STD_LOGIC;
 Ci: in STD_LOGIC;
 Co: out STD_LOGIC;
 S : out STD_LOGIC);
end FullAdder;
```

# Bộ cộng: thiết kế bằng block diagram

Mô tả entity, biểu tượng



```
entity Adder_BD is
 port(
 Ci : in STD_LOGIC;
 A : in STD_LOGIC_VECTOR(3 downto 0);
 B : in STD_LOGIC_VECTOR(3 downto 0);
 Co : out STD_LOGIC;
 S : out STD_LOGIC_VECTOR(3 downto 0)
);
end Adder_BD;
```



# Bộ cộng: block diagram → VHDL

```
architecture structure of Adder_BD is
component FullAdder
 port (A : in STD_LOGIC;
 B : in STD_LOGIC;
 Ci: in STD_LOGIC;
 Co: out STD_LOGIC;
 S : out STD_LOGIC);
end component;

signal NET1 : STD_LOGIC;
signal NET2 : STD_LOGIC;
signal NET3 : STD_LOGIC;

begin
U1 : FullAdder
 port map(A => A(0),B => B(0),Ci => Ci,Co => NET1,S => S(0));
U2 : FullAdder
 port map(A => A(1),B => B(1),Ci => NET1,Co => NET2,S => S(1));
U3 : FullAdder
 port map(A => A(2),B => B(2),Ci => NET2,Co => NET3,S => S(2));
U4 : FullAdder
 port map(A => A(3),B => B(3),Ci => NET3,Co => Co,S => S(3));
end structure ;
```

# Bộ cộng: VHDL dạng dataflow

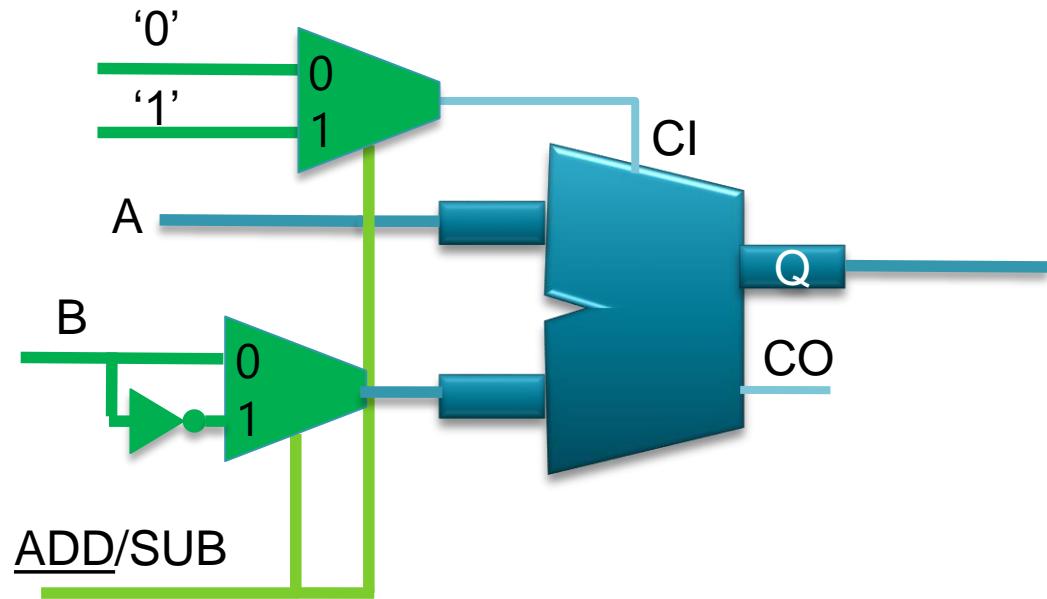
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
entity Adder is
 generic (BUS_WIDTH : integer := 8);
 port(
 Ci : in STD_LOGIC;
 A : in STD_LOGIC_VECTOR(BUS_WIDTH-1 downto 0);
 B : in STD_LOGIC_VECTOR(BUS_WIDTH-1 downto 0);
 S : out STD_LOGIC_VECTOR(BUS_WIDTH-1 downto 0);
 Co : out STD_LOGIC
);
end Adder;
-- Ci Ci Nguyen ly chung
-- A 0 A
-- + B + 0 B
-- ----- <--> -----
-- Co S Co S
architecture MyDataflow of Adder is
signal Aex: STD_LOGIC_VECTOR(BUS_WIDTH downto 0);-- Mo rong toan hang A
signal Bex: STD_LOGIC_VECTOR(BUS_WIDTH downto 0);-- Mo rong toan hang B
signal Sex: STD_LOGIC_VECTOR(BUS_WIDTH downto 0);-- Mo rong ket qua
begin
 Aex <= '0' & A ;
 Bex <= '0' & B;
 Sex <= Aex + Bex + Ci ;
 Co <= Sex(BUS_WIDTH);
end MyDataflow;
```

# Bộ cộng/trừ kết hợp

- IC thực hiện được cả phép cộng và trừ

- $A + B = A + B + 0$
- $A - B = A + \bar{B} + 1$

- IC cộng trừ kết hợp  
= IC cộng + 2 bộ MUX

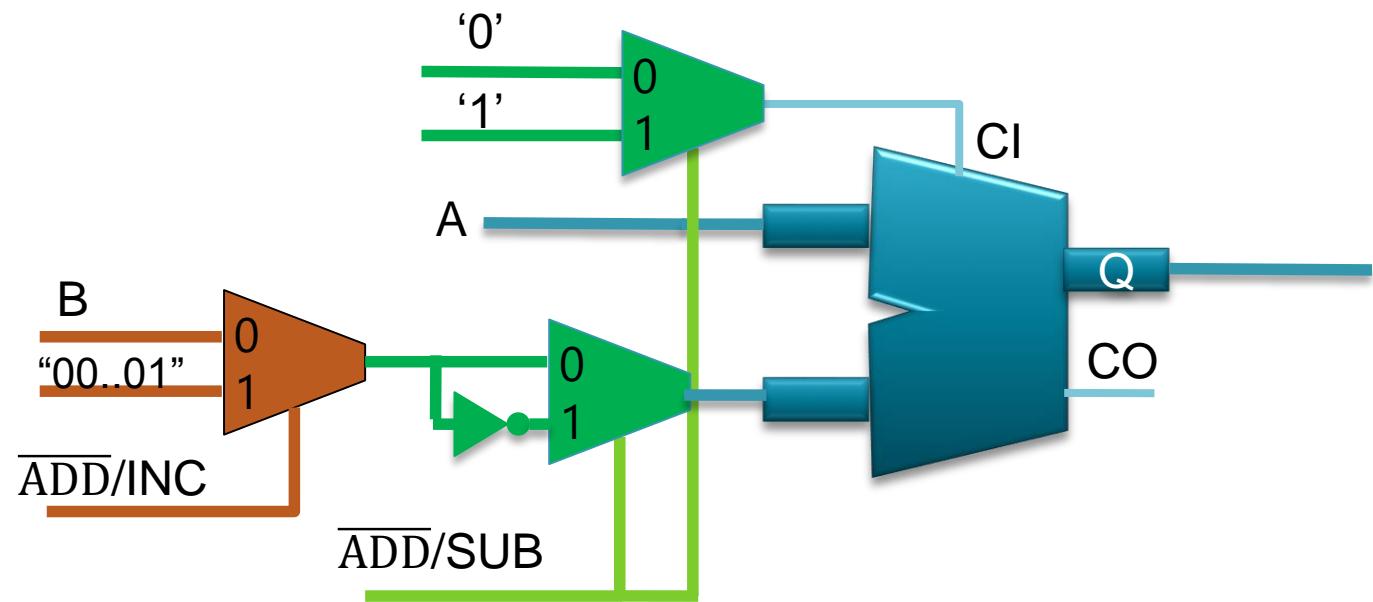


# Bộ cộng/trừ kết hợp đếm tăng giảm

## Phép đếm

- $A + B = A + B + 0$
- $A - B = A + \bar{B} + 1$
- $A + 1 = A + 1 + 0$
- $A - 1 = A + \bar{1} + 1$

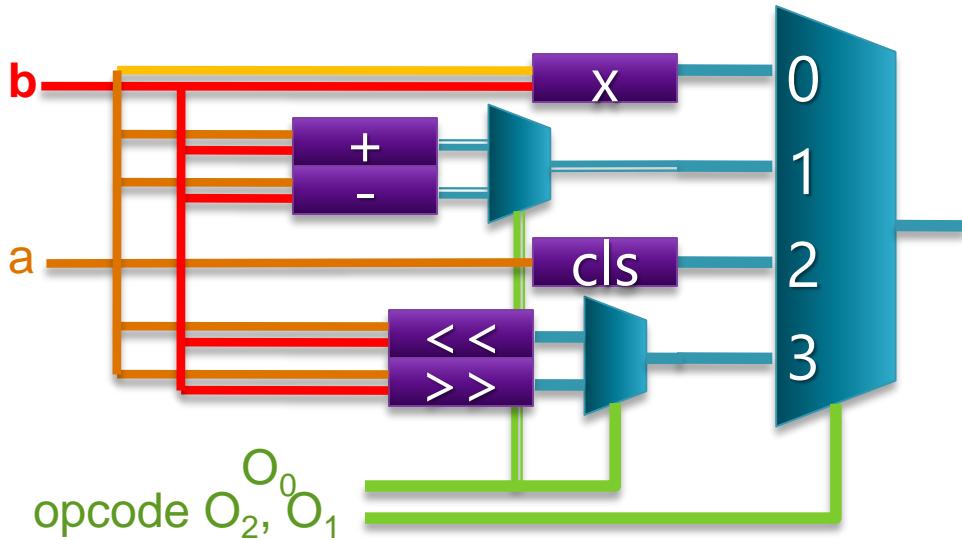
IC cộng trừ kết hợp đếm  
= IC cộng trừ kết hợp + 1 bộ MUX



# Thiết kế ALU

1/4

- ALU thường là mạch tổ hợp với 2 thành phần:
  - Các mạch thực hiện phép toán.
  - Các bộ ghép kênh.
- Sơ đồ nguyên lý



| Opcode                                       | ALU     |
|----------------------------------------------|---------|
| O <sub>2</sub> O <sub>1</sub> O <sub>0</sub> |         |
| 0 0 x                                        | a x b   |
| 0 1 0                                        | a + b   |
| 0 1 1                                        | a - b   |
| 1 0 x                                        | clear a |
| 1 1 0                                        | a << b  |
| 1 1 1                                        | a >> b  |

# Thiết kế ALU

2/4

- Ngoài ra, các bộ ghép kênh còn được sử dụng để phối ghép trạng thái cờ từ các phép toán, theo cách tương tự.
- Nhận xét: cho dù mã lệnh chỉ yêu cầu thực hiện 1 phép toán, nhưng tất cả các mạch phép toán đều hoạt động.
- Có thể tiết kiệm năng lượng bằng cách ngắt các mạch tính toán vô ích khỏi nguồn cấp.
  - Dùng chính opcode để điều khiển cấp nguồn điện.
  - Điều này về cơ bản không ảnh hưởng tới tốc độ tính toán, vì mọi mạch phép toán đều thực hiện song song.

# Thiết kế ALU

3/4

- Opcode dùng để chọn mạch tính toán trên ALU, chính là opcode của tập lệnh CPU, có thể với một chút thay đổi nhỏ.
- Thiết lập bộ mã cho opcode (ví dụ phép + ứng với mã 010) có ảnh hưởng tới hiệu năng của ALU, vì sẽ làm tăng/giảm hiệu quả giải mã opcode.
- Lưu ý thiết kế ALU:
  - Một số mạch phép toán có thể thực hiện nhiều phép toán (ví dụ bộ cộng/trừ). Khi đó opcode cho các phép toán này cần sai khác bit càng ít càng tốt → tối thiểu hóa tốt hơn → hiệu năng cao hơn.

# Thiết kế ALU

4/4

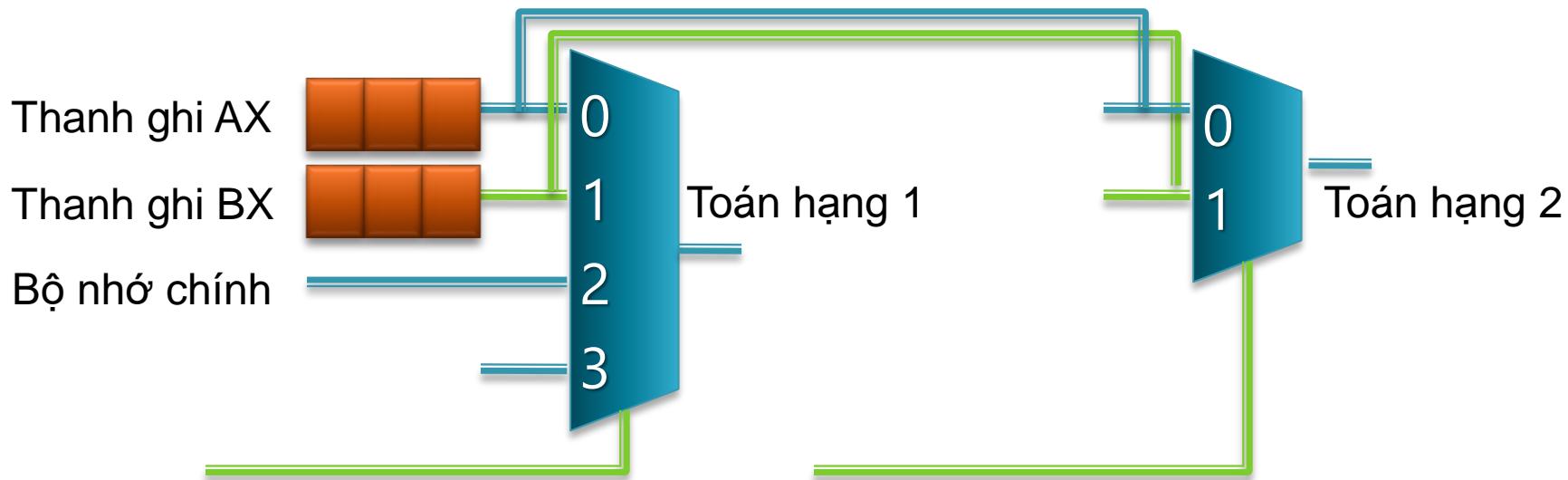
- Ví dụ về 2 cách mã opcode khác nhau. Cách thứ 2 gây khó khăn cho việc giải mã phép cộng → cần nhiều thời gian tính toán hơn, hiệu năng thấp.

| opcode     | ALU                       |
|------------|---------------------------|
| 00x        | $a \times b$              |
| <b>010</b> | <b><math>a + b</math></b> |
| <b>011</b> | <b><math>a - b</math></b> |
| 10x        | clear a                   |
| 110        | $a \ll b$                 |
| 111        | $a \gg b$                 |

| opcode     | ALU                       |
|------------|---------------------------|
| 00x        | $a \times b$              |
| <b>010</b> | <b><math>a + b</math></b> |
| 011        | clear a                   |
| <b>10x</b> | <b><math>a - b</math></b> |
| 110        | $a \ll b$                 |
| 111        | $a \gg b$                 |

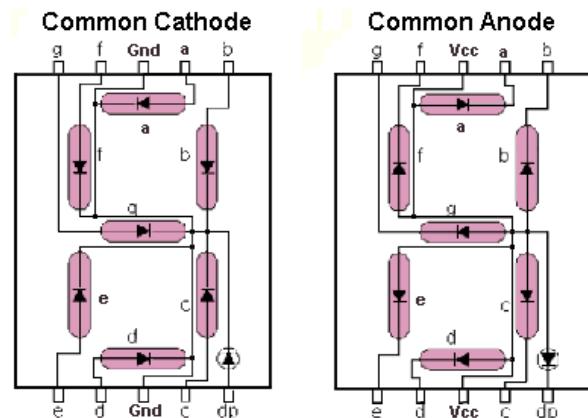
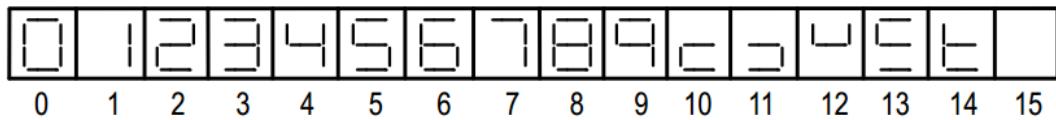
# Chọn toán hạng cho ALU

- ALU nhằm thực hiện phép toán
- PreALU cung cấp các toán hạng vào cho ALU
- Toán hạng vào có thể lấy từ tập thanh ghi, bộ nhớ chính, etc.
- 1 thanh ghi muốn đưa dữ liệu tới bao nhiêu module khác cũng được, nên có thể là toán hạng 1,2, đích. Nhưng bộ nhớ thì không.

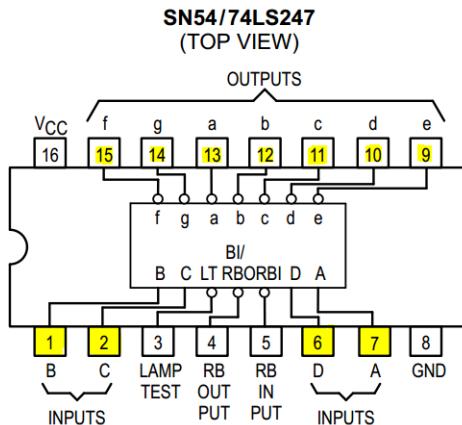


# Bộ giải mã led 7 đoạn

- Led 7-đoạn gồm 7 diot quang để tạo hình các chữ số từ 0 tới F<sub>16</sub>

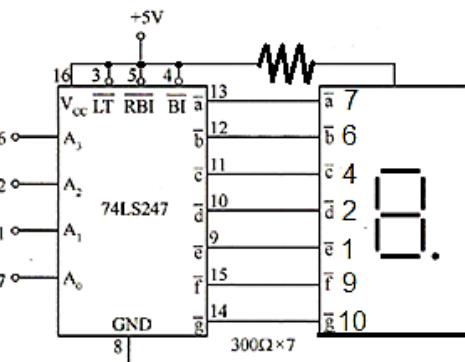


- Led 7-đoạn có 2 loại: chung Cathode(-) / chung Anode(+)
- Bộ giải mã led-7 đoạn dùng để hiển thị một số nhị phân 4 bit ra đèn led 7-đoạn



IC giải mã

Cách nối IC với led 7-đoạn

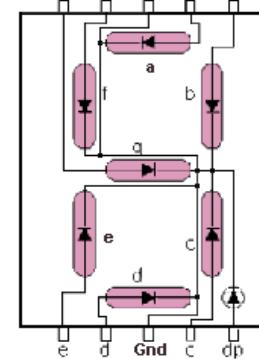


```
entity decoder7seg is
port(
 I : in STD_LOGIC_VECTOR
 (3 downto 0);
 a : out STD_LOGIC;
 b : out STD_LOGIC;
 c : out STD_LOGIC;
 d : out STD_LOGIC;
 e : out STD_LOGIC;
 f : out STD_LOGIC;
 g : out STD_LOGIC);
end decoder7seg;
```

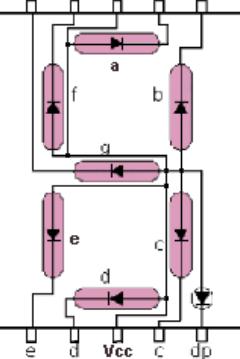
# Bộ giải mã led 7 đoạn

```
architecture behavior of decoder7seg is
 signal tmp: STD_LOGIC_VECTOR(6 downto 0); //~ abcdefg
begin
decode:process (I)
begin
 case I is
 when "0000" => tmp <= "0000001";
 when "0001" => tmp <= "1001111";
 when "0010" => tmp <= "0010010";
 when "0011" => tmp <= "0000110";
 when "0100" => tmp <= "1001100";
 when "0101" => tmp <= "0100100";
 when "0110" => tmp <= "0100000";
 when "0111" => tmp <= "0001111";
 when "1000" => tmp <= "0000000";
 when "1001" => tmp <= "0000100";
 when "1010" => tmp <= "0000010";
 when "1011" => tmp <= "1100000";
 when "1100" => tmp <= "0110001";
 when "1101" => tmp <= "1000010";
 when "1110" => tmp <= "0010000";
 when "1111" => tmp <= "0111000";
 when others => tmp <= (others => 'Z');
 end case;
end process decode;
a <= tmp(6);b <= tmp(5);c <= tmp(4);d <= tmp(3);e <= tmp(2);f <= tmp(1);g <= tmp(0);
end behavior;
```

Common Cathode



Common Anode



# Bộ so sánh

```
library IEEE;
use IEEE.std_logic_1164.all;

entity comp is
 port (
 inAeB : in std_logic;
 outAeB : out std_logic;
 A : in std_logic_vector(7 downto 0);
 B : in std_logic_vector(7 downto 0)
);
end entity;

architecture logic_level of comp is
begin
 outAeB <= '1' when (A = B) and (inAeB = '1') else '0';
end architecture logic_level ;

architecture gate_level of comp is
 signal same: std_logic;
begin
 same <= ((A(7) xor B(7)) = '0') and and ((A(0) xor B(0)) = '0')
 outAeB <= '1' when (same = '1') and (inAeB = '1') else '0';
end architecture gate_level;
```

--! Cho phép so sánh bằng  
--! Kết quả so sánh bằng. '1' là bằng nhau  
--! Toán hạng thứ 1  
--! Toán hạng thứ 2

--! Kiến trúc hoạt động mức logic

--! Kiến trúc hoạt động mức cỗng

# Phần tử nhớ D FlipFlop

**entity DFF is**

**port (**

CLR : in std\_logic;

--! Xóa bit nhớ, không đồng bộ

CE : in std\_logic

--! Cho phép Write

CLK : in std\_logic;

--! Tín hiệu đồng hồ, tích cực sườn lên

D : in std\_logic;

--! Bit dữ liệu vào

Q : out std\_logic

--! Bit dữ liệu ra

**);**

**end entity;**

**architecture behavior of DFF is**

**begin**

**process** (CLK, CLR) -- Process chỉ thực thi, khi có thay đổi trên CLK, CLR

**begin**

**if** CLR = '1' **then** -- Nếu tín hiệu xóa CLR=1, thì xóa đầu ra Q về 0

Q <= '0';

**elsif** rising\_edge(CLK) **then** -- Nếu CLR =0, và tại sườn lên của tín hiệu CLK

**if** CE = '1' **then** -- Nếu có tín hiệu cho phép Write, thì chốt lại

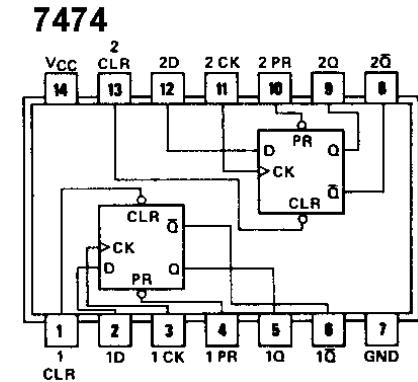
Q <= D;

**end if;** -- Các trường hợp còn lại, đầu ra Q không đổi, tức là nhớ.

**end if;**

**end process;**

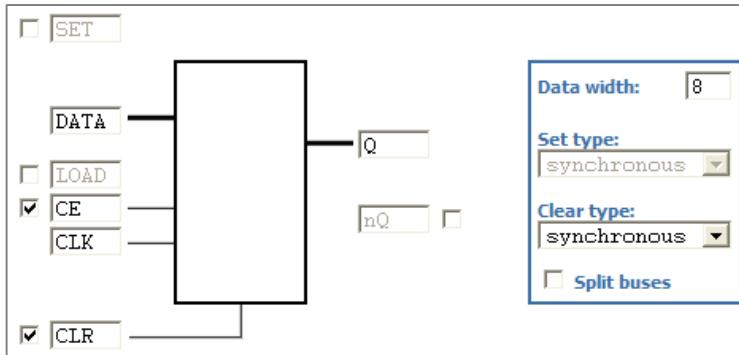
**end behavior;**



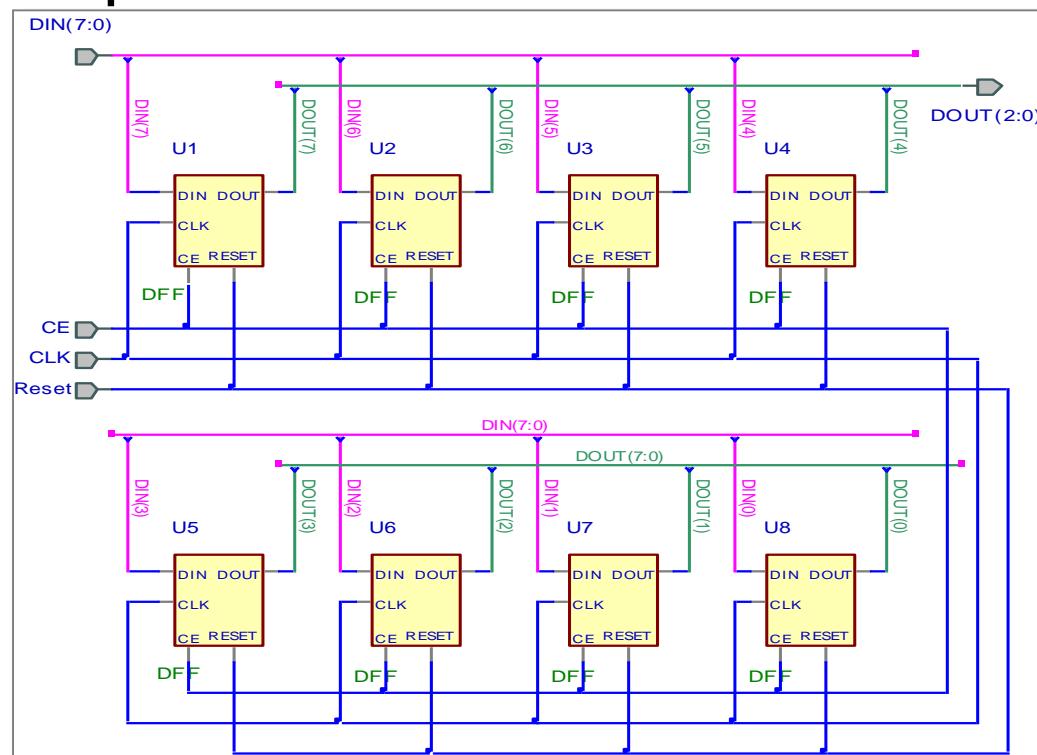
# Thanh ghi đa năng

1/2

- Thanh ghi dùng để lưu trữ thông tin bên trong CPU, có tốc độ cao nhất trong phân cấp bộ nhớ.
- Có thể chứa cả dữ liệu và địa chỉ.
- Chỉ gồm bit nhớ, không có phần cứng đặc thù kèm theo.



Dùng IP Core trong ActiveHDL  
để sinh mã, tạo thanh ghi



Thiết kế thanh ghi từ các D-FF bằng sơ đồ cấu trúc

# Thanh ghi đa năng

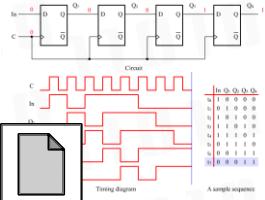
2/2

```
entity IPCore_GPR is
 port (
 CLR : in std_logic; -- xóa dữ liệu
 CLK : in std_logic; -- tín hiệu đồng hồ
 CE : in std_logic; -- cho phép h.động
 --! cổng vào dữ liệu, nội dung cần nhớ
 DATA : in std_logic_vector(7 downto
 0);
 --! cổng ra dữ liệu ~ nội dung đã nhớ
 Q : out std_logic_vector(7 downto 0)
);
end entity;
```

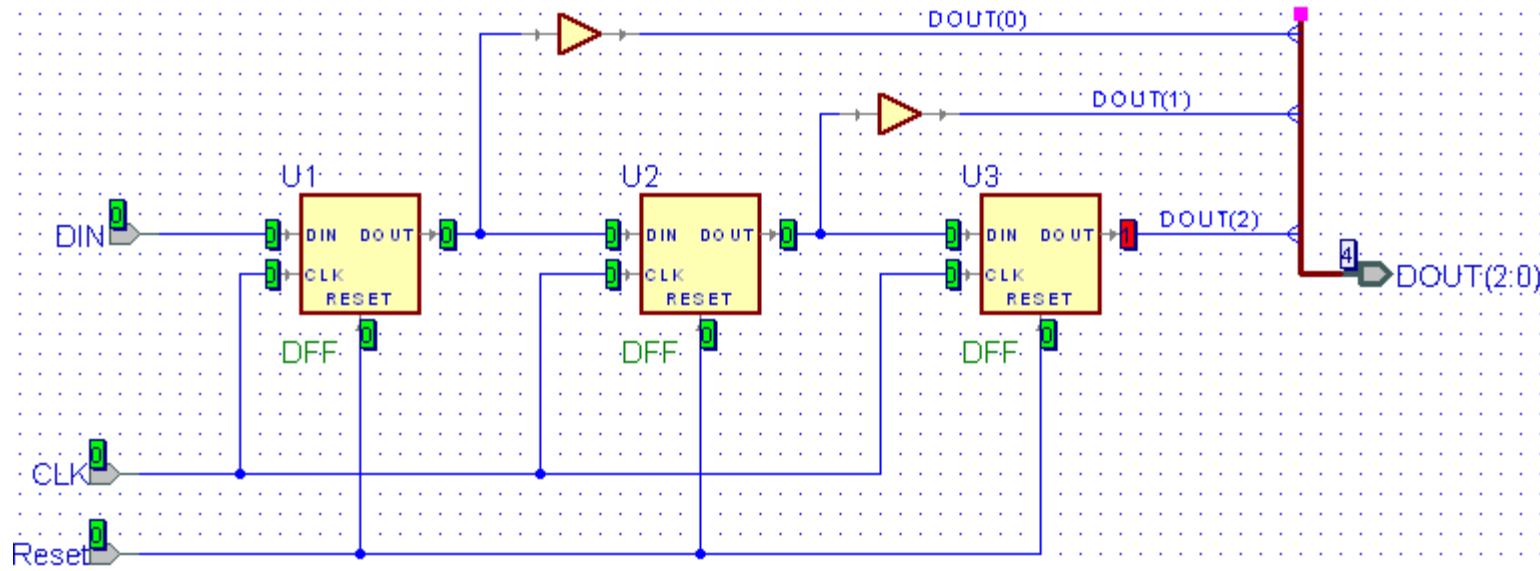
```
architecture GPR_arch of IPCore_GPR is
begin
 process (CLK)
 begin
 --! tại sườn dương của đồng hồ
 if rising_edge(CLK) then
 --! Nếu được phép hoạt động
 if CE = '1' then
 --! xóa dữ liệu đồng bộ
 if CLR = '1' then
 Q <= (others => '0');
 else
 --! hoặc chốt dữ liệu
 Q <= DATA;
 end if; -- CLR
 end if; -- CE
 end if; -- clk
 end process;
end GPR_arch;
```

# Thanh ghi dịch

- Có 2 loại:
  - Vào nối tiếp, ra song song (hình dưới);
  - Vào song song, ra nối tiếp;



- Ứng dụng:
  - Chuyển đổi dữ liệu nối tiếp  $\leftrightarrow$  song song. Ví dụ UART
  - Chia tần số.
  - Tạo tín hiệu tuần hoàn có dạng sóng phức tạp.



# Bài tập:

- Hãy thiết kế thanh ghi dịch **vào song song, ra nối tiếp** 4 bit.

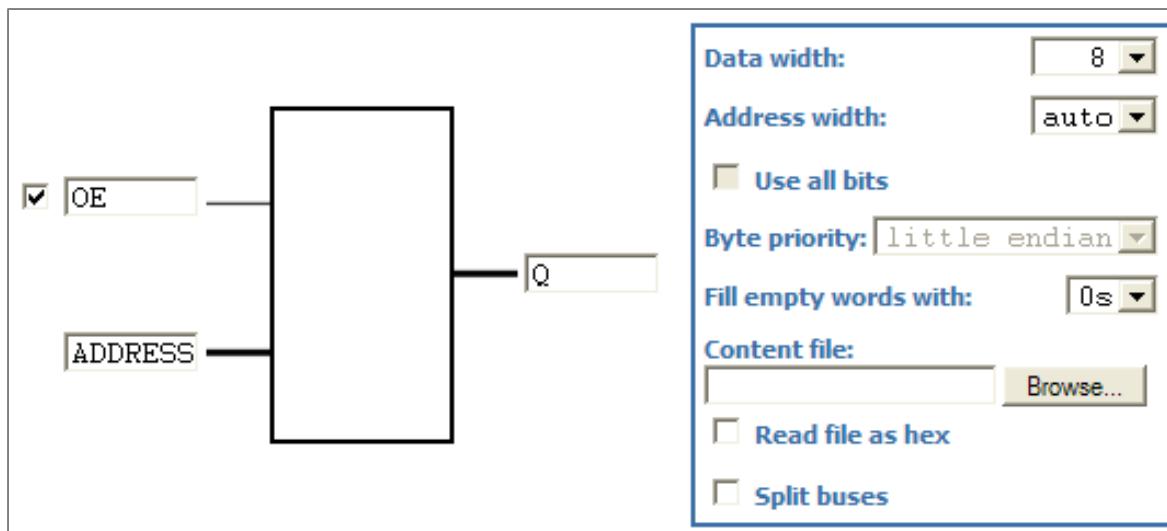
• **Gợi ý nguyên lý:**

**Thiết kế:**

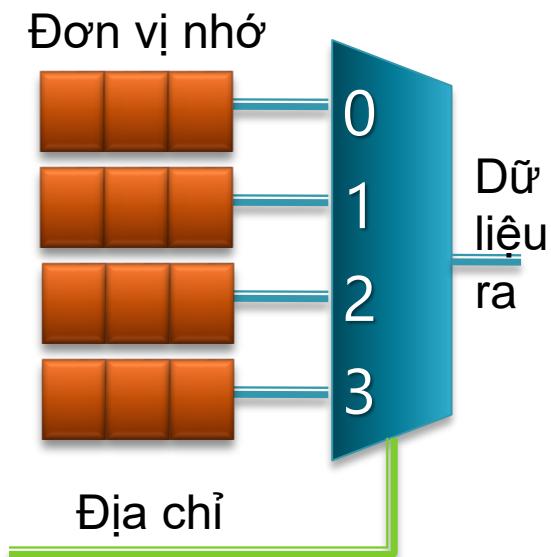
# ROM

1/2

- Thiết kế giống như một mảng các thanh ghi đa năng, nhưng mảng chỉ cho phép đọc dữ liệu.



Dùng IP Core trong ActiveHDL để sinh mã,  
tạo ROM trống, hoặc tự lấy thông tin từ 1 file hex



Thiết kế ROM từ mảng các thanh ghi  
kết hợp với bộ chọn kênh MUX

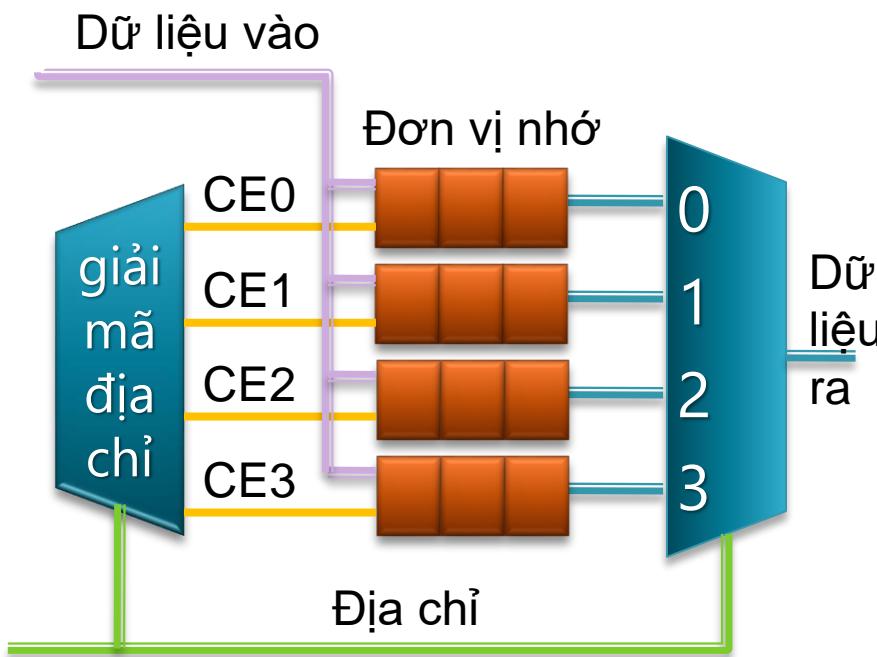
# ROM

2/2

```
entity IPCore_rom is
port (
 OE : in std_logic; -- cho phép dữ liệu
 -- đường địa chỉ
 ADDRESS : in std_logic_vector(2 downto 0);
 -- đường dữ liệu
 Q : out std_logic_vector(7 downto 0)
);
end entity;
```

```
architecture rom_arch of IPCore_rom is
begin
 process (ADDRESS, OE)
 variable ADDR_TEMP: std_logic_vector(1
downto 0);
 begin
 ADDR_TEMP := ADDRESS(1 downto 0);
 if (OE = '1') then
 if (ADDRESS(2) = '0') then
 case (ADDR_TEMP) is
 when "00" => Q <= "01100001";
 when "01" => Q <= "01100010";
 when "10" => Q <= "00110100";
 when others => Q <= "00000000";
 end case;
 else
 – mọi giá trị còn lại trong ROM = 0
 Q <= "00000000";
 end if;
 else – nếu OE = '0'
 Q <= "ZZZZZZZZ"; -- trả kháng cao
 end if;
 end process;
end architecture;
```

- Tương tự như ROM, nhưng cho phép ghi dữ liệu.
- Có nhiều kiểu thiết kế khác nhau: đơn/đa cỗng, chung/riêng đường địa chỉ/dữ liệu.



# RAM

3/3

```
entity RAM is
 generic (ADDR_WIDTH : integer := 8;
 DATA_WIDTH : integer := 8);
 port (
 WE : in STD_LOGIC; -- Tin hieu cho phep ghi vao RAM
 OE : in STD_LOGIC; -- Tin hieu cho phep doc RAM
 CLK : in std_logic;
 ADDR : in std_logic_vector(ADDR_WIDTH-1 downto 0);
 DIN : in std_logic_vector(DATA_WIDTH-1 downto 0);
 DOUT : out std_logic_vector(7 downto 0)
);
end entity;
```

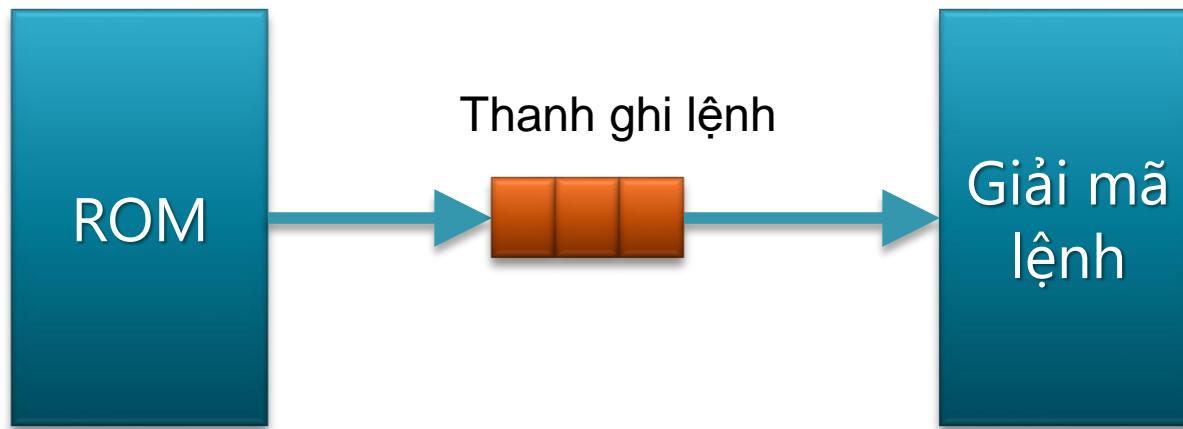
# RAM

3/3

```
architecture ram_arch of RAM is
 -- dung mang du lieu de chua gia tri
 type ram_mem_type is array (2**ADDR_WIDTH-1 downto 0) of
std_logic_vector(DATA_WIDTH-1 downto 0);
 signal ram_mem: ram_mem_type;
begin
 WRITE: process (CLK) -- chi ghi du lieu tai suon duong clk và WE=1
begin
 if rising_edge(CLK) then
 if (WE = '1') then
 ram_mem(CONV_INTEGER(ADDR)) <= DIN;
 end if;
 end if;
end process WRITE;
 READ: process (OE) -- chi doc du lieu khi OE = 1
begin
 if (OE = '1') then
 DOUT <= ram_mem(CONV_INTEGER(ADDR));
 else
 DOUT <= (others => 'Z');
 end if;
end process READ;
end architecture;
```

# Thanh ghi lệnh

- Thanh ghi lệnh IR, Instruction Register, chứa mã lệnh đang được giải mã.
- Về thiết kế, chỉ là một thanh ghi ghi/đọc bình thường
- Đặc trưng: đường dữ liệu vào được nối trực tiếp với bộ nhớ chương trình, và dữ liệu đầu ra đưa tới bộ giải mã lệnh.



# Thanh ghi bộ đếm c.trình

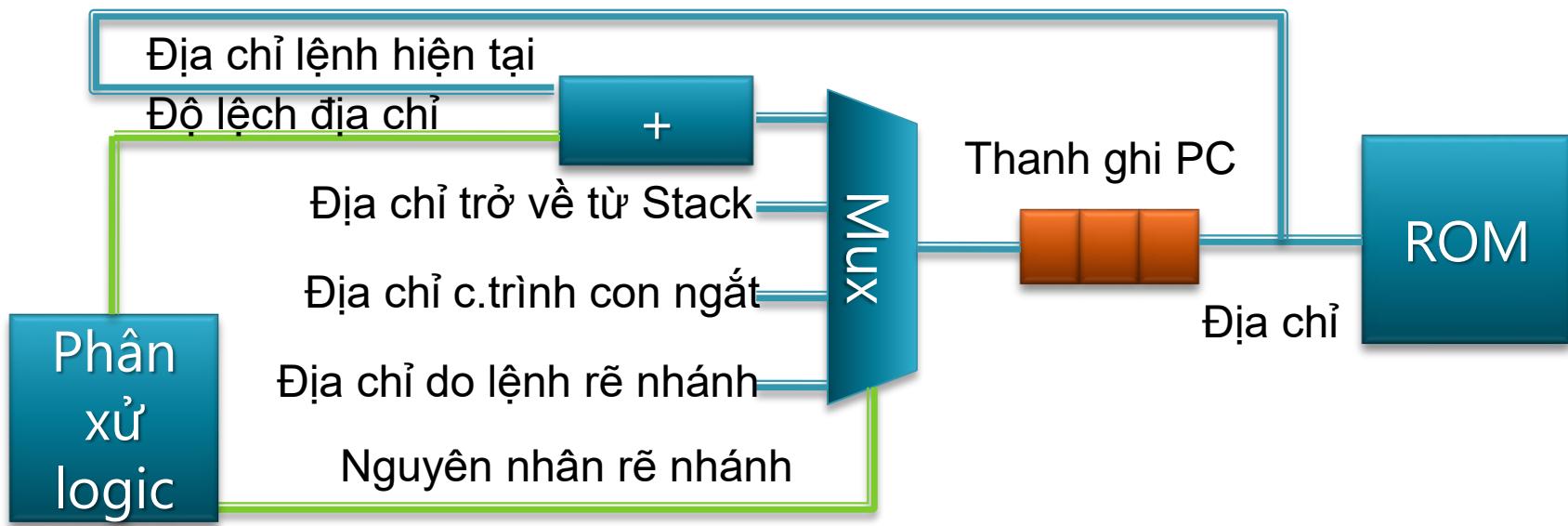
1/5

- Thanh ghi **bộ đếm** chương trình PC, Program Counter hoặc Instruction Pointer, chứa địa chỉ của lệnh sẽ được thực hiện.
- Phần lớn các lệnh trong chương trình là tuần tự  
→ địa chỉ của lệnh kế tiếp = địa chỉ của lệnh hiện tại + độ lệch địa chỉ giữa 2 lệnh đó. → *dữ liệu vào*.  
*Với bộ xử lý RISC, độ lệch này là không đổi.*  
→ Thiết kế PC sẽ **bao gồm một bộ cộng**.
- Với lệnh rẽ nhánh, PC phải ghi nhớ địa chỉ mới (địa chỉ nhảy tới) bất kỳ.* → *dữ liệu vào*.
- Ngắt xảy ra* → *rẽ nhánh tới 1 chương trình con phục vụ ngắt ở địa chỉ nào đó* → *dữ liệu vào*.

# Thanh ghi bộ đếm c.trình

2/5

- Rẽ nhánh do trở về từ chương trình con → địa chỉ trở về sẽ được đưa vào PC từ Stack → *dữ liệu vào*.
- Do có nhiều nguồn dữ liệu vào nên thiết kế PC phải **bao gồm một bộ ghép kênh**.



# Thanh ghi bộ đếm c.trình

3/5

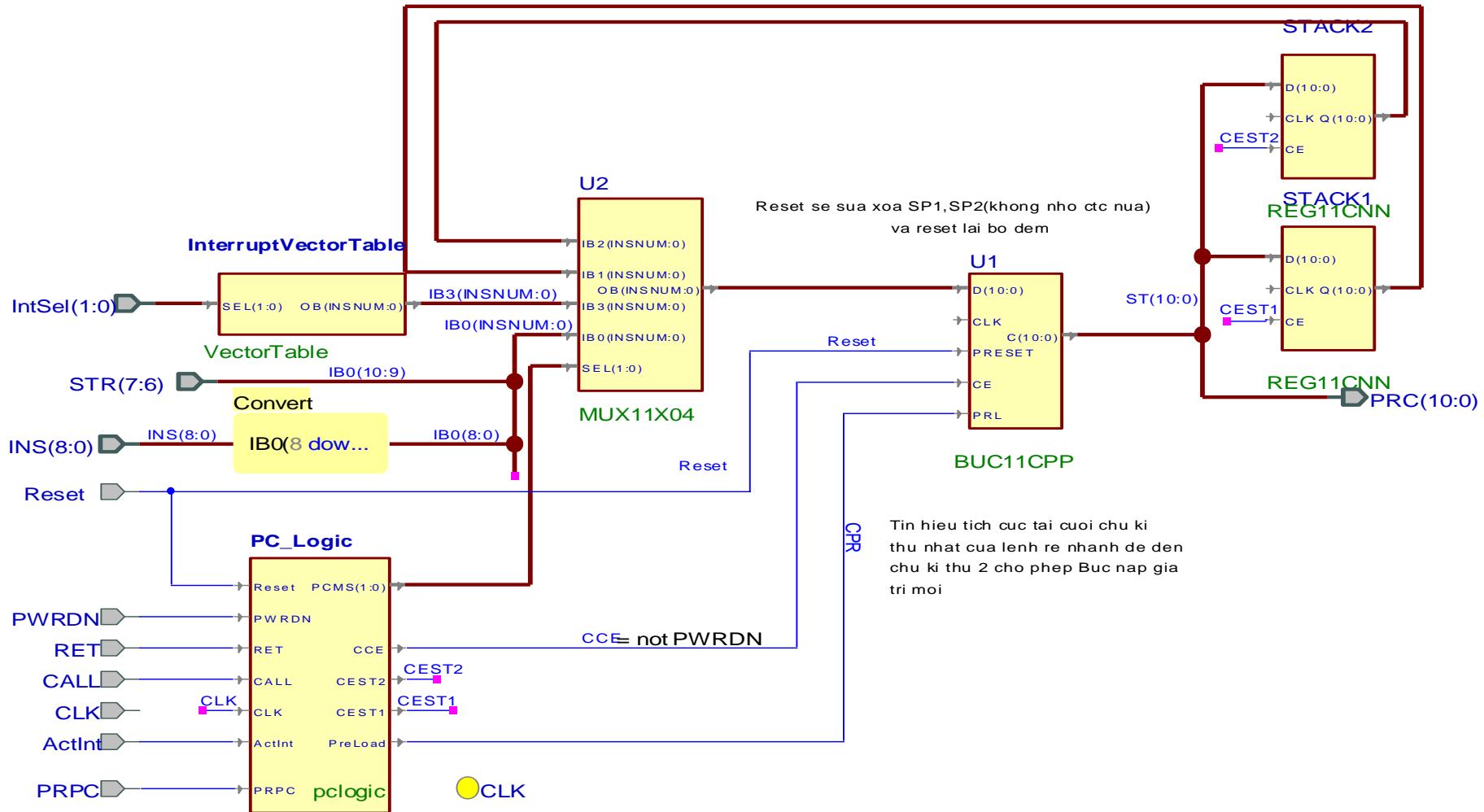
- Bộ phân xử nhằm phân tích:

- opcode của mã lệnh hiện tại (chứa trong thanh ghi IR)
- một số ngữ cảnh hiện tại như có ngắt không, kết quả giải mã lệnh hiện tại, etc.

→ Đề ra quyết định điều khiển các thành phần còn lại.

# Thanh ghi bộ đếm c. trình

4/5



Tin hieu tich cuc khi dung hoat dong, tiet kiem nang luong

# Thanh ghi bộ đếm c.trình

5/5

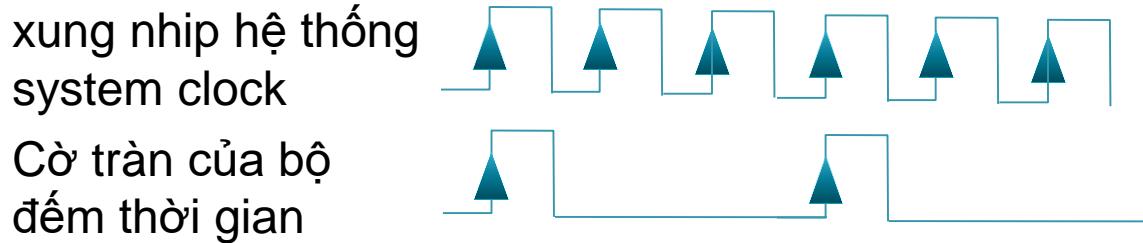
## • Lưu ý:

- Tín hiệu reset giá trị trong thanh ghi PC sẽ đưa địa chỉ khởi động hệ thống.

# Bộ đếm thời gian

1/2

- Bộ đếm thời gian, Time Counter, đếm số chu kỳ xung nhịp.
- Ứng dụng: tạo xung nhịp cố định với chu kỳ là số nguyên lần xung nhịp hệ thống, tạo hàm delay()

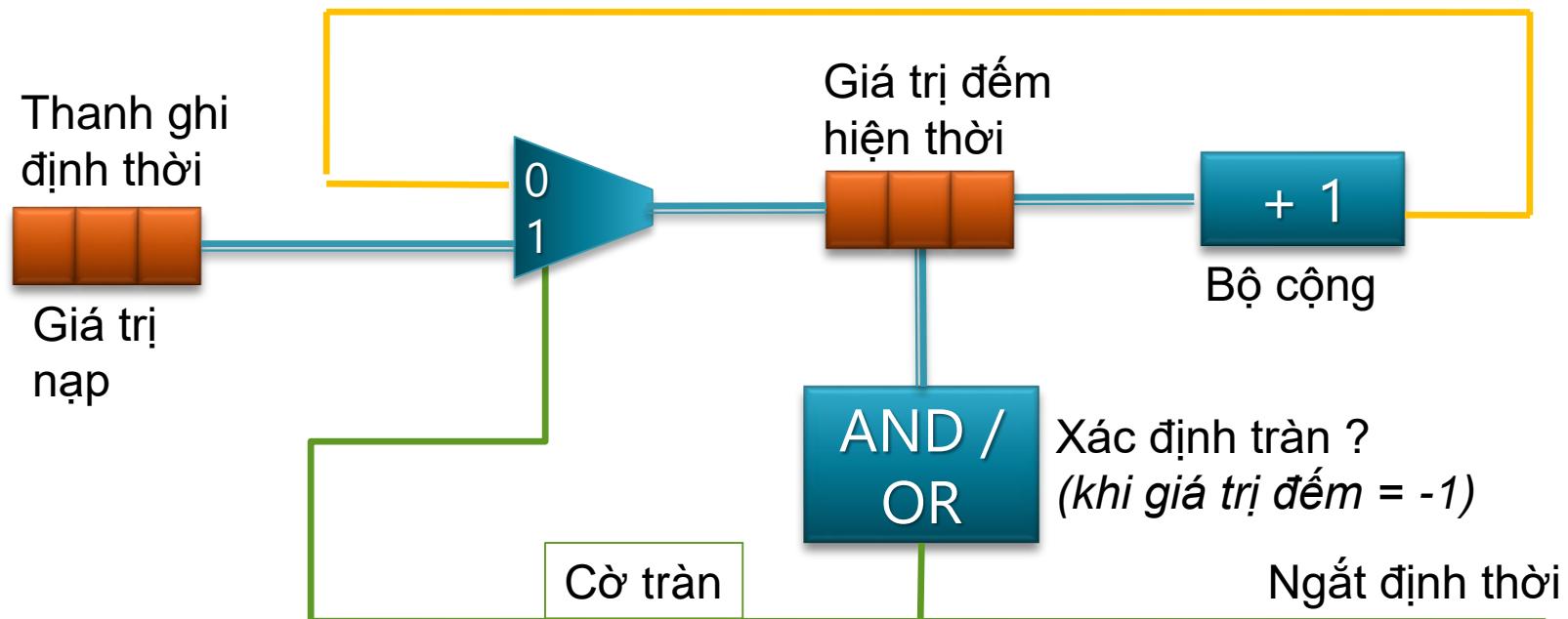


## Thiết kế:

- Một bộ cộng để đếm số chu kỳ xung nhịp hệ thống.
- Thanh ghi lưu trữ kết quả đếm hiện tại của bộ cộng.
- Thanh ghi lưu trữ giá trị khởi tạo của bộ cộng.
- Xác định thời điểm tràn bộ cộng (giá trị đếm = -1).

# Bộ đếm thời gian

2/2



## Lưu ý:

- Giá trị đếm (ví dụ 3 bit) sẽ có dạng 5, 6, 7<sup>tràn</sup>, 5, 6, 7<sup>tràn</sup>, 5,..
- Thanh ghi định thời là một thành phần của tập thanh ghi, nhận giá trị từ ALU.
- Ngắt định thời sẽ được chuyển tới bộ giải mã xử lý, để gọi chương trình con phục vụ ngắt tương ứng.

# Bài tập: Bộ đếm theo ngưỡng

- Hãy thiết kế bộ đếm thời gian nhưng thanh ghi định thời không chứa giá trị nạp, mà chứa giá trị ngưỡng, để khi giá trị đếm = ngưỡng thì báo ngắt định thời?
- **Gợi ý nguyên lý:**

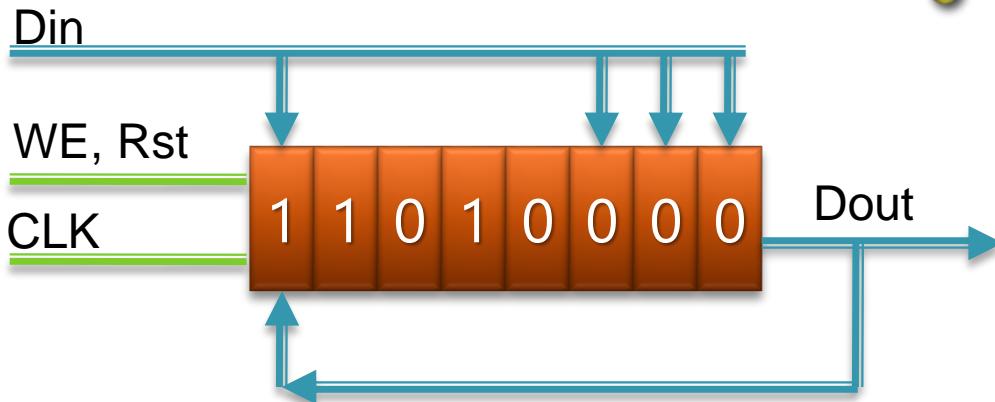
**Thiết kế:**

# Tạo tín hiệu có dạng sóng cho trước

- Tạo tín hiệu tuần hoàn cố định



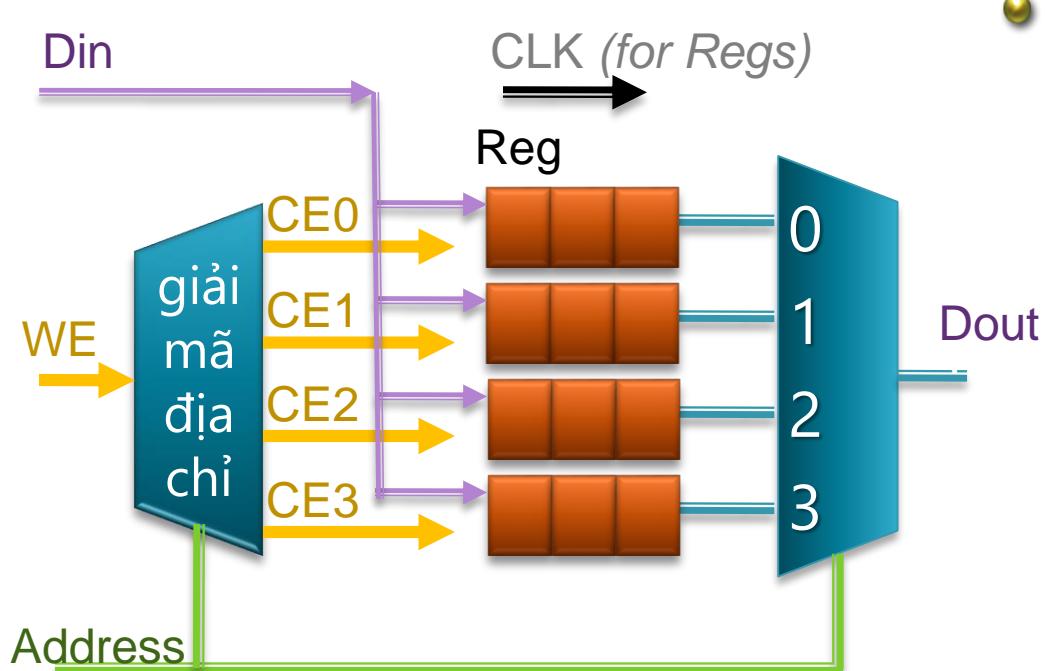
- Sử dụng thanh ghi dịch xoay vòng



- Các bước thực hiện
  - Tạo 8 bit nhớ bằng D Flip Flop với tín hiệu Din, WE, CLK, Reset.
  - Mux 2x1 cho mỗi DFF
  - T.kế Block Diagram.
  - Chạy giả lập

# Tự xây dựng RAM 4 byte

- Xây dựng RAM 4 byte từ 4 thanh ghi 1 byte



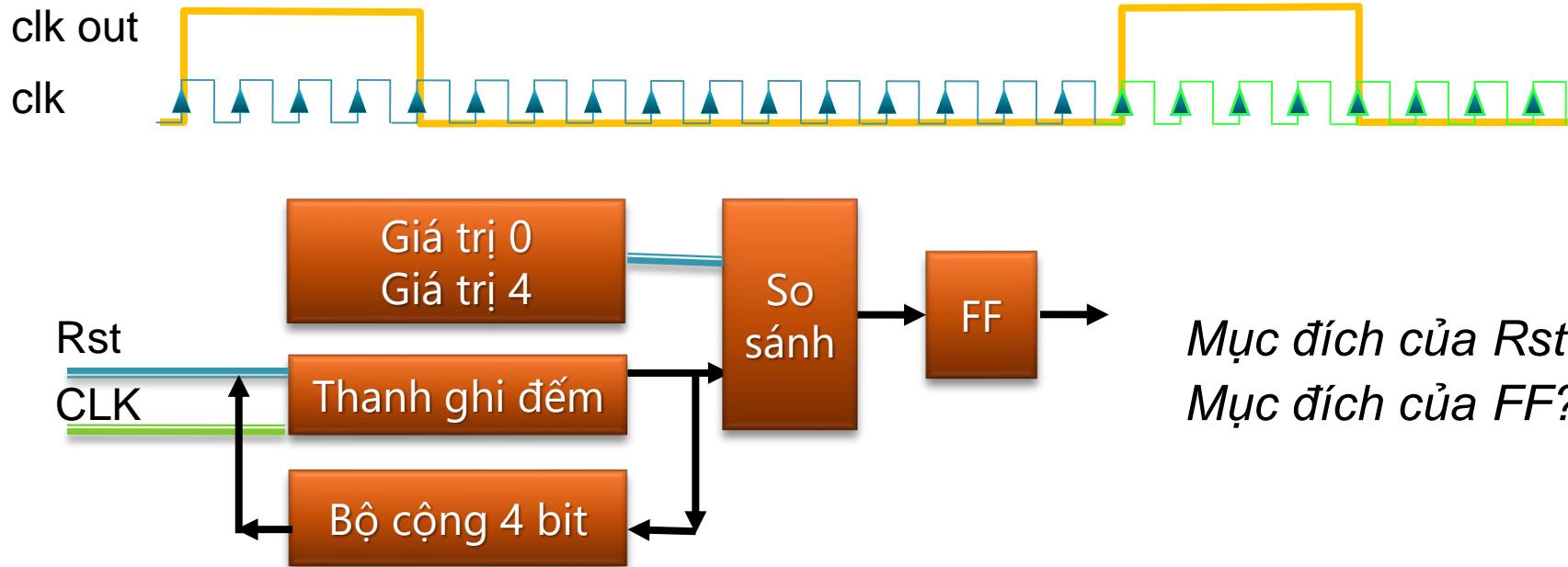
## Gợi ý:

- Các thanh ghi làm việc tại sườn dương của CLK, hoặc của CE (tùy chọn)
- Giải mã địa chỉ có thể dùng các mạch AND 3 đầu vào của Address với WE, hoặc dùng Demux.
- T.кế Block Diagram.
- Chạy giả lập

# Bộ chia tần số div16, 25%

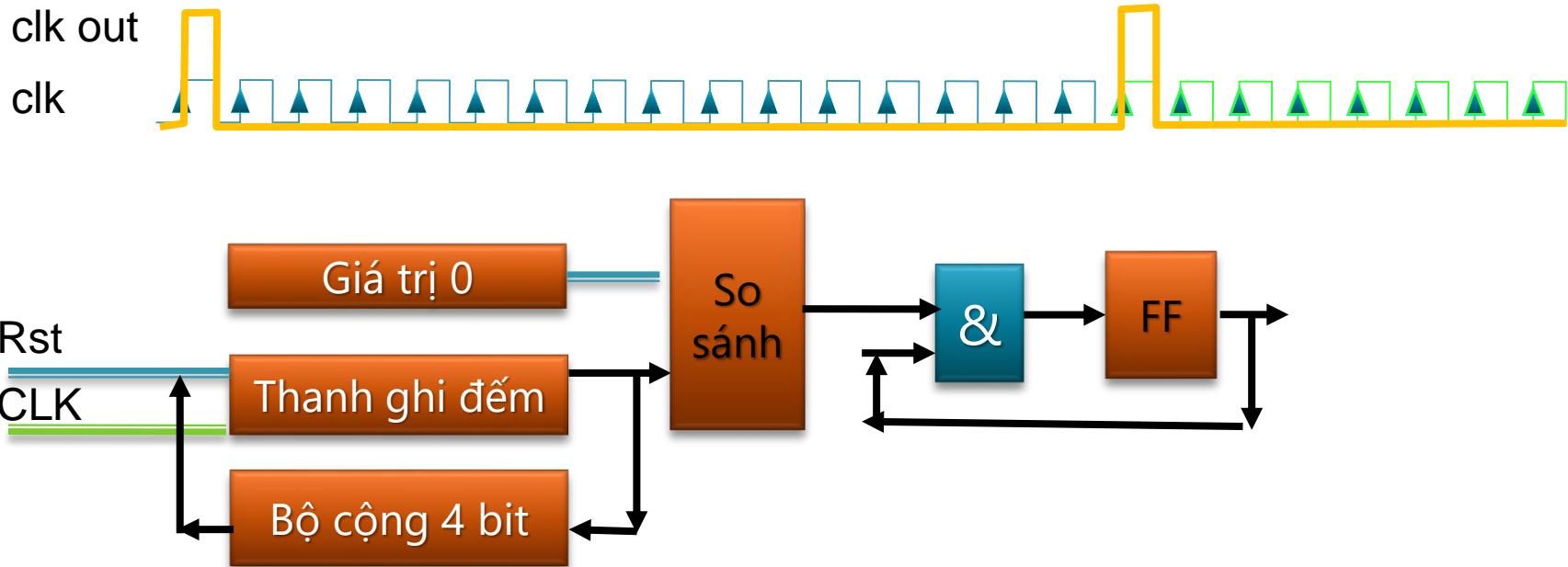
- Bộ chia tần nhận tín hiệu đồng hồ vào tốc độ cao, để tạo ra tín hiệu đồng hồ ra tốc độ chậm div16 với độ rộng xung 25%

- Gợi ý:
  - Bộ cộng 4 bit
  - Thanh ghi đếm 4 bit, Rst
  - Mạch so sánh với 0, với 4
  - Thanh ghi chốt 1 bit clk out



# Bộ chia tần số div16, 1/16

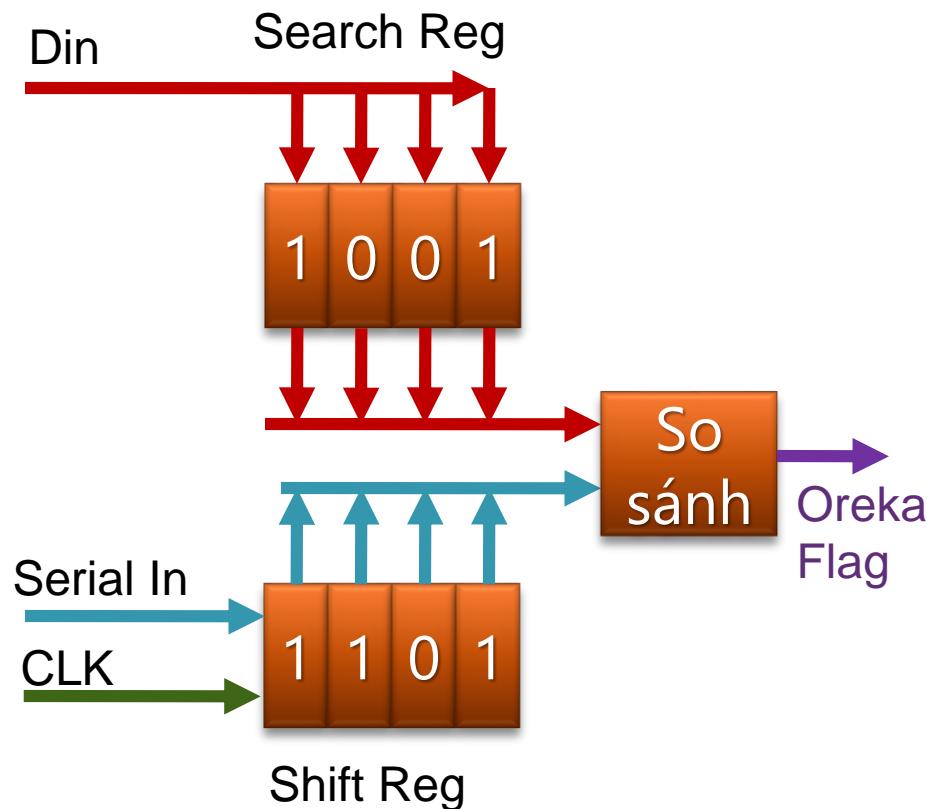
- Bộ chia tần nhận tín hiệu đồng hồ vào tốc độ cao, để tạo ra tín hiệu đồng hồ ra tốc độ chậm div16 với độ rộng xung 6.25%
- Gợi ý:
  - Bộ cộng 4 bit
  - Thanh ghi đếm 4 bit, Rst
  - Mạch so sánh với 0
  - Thanh ghi chốt 1 bit clk out



# Phát hiện chuỗi bit đầu vào

1/2

- Hãy xây dựng mạch để phát hiện chuỗi bit đầu vào bằng “1001” bằng 3 cách:
- Ứng dụng: xây dựng phần cứng xử lý giao thức.
- Cách 1: Thanh ghi dịch
  - + Thanh ghi 4 bit x 2
  - + Lập trình được

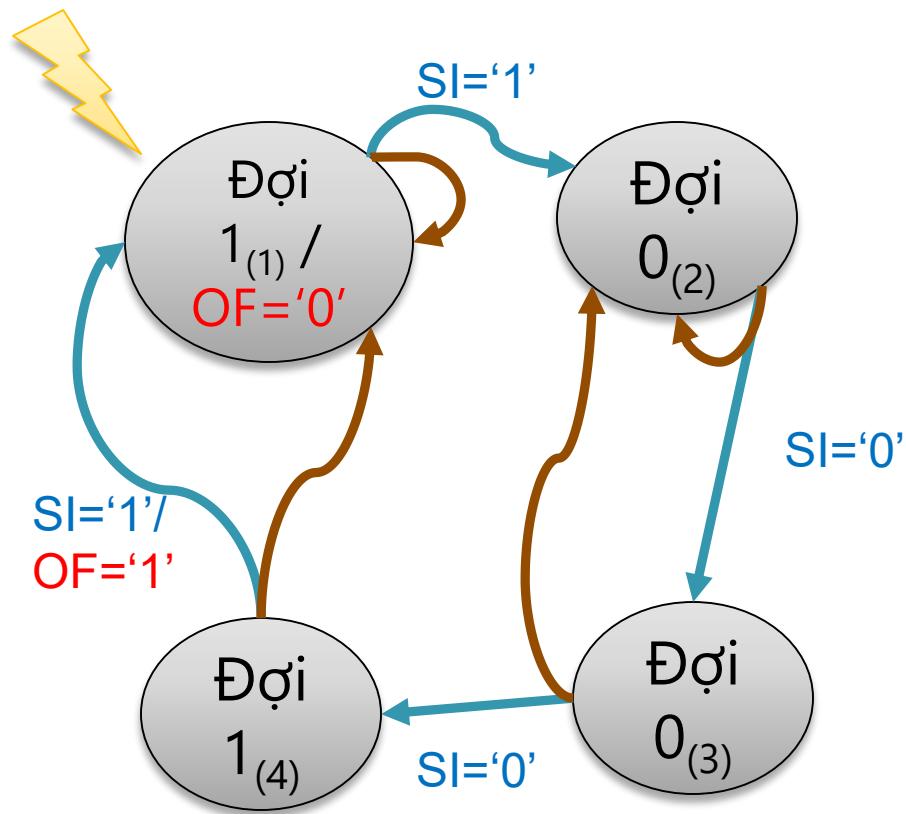


# Phát hiện chuỗi bit đầu vào

2/2

## Cách 2: FSM

- + 2 bit nhớ
- + Không lập trình được



## Cách 3: VHDL

- + Đặt bên trong process
- + Doxygen Comment

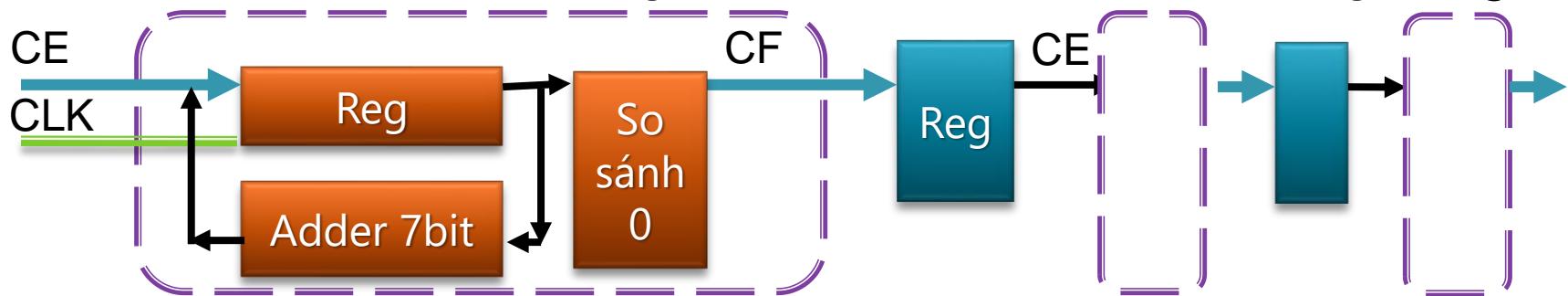
[http://www.youtube.com/watch?v=Z\\_gsu84KLLo](http://www.youtube.com/watch?v=Z_gsu84KLLo)

```
--! @brief Architure definition of the MUX
--! @details More details about this mux
architecture myarch of SyntaxMod is
begin
process (CLK)
begin
if rising_edge(CLK) then
ShiftReg <= Din & ShiftReg(3 downto 1);
if
end if;
end process;
end architecture;
```



# Bộ đếm 2 triệu

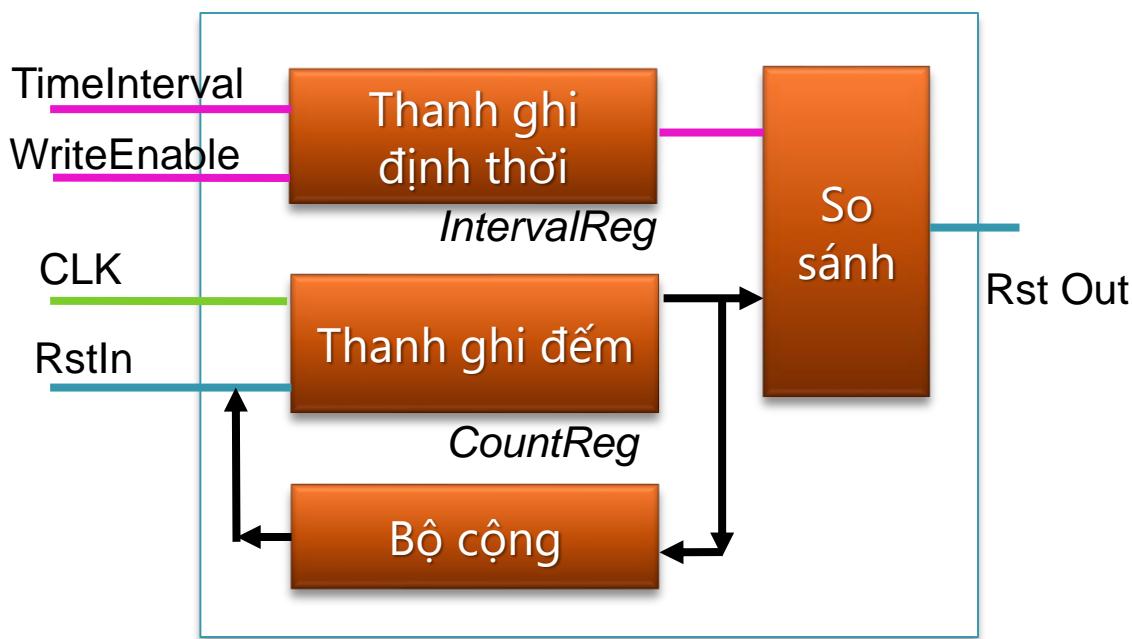
- Xây dựng bộ đếm (+1) 2 triệu đơn vị
  - Bộ cộng Ripper với 21 bit.
  - nhưng tách thành 3 bộ cộng 7 bit, có CE(cho phép +1)
  - bởi 2 lát cắt, thành 3 công đoạn.
  - Bộ đếm 7 bit bằng VHDL, bộ đếm 21 bit bằng Diagram



- Yêu cầu giả lập
  - Cho CF trễ 100ns:  $CF = CF'$  after 100 ns; ( $\sim 10\text{MHz}$ )
  - Chạy giả lập với CLK bằng 100MHz (10ns)
  - Chạy giả lập với CLK bằng 10MHz (100ns)

# Thiết kế bộ watchdog

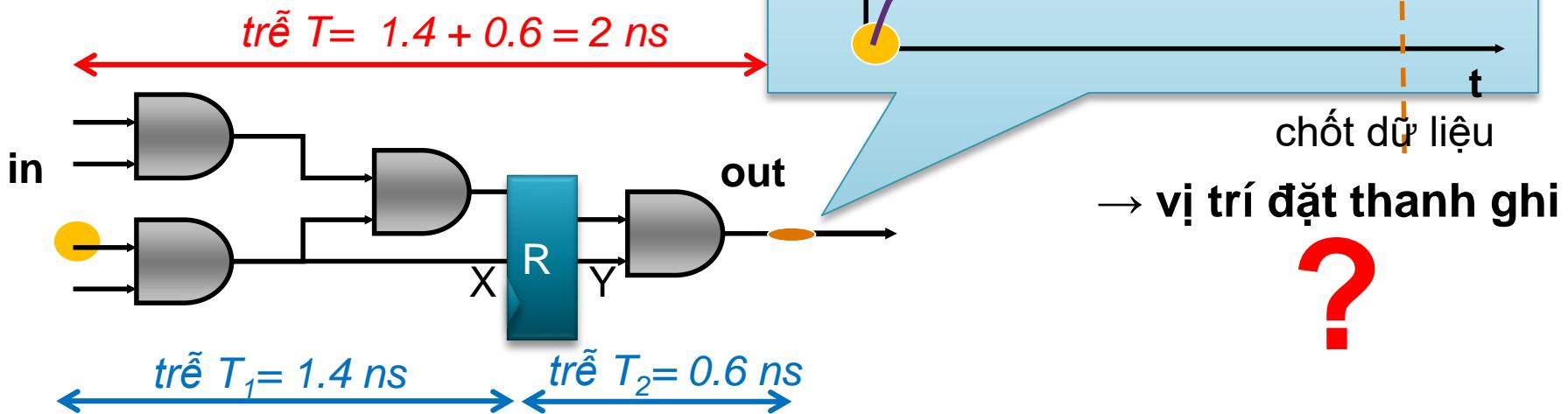
- Mạch watchdog là mạch phụ trợ, sẽ gửi tín hiệu reset (RstOut) tới mạch chính, nếu mạch chính không gửi reset tới mạch watchdog (Rst\_In) sau khoảng thời gian nào đó.



- Các bước thực hiện
  - Bộ cộng
  - Thanh ghi

# Pipeline: nguyên lý cơ bản

Khi chưa có thanh ghi R, độ trễ T của mạch là tổng độ trễ của mọi thành phần trong mạch



Sau khi có thanh ghi, mạch chia thành 2 nửa. Độ trễ  $T_R$  của mạch là  $\max(T_1, T_2)$ . Vậy hiệu năng xử lý đã tăng  $2 / 1.4 = 1.43$  lần

Pipeline: thêm thanh ghi để ngăn quá trình quá độ của lệnh mới, với giá trị đã ổn định của lệnh cũ.

# Pipeline

2/3

- Công đoạn

Lệnh 1

Stage 1

Stage 2

Stage 3

Stage 4

Stage 5

Lệnh 2

Stage 1

Stage 2

Stage 3

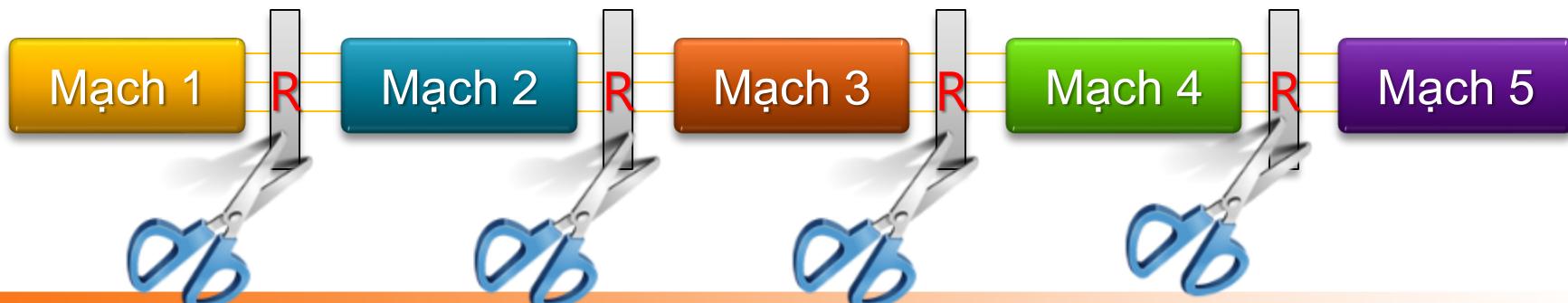
Stage 4

Stage 5

- Mạch số trước khi có pipeline

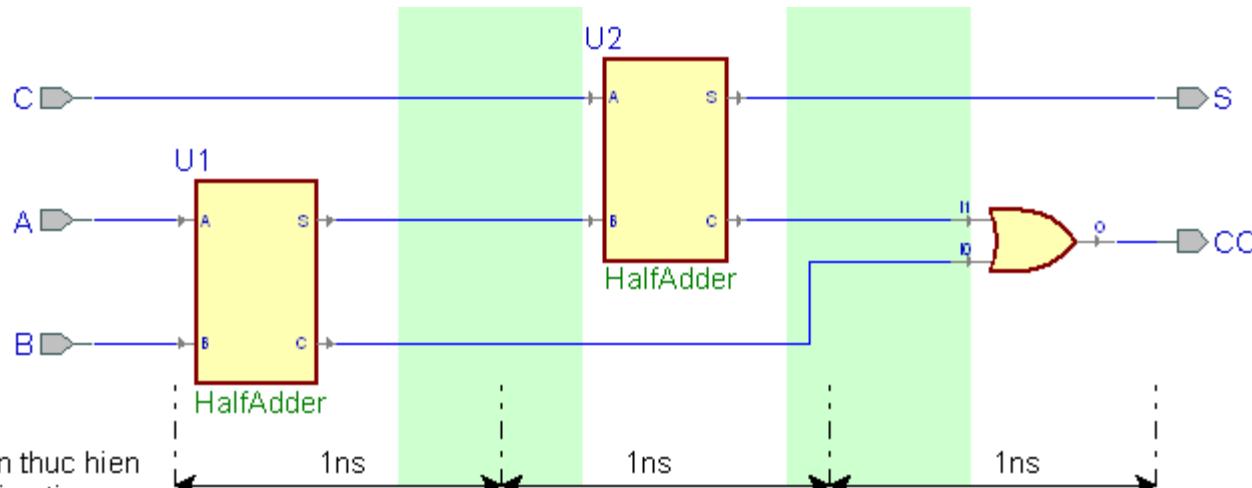
Mạch 1 nối tiếp mạch 2, ... nối tiếp mạch 5

- Mạch số khi đã tiến hành pipeline hóa

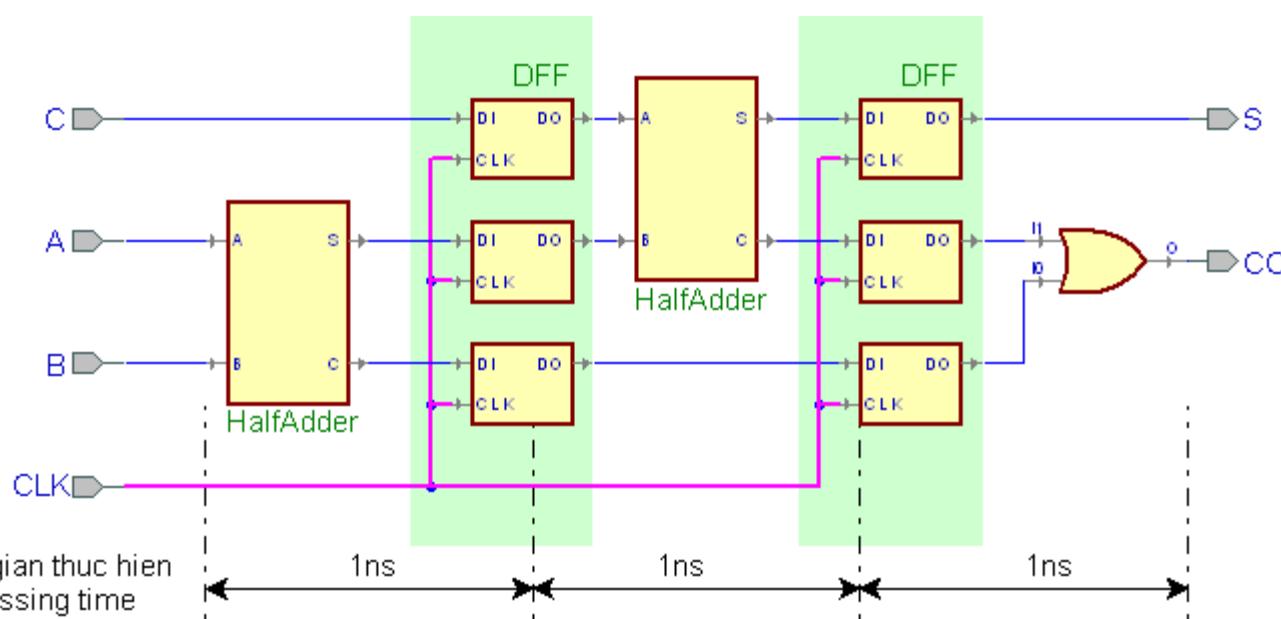


# Pipeline bộ cộng

3/3



Bộ cộng



Bộ cộng  
sau pipeline  
với  
3 công đoạn



Phần VI:

# Thiết kế bộ vi xử lý

- Thiết kế mức đỉnh
- Thiết kế mức thành phần
- Tổng hợp và mô phỏng
- Đặt thiết kế lên FPGA



# Về bộ xử lý SLC1657

- Bộ xử lý 8 bit, mã nguồn mở
- Phiên bản nâng cấp pT-BDC 8x có thiết kế dạng Block Diagram trực quan.



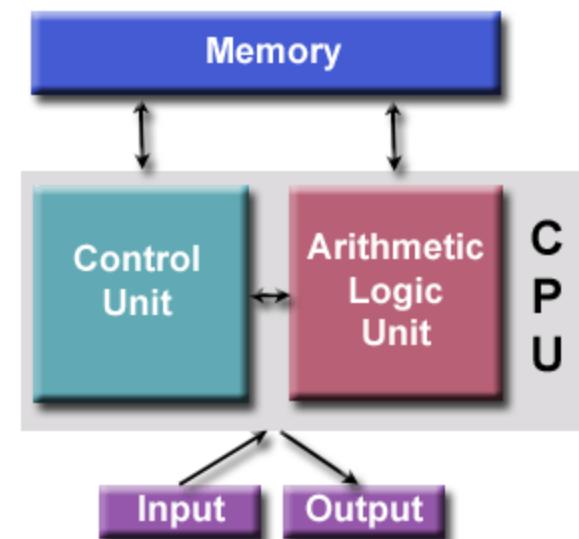
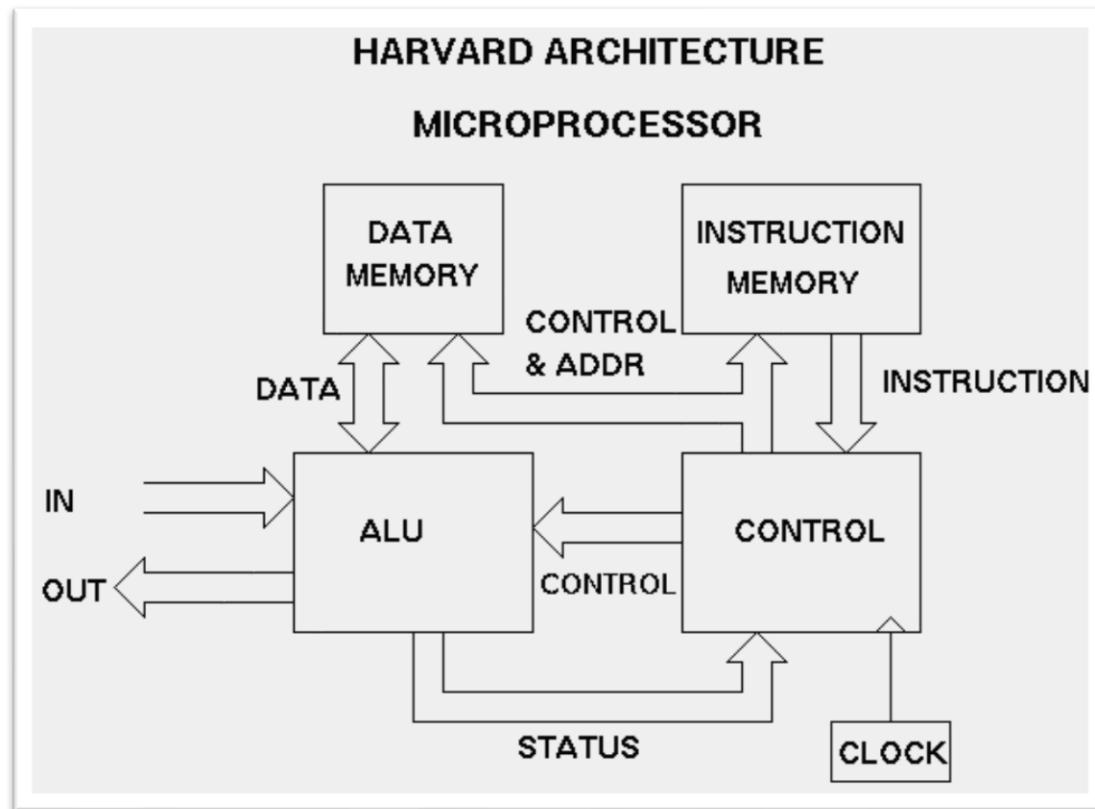
# Đặc trưng

- Kiến trúc Harvard.
  - Độ rộng bus dữ liệu, địa chỉ: 8 bit.
  - Độ rộng bus lệnh: 13 bit.
- Lệnh chứa 1 địa chỉ toán hạng.
- Có chế độ giảm tiêu thụ năng lượng
- Giao tiếp RS232, ngắt
- Hệ thống pipeline 4 công đoạn
- Thiết kế chi tiết mức phần tử logic cơ bản

# Kiến trúc Harvard

1/2

- Là kiến trúc máy tính mà trong đó phân biệt rõ ràng bộ nhớ dữ liệu và bộ nhớ chương trình.

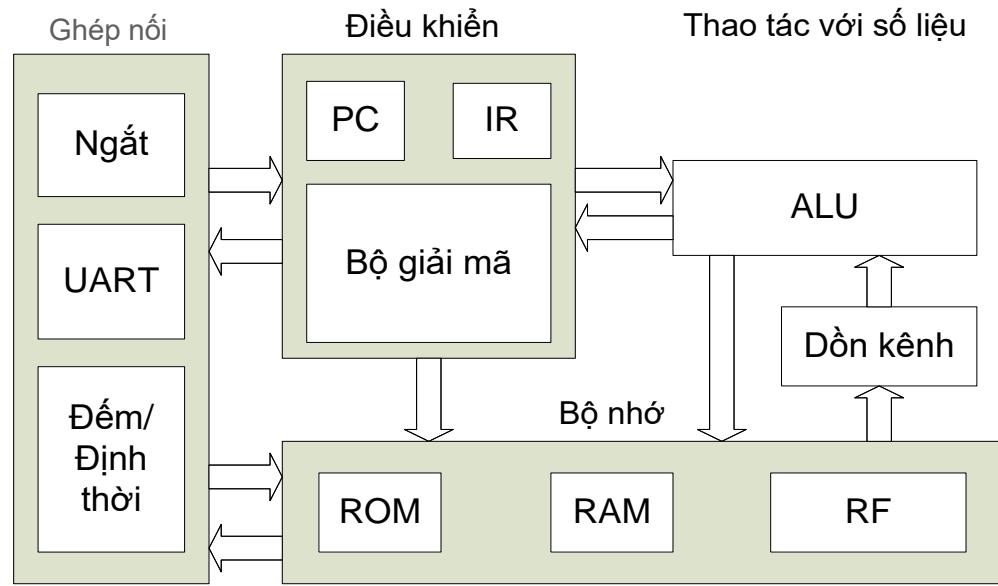
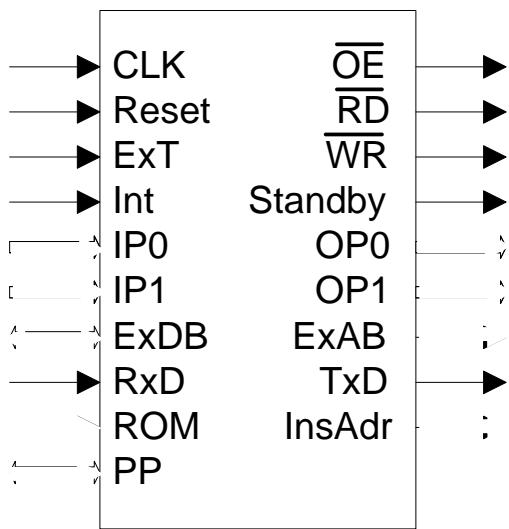


# Kiến trúc Harvard

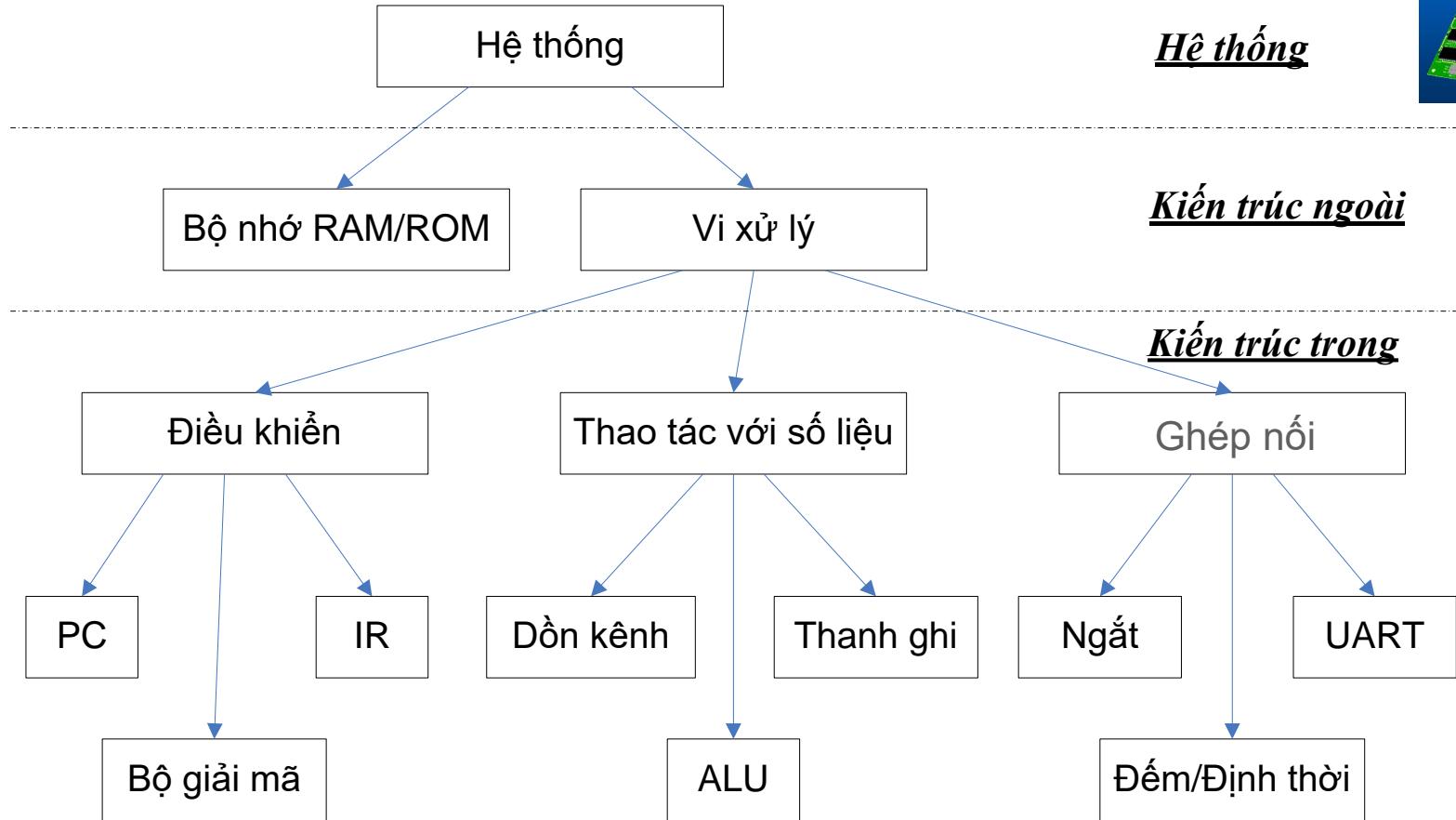
2/2

- Ngày nay, CPU tốc độ cao ngày nay thường kết hợp hai kiến trúc Harvard và Von Neumann.
- Bộ nhớ cache trên chip được phân thành cache chương trình và cache dữ liệu.  
→ Kiến trúc Harvard được dùng khi CPU truy cập vào cache.
- Trong trường hợp không có cache, dữ liệu được lấy từ bộ nhớ chính, mà bộ nhớ chính không được chia thành vùng nhớ chương trình và vùng nhớ dữ liệu.  
→ Kiến trúc Von Neumann dùng ở mức truy cập bộ nhớ chính.

# Tổng quan kiến trúc



# Tổng quan kiến trúc (2)



# Tập lệnh

- Tập thao tác gồm 27 phép thao tác, mã hoá bởi 5 bit, điều khiển ALU.
- Tập lệnh gồm 50 lệnh với 4 nhóm: lệnh số học, lệnh logic, lệnh chuyển dữ liệu, lệnh điều khiển .
- Sử dụng 4 phương pháp địa chỉ toán hạng: tức thì, địa chỉ trực tiếp, địa chỉ thanh ghi, địa chỉ gián tiếp qua thanh ghi.
- Thiết kế tập lệnh có vai trò quyết định tới hiệu quả xử lý.

# Tập lệnh (2)

| Opcode                      | Tên lệnh | ALU  | ALF   | Số chu kì |
|-----------------------------|----------|------|-------|-----------|
| 00000 0000 0000             | NOP      | PASN | 00000 | 1         |
| 00000 0000 0001             | PWRDN    | PASN | 00000 | 1         |
| 00000 0000 0011             | RET      | PASN | 00000 | 2         |
| 00011 1TDR RRRR             | CMP      | CMP  | 00111 | 1         |
| 01100 0PDR RRRR             | CAZP     | CAZP | 11000 | 1         |
| 10000 VVVR RRRR             | CLRB     | CLRB | 11000 | 1         |
| 10001 VVV <sub>R</sub> RRRR | SETB     | SETB | 11001 | 1         |
| 1011V VVVV VVVV             | JC       | PASN | 00000 | 1(2)      |
| 11000 VVVV VVVV             | MOVI     | PASN | 00000 | 1         |
| 11011 VVVV VVVV             | XORI     | XOR  | 01111 | 1         |
| 1110V VVVV VVVV             | CALL     | PASN | 00000 | 2         |

# Giải mã lệnh

1/8

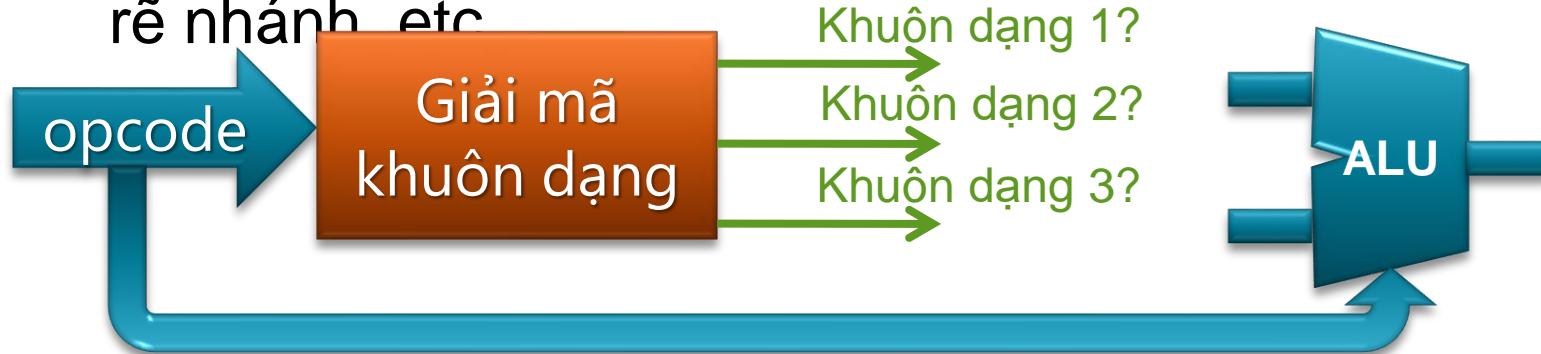
- Bộ giải mã lệnh nhằm để biến đổi các opcode đầu vào thành các tín hiệu điều khiển **chọn toán tử** và **chọn toán hạng** đầu ra.
- Tín hiệu điều khiển **chọn toán tử** thường là chân Select của các bộ ALU Mux, để cho biết phép toán cần thực hiện là gì.
- Tín hiệu điều khiển **chọn toán hạng** là chân Chip Enable của các thanh ghi và bộ nhớ → để cho biết kết quả phép toán sẽ được lưu trữ vào đâu.



# Giải mã lệnh

2/8

- Opcode của tập lệnh thường bao gồm:
  - các sub opcode chọn toán tử của ALU.
  - và các opcode cho các lệnh không dùng tới ALU như rẽ nhánh, etc



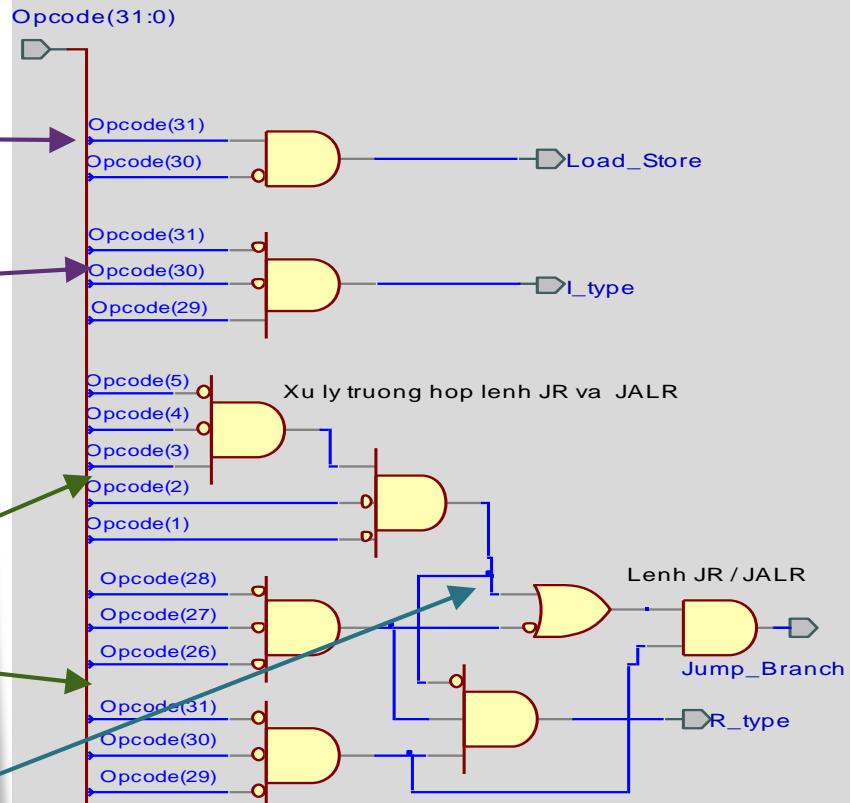
- Nhiều lệnh trong tập lệnh có thể chỉ tương ứng với một toán tử duy nhất trong ALU. Ví dụ, phép  $rd = rs + 10$  và phép  $rd = rs + rt$  với ALU là như nhau. → phải biến đổi opcode của lệnh một chút, trước khi gửi tới chân chọn toán tử của ALU

# Giải mã lệnh

3/8

| 31                                | 26     | 25    | 21     | 20     | 16                  | 15      | 11        | 10 | 6 | 5 | 0 |        |
|-----------------------------------|--------|-------|--------|--------|---------------------|---------|-----------|----|---|---|---|--------|
| opcode                            | rs     | rt    | rd     |        | shamt               |         | funct     |    |   |   |   | R-type |
| opcode                            | rs     | rt    |        |        |                     |         | immediate |    |   |   |   | I-type |
| opcode                            |        |       |        | target |                     |         |           |    |   |   |   | J-type |
| Load and Store Instructions       |        |       |        |        |                     |         |           |    |   |   |   |        |
| 100000                            | base   | dest  |        |        | signed offset       |         |           |    |   |   |   |        |
| 100001                            | base   | dest  |        |        | signed offset       |         |           |    |   |   |   |        |
| 100011                            | base   | dest  |        |        | signed offset       |         |           |    |   |   |   |        |
| 100100                            | base   | dest  |        |        | signed offset       |         |           |    |   |   |   |        |
| 100101                            | base   | dest  |        |        | signed offset       |         |           |    |   |   |   |        |
| 100110                            | base   | dest  |        |        | signed offset       |         |           |    |   |   |   |        |
| 100111                            | base   | dest  |        |        | signed offset       |         |           |    |   |   |   |        |
| I-Type Computational Instructions |        |       |        |        |                     |         |           |    |   |   |   |        |
| 000001                            | src    | dest  |        |        | signed immediate    |         |           |    |   |   |   |        |
| 000010                            | src    | dest  |        |        | signed immediate    |         |           |    |   |   |   |        |
| 000011                            | src    | dest  |        |        | signed immediate    |         |           |    |   |   |   |        |
| 001000                            | src    | dest  |        |        | zero-ext. immediate |         |           |    |   |   |   |        |
| 001001                            | src    | dest  |        |        | zero-ext. immediate |         |           |    |   |   |   |        |
| 001100                            | src    | dest  |        |        | zero-ext. immediate |         |           |    |   |   |   |        |
| 001110                            | src    | dest  |        |        | zero-ext. immediate |         |           |    |   |   |   |        |
| 001111                            | 00000  | dest  |        |        | zero-ext. immediate |         |           |    |   |   |   |        |
| R-Type Computational Instructions |        |       |        |        |                     |         |           |    |   |   |   |        |
| 000000                            | 00000  | src   | dest   | shamt  | 0000000             |         |           |    |   |   |   |        |
| 000000                            | 00000  | src   | dest   | shamt  | 000010              |         |           |    |   |   |   |        |
| 000000                            | 00000  | src   | dest   | shamt  | 000011              |         |           |    |   |   |   |        |
| 000000                            | rshamt | src   | dest   |        | 00000               | 0000100 |           |    |   |   |   |        |
| 000000                            | rshamt | src   | dest   |        | 00000               | 000110  |           |    |   |   |   |        |
| 000000                            | rshamt | src   | dest   |        | 00000               | 000111  |           |    |   |   |   |        |
| 000000                            | src1   | src2  | dest   |        | 00000               | 1000001 |           |    |   |   |   |        |
| 000000                            | src1   | src2  | dest   |        | 00000               | 100011  |           |    |   |   |   |        |
| 000000                            | src1   | src2  | dest   |        | 00000               | 100100  |           |    |   |   |   |        |
| 000000                            | src1   | src2  | dest   |        | 00000               | 100101  |           |    |   |   |   |        |
| 000000                            | src1   | src2  | dest   |        | 00000               | 100110  |           |    |   |   |   |        |
| 000000                            | src1   | src2  | dest   |        | 00000               | 100111  |           |    |   |   |   |        |
| 000000                            | src1   | src2  | dest   |        | 00000               | 101010  |           |    |   |   |   |        |
| 000000                            | src1   | src2  | dest   |        | 00000               | 101011  |           |    |   |   |   |        |
| Jump and Branch Instructions      |        |       |        |        |                     |         |           |    |   |   |   |        |
| 000010                            |        |       | target |        |                     |         |           |    |   |   |   |        |
| 000011                            |        |       | target |        |                     |         |           |    |   |   |   |        |
| 000000                            | src    | 00000 | 00000  | 00000  | 001000              |         |           |    |   |   |   |        |
| 000000                            | src    | 00000 | dest   | 00000  | 001001              |         |           |    |   |   |   |        |
| 000100                            | src1   | src2  |        |        | signed offset       |         |           |    |   |   |   |        |
| 000101                            | src1   | src2  |        |        | signed offset       |         |           |    |   |   |   |        |
| 000110                            | src    | 00000 |        |        | signed offset       |         |           |    |   |   |   |        |
| 000111                            | src    | 00000 |        |        | signed offset       |         |           |    |   |   |   |        |
| 000001                            | src    | 00000 |        |        | signed offset       |         |           |    |   |   |   |        |
| 000001                            | src    | 00001 |        |        | signed offset       |         |           |    |   |   |   |        |

Giải mã khuôn dạng lệnh MIPS



# Giải mã lệnh

4/8

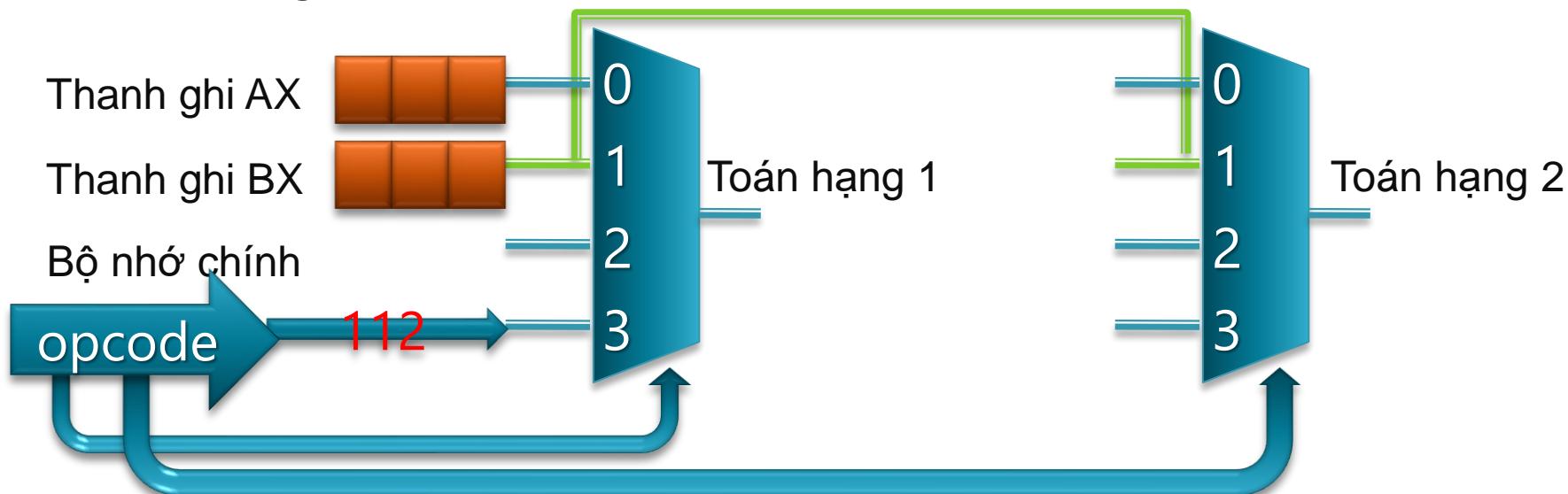
- Trong một khuôn dạng, ý nghĩa và kích thước của các nhóm bit là giống nhau.
  - chung cách giải mã
  - dùng chung module giải mã.
- Các lệnh cùng khuôn dạng thì thường chung đặc tính điều khiển như thay đổi cờ, rẽ nhánh, etc.



# Giải mã lệnh

5/8

- Trong mã lệnh, chuỗi bit chứa thông tin về toán hạng nguồn sẽ được gửi tới các bộ mux để chọn toán hạng đầu vào cho ALU.



- Opcode cũng có thể là chứa luôn toán hạng theo phương pháp địa chỉ tức thì. Ví dụ  $c = a + 112$ .
- Với toán hạng ngầm định thì cần tiến hành giải mã theo qui tắc ngầm định đã được đặc tả.

# Giải mã lệnh

6/8

- Bộ giải mã lệnh còn thực hiện các công việc:
  - Điều khiển các quá trình song song bên trong bộ xử lý.
  - Điều khiển các ngắt (Vì ngắt cũng chỉ là một lệnh rẽ nhánh đặc biệt)
  - Điều khiển rẽ nhánh.

# Giải mã lệnh rẽ nhánh

7/8

- Lệnh rẽ nhánh cần được xử lý trong 2 chu kỳ xung nhịp.
- Các vấn đề phát sinh:
  1. Lệnh rẽ nhánh không sử dụng tới ALU. Vậy ALU làm gì trong lúc lệnh rẽ nhánh thực hiện?
  2. Trong lúc rẽ nhánh, các lệnh đang tồn tại dở trong ống lệnh pipeline sẽ phải xử trí thế nào?
  3. Ống lệnh có 5 công đoạn. Vậy khi lệnh đầu tiên sau rẽ nhánh mới chỉ nằm ở công đoạn đầu tiên, các công đoạn sau sẽ làm gì?

# Giải mã lệnh rẽ nhánh

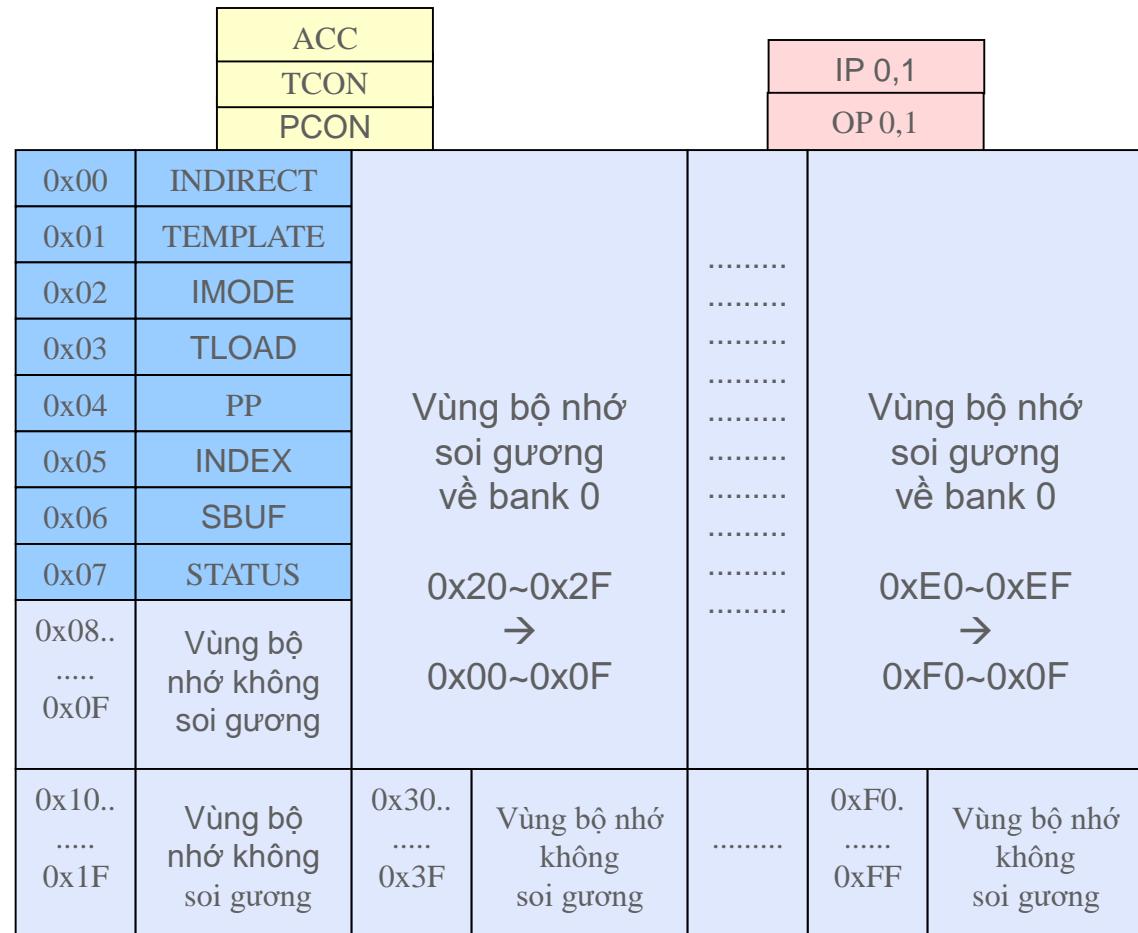
8/8

## Nguyên tắc xử lý rẽ nhánh:

- Đánh dấu các chu kỳ rẽ nhánh để có thể nhận biết được CPU đang ở giai đoạn nào của lệnh rẽ nhánh.
- Nếu một thành phần nào đó của CPU không được dùng đến, (ví dụ ALU, cá công đoạn sau) thì phải chèn lệnh NOP vào các thành phần đó (ví dụ NOP là lệnh trung tính).
- Với ALU: bộ giải mã phải đưa tín hiệu chọn toán tử NOP tới ALU.
- Với điều khiển công đoạn: bộ giải mã đẩy giá trị NOP vào các thanh ghi điều khiển công đoạn, bằng cách reset các thanh ghi này.
- Bộ giải mã phải tính toán số lượng lệnh NOP được đưa vào ALU và các công đoạn sao cho phù hợp với số công đoạn bị thiếu trong ống khi rẽ nhánh. Giải pháp: sử dụng thanh ghi dịch.

# Bộ nhớ và tập thanh ghi

- Tập thanh ghi không lập trình được: SSR, LSR, STACK...
- Tập thanh ghi lập trình được: SBUF, Acc...
- Tập lệnh không có bit phân biệt địa chỉ thanh ghi và địa chỉ ô nhớ → qui định soi gương.

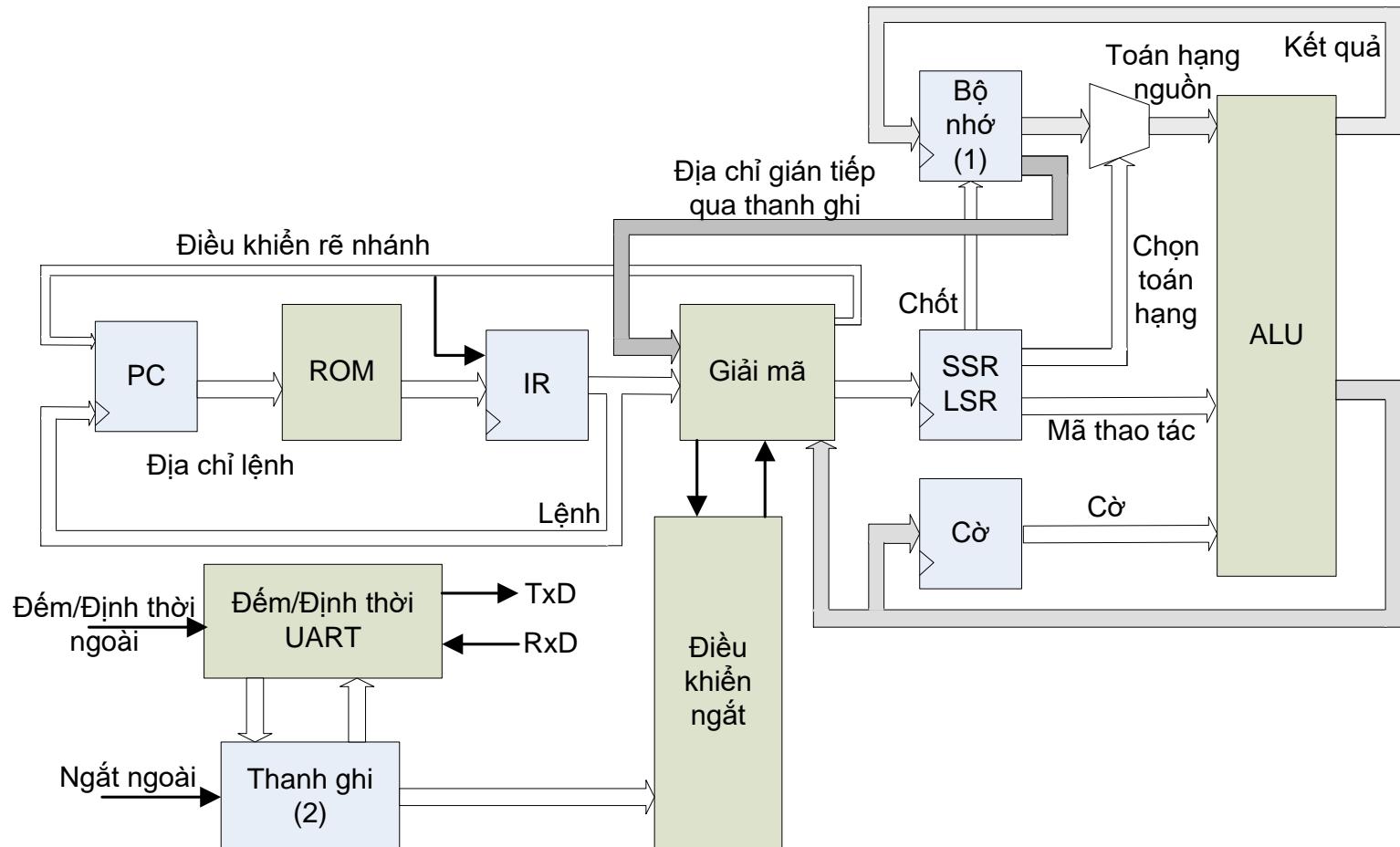


# Giải thích về soi gương bộ nhớ

- Tập lệnh **không có bit phân biệt** địa chỉ thanh ghi và địa chỉ ô nhớ. Ví dụ: lệnh yêu cầu thao tác với địa chỉ 0xEF, là địa chỉ thanh ghi hay bộ nhớ?
  - Qui định mọi địa chỉ có dạng xx0x.xxxx đều qui về địa chỉ 0000.xxxx → Qui tắc soi gương, giải mã lệnh nhanh trong 1 chu kỳ lệnh, tồn không gian địa chỉ.
  - Qui định duy nhất địa 0000.xxxx là địa chỉ thanh ghi → tiết kiệm không gian địa chỉ, khó giải mã, có thể tồn nhiều chu kỳ đồng hồ mới xác định được địa chỉ.
    - **Địa chỉ cuối cùng = thông tin trên mã lệnh (DI) [ kết hợp giá trị trong thanh ghi địa chỉ (WO) ]**

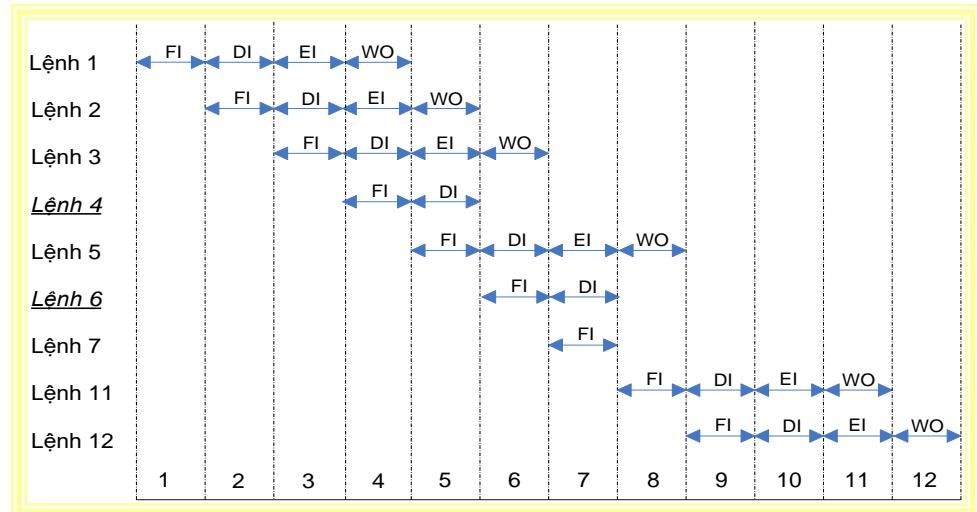
→ Tập lệnh chỉ là qui định mã bit, nhưng có vai trò hết sức quan trọng trong kiến trúc và hiệu năng của bộ xử lý.

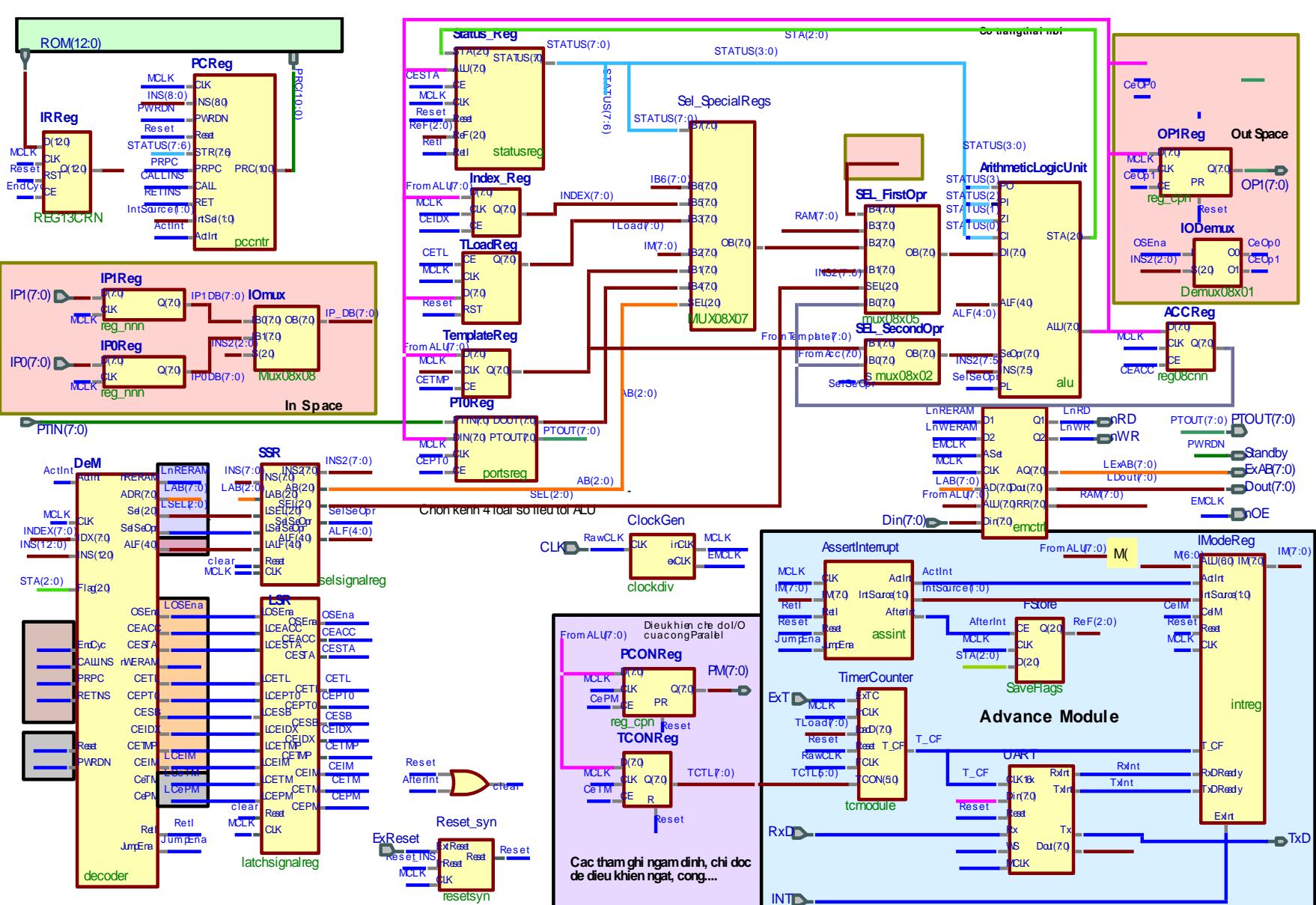
# Nguyên lý hoạt động



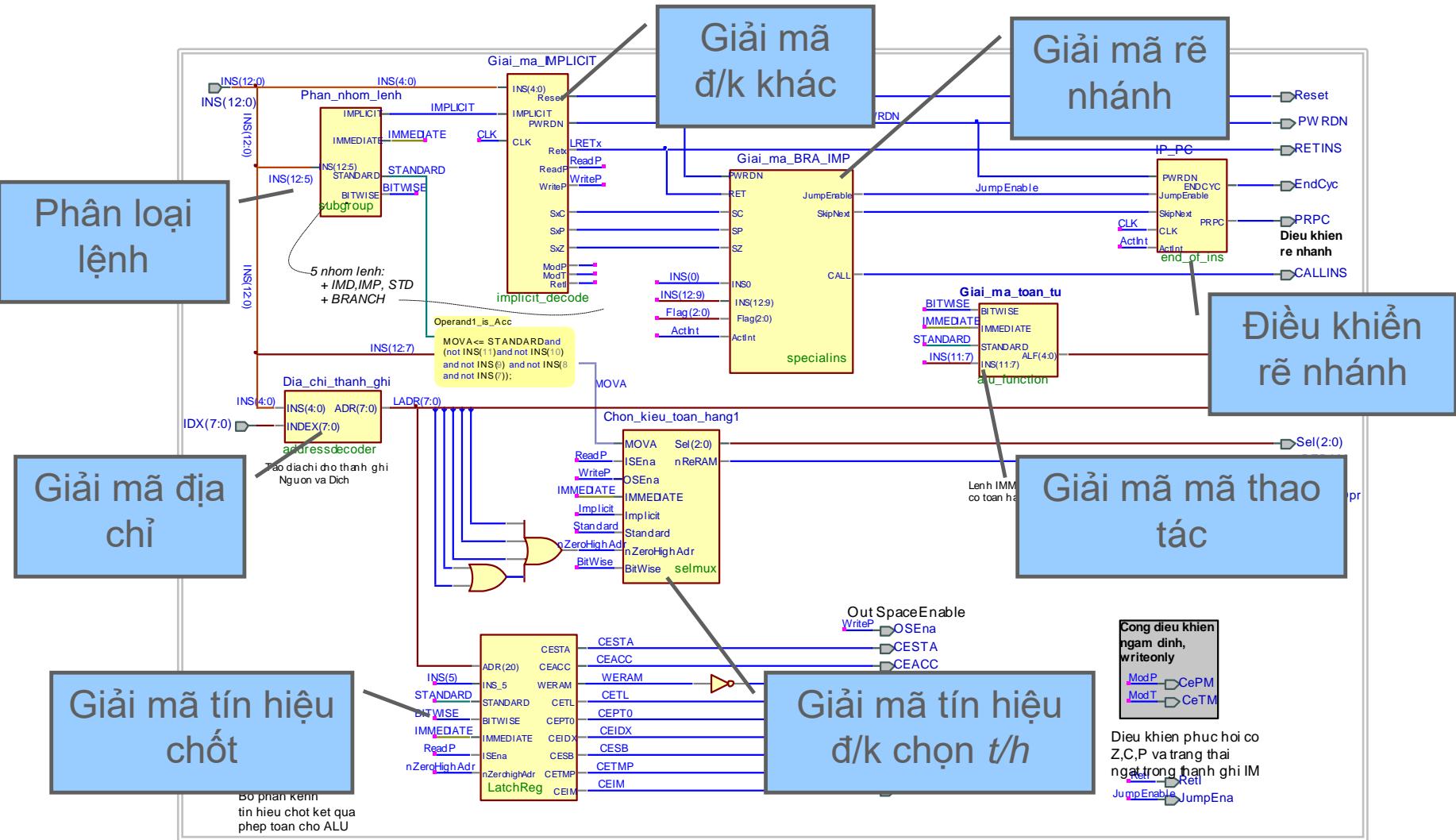
# Pipeline

- Nhận lệnh: PC, IR
- Giải mã lệnh: tuần tự, rẽ nhánh, ngắt.
- Thực hiện lệnh: nhận toán hạng, thực hiện.
- Cắt toán hạng: tín hiệu cho phép chốt từ bộ giải mã lệnh.



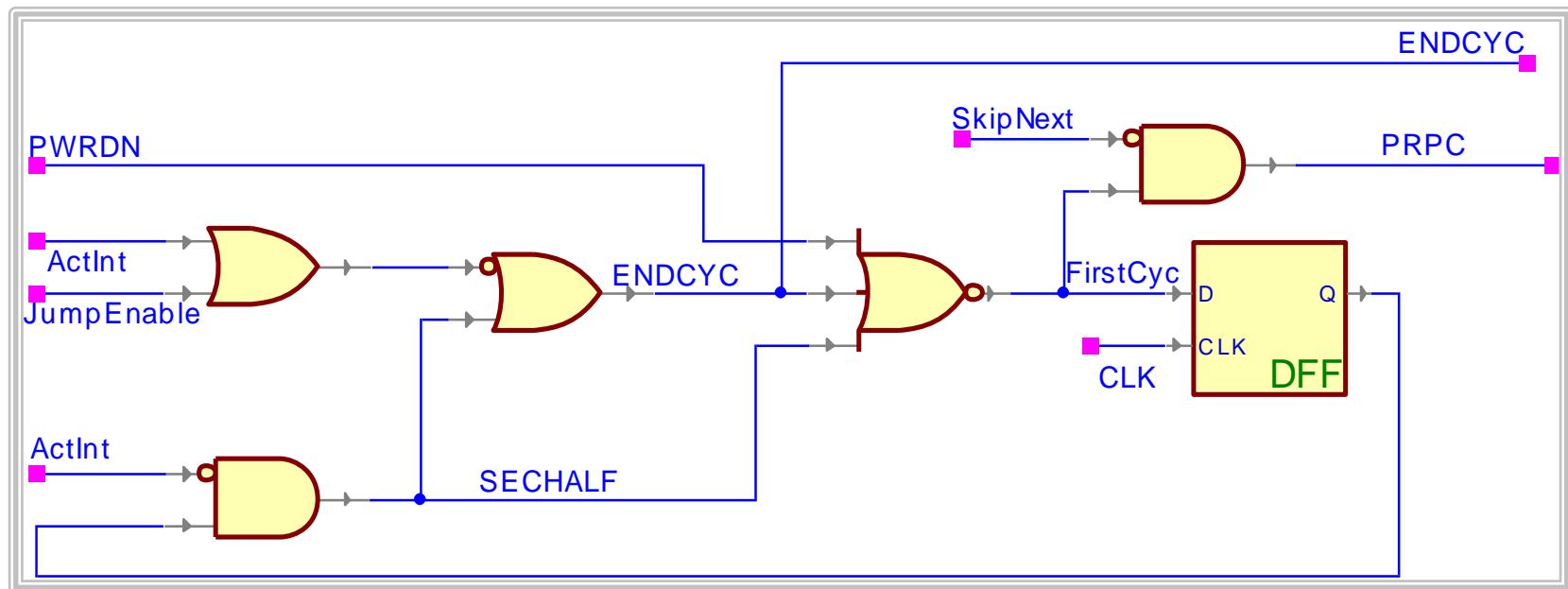


# Giải mã lệnh

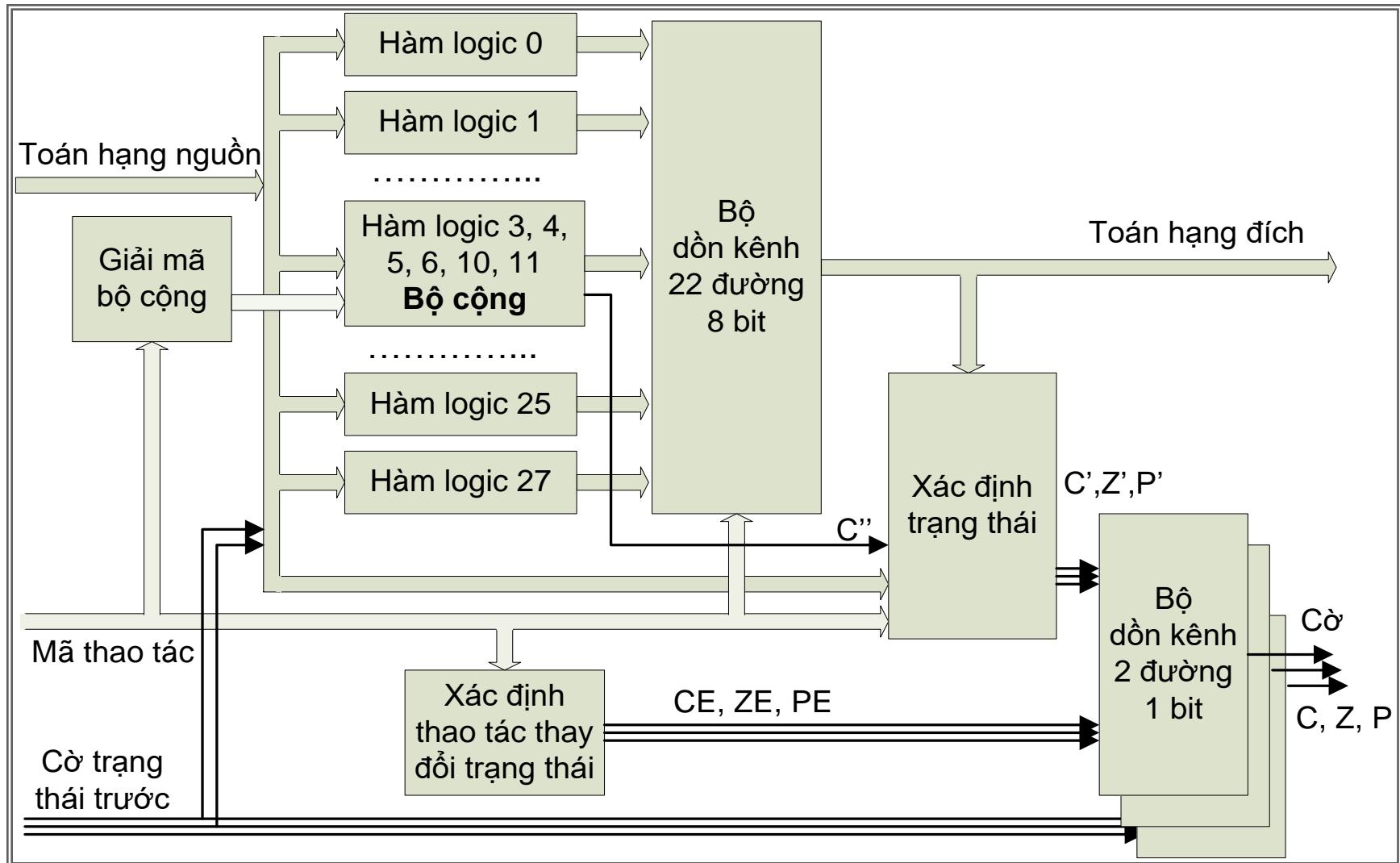


# Giải mã lệnh (2)

| Lệnh                    | SecHalf | EndCyc | FirstCyc | PC           | IR         |
|-------------------------|---------|--------|----------|--------------|------------|
| Lệnh tuần tự            | 0       | 1      | 0        | Cộng 1       | Chốt lệnh  |
| Lệnh rẽ nhánh hoặc ngắt | CK1     | 0      | 0        | Chốt địa chỉ | Không chốt |
|                         | CK2     | 1      | 1        | 0            | Cộng 1     |
|                         |         |        |          |              | Chốt lệnh  |



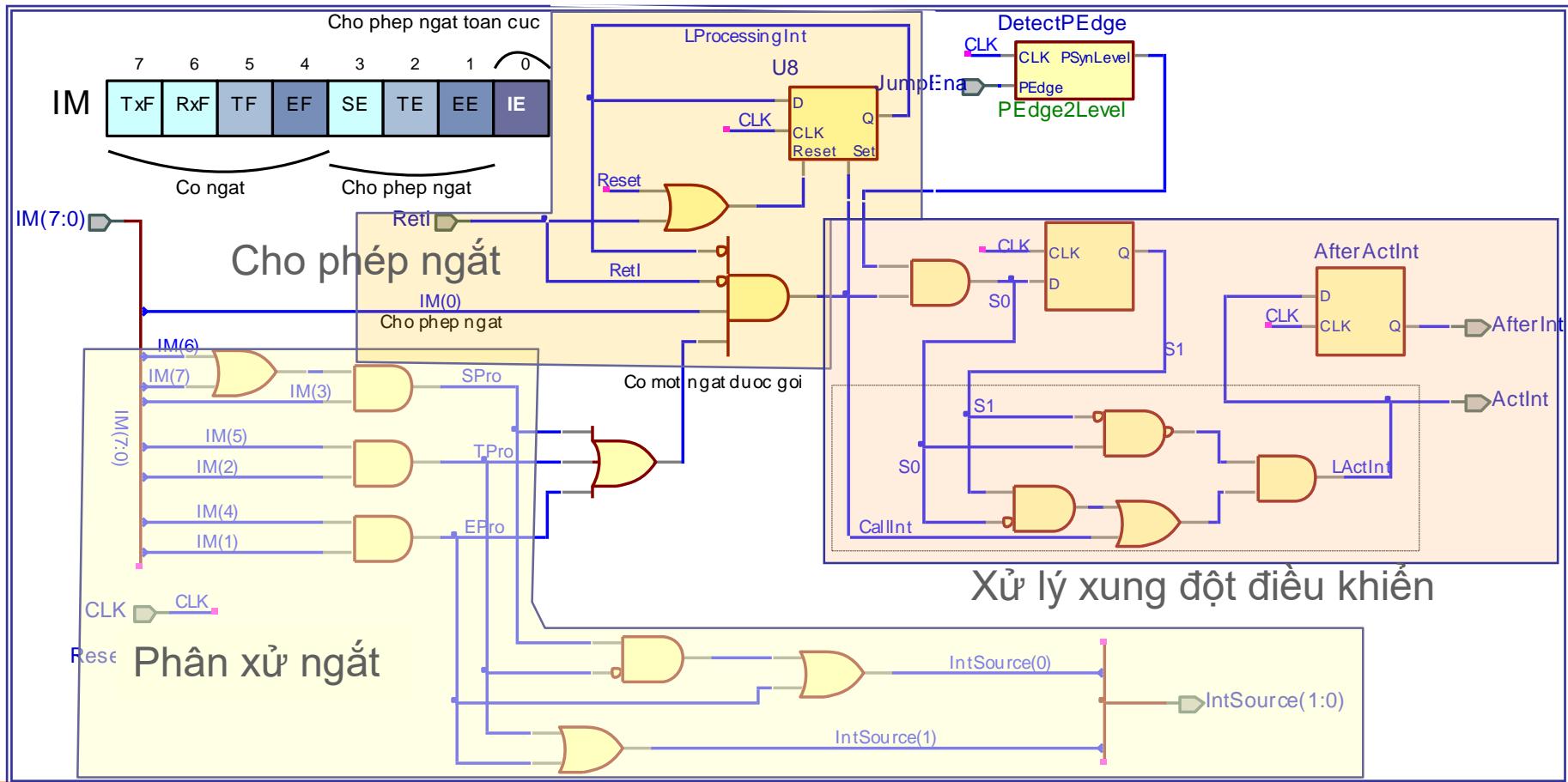
# 3.11 Thực hiện lệnh



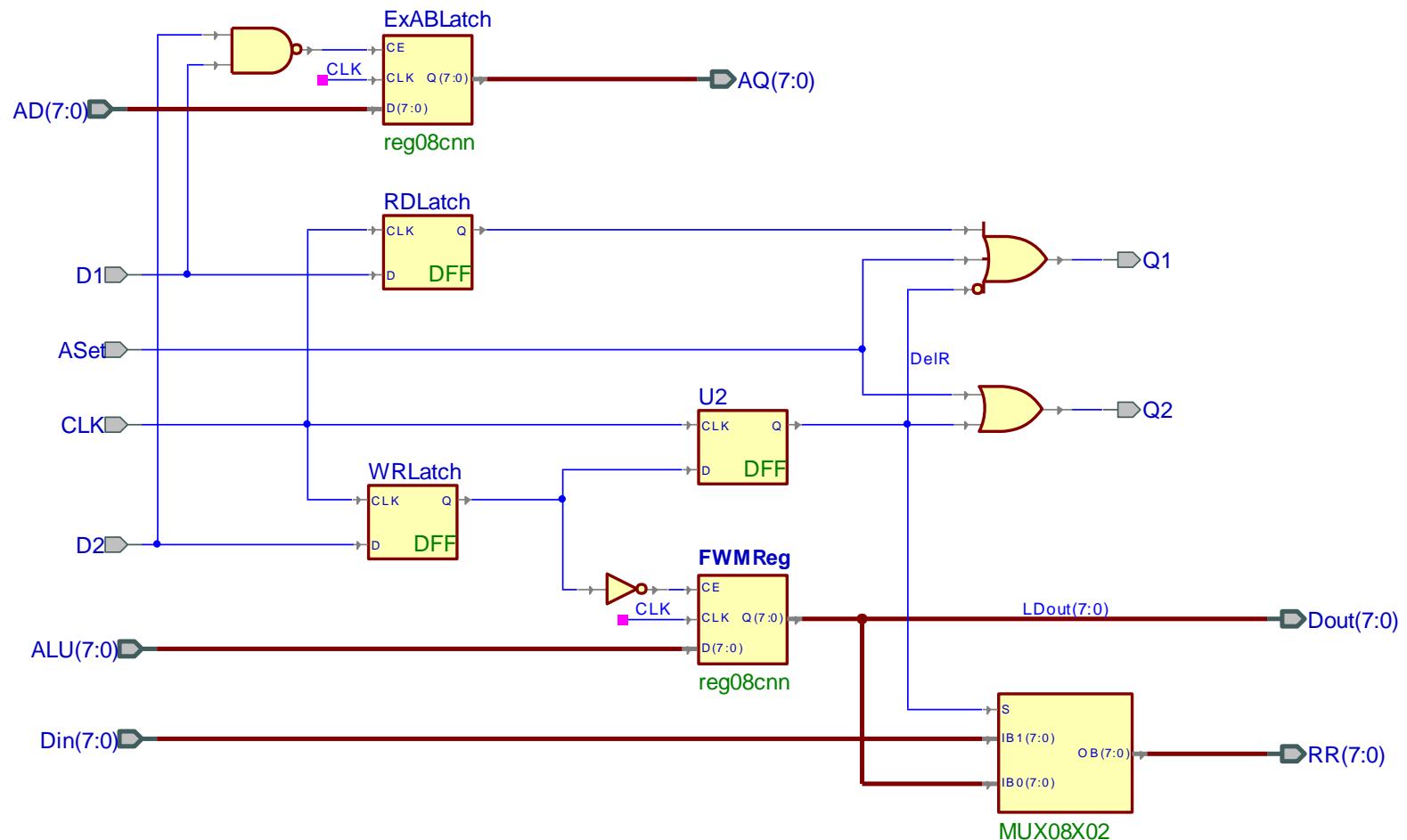
### 3.12 Các module ghép nối

- Đếm/định thời: dùng nguồn xung định thời bên trong hoặc nguồn xung đếm ngoài
- Module UART: phục vụ truyền, nhận tiếp, tốc độ boud thay đổi được, khung truyền cố định
- Ngắt:
  - 3 vector ngắt
  - 4 cờ ngắt: ngắt ngoài, ngắt đếm/định thời, ngắt truyền và nhận nối tiếp
  - Cờ ngắt ngoài và đếm/định thời xoá bằng phần cứng, cờ nối tiếp xoá bằng phần mềm
  - Chống xung đột điều khiển với lệnh rẽ nhánh

# Các module ghép nối (2)



# 3.13 Điều khiển truy xuất ExM



# 4.1 Cài đặt FPGA

- Yêu cầu tài nguyên

| FPGA            | Cổng IO  | LUT | Flip Flop  | Fmax |
|-----------------|----------|-----|------------|------|
| XC2S100-5PQ208  | 89 / 144 | 62% | 712 / 2400 | 29%  |
| XC2S30-5PQ208   | 89 / 136 | 75% | 612 / 864  | 70%  |
| XC2S50E-7TQ144  | 89 / 102 | 82% | 711 / 1536 | 46%  |
| XC2S100E-6PQ208 | 89 / 146 | 61% | 712 / 2400 | 29%  |
| XC2S300E-6FG456 | 89 / 329 | 27% | 764 / 6144 | 12%  |

## 4.2 Chức năng bo mạch

- Cấu hình FPGA để hoạt động theo thiết kế thông qua cổng JTAG
- Kiểm tra quá trình cấu hình FPGA
- Ghép nối chip với bộ nhớ ngoài
- Thiết kế các jumper tương ứng với các chân vào ra
- Xây dựng sẵn một số thiết bị vào ra cơ bản để kiểm tra hoạt động của phần mềm như DIP, Led...

→ Bo mạch phát triển đã thực hiện tất cả các ghép nối phần cứng cơ bản.

→ Người sử dụng chỉ cần nạp chương trình vào ROM để sử dụng chip

# 5.1 Thiết kế phần mềm

- Đồng thiết kế phần cứng - phần mềm: việc thiết kế phần cứng và phần mềm diễn ra song song
- Sử dụng phần mềm để kiểm thử hoạt động của chip
- Các giai đoạn phát triển phần mềm
  - Viết chương trình từng bit một.
  - Sử dụng chương trình dịch pT-BDC Compiler

# 5.2 Chương trình

The screenshot shows the TestExInt - Asip software interface. The main window displays assembly code with comments. The code includes procedures like ExIntProcedure, INCVAR, and NextProcess, along with various instructions such as READP, ANDI, JZ, DEC, JMP, INC, and RFAND. A tooltip 'Compiling...' is visible above the error list. The bottom status bar says 'For Help, press F1'.

```
TestExInt - Asip
File Edit View Build Tools Help
, ---

ExIntProcedure:
 READP #0d ; Acc:=IP0
 ANDI #00000001b ; Xet bit 0
 JZ INCVAR ; Neu bit 0 bang 0 thi tang dan Index
 ; Neu bit 0 bang 1 thi giam dan Index

 DEC Index,1 ; Index:=Index-1
 JMP NextProcess

INCVAR:
 INC Index,1 ; Index:=Index+1
NextProcess:
 RFAND #0d ; Acc:=TPA

Compiling...
Error: Unrecognized command. Line 2
Error: Unrecognized command. Line 7
Error: Unrecognized command. Line 34
Error: Unrecognized command. Line 40
Error: Unrecognized command. Line 43
Error: Unrecognized command. Line 50

For Help, press F1
```



Bộ môn Kỹ thuật Máy tính, Viện Công nghệ thông tin và Truyền thông, ĐHBKHN  
www.dce.hust.edu.vn

The End

HUE HUST

Msc. Nguyễn Đức Tiến  
tiennd@soict.hut.edu.vn  
+84-91-313-7399



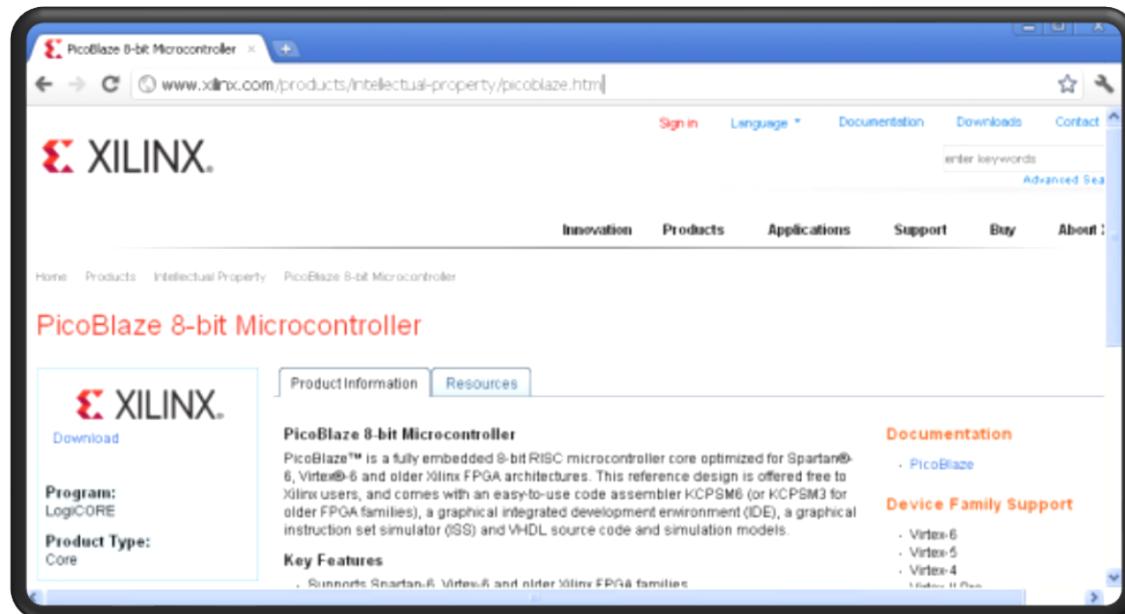
Bộ xử lý mã nguồn mở

# Picoblaze

# Về bộ xử lý Picoblaze

## PicoBlaze

- <http://www.xilinx.com/products/intellectual-property/picoblaze.htm>
- 100% code VHDL.



PicoBlaze  
Userguide

# Picoblaze: tổng quan

- Vi điều khiển nhúng 8 bit có cấu trúc RISC.
- Được tối ưu phát triển cho các họ FPGA của Xilinx như Spartan 3, Virtex II và Virtex II Pro.
- Vi điều khiển PicoBlaze được tối ưu về mặt hiệu suất và chi phí phát triển thấp.
- Mã nguồn mở VHDL, lõi mềm.
- PicoBlaze FPC được hỗ trợ bởi các công cụ assembler và IDE lập trình, giả lập, etc , và bởi các công cụ của Xilinx như System Generator hay ISE.

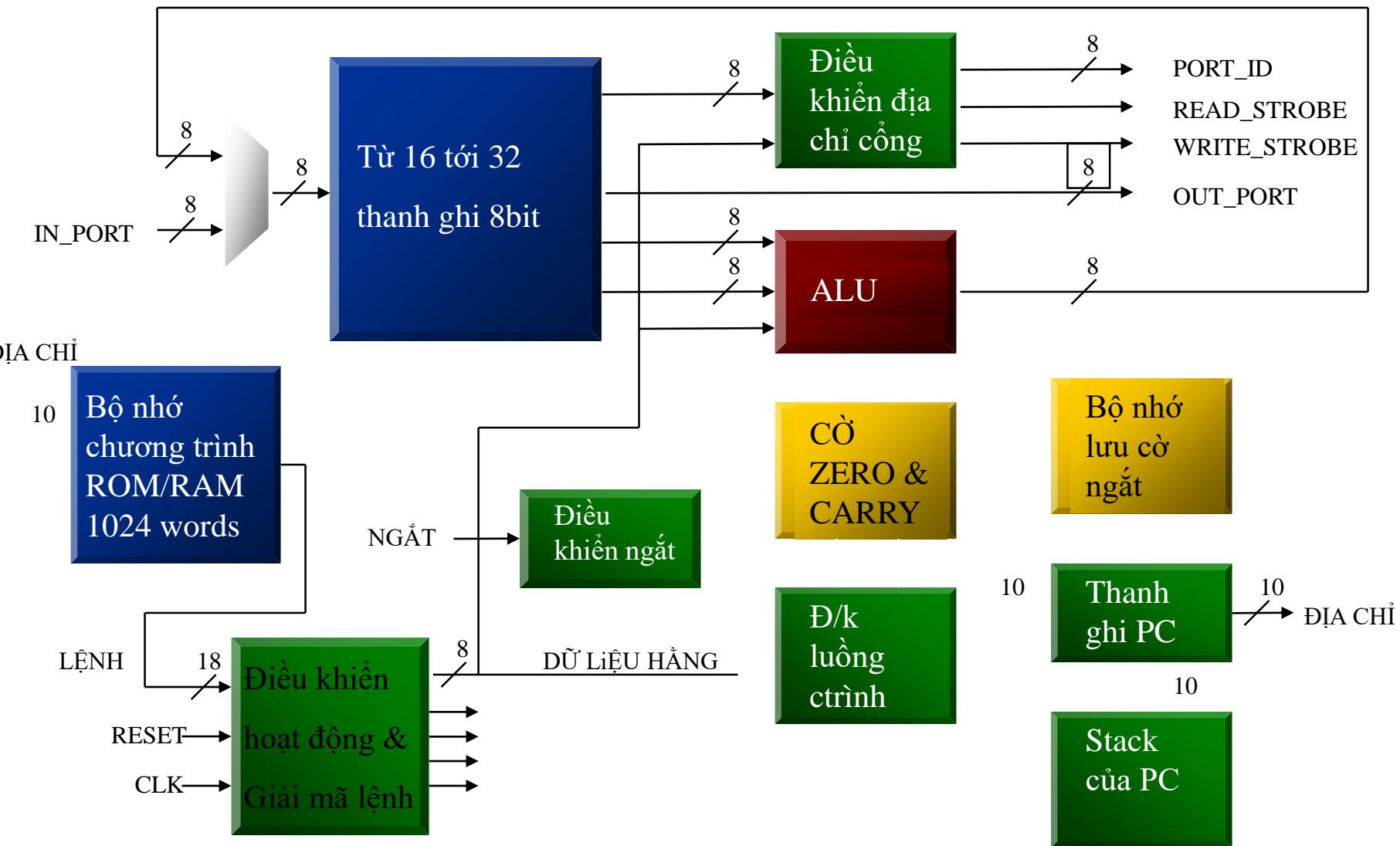
# Picoblaze: đặc trưng

- 16 thanh ghi dữ liệu chức năng chung có độ rộng 8 bit.
- Lưu trữ được 1K lệnh trong chương trình có thể lập trình được trong chip và tự động nạp khi cấu hình FPGA hay khi khởi động FPGA.
- ALU với các cờ CARRY và ZERO.
- Một bảng RAM 64 byte.
- 256 đầu vào và 256 đầu ra dễ dàng có thể mở rộng thêm.
- Stack cho phép gọi lồng 31 lần CALL/RETURN.

# Picoblaze: đặc trưng

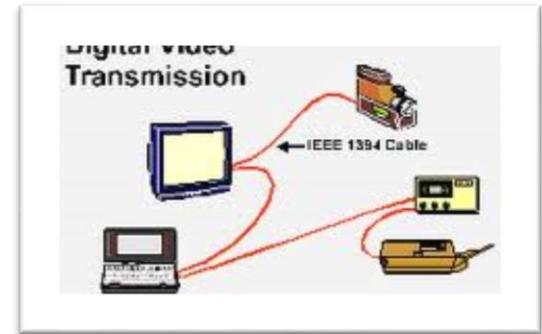
- Predictable performance, luôn luôn dùng 2 xung nhịp hệ thống cho một câu lệnh, có thể đạt tới 200 MHz hoặc 100 MIPS trong Virtex – II Pro FPGA.
- Đáp ứng ngắt nhanh; worst-case 5 clock cycles.
- Được tối ưu cho cấu trúc Spartan – 3, Virtex II, và Virtex II Pro FPGA của Xilinx chỉ chiếm 96 slices và 0.5 tới 1 block RAM.
- Hỗ trợ mô phỏng tệp lệnh assembler.

# Kiến trúc Picoblaze



# Ứng dụng của Picoblaze

- Chức năng chuyển mạch ở Front panel và chức năng hiện thị trên Set Top Box.
- Link layer trong IEEE 1394 Interface.
- Bộ vi điều khiển trong Compact Flash Programming engine.
- DECT Radio/Repeater.



# Ứng dụng của Picoblaze

- Điều khiển lập trình cho PCI board.
- Điều khiển truyền thông.
- Thực hiện tiền xử lý cho các bộ xử lý network.
- Bộ điều khiển động cơ.
- Điều khiển các nguồn cấp lập trình được.
- Là một thành phần của Media Access Controller
- Vi điều khiển trong các thiết bị broadcast video.

# Ứng dụng của Picoblaze





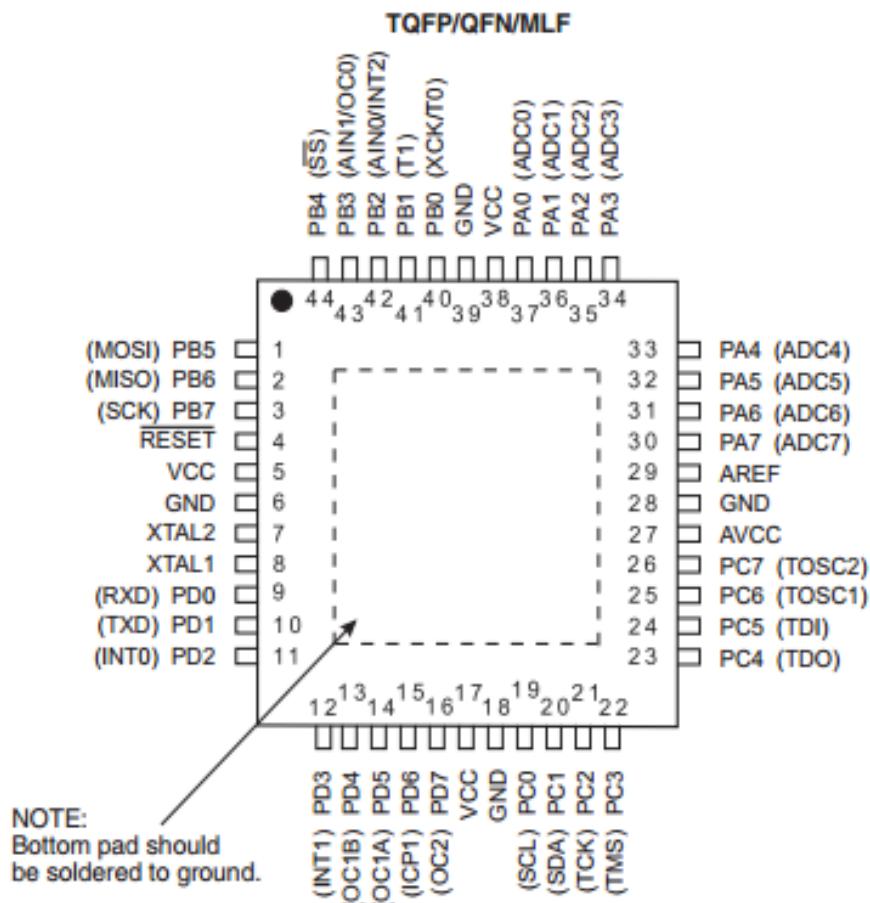
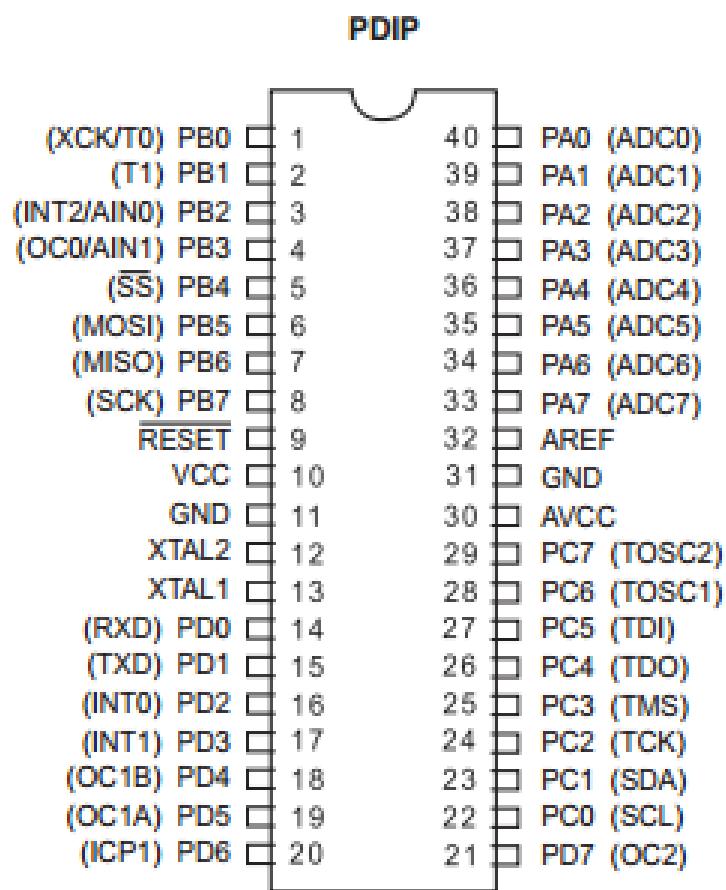
Bộ xử lý thương mại

# Atmega16

# Atmega16: Đặc tính

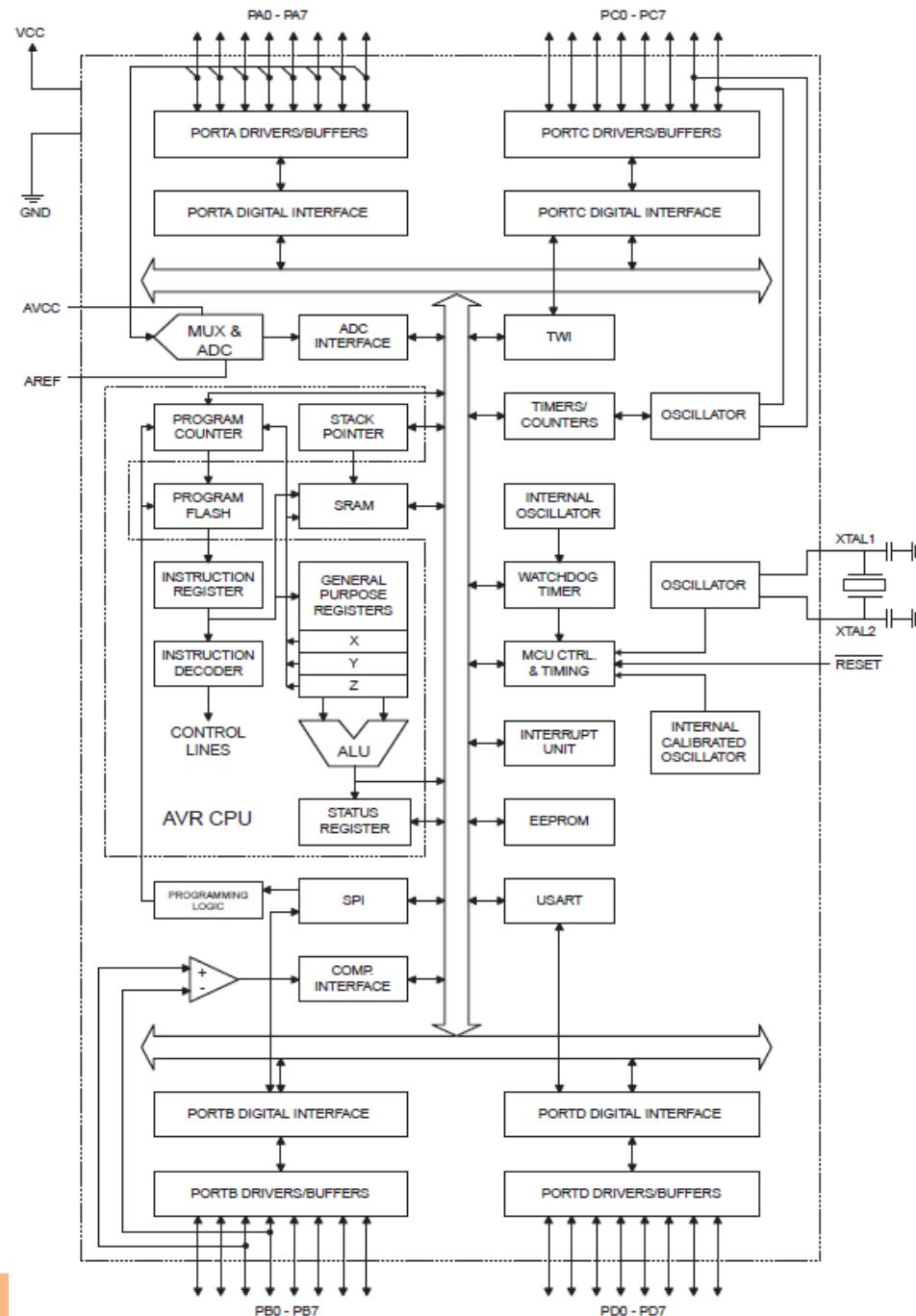
- Bộ xử lý 8 bit CMOS
- Kiến trúc RISC
- Atmega16 = bộ xử lý AVR + bộ nhớ + vào ra
- Năng lực tính toán: 1 MIPS, 1 lệnh/1 chu kỳ

# Atmega16: Đóng gói



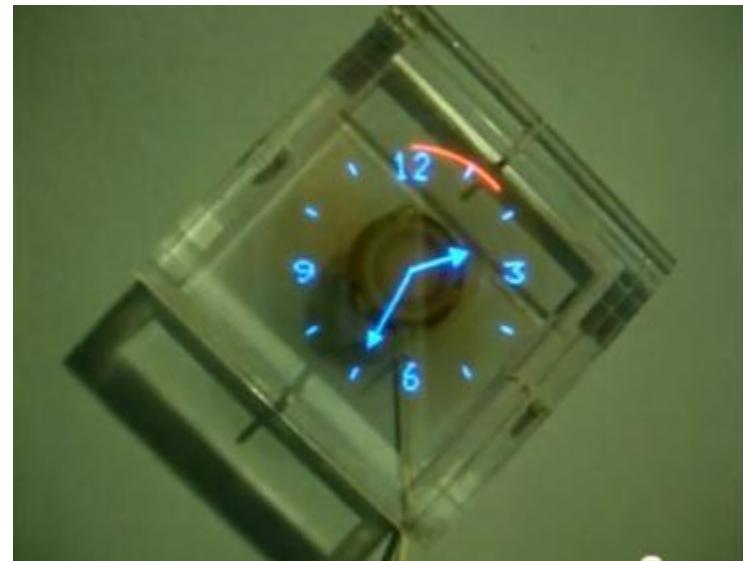
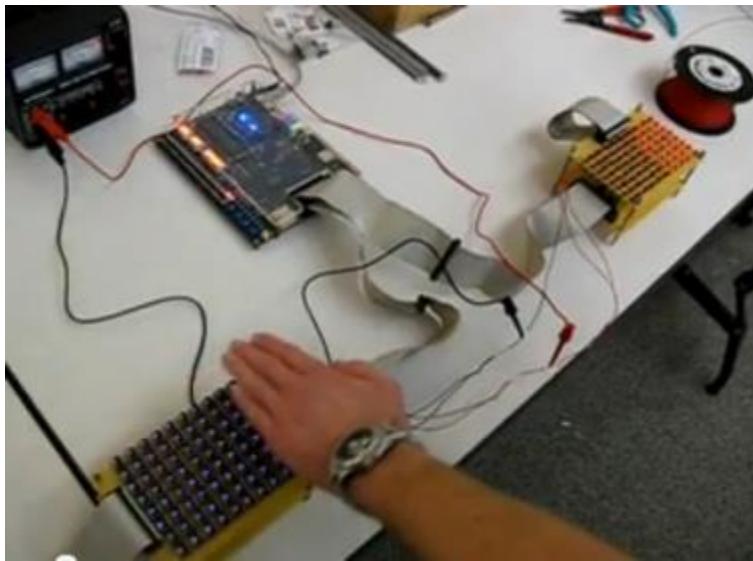
# Sơ đồ khối

| PDIP            |    |
|-----------------|----|
| (XCK/T0) PB0    | 1  |
| (T1) PB1        | 2  |
| (INT2/AIN0) PB2 | 3  |
| (OC0/AIN1) PB3  | 4  |
| (SS) PB4        | 5  |
| (MOSI) PB5      | 6  |
| (MISO) PB6      | 7  |
| (SCK) PB7       | 8  |
| RESET           | 9  |
| VCC             | 10 |
| GND             | 11 |
| XTAL2           | 12 |
| XTAL1           | 13 |
| (RXD) PD0       | 14 |
| (TXD) PD1       | 15 |
| (INT0) PD2      | 16 |
| (INT1) PD3      | 17 |
| (OC1B) PD4      | 18 |
| (OC1A) PD5      | 19 |
| (ICP1) PD6      | 20 |
| PA0             | 40 |
| PA1             | 39 |
| PA2             | 38 |
| PA3             | 37 |
| PA4             | 36 |
| PA5             | 35 |
| PA6             | 34 |
| PA7             | 33 |
| AREF            | 32 |
| GND             | 31 |
| AVCC            | 30 |
| PC7 (TOSC2)     | 29 |
| PC6 (TOSC1)     | 28 |
| PC5 (TDI)       | 27 |
| PC4 (TDO)       | 26 |
| PC3 (TMS)       | 25 |
| PC2 (TCK)       | 24 |
| PC1 (SDA)       | 23 |
| PC0 (SCL)       | 22 |
| PD7 (OC2)       | 21 |



- Một số kit tại CSLab.
- Hướng dẫn phát tín hiệu VGA bằng FPGA.  
<http://www.fpga4fun.com/PongGame.html>
- FPGA Based VGA driver and Arcade game.  
[http://static.armandas.lt/res/fpga\\_based\\_vga\\_driver\\_and\\_arcade\\_game.pdf](http://static.armandas.lt/res/fpga_based_vga_driver_and_arcade_game.pdf)
- Hướng dẫn sử dụng Altera Quatus
- Hướng dẫn sử dụng Altera Quatus, Xillinx ISE, Actel Libero
- Hướng dẫn sử dụng kit Spartan II LC VN

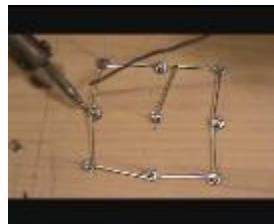
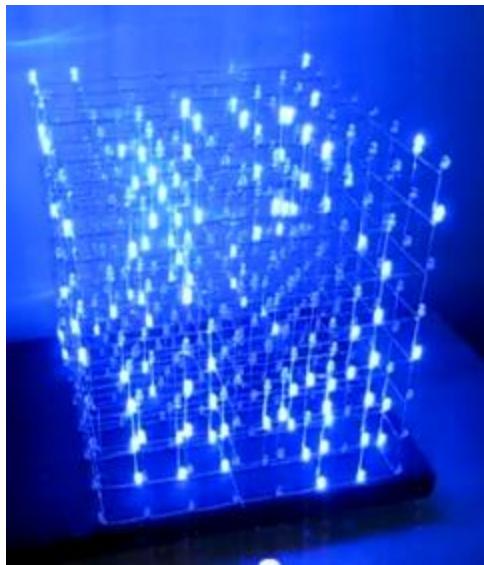
# Một số ứng dụng thú vị



- Sử dụng 80 cảm biến quang để gửi tín hiệu tới DE2 board qua các thanh ghi vào song song-ra nối tiếp (165) và gửi kết quả tới ma trận led 8x10 thông qua các thanh ghi dịch 8-bit vào nối tiếp-ra song song (595).
- <http://youtu.be/LCljWp7LDI8>

- Hướng dẫn thực hiện  
[http://youtu.be/no2\\_M\\_b059g](http://youtu.be/no2_M_b059g)  
<http://www.elektroda.pl/rtvforum/topic944484.html>

# Một số ứng dụng thú vị



- Create your own 8x8x8 LED Cube 3-dimensional display. Guide to build.
- <http://youtu.be/6mXM-oGggrM>
- <http://youtu.be/ea8aG2aQ5FY>
- <http://www.instructables.com/id/Led-Cube-8x8x8/?ALLSTEPS>