

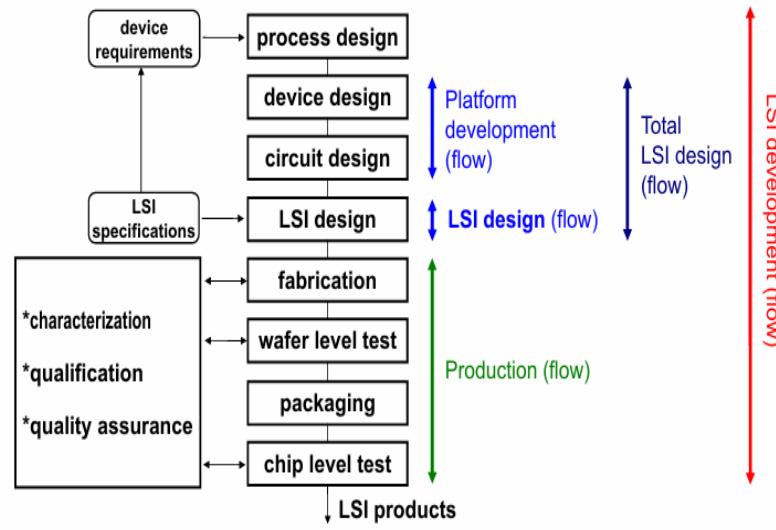
CHAPTER 3: LSI Logic Design

3.1. LSI Design Flow.

Tổng Quan về LSI Development Flow

Tên gọi	Giai đoạn bao gồm
Platform Development Flow	Process Design → Device Design → Circuit Design
LSI Design Flow	Circuit Design → LSI Design
Total LSI Design Flow	Platform Development + LSI Design
LSI Development Flow	Toàn bộ quá trình từ yêu cầu đến sản phẩm cuối cùng
Production Flow	Bắt đầu từ Fabrication → Chip Level Test

LSI Development Flow (Review)



CHI TIẾT CÁC BƯỚC

1. Device Requirements: Yêu cầu chức năng, hiệu năng, chi phí, môi trường sử dụng của sản phẩm.

2. Process Design: Thiết kế quy trình chế tạo bán dẫn (Công nghệ CMOS, FinFET, v.v.), Quy định về điện áp, loại transistor, lớp kim loại...

3. Device Design Mô hình hóa các linh kiện vật lý như transistor, diode, MOSFET. Phân tích vật lý, đặc tính điện.

4. Circuit Design

- Thiết kế mạch cấp thấp (Adders, Flip-Flops, Multipliers, Buffers...).
- Sử dụng thư viện cell chuẩn (standard cell library).

5. LSI Design

Thiết kế toàn chip: RTL → Synthesis → P&R → Verification. Mô phỏng, kiểm tra logic, timing, tiêu thụ điện, nhiễu...

6. Fabrication

Sản xuất wafer silicon với layout thiết kế đã hoàn chỉnh. Giai đoạn tốn kém nhất.

7. Wafer Level Test

- Kiểm tra chip khi còn trên wafer.
- Đảm bảo chức năng cơ bản, loại bỏ die lỗi.

8. Packaging

- Cắt die, đóng gói (package) thành chip hoàn chỉnh.
- Đảm bảo kết nối chân, nhiệt độ, bảo vệ vật lý.

9. Chip Level Test

- Kiểm thử toàn bộ chip sau khi đóng gói.
- Kiểm tra chức năng, hiệu năng, độ tin cậy.

CÁC GIAI ĐOẠN ĐÁNH GIÁ KHÁC

- Characterization:** Đo lường các đặc tính điện của chip.
- Qualification:** Kiểm định độ tin cậy lâu dài.
- Quality Assurance (QA):** Đảm bảo chất lượng toàn bộ quy trình sản xuất.

Kết quả cuối cùng: LSI Products → Chip hoàn chỉnh, sẵn sàng tích hợp vào hệ thống.

LSI Design Flow – Quy trình thiết kế logic

Input ban đầu: System spec (đặc tả hệ thống):

→ Xác định yêu cầu chức năng, hiệu năng, công suất tiêu thụ, kích thước, môi trường hoạt động, v.v.

Output cuối cùng của giai đoạn này: Mask making

→ Dữ liệu layout được sử dụng để tạo các mask quang học cho quy trình quang khắc (photolithography) trong chế tạo chip.

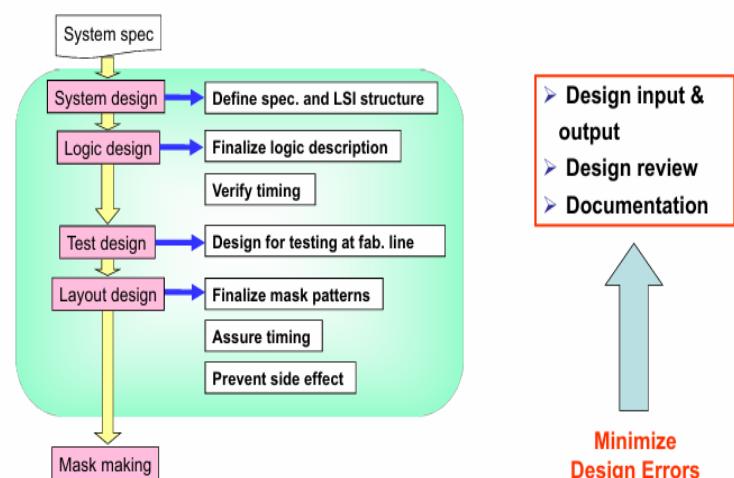
Ghi chú quan trọng ở bên phải hình:

- Design input & output:** Đảm bảo đầu vào & đầu ra giữa mỗi bước là rõ ràng và được kiểm định.
- Design review:** Có kiểm tra chéo ở mỗi giai đoạn để phát hiện lỗi sớm.
- Documentation:** Ghi chép đầy đủ để hỗ trợ bảo trì, kiểm tra, và debug sau này.

Mục tiêu chính: Minimize Design Errors (Tối thiểu hóa lỗi thiết kế)

Các bước chính trong thiết kế LSI

LSI Design Flow (1/2)



Bước	Hoạt động chính
1. System Design	<ul style="list-style-type: none"> - Định nghĩa kiến trúc tổng thể của hệ thống LSI - Phân rã module chức năng
2. Logic Design	<ul style="list-style-type: none"> - Mô tả logic (thường dùng Verilog/VHDL) - Verify timing (Static Timing Analysis) - Hoàn thiện logic
3. Test Design	<ul style="list-style-type: none"> - Thiết kế các cấu trúc testability (scan chain, BIST...) - Đảm bảo khả năng kiểm tra tại dây chuyền sản xuất
4. Layout Design	<ul style="list-style-type: none"> - Tạo mặt nạ (mask patterns) dùng trong sản xuất wafer - Assure timing sau khi bố trí (Layout vs Schematic, DRC/LVS) - Prevent side effects: nhiễu, coupling, IR drop, electromigration, v.v.

Tổng kết nhanh Giai đoạn này gồm: Spec → Logic → Test → Layout → Mask making

👉 Với các yếu tố then chốt: timing, testability, noise, và độ chính xác về layout.

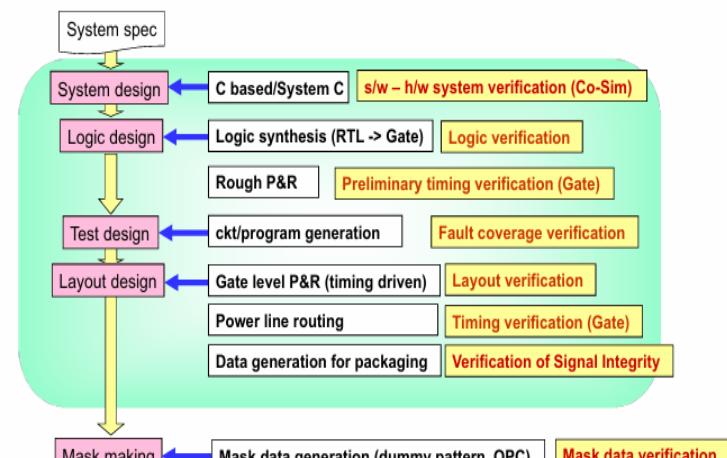
Chi tiết các bước chính và hoạt động tương ứng

Bước	Hoạt động kỹ thuật	Mục đích kiểm tra / xác minh
System Design	- Mô phỏng bằng C-based / SystemC	SW-HW co-simulation để kiểm chứng kiến trúc hệ thống
Logic Design	<ul style="list-style-type: none"> - Tổng hợp logic: RTL → Gate-level netlist - Rough P&R (sắp xếp sơ bộ) 	<i>Logic verification, Preliminary timing (Gate)</i>
Test Design	- Tạo chương trình kiểm thử (test pattern/program generation)	<i>Fault coverage verification</i>
Layout Design	<ul style="list-style-type: none"> - Gate-level P&R (timing-driven) - Power line routing - Dữ liệu đóng gói 	<i>Layout verification</i> <i>Timing (Gate)</i> <i>Signal integrity check</i>
Mask Making	- Tạo dữ liệu mask: dummy pattern, OPC, v.v.	<i>Mask data verification</i>

✓ Các điểm xác minh (Verification Points)

Verification Loại	Vị trí/Hoạt động liên quan
Co-simulation SW/HW	Trong System Design – dùng SystemC để mô phỏng tích hợp
Logic Verification	Sau quá trình Logic Synthesis (RTL → Gate)
Preliminary Timing (Gate-level)	Sau bước Rough P&R
Fault Coverage Verification	Trong Test Design
Layout Verification	Gate-level P&R trong Layout Design
Timing Verification (Gate-level)	Power Routing
Signal Integrity Verification	Trước khi tạo dữ liệu đóng gói
Mask Data Verification	Sau khi tạo dữ liệu mask

LSI Design Flow (2/2)



Ghi chú:

- Co-Simulation:** Cần xác minh sớm tính đúng đắn của hệ thống phần mềm–phần cứng trước khi đi vào chi tiết logic.
- OPC (Optical Proximity Correction):** Kỹ thuật xử lý mask để đảm bảo pattern in ra đúng dù có hiệu ứng quang học.
- Signal Integrity:** Bao gồm nhiễu chéo, ringing, IR drop – rất quan trọng trong tốc độ cao.

3.2. System Design.

Tầng 1: Device (Tr, C, R) Transistor (Tr), Capacitor (C), Resistor (R).

Là phần tử điện tử cơ bản để xây dựng các mạch.

Tầng 2: Circuits: Các mạch cơ bản được xây dựng từ các device:

- ADC, DAC:** Chuyển đổi tương tự \leftrightarrow số
- AND/OR, FF/Latch:** Các khối logic cơ bản
- Memory, DSP, CPU:** Khối chức năng số
- USB1/2, DDRI/II, LVDS, 1394, MPEG2/4:** Giao tiếp và xử lý tín hiệu

Tầng 3: Application Specific Ips - Các khối IP chuyên biệt cho từng ứng dụng:

- Được tích hợp lại từ các mạch trên
- Có thể tái sử dụng trong nhiều thiết kế khác nhau

Tầng 4: Systems - Tích hợp các IP để hình thành hệ thống hoàn chỉnh:

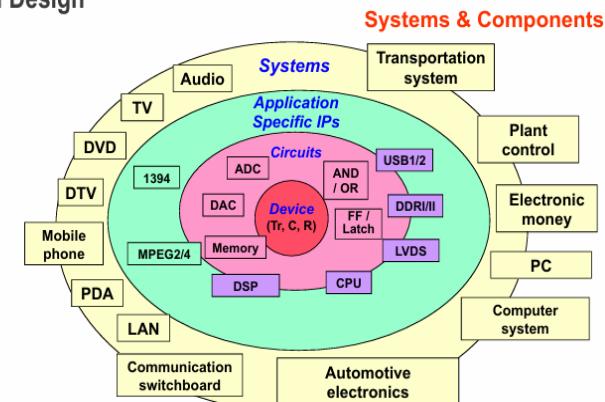
- Ví dụ: TV, DVD, Mobile Phone, PDA, LAN, Audio, Communication board

Tầng 5: Systems & Components – Ứng dụng cuối cùng

Sử dụng LSI trong các lĩnh vực công nghiệp/thương mại:

- Automotive electronics, Plant control, PC, Electronic money, Transportation system, etc.**

System Design



Ý nghĩa của sơ đồ này:

- Cho thấy **System Design** là quá trình đi từ **thiết bị cơ bản** \rightarrow **IP** \rightarrow **hệ thống hoàn chỉnh**
- LSI không chỉ phục vụ 1 ứng dụng cụ thể mà còn **có thể tái sử dụng và mở rộng cho nhiều lĩnh vực**
- Mức trừu tượng** tăng dần từ trong ra ngoài \rightarrow thiết kế LSI cần tính đến khả năng tích hợp đa lớp này.

System Integration – Tổng hợp hệ thống

Tiêu chí	SoC (System-on-Chip)	SiP (System-in-Package)
Mô tả	Tích hợp toàn bộ hệ thống vào 1 chip LSI duy nhất	Tích hợp nhiều chip (LSI) vào 1 gói đóng gói (package)
Cấu trúc	Một chip đơn duy nhất	Nhiều chip trong 1 package
Thành phần tích hợp	CPU, Memory, Peripherals	CPU, Memory, ASIC, v.v.
Ưu điểm chính	<ul style="list-style-type: none"> ⚡ Hiệu năng cao 🔋 Tiết kiệm điện 📦 Tiết diện nhỏ 	<ul style="list-style-type: none"> 🔧 Linh hoạt hơn trong thiết kế 📦 Dễ dàng tái sử dụng IP/chip
Ứng dụng phổ biến	Thiết bị di động, embedded	IoT, thiết bị chuyên dụng, wearable

Tiến trình tích hợp:

1. System board 1:

- CPU, Memory rời rạc
- Dùng các peripheral discrete ICs (74 series)

2. System board 2:

- Ghép thêm ASIC và giảm số lượng IC rời rạc
- Bắt đầu tích hợp dần

3. SoC:

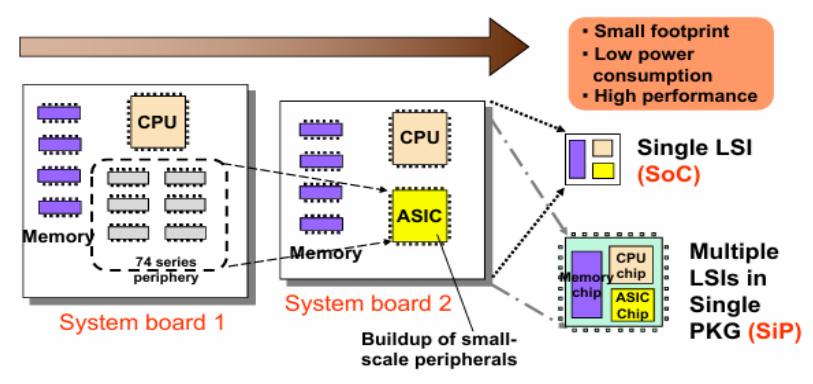
- Tích hợp tất cả vào một chip duy nhất
- \rightarrow Hiệu năng, tiết kiệm điện, nhỏ gọn

4. SiP:

- Nhiều chip (CPU, ASIC, Memory) trong 1 package
- Có thể kết hợp chip với công nghệ khác nhau (analog + digital)

SoC (System-on-Chip): System implementation in single LSI

SiP (System-in-Package): System implementation in 1 package



System Design Flow – Luồng thiết kế hệ thống

Giai đoạn System Design (Thiết kế hệ thống)

Bước	Nội dung	Mục tiêu
1. Investigation of spec	Nghiên cứu yêu cầu và chức năng	► Đề xuất đặc tả và tính năng ngoài
2. Architecture Definition	Lựa chọn kiến trúc, CPU core, tổ chức hệ thống	► Phân chia nhiệm vụ phần cứng (HW) và phần mềm (SW)
3. Evaluation	Đánh giá hiệu năng, chi phí	► Ước tính tổng quan trước khi đi sâu thiết kế

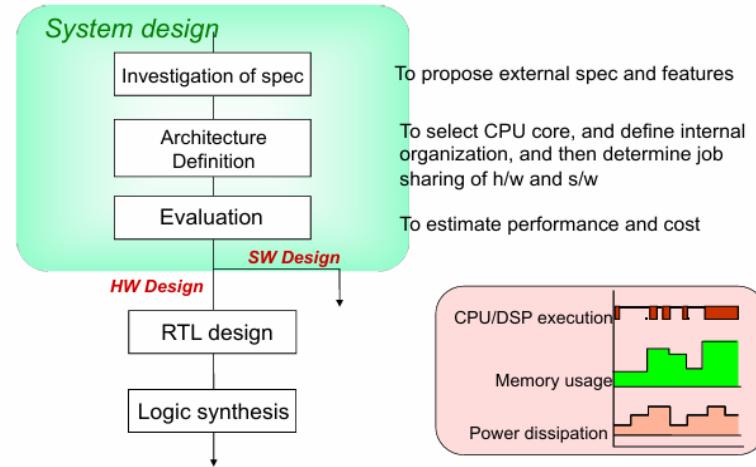
Kết quả của bước này sẽ phân nhánh thành:

- **HW Design** → Thiết kế RTL → Tổng hợp logic (synthesis)
- **SW Design** → Thiết kế phần mềm nhúng (embedded software)

HW Design Flow sau bước System Design:

- RTL Design: Viết mô tả phần cứng ở cấp độ thanh ghi (VHDL/Verilog)
- Logic Synthesis: Biến RTL thành mạng cổng logic thực tế

System Design Flow



Biểu đồ phụ (bên phải slide): Mô phỏng đặc tính hệ thống

Biểu diễn trực quan:

- **CPU/DSP execution:** Hoạt động xử lý trung tâm
- **Memory usage:** Dung lượng bộ nhớ được sử dụng
- **Power dissipation:** Mức tiêu thụ công suất

Dùng để:

- Kiểm tra hiệu năng hệ thống
- Phân tích chi phí tài nguyên
- Giúp điều chỉnh phân chia phần cứng/phần mềm hợp lý hơn

Requirements for System Design – Yêu cầu trong thiết kế hệ thống LSI

1. Tối ưu hóa đặc tả (LSI specifications optimization)

► Cần được thực hiện dựa trên:

- Xu hướng thị trường (market trend)
- Phản hồi từ khách hàng (customer needs)

Các yếu tố chính cần tối ưu:

- Hiệu năng (performance)
- Tính năng (features)
- Công suất tiêu thụ (power dissipation)
- Độ tin cậy (reliability)
- Giá thành (price) → liên quan đến:
 - Kích thước die (target die size)
 - Loại package sử dụng
 - Công nghệ chế tạo (process node)

2. Loại bỏ over-spec và bug trong đặc tả

- ◆ Over-spec: Thiết kế vượt quá yêu cầu thực tế → lãng phí tài nguyên, tăng chi phí
- ◆ Spec bugs: Lỗi logic hoặc mô tả mâu thuẫn trong yêu cầu hệ thống

3. Tránh thay đổi đặc tả trong giai đoạn thiết kế phần cứng

Lý do quan trọng:

- Time-to-market ngày càng gấp rút
- Độ phức tạp của thiết kế ngày càng tăng
- Hiệu năng yêu cầu ngày càng cao
- Chi phí chế tạo rất lớn

Một bộ mask fabrication có thể lên tới hơn \$1 triệu đô la
Nếu đặc tả thay đổi muộn → làm lại mask, thiết kế lại → rất tốn kém và mất thời gian

Early Prototype – Nguyên mẫu sớm trong thiết kế LSI

Mục tiêu: Tạo nền tảng phát triển phần mềm và xác minh hệ thống (system verification) trước khi bắt đầu thiết kế RTL.

Luồng xử lý (Flow):

1. Mô tả chức năng (Functional description) ➤ Được viết bằng ANSI C (phần mềm + phần cứng)

2. Phân chia hệ thống:

- Software section
- Hardware section
- Phần giao tiếp (I/F section) nằm ở ranh giới hai phần

3. Công cụ chuyển đổi: eXCite™ (YXI Co.)

- Dùng để:
 - I/F synthesis: tổng hợp phần giao tiếp
 - High-level synthesis: tổng hợp từ C → RTL
- Đầu ra là mô tả phần cứng RTL (VHDL/Verilog)

4. Tạo Prototype trên FPGA:

- Ghép nối RTL với các khối driver/glue logic
- Triển khai lên bo mạch thật để chạy thử ứng dụng
- Có Interlock giữa phần mềm và phần cứng để đồng bộ

Ứng dụng: Dùng để kiểm thử:

- Tốc độ xử lý cao (High-speed simulation)
- Môi trường phát triển phần mềm (Software development)

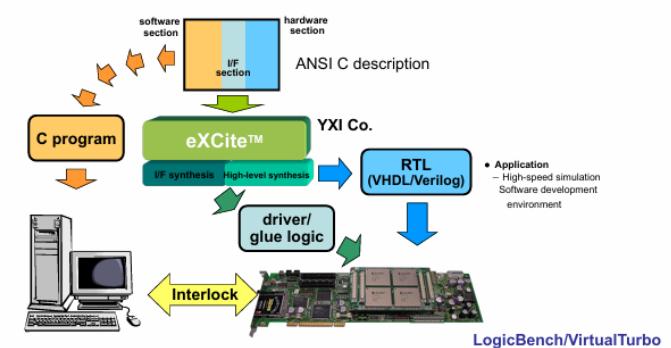
Ý nghĩa chính:

- Tiết kiệm thời gian bằng cách bắt đầu phát triển phần mềm sớm
- Cho phép kiểm tra tính đúng đắn của hệ thống trước khi chuyển sang RTL
- Hỗ trợ việc co-simulation (mô phỏng đồng thời) giữa phần mềm và phần cứng

Early Prototype

Functional description (algorithm) in ANSI C is converted to form FPGA-based early prototype

☞ Platform for software development, and system verification prior to initiation of RTL design.



3.3. Logic Design.

Logic Design Flow – Luồng thiết kế logic: Quy trình thiết kế logic nằm **giữa System Design và Test Design**, gồm các bước quan trọng từ mô tả RTL đến xác minh timing sơ bộ.

Các bước chi tiết:

1. RTL Design

- Ngôn ngữ dùng: Verilog HDL / VHDL
- Mục tiêu: Mô tả hành vi logic của mạch

2. RTL Verification

- Kiểm tra logic RTL có tuân thủ đúng đặc tả chức năng (functional spec) không
- Công cụ phổ biến:
 - NC-Verilog, VCS, Xcelium, ModelSim

- Mục tiêu: Xác minh chức năng của thiết kế

3. Logic Synthesis

- Biến RTL thành mô tả cổng logic (Gate Level)
- Công cụ:
 - Design Compiler, Genus Synthesis

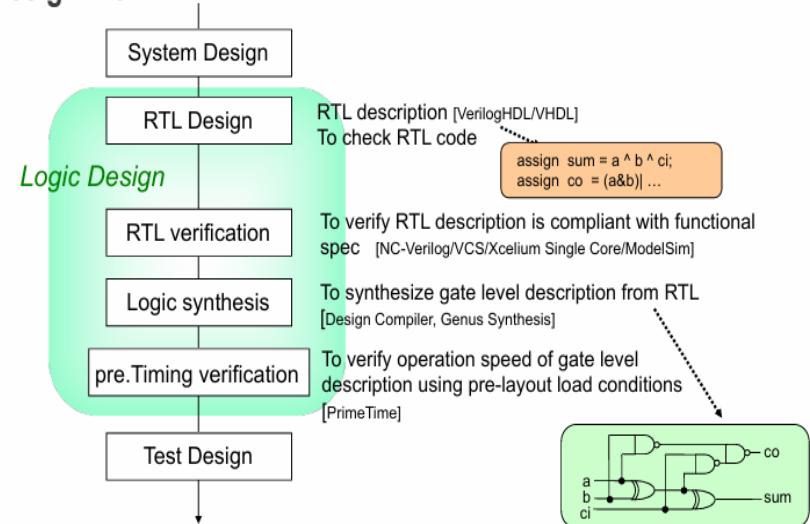
- Kết quả: Tập hợp các gate như AND, OR, NOT, flip-flop,...

4. Pre-Timing Verification

- Kiểm tra tốc độ hoạt động (timing) của mạch cổng trước khi đặt layout
- Điều kiện giả lập tải (pre-layout load conditions)
- Công cụ: PrimeTime

Mối liên kết với Test Design: Sau khi hoàn tất thiết kế logic và kiểm tra sơ bộ, thiết kế được chuyển sang Test Design để phát triển chiến lược kiểm thử (test strategy) và tạo các mạch kiểm tra (test circuitry).

Logic Design Flow



Design and Modeling – Tổng quan quy trình thiết kế phần cứng

Design Process và Khoảng Cách Ngôn Ngữ

Quy trình thiết kế từ Ý tưởng → Netlist:

- Ý tưởng (Idea)** → Mô tả chức năng tổng quát, yêu cầu ban đầu.
- Specification (Đặc tả)** → Viết bằng ngôn ngữ tự nhiên hoặc công cụ mô hình hóa (UML, C, v.v.)
- Chuyển đổi sang Netlist** → Gồm các bước như viết HDL, tổng hợp logic, tạo GDS và mask pattern để sản xuất chip.

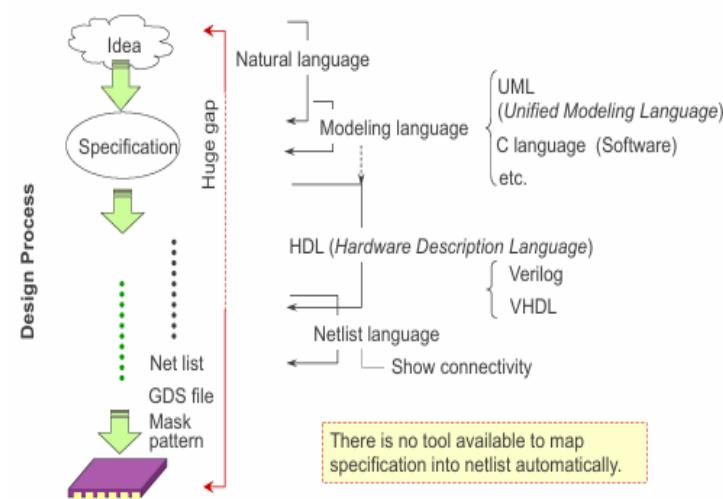
Vấn đề quan trọng:

"Huge gap" – Khoảng cách lớn giữa mô tả đặc tả bằng ngôn ngữ tự nhiên và Netlist.

→ Hiện chưa có công cụ nào hoàn toàn tự động chuyển đổi đặc tả thành netlist.

Ngôn ngữ sử dụng:

Loại ngôn ngữ	Ví dụ	Mục đích
Ngôn ngữ tự nhiên	Tiếng Anh/Tiếng Việt	Viết đặc tả, yêu cầu
Ngôn ngữ mô hình hóa	UML, C	Mô hình hóa chức năng
HDL	Verilog, VHDL	Mô tả phần cứng chính xác
Netlist		Kết nối cổng logic, cho tổng hợp



Mapping giữa các cấp độ trừu tượng

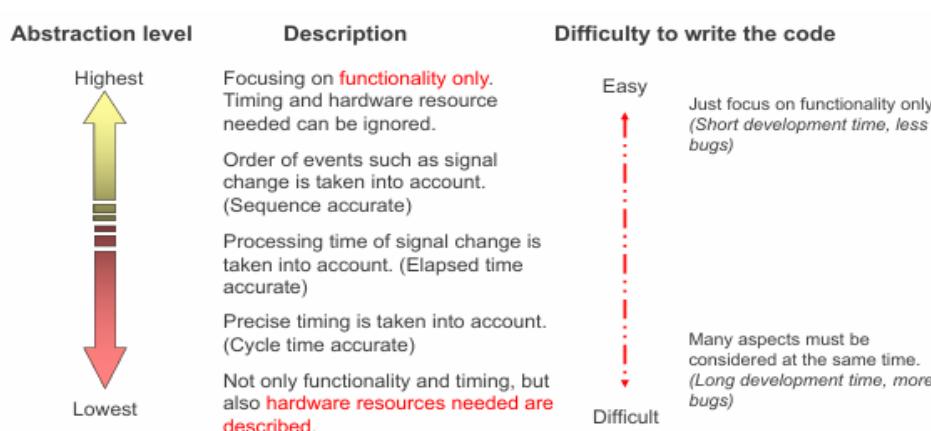
Luồng thiết kế từ cấp trừu tượng cao đến thấp:

Cấp độ	Mô tả	Model
Natural language	Tài liệu yêu cầu người dùng	L1
UML	Mô hình chức năng, kiến trúc	L2
C language	Mô tả hành vi phần mềm/hệ thống	L3
Verilog RTL	Mô tả phần cứng cụ thể	L4

Thực tế có thể bỏ qua một số bước như UML/C nếu không cần thiết.

Ghi chú thêm: Có thể tạo nhiều cấp tài liệu từ ngôn ngữ tự nhiên:

- Yêu cầu người dùng
- Đặc tả chức năng / giao tiếp
- Đặc tả thiết kế nội bộ v.v.



Các mức trừu tượng và độ khó:

Bảng mô tả theo cấp độ trừu tượng:

Cấp độ trừu tượng	Nội dung mô tả	Độ khó
Cao nhất	Chỉ mô tả chức năng, bỏ qua timing, tài nguyên phần cứng	Dễ
	Xác định thứ tự thay đổi tín hiệu (sequence accurate)	Trung bình
	Tính đến thời gian xử lý tín hiệu (elapsed time)	Trung bình-khó
	Tính đến chu kỳ đồng hồ chính xác (cycle time)	Khó
Thấp nhất	Mô tả tài nguyên phần cứng (cổng logic, mux, ff...)	Rất khó

Nhận định:

- Càng xuống cấp trừu tượng thấp, độ chính xác cao hơn → cần mô tả timing, tài nguyên → phức tạp hơn.
- Ở cấp cao nhất, tập trung vào "logic hành vi" giúp rút ngắn thời gian phát triển, ít bug.

Tóm tắt điểm chính:

- Không có công cụ nào tự động chuyển từ đặc tả sang Netlist → cần làm thủ công qua nhiều lớp mô hình.
- Luồng thiết kế cần chuyển dần từ ý tưởng → ngôn ngữ mô hình hóa → HDL → Netlist.
- Các mức trừu tượng càng thấp thì càng khó viết nhưng mang lại độ chính xác cao cho phần cứng.
- Luôn có nhiều mức tài liệu viết bằng ngôn ngữ tự nhiên → đây là bước quan trọng trong mô hình hóa.

Modeling Level Example – Ví dụ về các mức độ mô hình hóa trong thiết kế phần cứng

Bảng so sánh các mức mô hình hóa theo độ chính xác mô tả và khả năng sử dụng

Tiêu chí mô tả	Mức 1 (Transaction-level)	Mức 2 (Cycle accurate)	Mức 3 (RTL)	Mức 4 (Gate-level)
Sequence	accurate	accurate	accurate	accurate
Elapsed time	No / Yes	accurate	accurate	accurate
Machine cycle (chu kỳ)	No / Yes	Yes	Yes	Yes
Hardware resource	No	No / Yes	Yes	Yes
Synthesis	* ¹ No / * ² No	Yes	Yes	—
Simulation	Yes	Yes	Yes	Yes
Ví dụ mô hình	Transaction Level Modeling	Cycle-accurate Model	RTL model	Gate-level model
Ngôn ngữ mô tả	C, C++, SystemC	SystemC, Verilog/VHDL	Verilog/VHDL	Verilog + Netlist

Chú thích chi tiết các mức mô hình:

◆ Mức 1 – Transaction Level Modeling (TLM)

- Ngôn ngữ: C, C++, SystemC
- Không mô tả tài nguyên phần cứng, không mô tả chu kỳ chính xác → phù hợp cho phát triển phần mềm, kiểm thử sớm.
- Dùng để: mô phỏng nhanh, phát triển phần mềm hệ thống.

◆ Mức 2 – Cycle Accurate Simulation Model

- Ngôn ngữ: SystemC, Verilog, VHDL
- Mô phỏng chính xác theo chu kỳ đồng hồ.
- Tài nguyên phần cứng có thể được mô tả hoặc không.

◆ Mức 3 – RTL (Register Transfer Level) Model

- Ngôn ngữ: Verilog, VHDL
- Mô tả rõ ràng tài nguyên phần cứng, có thể tổng hợp được.
- Phổ biến nhất trong thiết kế logic hiện nay.

◆ Mức 4 – Gate-Level Model

- Ngôn ngữ: Verilog netlist
- Được tạo ra sau quá trình tổng hợp từ RTL.
- Dùng để kiểm tra timing, tiêu thụ điện năng, mô phỏng hậu tổng hợp.

Description level			Synthesis ()	Simulation ()	Example	Description language
Timing						
Sequence	Elapsed time	Machine cycle				
accurate	No	No	No	* ¹ No	Yes	transaction level modeling untimed
	Yes					timed
accurate	accurate	Yes	No	* ² No	Yes	Cycle accurate simulation model
accurate	accurate	Yes	Yes	Yes	Yes	System C Verilog VHDL
accurate	accurate	Yes	Yes	—	Yes	RTL model

💡 **Ghi chú từ bảng (1 và 2):

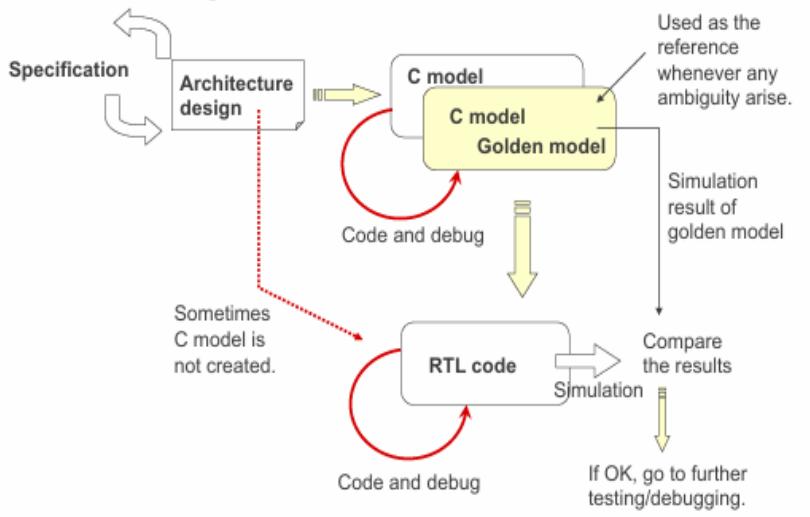
- ¹: Nếu mô hình không có tính song song và không có timing, có vài công cụ hỗ trợ tổng hợp hành vi – nhưng chưa phát triển hoàn chỉnh.
- ²: Một số công cụ tổng hợp hành vi đã tồn tại, nhưng vẫn đang trong quá trình hoàn thiện.

Tóm tắt vai trò từng mức mô hình hóa:

Mức độ	Mục tiêu chính	Ưu điểm	Nhược điểm
TLM	Mô phỏng nhanh, phát triển phần mềm	Dễ viết, tốc độ cao	Thiếu chính xác phần cứng
Cycle	Mô phỏng chính xác theo chu kỳ	Mô tả thời gian tốt	Chưa mô tả đầy đủ phần cứng
RTL	Thiết kế phần cứng chi tiết	Tổng hợp được, mô tả rõ ràng	Phức tạp hơn
Gate	Kiểm tra timing, điện, layout	Chính xác, phù hợp hậu tổng hợp	Khó viết tay, thường tự động

Design Process Using Models – Quy trình thiết kế sử dụng mô hình

Design Process Using Models



Mục tiêu chính của quy trình: Tăng độ chính xác, hiệu quả kiểm thử, và giảm lỗi thiết kế bằng cách sử dụng các mô hình phần mềm (C model/golden model) trước khi triển khai RTL code.

Quy trình chi tiết:

- Specification (Đặc tả yêu cầu):** Đầu vào ban đầu mô tả chức năng hệ thống cần thiết.
- Architecture Design (Thiết kế kiến trúc):**
 - Phân tích đặc tả để tạo ra kiến trúc tổng thể của hệ thống.
 - Từ đây, có 2 nhánh đi song song:
 - Sinh ra C model (mô hình phần mềm)
 - Hoặc đi thẳng đến RTL code (nếu bỏ qua bước mô hình phần mềm)

C Model (Golden Model):

- Còn gọi là mô hình chuẩn (golden model) – dùng để kiểm thử tính đúng đắn.
- Code and debug:
 - C model được viết và kiểm thử độc lập.
 - Dùng để xác minh logic và chức năng trước khi vào RTL.
- Ưu điểm:
 - Dễ viết, dễ kiểm thử.
 - Tốc độ mô phỏng cao.
 - Dùng làm chuẩn để so sánh kết quả với RTL.

RTL Code (Verilog/VHDL):

- Viết mô tả phần cứng chi tiết.
- Code and debug:
 - Mô phỏng RTL và so sánh kết quả với C model.
 - Nếu kết quả giống nhau → tiếp tục debug thêm, hoặc chuyển sang kiểm thử sâu hơn.

Tóm tắt vai trò C Model:

- Mô phỏng chức năng:** Đảm bảo logic hoạt động đúng trước khi viết RTL
- So sánh với RTL:** Phát hiện lỗi RTL bằng cách đối chiếu output
- Golden Model:** Là cơ sở đối chiếu trong mọi bước xác minh.

Một số lưu ý từ slide:

- “Sometimes C model is not created”: Có thể bỏ qua bước C model, nhưng điều này giảm khả năng debug sớm và làm khó xác minh chức năng.
- C model được dùng làm chuẩn (reference): Trong trường hợp có bất kỳ sự mơ hồ nào, người thiết kế sẽ tra lại kết quả từ golden model để xác định đúng/sai.

3.4. RTL Design and Verification.

Objective of Logic Design:

Mục tiêu cuối cùng của thiết kế logic là: Tạo ra photo masks phù hợp với đặc tả chức năng, để chế tạo chip (fabrication).

2 yêu cầu bắt buộc của RTL Code:

- Simulatable:** Mã phải được EDA tool mô phỏng được để xác minh chức năng
- Synthesizable:** Mã phải cho phép EDA tool tạo được netlist để tổng hợp và chế tạo
- Chúng ta phải viết mã RTL để vừa **mô phỏng** được (kiểm thử), vừa **tổng hợp** được (triển khai thật).

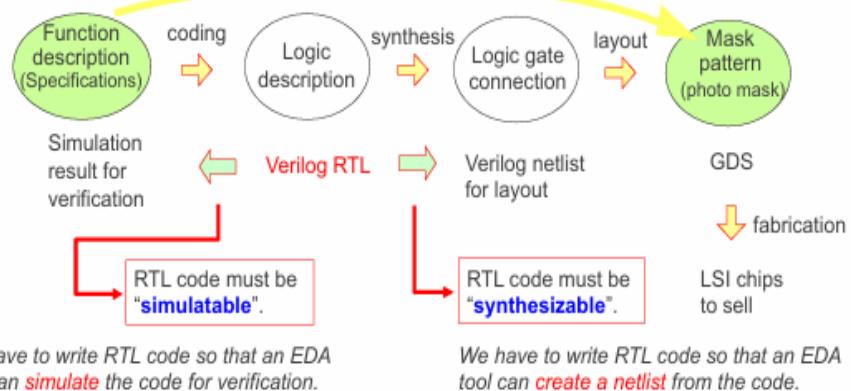
Quy trình từ đặc tả đến chip vật lý:

1. **Function Description (Đặc tả chức năng)**
 - o Bắt đầu từ mô tả chức năng hệ thống (specification).
2. **Logic Description (Mô tả logic)**
 - o Viết RTL code (Verilog RTL).
3. **Synthesis (Tổng hợp)**
 - o Biến RTL thành netlist cổng logic (Verilog netlist).
4. **Layout (Bố trí vật lý)**
 - o Dựa trên netlist để tạo bố cục mặt nạ (Mask pattern / GDS file).
5. **Fabrication (Chế tạo)**
 - o Dùng mask để sản xuất LSI chip thực tế.

Objective of Logic Design

Objective of logic design

= Get photo masks equivalent to the specifications, for fabrication purpose.



Tổng hợp: Design Models

1. Lý do cần chọn cấp độ trừu tượng (Abstraction level):

- Phụ thuộc vào **mục đích mô hình hóa (modeling target logic)**:
 - o Mô phỏng chức năng? → Dùng C code.
 - o Chuẩn bị tổng hợp thành phần cứng? → Dùng Verilog RTL với cú pháp phù hợp.

2. Phân loại theo cấp độ trừu tượng (Abstraction level):

Cấp độ trừu tượng	Ngôn ngữ phù hợp	Đặc điểm chính
Cao (high)	C	Untimed, simulatable
Thấp (low)	Verilog RTL	Cycle-accurate, synthesizable

Design Models

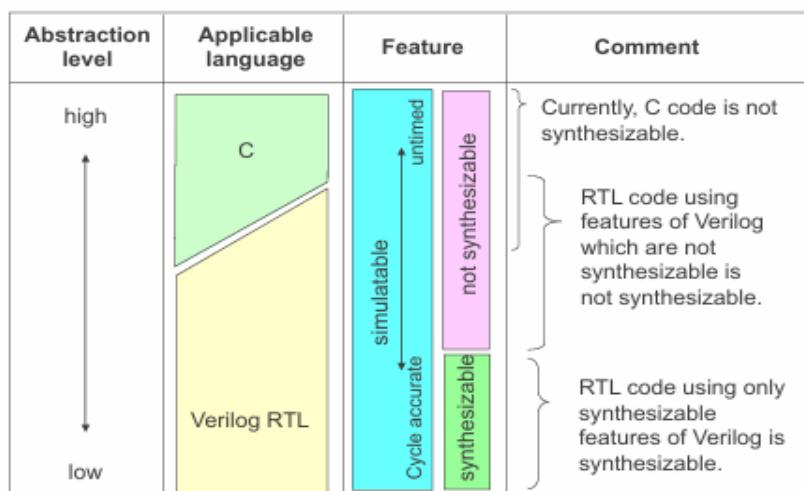
3. Các đặc tính của mô hình (Feature):

Feature	Ý nghĩa
Untimed	Không quan tâm đến thời gian thực → phù hợp cho mô phỏng chức năng
Simulatable	Có thể mô phỏng để kiểm tra logic
Cycle Accurate	Mô phỏng chính xác theo từng chu kỳ xung clock
Synthesizable	Có thể tổng hợp để tạo netlist cho sản xuất thực tế

We must select a suitable abstraction level depending on the purpose of modeling the target logic.

4. Các ghi chú quan trọng (Comment):

- **C code:**
 - o Dùng cho mô hình cấp cao (high-level model)
 - o ✗ Không thể tổng hợp → chỉ dùng để kiểm tra logic (Golden Model)
- **Verilog RTL:**
 - o Có thể dùng các cú pháp không tổng hợp được (not synthesizable) → cần cẩn trọng.
 - o Chỉ những cú pháp RTL **chuẩn và tổng hợp được** mới dùng để đưa vào chế tạo chip.



Verilog RTL Design

- RTL (Register Transfer Level): Là cấp độ mô tả luồng dữ liệu giữa các thanh ghi (registers) và cách logic xử lý tín hiệu giữa chúng.
- Trong thiết kế front-end, RTL Verilog mô tả:
 - o Dữ liệu đi vào và ra khỏi các thanh ghi.
 - o Logic xử lý giữa các khối đăng ký.
- Không cần mô tả chi tiết các cổng logic hoặc mạch cụ thể.

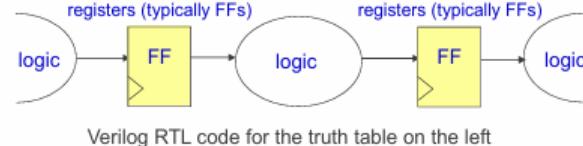
Ưu điểm của thiết kế RTL cho frontend designer:

- Frontend designers **không cần quan tâm** đến:
- Kết hợp loại cổng logic nào.
 - Mạch điện thực tế để hiện thực hóa logic đó.
 - ➡ Chỉ tập trung vào: **logic chức năng & hành vi mong muốn.**

Verilog RTL Design (1/2)

There are several levels of description in Verilog language. For front end design, we generally use **Register Transfer Level (RTL Level)**, which can describe how signals go into registers, memory elements, as shown beside.

Renesas mainly uses Verilog as a high level hardware description language



Verilog RTL code for the truth table on the left

Truth table				
a[3]	a[2]	a[1]	a[0]	y
1	x	x	x	q1
0	1	x	x	q2
0	0	x	0	q3
0	0	0	1	q4
0	0	1	1	q5



```
casez ( a[3:0] )  
 4'b1??? : y = q1 ;  
 4'b01?? : y = q2 ;  
 4'b00?0 : y = q3 ;  
 4'b0001: y = q4 ;  
 4'b0011: y = q5 ;  
 default : y = 4'bxxxx ;  
endcase  
⋮
```

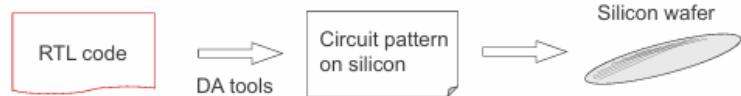
Verilog RTL Design (2/2)

When writing RTL code, frontend designers do not have to concern on:

how to combine what kind of gates, and the circuits of the gates which are necessary to realize the logic in the code.

Therefore, frontend designers can be free from the physical design and focus on the logic itself.

The mapping between RTL code and the circuit on the silicon is done by backend designers with a help of design automation tools.



RTL Programming Summary

(TƯỚI ÔN)

Nguồn gốc của Bug trong RTL Verification

Mục tiêu chính của RTL Verification: Đảm bảo rằng **RTL code** (Register Transfer Level) hoạt động đúng như **kỳ vọng của khách hàng** và **đặc tả thiết kế**.

Nguồn gốc lỗi (Bugs) đến từ đâu?

1. Customer's Expectation (Kỳ vọng khách hàng):

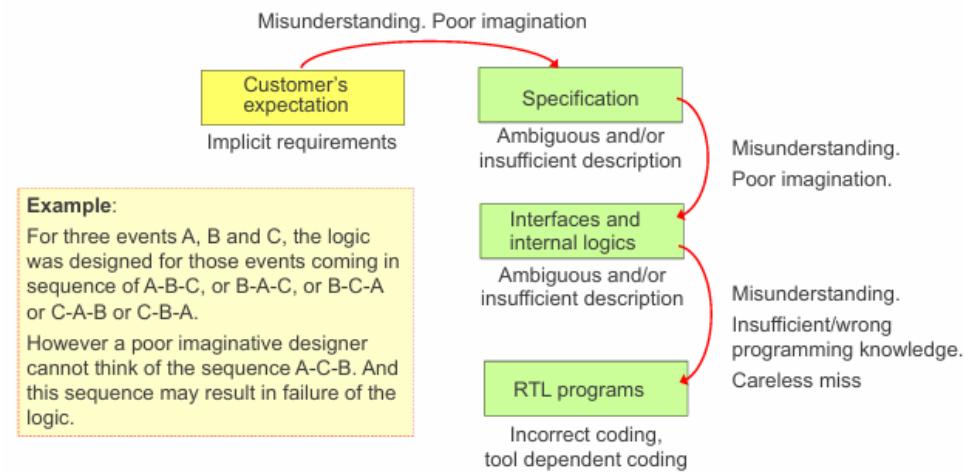
- Bao gồm **những yêu cầu ngầm hiểu** (implicit requirements).
- Không phải lúc nào cũng được viết rõ ràng trong đặc tả.

Gây ra: **Hiểu sai**, thiếu tưởng tượng từ phía kỹ sư thiết kế.

2. Specification (Đặc tả thiết kế):

- **Mơ hồ** hoặc **thiếu chi tiết**.
- Dễ dẫn đến hiểu sai, hoặc tưởng tượng không đầy đủ về logic hệ thống.
→ Dẫn tới bug do sai phạm trong logic tổng thể.

RTL Verification – Where Bugs come from?



3. Interfaces and Internal Logics (Giao tiếp & logic bên trong):

- Mô tả không đầy đủ / không rõ ràng.
- Lỗi đến từ việc kết nối các khối logic / giao diện với nhau không chính xác.
Sai do: **hiểu sai**, **thiếu kiến thức**, hoặc **hiểu nhầm về ngữ nghĩa interface**.

4. RTL Programs:

- **Code sai**, phụ thuộc vào công cụ, thiếu kiểm tra chéo.
 - Ví dụ:
 - Viết nhầm thứ tự dòng.
 - Dùng cấu trúc không tổng hợp được (unsynthesizable).
 - Lỗi ngữ nghĩa trong cú pháp Verilog/VHDL.
- Gây ra lỗi do:
- Kiến thức lập trình chưa đủ.
 - **Cấu thả khi lập trình**.

Ví dụ minh họa thực tế:

Với ba sự kiện A, B, C — logic thiết kế giả định các chuỗi A-B-C, B-A-C, B-C-A...

Nhưng **không xét tới chuỗi A-C-B** — dẫn đến **failure** khi tình huống đó xảy ra.

Gốc rễ: Thiếu tưởng tượng, không bao quát tất cả các khả năng.

Bài học rút ra:

Vấn đề	Hậu quả
Yêu cầu ngầm hiểu	Không được thể hiện trong đặc tả
Đặc tả không rõ ràng	Dễ dẫn tới hiểu sai
Kỹ sư không lường hết case	Dễ bỏ sót trường hợp gây lỗi logic
Lập trình cấu thả	Code dễ dính bug khi mô phỏng hoặc tổng hợp

RTL Verification – Desktop Checking (1/3)

"Kiểm tra thủ công (desktop checking) vẫn rất hiệu quả để phát hiện bug trong logic."

"On a desk by hand (eyes)":

- Đây là hành động bạn **ngồi nhìn, đọc kỹ mã RTL**, vẽ sơ đồ hoặc tưởng tượng các trường hợp khác nhau có thể xảy ra.
- Giống như debug bằng **trí tưởng tượng** và kiểm tra logic thủ công **trước khi chạy mô phỏng**.

Vì sao vẫn cần desktop check?

- Dù có **RTG (Random Test Generator)** hỗ trợ tạo dữ liệu test, nhưng:
 - RTG chỉ phát hiện lỗi với những tình huống **nó có thể tưởng tượng ra**.
 - RTG không thay thế được tư duy con người**, nhất là trong các trường hợp ranh giới, điều kiện bất thường.

Cảnh báo:

"Logic được thiết kế bởi kỹ sư **thiếu trí tưởng tượng** thường có rất nhiều lỗi."

RTL Verification – Desktop Checking (2/3)

"**Kỹ sư thường chỉ code theo trình tự mình nghĩ đến. Nhưng giả định đó có thể sai khi vào thực tế.**"

Tình huống minh họa:

- Bạn **code logic** dựa theo một trình tự giả định:
 - Signal A đến → thực hiện action.
 - Rồi signal B đến → làm bước tiếp theo.
- Trong thực tế:
 - Có thể **B đến trễ**, hoặc **B đến sớm hơn**, hoặc môi trường thay đổi khiến **giả định ban đầu không còn đúng**.

Thông điệp cốt lõi:

- Luôn luôn kiểm tra lại các giả định cơ bản của bạn!**
- Khi bạn viết code dựa trên giả định "sau A thì B", hãy **tự hỏi**:
 - Nếu **B đến trễ** thì sao?
 - Nếu **B đến trước** thì sao?
 - Nếu **B không bao giờ đến** thì sao?

Đây chính là **desktop checking**: tưởng tượng ra tất cả các khả năng và chắc chắn rằng logic vẫn ổn.

RTL Verification – Desktop Checking (3/3)

"**Những điều tưởng như chắc chắn trong thiết kế thường không đúng trong thực tế.**"

Điều cần kiểm tra	Ý nghĩa	Ví dụ
X Hành vi không xảy ra như bạn hy vọng	Bạn nghĩ tín hiệu sẽ đến vào thời điểm cụ thể, nhưng không	Bạn mong ack đến sau req, nhưng nó bị delay
X Sự kiện xảy ra không đúng thứ tự bạn nghĩ	Bạn nghĩ A sẽ đến trước B, nhưng B đến trước	Tín hiệu reset đến sau khi enable đã bật
X Giả định không được hiện thực hóa	Nguồn, clock, reset, init có thể chưa sẵn sàng	Clock chết Block chưa cấp nguồn Một module chưa khởi tạo
X Giá trị cần thiết chưa được định nghĩa	Giá trị chưa được gán, bị ngẫu nhiên, hoặc không ổn định	Register chưa reset Giá trị input là X (undefined)

RTL Verification – Desktop Checking (1/3)

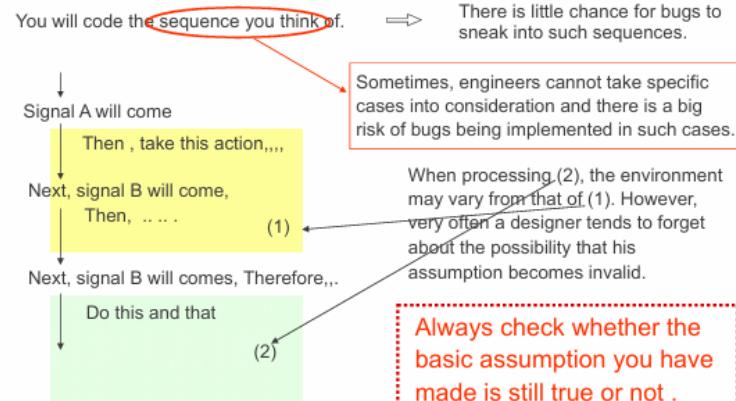
Checking a logic **on a desk by "hand" (eyes)** is still an effective way to verify logics if it is done in a way that every assumption a designer made is checked whether they are really true or not.

You must do desktop check to make sure that you really understand the issues in your design.

Sometimes RTG (Random Test Generator) can checkout bugs by providing test data which a designer may never imagine. However, this does not mean RTG can replace desktop checking.

→ Logics designed by an engineer with poor imagination have a lot of bugs.

RTL Verification – Desktop Checking (2/3)



Especially, following points must be checked all the time during the design activities.

What happens might **not happen in the way you hope**.

(The input which you expect may not come when you think it will come).

Events may happen in a **different sequence to that you expected**.

(Event A will come before Event B, when you assume B will come before A).

Or Event A and B happen at the same time, when you expect them to come in a sequence).

A presupposed condition **may not realized**.

Some blocks may have no power when you think all the blocks are powered up.

Some blocks may not be initialized, when you think all the blocks are initialized.

A clock signal is dead, when you think the clock is always alive.

Some voltage may be supplied, even after the power is off.

A value which must be fixed, **may not be defined yet**.

(Not initialized, Not given a predetermined value, Unstable, . . .)

RTL Verification – RTL Checker (Style Check)

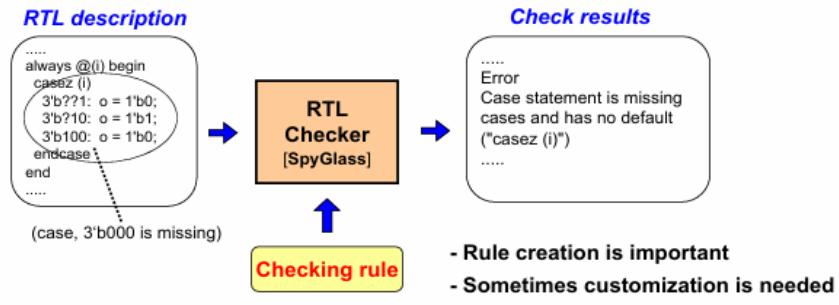
Mục tiêu chính: Phát hiện các lỗi về **quy tắc đặt tên, cách viết mã RTL, và coding style** có thể gây lỗi hoặc không nên sử dụng.

RTL Verification – RTL Checker (Style Check)

Checkout naming and coding style which are prohibited or not recommended.

Vấn đề ở đây:

- Dòng 3'b000 bị thiếu (giá trị này không được xử lý).
- Không có **default** trong casez, nghĩa là nếu giá trị đầu vào không khớp với bất kỳ case nào, **o sẽ không được định nghĩa**, dẫn đến **lỗi ngẫu nhiên (X-state propagation)** trong mô phỏng hoặc tổng hợp.



RTL Checker (giữa)

- Đây là công cụ tự động kiểm tra coding style. Ở đây là **SpyGlass**, nhưng có thể là các tool khác như:
 - VeriLint
 - AscentLint (Real Intent)
 - HDL Designer
 - hoặc các plugin của EDA toolchain (Synopsys, Cadence...)

Chức năng chính:

- Tự động phát hiện:
 - Thiếu default trong case/casez
 - Case không bao phủ đủ input
 - Viết lại tín hiệu nhiều lần
 - Đặt tên sai quy chuẩn
 - Sử dụng các construct nguy hiểm trong Verilog (ví dụ: latch ẩn)

Checking rule = các quy tắc kiểm tra (style rules)

- Bạn có thể dùng rule mặc định của công cụ
- Nhưng nên **tùy chỉnh (customize)** để phù hợp với dự án hoặc quy chuẩn của

Static Functional Verification

Mục đích của Static Functional Verification

"Given a set of properties, check if the design always satisfies them."

- Properties:** Các yêu cầu logic được định nghĩa rõ ràng, ví dụ như:
 - A xảy ra thì B phải xảy ra trong vòng 2 chu kỳ clock.
 - FIFO không bao giờ **overflow** hoặc **underflow**.
 - Dữ liệu đọc ra bằng dữ liệu viết vào sau đúng thời gian trễ.
- Không cần **input test vectors**, không cần viết testbench → không giống như mô phỏng truyền thống (simulation).

Kỹ thuật formal verification hoạt động thế nào?

- Dựa vào các **thuật toán logic toán học (model checking)** để chứng minh rằng:
 - Một điều kiện/thuộc tính (property) **luôn đúng (hold)** trong tất cả trường hợp có thể.
 - Nếu không đúng → trả về **counterexample** (tức là một chuỗi trạng thái chứng minh điều kiện bị vi phạm).

Nếu đúng → bạn hoàn toàn yên tâm.

Nếu sai → nhận được một chuỗi các tín hiệu minh chứng rằng logic **có bug**.

Những gì được kiểm tra?

- Các **thuộc tính logic (properties)** do designer định nghĩa.
 - Dạng phổ biến nhất là **assertion-based verification (ABV)**.
 - Sử dụng ngôn ngữ như SystemVerilog Assertions (SVA).

Nếu một thuộc tính sai:

- Công cụ sẽ trả về **counterexample trace** để bạn xem lại các tín hiệu → giúp sửa lỗi.

Khi nào nên dùng static functional verification?

Đặc biệt hiệu quả cho:

- **ECC checker/corrector:** quá nhiều khả năng kết hợp input bit.
- **FIFO control logic:** cần kiểm tra các trường hợp corner-case như full/empty/overflow.
- **Protocol compliance checking:** kiểm tra nếu giao thức luôn đúng, ví dụ handshake.

RTL Verification – Static Functional Verification

Given a set of properties that describe the behavior of the design, static functional verification tools can prove proper functionality of the design using formal verification techniques.

What are checked out?

Properties given by a designer hold true or not.
If not true, it gives a counterexample.

Available tools

Solidify (Averant), 0-In (Mentor)

Although static functional verification cannot handle large size logic, it is suitable for testing logics which need huge combination of input patterns such as ECC, pattern matching logics, or FIFO, etc.

Static verification does not do logic simulation, therefore no test input is needed. It checks if properties given are true or not.

Giải thích

Các thuật toán formal verification rất nặng và không scale tốt

Nếu property không chính xác → kết quả kiểm tra không có ý nghĩa

Không dễ như viết testbench cơ bản

RTL Verification – Assertion Based Verification

Assertion là gì?

- Assertion là một câu lệnh kiểm tra các điều kiện phải đúng (must) hoặc không được đúng (must not) trong hệ thống RTL.
- Được viết bằng ngôn ngữ chuyên dụng như:
 - PSL (Property Specification Language)
 - SVA (SystemVerilog Assertions – phổ biến hơn hiện nay)

RTL Verification – Assertion Based Verification

Define properties of signals in target modules using a language such as PSL (Property Specification Language)

Run simulation on a simulator supporting assertion, such as NC-Verilog (LDV5.0 or later)

When violation detected, error report is given.

"Must" and "must not" can be declared.

Signal A must rise 5 clock cycles after signal B falls.
Signal A and B must not be 1 at the same time.

Especially useful for checking improper reuse of modules.

Expected to improve reusability or reduce troubles using IPs.

Lợi ích lớn nhất của ABV

- Phát hiện **sai phạm nhỏ và tinh vi** mà mắt thường hoặc testbench thường khó thấy.
- Giúp **documentation hóa rõ ràng** các yêu cầu logic.
- Dễ dàng **reuse IP** khi tích hợp nhiều lần ở các môi trường khác nhau.

Tác dụng chính của Assertion-Based Verification

Tác dụng

Giải thích

✓ Giúp phát hiện bug sớm

Vì phạm logic sẽ bị bắt ngay

✓ Không cần viết quá nhiều testbench

Vì assertion tự động kiểm tra trong mô phỏng

✓ Tăng khả năng tái sử dụng (reuse) IP

Đảm bảo logic đúng ở bất kỳ môi trường tích hợp nào

✓ Tối ưu khi dùng IP (IP Integration)

Cảnh báo nếu IP dùng sai cách hoặc sai thứ tự tín hiệu

Ví dụ về các assertion

Điều kiện kiểm tra

Viết bằng lời

A phải xuất hiện sau B rớt

Signal A must rise 5 clock cycles after signal B falls

A và B không được cùng lúc là 1

Signal A and B must not be 1 at the same time

Một FSM không được vào trạng thái sai

Certain states must not be entered

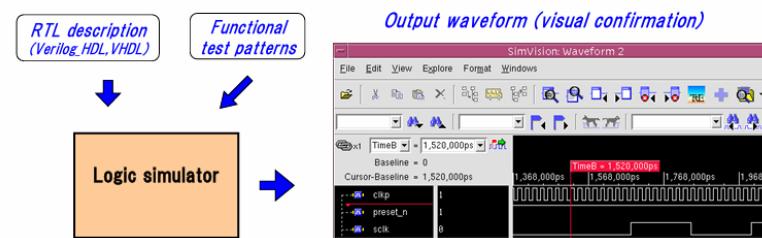
Functional Logic Simulation (1/4)

Mục tiêu: Kiểm tra xem mô tả RTL có hoạt động như mong muốn không bằng cách mô phỏng đầu ra dựa trên các mẫu kiểm tra chức năng (Functional Test Patterns).

Thành phần chính:

- RTL description: Mô tả phần cứng bằng Verilog hoặc VHDL.
- Functional test patterns: Bộ tín hiệu đầu vào được thiết kế để kích hoạt các chức năng cụ thể của mạch.
- Logic simulator: Mô phỏng phản hồi logic từ mô tả RTL và tạo ra output waveform để xác nhận trực quan.

To check expected simulation output by applying functional test patterns to RTL description

**Functional Logic Simulation (2/4)****Nhận định quan trọng:**

- **RTL simulation rất mất thời gian.**
 - Cần chuẩn bị dữ liệu kiểm tra một cách cẩn thận để nhắm vào các **trường hợp ngóc ngách (corner cases)** hoặc **tình huống quan trọng (critical cases)**.

Giải pháp để rút ngắn thời gian:

- (a) Sử dụng **RTG (Random Test Generator)** để giảm công sức viết tay test pattern.
- (b) Dùng **FPGA hoặc Emulator** cho mô phỏng sớm → tăng tốc độ thực thi mô phỏng hàng trăm lần.

RTL Verification – Functional Logic Simulation (2/4)

By using RTL simulator, we can see if the logic works as intended by checking the outputs.

RTL simulation takes time. Therefore, to make it time efficient, test data must be carefully prepared to test critical or corner cases.

There are several ways to help RTL simulation time short.

- (a) To reduce test data preparation time: Random Test Generator (RTG)
- (b) To reduce simulation time: Early prototyping with FPGA or emulator

Functional Logic Simulation (3/4)**Vấn đề:**

- RTG tạo ra quá nhiều mẫu dữ liệu → **khó kiểm tra kết quả bằng mắt**.

Giải pháp:

- Dùng **Golden Model**:
- Là một mô hình tham chiếu (thường là **C Model**) được tin tưởng là đúng.
- So sánh đầu ra của RTL với đầu ra của Golden Model để xác định đúng/sai.

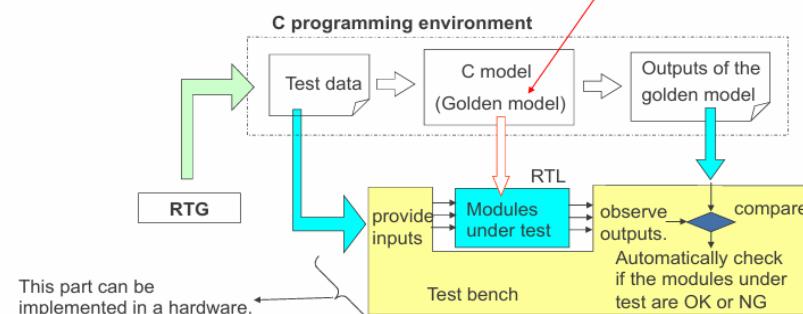
Quy trình:

- RTG tạo **Test data**.
- C Model xử lý test data để sinh **expected output**.
- RTL được mô phỏng với cùng test data.
- So sánh kết quả RTL và Golden Model bằng **Testbench**.

RTL Verification – Functional Logic Simulation (3/4)

RTG creates so many data patterns, therefore it is not feasible to see the result and check if it is OK or not by hand. Therefore, usually we use a **golden model** as a reference and compare the result as shown below.

A model which is believed to be correct and can be used as a reference for RTL code is called the golden model.

**Functional Logic Simulation (4/4)****RTL Verification – Functional Logic Simulation (4/4)****Nâng cấp quy trình:**

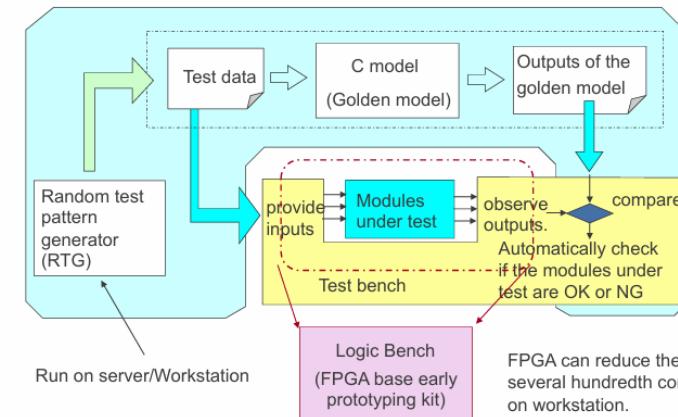
- Thêm **Logic Bench (FPGA Base Early Prototyping Kit)** → chạy mô hình thực để kiểm tra kết quả nhanh chóng.

Quy trình mở rộng:

- RTG chạy trên server hoặc workstation → tạo Test data.
- C Model sinh kết quả mong đợi.
- **Modules under test** nhận input và sinh output trong môi trường Testbench.
- Kết quả được **so sánh tự động** với Golden Model.

Ưu điểm:

- **FPGA** tăng tốc thời gian kiểm tra mô phỏng gấp **hàng trăm lần** so với simulator truyền thống.



RTL Verification – Test Bench vs Target Code

Mục tiêu chính:

Xác nhận rằng việc mô phỏng cho kết quả đúng không có nghĩa là mã RTL đúng – vì Test Bench có thể sai và che giấu lỗi trong mã đích.

Bảng so sánh – 4 trường hợp kết hợp:

Target Code	Test Bench	Kết quả	
✓ Good	✗ Wrong	Mã đúng bị từ chối (False Negative)	✗
✗ Wrong	✗ Wrong	Mã sai được chấp nhận (False Positive) → bug bị bỏ sót	✗
✗ Wrong	✓ Good	Mã sai bị phát hiện và loại	✓
✓ Good	✓ Good	Mã đúng được xác nhận	✓

Điểm nhấn quan trọng:

Cảnh báo: "Do not think: My code is correct, because the simulation result is good." → Đừng nghĩ mã đúng chỉ vì mô phỏng cho ra kết quả như mong đợi!

Lý do: Test Bench sai lầm có thể gây hậu quả nghiêm trọng hơn mã sai. Vì test bench không kiểm tra đúng chức năng, bug sẽ bị che giấu → phần cứng sản xuất ra bị lỗi mà không hề biết.

Kết luận then chốt: Test bench quan trọng hơn nhiều so với target code trong quá trình kiểm chứng sản phẩm.

Highlight ở dưới cùng: "Test bench is much more vital for our products to assure that our products are bug free."

RTL Verification – Test Data

Nhấn mạnh sự quan trọng của việc thiết kế, lựa chọn dữ liệu kiểm thử (test data) trong quá trình lập trình và xác minh RTL

Lập trình RTL cần chuẩn bị sẵn cho các kiểu dữ liệu đầu vào khác nhau

- Bạn cần tưởng tượng ra **các kiểu dữ liệu có thể xảy ra**, và viết code sao cho xử lý đúng với tất cả chúng.
- Nếu bạn **không lường trước dữ liệu đầu vào bất ngờ**, mã RTL **sẽ hoạt động sai** với những dữ liệu đó.

Câu nhấn mạnh cực kỳ quan trọng: "Your code will not work properly for data which you did not expect to come."

Tư duy tưởng tượng là chìa khóa: "Your imaginative power decides the quality of your program."

Kỹ sư cần chọn dữ liệu kiểm thử sao cho bao phủ hết tất cả các đường đi (paths) có thể của mã RTL.

➡ Nghĩa là phải kiểm thử **đầy đủ** các điều kiện và khả năng xảy ra trong logic thiết kế.

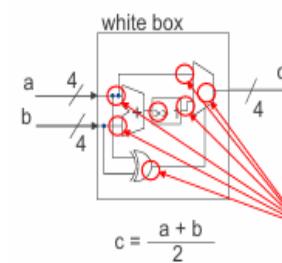
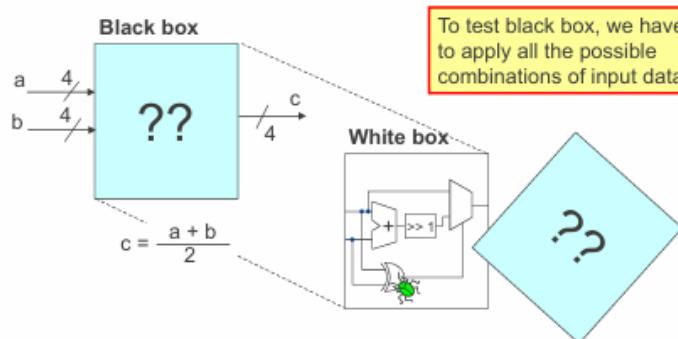
Các loại dữ liệu kiểm thử cần thiết:

Loại dữ liệu	Giải thích
Typical cases	Dữ liệu được sử dụng phổ biến, phản ánh hoạt động thông thường.
All possible state changes	Dữ liệu gây ra chuyển trạng thái , giúp kiểm tra tính chính xác khi FSM (finite state machine) hoặc logic thay đổi.
Corner cases	Dữ liệu biên, đặc biệt , thường dễ gây lỗi nếu không xử lý đúng. Ví dụ: reset, overflow, underflow...

RTL Verification – Black Box vs White Box Test

{ Black box test : Test **without** knowledge of internal structure and logic.

White box test : Test **with** knowledge of internal structure and logic.



To test white box, we do not have to apply all the possible combinations of input data. We can apply selective input data to check specific part of the design.

If the target is white box, we can create selective input data to activate specific part of the logic.

And once checked with some data, it may not have to be checked by using all the possible values. **Typical and corner case data may be enough to be applied.**

Slide 1/2 – Định nghĩa và sự khác biệt cơ bản:

Black Box Test

- Không biết cấu trúc hoặc logic bên trong.
- Chỉ kiểm tra **đầu vào và đầu ra**.
- Để đảm bảo toàn diện, ta **phải thử tất cả các tổ hợp dữ liệu đầu vào**.
- → Cách này **tốn thời gian**, nhưng hữu ích khi không có quyền truy cập nội bộ hệ thống (ví dụ: kiểm thử độc lập hoặc bởi bên thứ ba).

Ví dụ: Nếu $c = (a + b)/2$ mà ta không biết công thức này, ta sẽ test mọi cặp (a, b) có thể (ví dụ: 4-bit \rightarrow 256 tổ hợp).

White Box Test

- **Biết rõ cấu trúc và logic nội bộ.**
- Có thể hiểu cách hoạt động bên trong và từ đó tạo ra test case **một cách có chọn lọc**.
- Không cần thử hết các tổ hợp, chỉ cần các trường hợp **đại diện và đặc biệt** là đủ.

Insight: “To test white box, we can apply selective input data to check specific part of the design.”

Slide 2/2 – Ứng dụng White Box hiệu quả hơn:

- **White Box cho phép bạn tạo test case tập trung** để kích hoạt những phần logic cụ thể bên trong mạch.
- Sau khi được kiểm tra bằng các giá trị chọn lọc, có thể **không cần thử hết toàn bộ đầu vào**.

Các loại test data đủ dùng:

- **Typical case** (thường gặp)
- **Corner case** (biên, dễ lỗi) → Thường là **đủ để phát hiện lỗi**, không cần brute-force toàn bộ input.

Kết luận tổng hợp:

Tiêu chí	Black Box Test	White Box Test
Kiến thức về thiết kế	X Không biết	✓ Biết rõ
Dữ liệu đầu vào	Toàn bộ tổ hợp (exhaustive test)	Chọn lọc thông minh (selective)
Tốn thời gian	Nhiều	Ít hơn
Phạm vi kiểm thử	Bè ngoài (input \rightarrow output)	Sâu bên trong (cấu trúc logic)
Khả năng bao phủ lỗi	Tốt nếu đủ test case	Tốt nếu hiểu rõ logic

Lưu ý quan trọng:

“Typical and corner case data may be enough to be applied” (trong white box).

→ Đây là cách để **tối ưu hóa thời gian kiểm thử** mà vẫn hiệu quả.

3.5. Logic Synthesis and Cell-based Design.

Logic Synthesis là gì? Logic Synthesis là quá trình **chuyển mã RTL (Verilog)** thành một **mạng lưới cổng logic (Gate Netlist)** mà sau này có thể được chuyển sang **mặt nạ quang học (Photo mask)** để **in lên silicon wafer**, tạo ra chip thật.

Quy trình tổng quát:

1. RTL Source Code (Verilog)

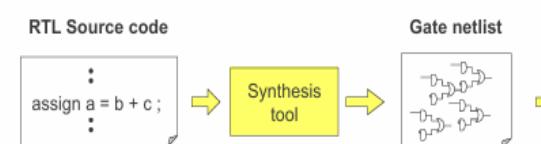
Ví dụ: assign a = b + c;
→ Đây là code cấp cao mô tả hành vi của mạch số.

Logic Synthesis (1/3)

A synthesis tool uses a Verilog RTL source program as an input and generates a gate netlist which can be directly mapped to circuits on a silicon wafer.

2. Synthesis Tool

→ Là công cụ như *Design Compiler*, Vivado, Genus,...
→ Nó dịch code RTL thành tập hợp các **cổng logic cơ bản** (AND, OR, NOT, FF,...).



3. Gate Netlist

→ Mô tả rõ ràng từng cổng logic và kết nối giữa chúng.
→ Là đầu vào cho bước "Place & Route" sau này.

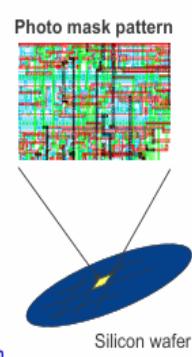
The tool will generate minimum set of logic gate blocks and will not generate gates which are not specified, explicitly or implicitly, in the RTL source code.

This means that a designer has to write all the logic blocks necessary to make them work properly themselves.

4. Photo Mask Pattern → Silicon Wafer

→ Thiết kế cuối cùng được chuyển thành mặt nạ dùng trong quy trình chế tạo chip (lithography).

This is very much different from a case of software application program.



Silicon wafer

Ý nghĩa quan trọng:

Cần khai báo đầy đủ logic!

“The tool **will not generate gates** which are not specified, explicitly or implicitly.”

→ Điều này nghĩa là:

- Bạn **phải mô tả tất cả** các khối logic cần thiết trong code RTL.
- Nếu bạn **quên khai báo** hoặc **mô tả sai**, synthesis tool sẽ **không tự động thêm hoặc sửa giúp bạn** (khác với phần mềm ứng dụng nơi thư viện runtime hỗ trợ nhiều logic sẵn).

So sánh với phần mềm truyền thống:

- Viết phần mềm → có thể phụ thuộc vào thư viện/hàm hệ thống.
- Viết RTL → **mọi thứ phải được mô tả tường minh** nếu bạn muốn nó tồn tại trong chip!

Kết luận:

Đặc điểm

Ý nghĩa

RTL chỉ là mô tả hành vi Phải được tổng hợp (synthesis) để tạo ra mạch logic thật

Synthesis không thêm gì ngoài yêu cầu Logic nào bạn không viết, thì sẽ **không có** trong netlist

Mạch phần cứng khác với phần mềm Vì **không có runtime environment** giúp xử lý logic còn thiếu

Logic Synthesis (2/3) – So sánh C Code vs RTL Code

Trường hợp phần mềm (Software):

C source code: $a = b + c \rightarrow$ Được biên dịch bởi

Compiler thành các lệnh: load b, add c, store a

Những lệnh này chỉ là chỉ thị cho CPU thực hiện,

bản thân chúng không làm được gì nếu không có

CPU. Bạn phải chạy trên hệ thống có CPU + RAM, thì chương trình mới hoạt động.

Trường hợp phần cứng (Hardware - RTL): RTL code

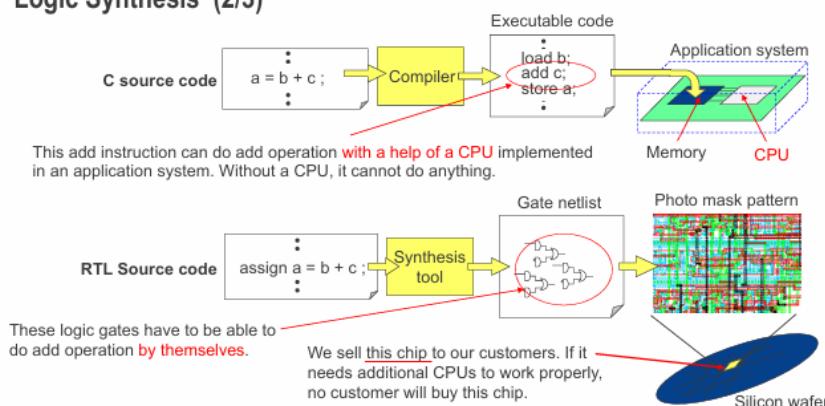
(Verilog): assign a = b + c; → Được Tổng hợp bởi

Synthesis Tool thành **mạch cộng thực tế**: Bao gồm

các cổng AND, OR, XOR, FA (Full Adder)...

Mạch cộng này sẽ được chế tạo thật trên silicon wafer → Nó **tự thực hiện phép cộng**, không cần đến CPU nào hỗ trợ.

Logic Synthesis (2/3)



So sánh cực kỳ quan trọng:

Đặc điểm	C Code (Software)	RTL Code (Hardware)
Ai thực hiện phép tính?	CPU trong hệ thống ứng dụng	Chính mạch logic được tạo ra từ RTL
Phụ thuộc phần cứng khác?	Cần có CPU mới chạy được	Không cần CPU – mọi logic phải tự đủ để hoạt động
Có thể bán độc lập không?	Không (cần hệ thống chạy nó)	Có – bán như là chip độc lập , hoạt động standalone

Ý nghĩa thương mại: “Nếu chip của bạn **phụ thuộc CPU để chạy**, khách hàng sẽ **không bao giờ mua!**”

→ Bởi vì khách hàng kỳ vọng khi **mua một chip**, nó sẽ **tự hoạt động** được – không yêu cầu phần cứng phụ trợ.

Kết luận:

- Viết phần mềm: bạn **chỉ mô tả logic**, còn CPU sẽ thực thi.
- Viết RTL: bạn **phải xây cả hệ thống logic để thực thi chính logic đó**.

Logic Synthesis (3/3)

Biến tả RTL (Verilog/VHDL) → Gate-level netlist để triển khai trên silicon.

Đầu vào của quá trình tổng hợp:

1. RTL Description (Verilog hoặc VHDL):

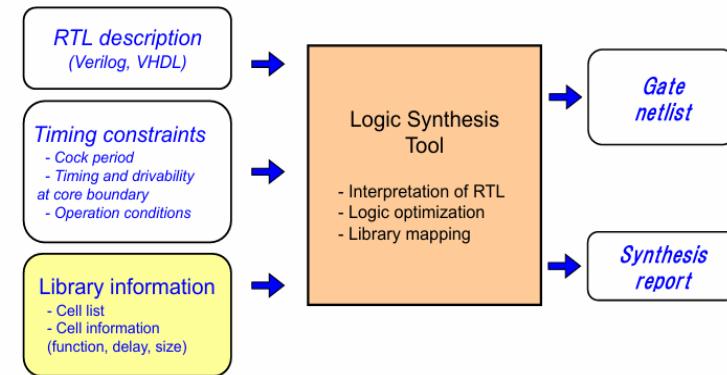
- Mô tả hành vi của mạch số theo logic đồng bộ.
- Ví dụ: assign sum = a + b;

Logic Synthesis (3/3)

2. Timing Constraints:

- Định nghĩa **giới hạn thời gian** để tool tổng hợp tối ưu logic phù hợp:
 - Clock period: Chu kỳ xung nhịp yêu cầu.
 - Input/Output delay: Thời gian truyền dữ liệu vào/ra khối thiết kế.
 - Driving strength: Khả năng truyền tải logic ở biên giao tiếp.
 - Operation conditions: Điện áp, nhiệt độ hoạt động, v.v.

Synthesis: RTL description is converted to gate-level description.



3. Library Information:

- Mô tả thư viện cổng logic khả dụng để tool sử dụng:
 - Cell list:** Danh sách các cổng như AND, OR, Flip-Flop, MUX, v.v.
 - Cell info:** Bao gồm:
 - Chức năng (function)
 - Độ trễ (delay)
 - Kích thước (size)

Logic Synthesis Tool thực hiện:

- Diễn giải RTL:** Hiểu ý nghĩa thiết kế từ code.
- Tối ưu logic:** Giảm số lượng cổng, tiết kiệm diện tích hoặc giảm trễ.
- Ánh xạ thư viện:** Gán logic của bạn vào các phần tử có thật từ thư viện (gọi là *library mapping*).

Kết quả đầu ra:

Output	Ý nghĩa
Gate Netlist	Mạch ở mức cổng – dùng để chạy STA, layout, hoặc tổng hợp tiếp
Synthesis Report	Gồm thông tin về: timing, area, power, số lượng cell, lỗi nếu có,...

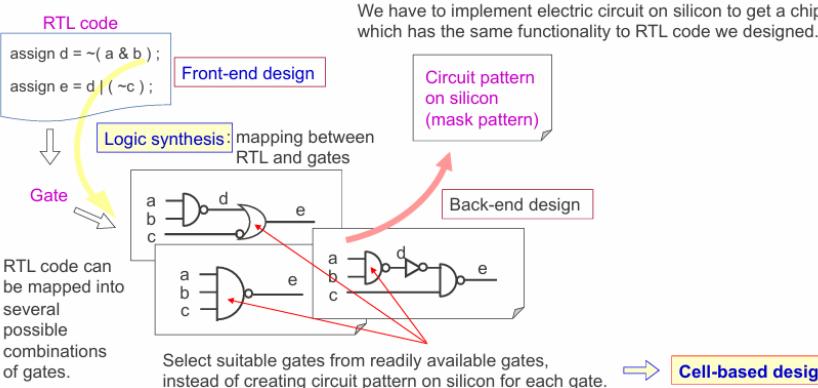
Ghi nhớ:

- Không có **Timing constraints** hoặc **Library**, synthesis tool **không biết tối ưu theo hướng nào** hoặc **sử dụng cell gì**.
- Đây là **bước chuyển từ “logic mô tả” sang “logic vật lý cụ thể”**.

Logic Synthesis and Cell-Based Design (1/4)

Mục tiêu: Biến mô tả RTL (dạng code logic như Verilog) thành mạch điện thực tế trên silicon (IC).

Logic Synthesis and Cell-Based Design (1/4)



Logic Synthesis

- Là quá trình ánh xạ code RTL thành các tổ hợp cổng logic (AND, OR, NOT,...).
- Có thể tồn tại nhiều cách ánh xạ khác nhau cho cùng một đoạn RTL code.

Cell-Based Design: Chọn những cổng (gates) từ thư viện chuẩn (Standard Cell Library) đã được thiết kế sẵn.

- Không cần tự thiết kế lại transistor bên trong mỗi cổng.
- Giúp tăng tốc độ thiết kế, tối ưu diện tích và đơn giản hóa quá trình backend (layout & mask).

Front-End Design	Back-End Design
RTL Code → Logic Synthesis	Gate-level → Physical Implementation
Output: Netlist	Output: Silicon Mask Pattern

Cell-Based Design Là Gì?

- Là thiết kế mạch bằng cách **sử dụng lại các cell chuẩn**, giống như lắp ráp LEGO.
- Cell đã có sẵn thông tin: Logic function, Delay, Area, Power, Layout

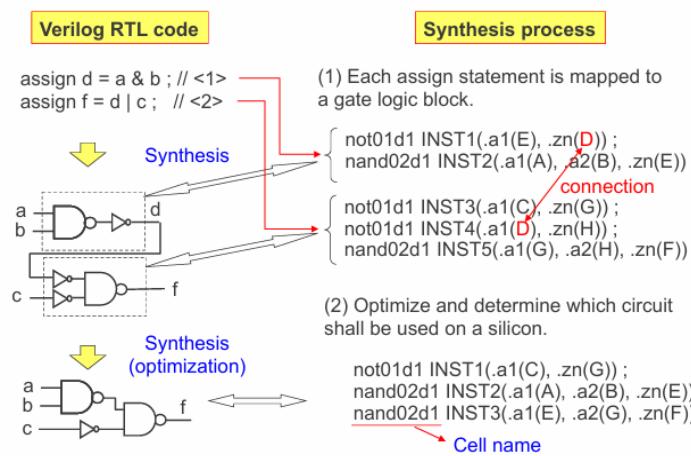
Giúp giảm thiểu rủi ro sai sót khi thiết kế transistor từ đầu.

Logic Synthesis and Cell-Based Design (2/4)

Mục tiêu: Minh họa một ví dụ đơn giản về quá trình tổng hợp (synthesis) từ RTL Verilog code thành **cổng logic** và sau đó là tối ưu hóa.

Logic Synthesis and Cell-Based Design (2/4)

Example.



Synthesis Process (Quá trình tổng hợp):

Bước 1: Chuyển từng lệnh assign sang tổ hợp cổng logic

Bước 2: **Tối ưu hóa (Optimization): Giảm số lượng cổng, giảm độ trễ, diện tích, hoặc điện năng tiêu thụ.**

Tóm tắt:

- **Synthesis** = ánh xạ RTL → logic → netlist (các cell).
- **Synthesis Optimization** = chọn tổ hợp cell hiệu quả hơn, giảm số lượng cổng hoặc kết nối.
- **Cell-based design** = tận dụng cell sẵn có trong thư viện thay vì tự thiết kế từng transistor.

Logic Synthesis and Cell-Based Design (3/4)

Mục tiêu: Giải thích về việc chọn **cell phù hợp** từ thư viện cổng chuẩn (standard cell library), đặc biệt là theo tiêu chí hiệu suất, diện tích, và **khả năng điều khiển (drivability)**.

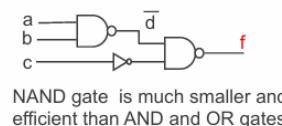
Khái niệm drivability (khả năng điều khiển)

- Là số lượng **fanout** (đầu ra) mà một cổng logic có thể **kéo/dẩy (drive)** được mà không làm giảm hiệu suất (tốc độ, độ ổn định).
- Nếu đầu ra f cần truyền tín hiệu đến **4 cổng khác**, ta cần cell có **drivability cao hơn** như nand02d4 (drive strength = 4).

Tóm tắt:

- Sử dụng **thư viện cell** là cốt lõi của **cell-based design**.
- Logic synthesis chọn cell phù hợp với **tải, tốc độ, diện tích, công suất**.
- **Drive strength** cao (ví dụ: d4) dùng để đảm bảo tín hiệu vẫn ổn định khi có nhiều fanout.

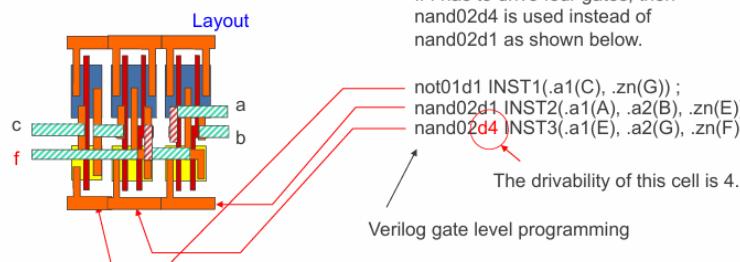
Logic Synthesis and Cell-Based Design (3/4)



Cells are selected from a standard cell library depending on a design constraint such as minimize area and/or power consumption, maximize speed, etc.

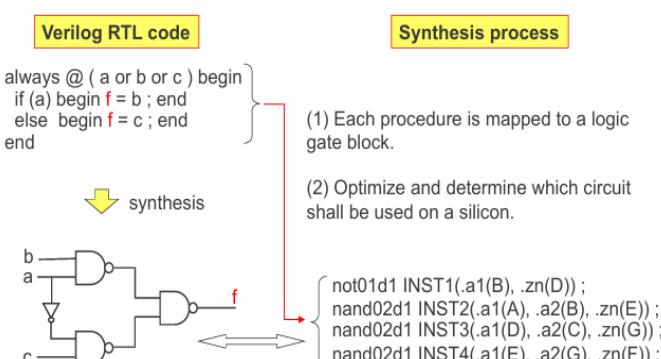
Example:

If f has to drive four gates, then nand02d4 is used instead of nand02d1 as shown below.



Logic Synthesis and Cell-Based Design (4/4)

While synthesizing, procedures are mapped into a logic gate blocks procedure by procedure base.



Logic Synthesis and Cell-Based Design (4/4)

Mục tiêu: Giải thích quá trình **synthesis** từ **procedural logic (mã điều kiện/luồng điều khiển)** sang **mạch logic cổng**, và quá trình **tối ưu hóa cell** trong quá trình này.

Synthesis thành cổng logic

- Công cụ **logic synthesis** chuyển đoạn mã procedural trên thành tổ hợp **cổng logic cơ bản**.
- Mạch được ánh xạ gồm: NOT, NAND (thay cho AND/OR vì hiệu quả hơn).
- **Sơ đồ mạch sau synthesis:**
 - **NOT gate** xử lý điều kiện $\sim a$
 - Hai **NAND gate** dùng để tạo chức năng AND/OR
 - Một **NAND gate** cuối để kết hợp và tạo ra f

Tối ưu hóa: **Synthesis tool** không chỉ ánh xạ logic mà còn:

- Chọn cell từ **standard cell library**
- Cân nhắc hiệu năng, diện tích, điện năng
- Ánh xạ các procedural block sang các mạch logic tương đương

Tóm lại:

- **Procedural Verilog code** (luồng điều khiển) → được ánh xạ sang **logic cỗng tổ hợp**.
- Synthesis tool **hiểu và tái cấu trúc** theo cách tối ưu cho phần cứng.
- Đây là bước quan trọng giúp ta đi từ mô tả hành vi (behavioral) đến mô tả cỗng logic (gate-level netlist) chuẩn bị cho layout trên silicon.

3.6. Gate Level Design and Verification

Mục tiêu: Mô tả các loại thông tin mà công cụ EDA (Electronic Design Automation) cần để **hiện thực hóa logic lên silicon**, và cách thư viện (library) hỗ trợ việc đó.

Công cụ EDA cần rất nhiều loại thông tin khác nhau (về chức năng, vật lý, wiring...) để chuyển mạch logic thành layout thật. Tất cả thông tin này được lấy từ các loại **thư viện chuẩn** (standard cell libraries).

Thông tin cần thiết mà EDA tool sử dụng:

1. Kết nối giữa các cell

→ Để biết mạch kết nối thế nào giữa các thành phần logic

2. Thông tin đặc trưng của cell

→ Chức năng, loại ngõ vào/ra, công suất kéo (drive strength)...

3. Thông tin về kích thước và hình học cell

→ Rộng, cao, vị trí terminal để layout chính xác

4. Thông tin cấu trúc bên trong cell

→ Các lớp diffusion, cỗng, dây nội bộ...

5. Luật layout

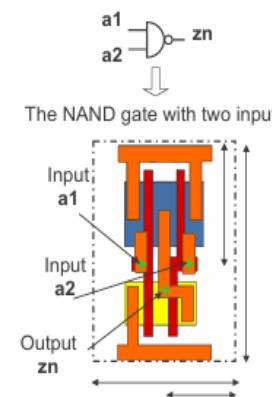
→ Quy định khoảng cách tối thiểu giữa các cell

6. Luật đi dây (wiring rules)

→ Độ rộng dây, khoảng cách dây, số lượng via...

Implement The Logic on Silicon (1/4)

EDA tools handle the following information for implementing the logic on to the silicon



- 1.The information connection between cells
- 2.The feature information of the cell (Function, input/output, the drive power, etc.)
- 3.The information on the structure of the cell (Width, the height, the terminal position, etc.)
- 4.The information on the inner structure of the cell (Diffusion layer size/position, gate, inner wiring, etc.)
- 5.Layout rules (Minimum distance between cells, etc.)
- 6.Wiring rules (Minimum pitch, multi-via, etc.)

Library information

- The frontend library
Information necessary to choose the cells
Information about delay and drivability
- The backend library
The information necessary for place and route
- The layout library
The pattern information to make photo mask.

The cell library

- The frontend library
Information necessary to choose the cells
Information about delay and drivability
- The backend library
The information necessary for place and route
- The layout library
The pattern information to make photo mask.

This part can be provided to the customers who synthesize logic by themselves.

This part may be provided to the customers who do layout jobs by themselves.

This part is strictly confidential.

The cell library contains a lot of information which is highly dependent on particular process generation and fabrication line.

In particular, macro cells and custom cells have no compatibility between fabrication lines, therefore avoid using these cells whenever possible.

Library Information: Tất cả thông tin trên được EDA tool lấy từ **library** dưới đây:

Library Type	Chức năng chính
Frontend Library	- Dùng để chọn cell - Thông tin delay, drive strength
Backend Library	- Phục vụ cho việc place and route
Layout Library	- Thông tin layout để tạo ra photo mask trong chế tạo silicon wafer

Ghi chú đặc biệt:

- Cell library phụ thuộc rất lớn vào công nghệ chế tạo **cụ thể**, vì mỗi hãng/fab có quy trình riêng.
- **Macro cell** và **custom cell** thường không tương thích giữa các dây chuyền chế tạo → tránh sử dụng khi không cần thiết.

Tóm lược:

- **Frontend:** Giúp lựa chọn cell khi viết RTL → synthesis.
- **Backend:** Hỗ trợ routing, xác định vị trí cell trên silicon.
- **Layout:** Chứa pattern thật để **in lên wafer** – cực kỳ nhạy cảm, nên luôn được giữ kín trong nội bộ fab.

Design Rule là gì?

► “Design Rule” là tập hợp các quy định kỹ thuật cho việc layout IC **dựa theo dây chuyền công nghệ (fabrication line)**.

► Bao gồm:

- **Minimum spacing** giữa các đường dây hoặc phần tử layout.
- **Minimum width** của dây dẫn hoặc layer.
- **Vị trí allowed/not allowed** cho via, diffusion, polysilicon, metal layers...
- **Electrical constraints** như độ tương thích giữa các lớp kim loại.

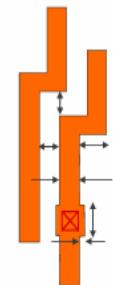
Những quy tắc này **bắt buộc phải tuân thủ** nếu không **mask sẽ không in chính xác lên wafer**, dẫn đến lỗi chip!

Implement The Logic on Silicon (3/4)

- 1.The information connection between cells
- 2.The feature information of the cell
(Function, input/output, the drive power, etc.)
- 3.The information on the structure of the cell
(Width, the height, the terminal position, etc.)
- 4.The information on the inner structure of the cell
Diffusion layer size/position, gate, inner wiring, etc.
- 5.Layout rules
(Minimum distance between cells, etc.)
- 6.Layout rules
(wiring rule, etc.)

The design rule

Minimum spacing, wiring width, and other conditions defined specific to a fabrication line.



Tóm lược Design Rule:

- **Design Rule là cẩm nang bắt buộc** cho layout engineer.
- Mỗi fabrication line có **design rule riêng**.
- Vi phạm những quy tắc này có thể **dẫn đến chip lỗi hoặc không thể sản xuất**.

Netlist là gì?

- **Netlist** là **tập hợp mô tả kết nối giữa các cổng logic (cells)** thông qua các dây (nets).
- Là kết quả **đầu ra từ quá trình tổng hợp (synthesis)** và là **đầu vào cho quá trình layout (place & route)**.

Mỗi dòng netlist cho biết:

- Sử dụng **cell nào** từ thư viện chuẩn (standard cell).
- Tên của **phiên bản cụ thể** của cell (instance name).
- **Kết nối giữa các chân (I/O)** của cell với **wire** cụ thể trong mạch.
- Ví dụ: INST3(.a1(C), .a2(B), .zn(E)); nghĩa là: NAND gate nối C và B vào, đầu ra là E.

Implement The Logic on Silicon (4/4)

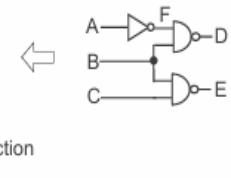
- 1.The information connection between cells
- 2.The feature information of the cell
(Function, input/output, the drive power, etc.)
- 3.The information on the structure of the cell
(Width, the height, the terminal position, etc.)
- 4.The information on the inner structure of the cell
Diffusion layer size/position, gate, inner wiring, etc.
- 5.Layout rules
(Minimum distance between cells, etc.)
- 6.Layout rules
(wiring rule, etc.)

The Netlist

```
arbitrary unique name
not01d1 INST1(.a1(A), .zn(F));
nand02d1 INST2(.a1(F), .a2(B), .zn(D));
nand02d1 INST3(.a1(C), .a2(B), .zn(E));
```

↓ ↓ ↓ ↓ ↓ ↓

Name of the Name of the Name of the Connection
cell used wire I/O of the cell



Tóm lược Netlist:

- **Netlist** là cầu nối giữa **thiết kế logic (RTL)** và **triển khai vật lý (layout)**. Nó là **mô tả kết nối của các cell tiêu chuẩn** trong thiết kế.
- Là output quan trọng của logic synthesis và input của backend design.

Gate Level Design – Thiết kế mức cổng

Định nghĩa: Gate level design là quá trình **tạo hoặc chỉnh sửa logic** bằng cách **kết nối các cổng logic cơ bản (primitive gates)** như AND, OR, NAND, NOR, NOT, v.v.

Gate level design thường xảy ra trong các trường hợp sau:

- Design for Testability (DFT): để chèn scan chain, BIST, hoặc các phần kiểm thử khác.
- Static Timing Analysis (STA): để phân tích chính xác độ trễ và timing path.
- Layout design: cần chính xác về vị trí, kết nối cell.

Gate Level Design

```
not01d1 INST1(.a1(A), .zn(F));
nand02d1 INST2(.a1(F), .a2(B), .zn(D));
nand02d1 INST3(.a1(C), .a2(B), .zn(E));
```

Generating a total logic or modifying a part of logic by selecting/connecting primitive gates as the above is called “gate level design”.

This situation may happen in Design for Testability (DFT), Static Timing Analysis (STA), Layout design.

Gate level design has some disadvantages as below:

- (1) Visibility of a code is very poor, hard to understand at a glance.
- (2) Portability of a design in net list is very bad because of the cells used in the design.
- (3) There are few compatibilities among the process generations.
- (4) It will be difficult to enjoy the benefit of the evolution of DA tools (such as optimization)

Hạn chế của Gate Level Design:

- Khó đọc & hiểu mã: Code gate-level rất chi tiết và phức tạp, khó hình dung chức năng.
- Khó tái sử dụng: Mỗi cell là cell cụ thể từ thư viện, nên thiết kế không linh hoạt.
- Kém tương thích: Mỗi thế hệ công nghệ có cell khác nhau, không thể mang sang trực tiếp.
- Khó tối ưu hóa: Không tận dụng được các tiến bộ của EDA tools (synthesis, PnR...).

Gate Level Verification – Functional Simulation

Mục tiêu: Gate-level simulation (GLS) là để kiểm tra hoạt động logic chính xác của mạch sau khi đã được tổng hợp (synthesis) thành netlist với các gate cụ thể.

Cách hoạt động: Event-Driven Simulation

- Đây là mô phỏng dựa trên sự kiện: khi một đầu vào thay đổi, sự thay đổi này sẽ lan truyền (propagate) qua các cổng logic.
- Mỗi sự thay đổi gọi là một event.

Diễn giải hình ảnh:

1. Initial

- Tất cả đầu vào và đầu ra có giá trị xác định (ở đây là 0 hoặc 1).

2. Event Occurred

- Đầu vào b thay đổi từ 1 → 0, tạo ra một event mới.

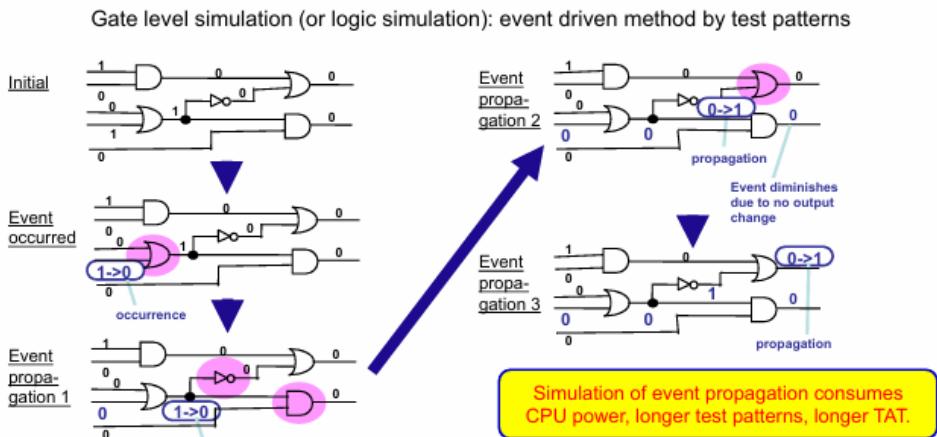
3. Event Propagation 1

- Event này truyền qua cổng NOT → đầu ra thay đổi từ 0 → 1.
- Đây là sự lan truyền đầu tiên của event.

4. Event Propagation 2 & 3

- Sự thay đổi tiếp tục lan sang các cổng logic kế tiếp.
- Nếu sự thay đổi không làm thay đổi đầu ra của một cổng, thì event bị tiêu biến (no propagation).

Gate Level Verification – Functional Simulation



"Simulation of event propagation consumes CPU power, longer test patterns, longer TAT."

- CPU power:** Mỗi lần event xảy ra → cần tính toán lại toàn bộ đầu ra liên quan → tốn tài nguyên máy.
- Longer test patterns:** Phải tạo nhiều mẫu kiểm tra (test patterns) để đảm bảo mọi đường logic được kiểm thử.
- Longer TAT (Turn-Around Time):** Thời gian kiểm thử lâu hơn do nhiều sự kiện lan truyền phải mô phỏng.

Khi nào dùng Gate-Level Simulation?

- Sau khi synthesis hoàn tất (Netlist có đầy đủ delay/cell cụ thể).
- Trước tape-out để đảm bảo không có lỗi logic do mapping/synthesis.
- Trong kiểm thử **Design for Testability (DFT)**, kiểm tra scan chain, v.v.

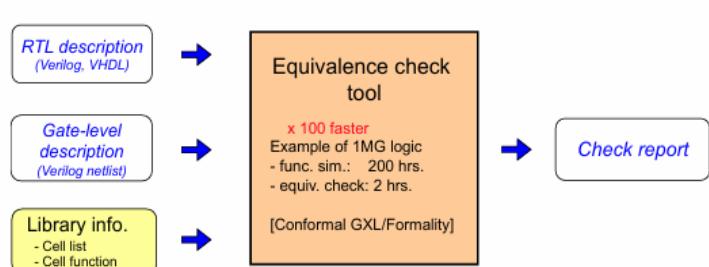
Gate Level Verification – Equivalence Check

Mục tiêu của Equivalence Check:

- Đảm bảo rằng mô tả RTL ban đầu (Verilog/VHDL) và kết quả sau khi tổng hợp (Gate-level Netlist) có chức năng tương đương tuyệt đối.
- Đây là bước bắt buộc trong flow để đảm bảo không có sai sót do synthesis.

Gate Level Verification – Equivalence Check

- Equivalence of two logic descriptions is mathematically verified by formality check: no need for test patterns.
- Equivalence check is exhaustive and faster than functional simulation.



Cơ chế hoạt động:

- So sánh toán học (formally) giữa hai mô tả logic:
 - RTL (trước synthesis)
 - Netlist (sau synthesis)
- Không cần test pattern như trong functional simulation.
- Sử dụng công cụ như **Synopsys Formality**, **Cadence Conformal**.

Các thành phần đầu vào cho công cụ Equivalence Check:

- RTL Description** – Verilog, VHDL: Mô tả thiết kế ở mức hành vi (behavioral/structural).
- Gate-level Description** – Verilog Netlist: Mô tả sau khi synthesis, gồm các gate cụ thể đã được ánh xạ.
- Library Info:** Thông tin về cell: danh sách cell (cell list), chức năng (cell function).

Hiệu quả: Nhanh gấp 100 lần so với Functional Simulation.

Ví dụ: Thiết kế 1M gate: **Functional sim:** 200 giờ > **Equivalence check:** 2 giờ

Output:

- **Check Report:** Báo cáo cho biết liệu RTL và Netlist có tương đương không.
- Nếu có khác biệt: công cụ sẽ chỉ rõ điểm sai lệch (dễ debug hơn).

Ưu điểm:

- **Nhanh chóng và triệt để (exhaustive).**
- Không phụ thuộc vào test pattern.
- Phát hiện lỗi synthesis mà functional sim có thể bỏ sót.

Floor Planning – For Pre-layout Timing Verification

Mục tiêu: Floor planning là bước đầu tiên trong giai đoạn physical design – **bố trí sơ bộ các khối chức năng** trên chip.

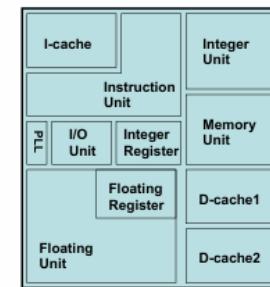
Mục tiêu là để chuẩn bị cho **timing estimation, routing**, và các bước tiếp theo của layout.

Những việc cần thực hiện trong Floor Planning:

1. **Đặt các khối chức năng (functional blocks):**
 - Dựa trên luồng tín hiệu (signal flow) trong thiết kế.
 - Đặt các block tương tác mạnh gần nhau để giảm độ trễ và tối ưu wiring.
2. **Xem xét packaging & bonding:**
 - Tối ưu theo gói đóng chip (package spec):
 - Giảm số lượng pin
 - Rút ngắn chiều dài dây nối
 - Tính toán vị trí bonding pad (pad để kết nối với package).

Floor Planning

For pre-layout timing verification



Example for floor plan of processor chip

Ứng dụng chính:

- Pre-layout timing verification: Giúp ước lượng sơ bộ độ trễ (timing) trước khi thực hiện layout chi tiết.
- Chuẩn bị cho bước placement và routing sau này.

Lợi ích:

- Tối ưu hóa hiệu suất và tiêu thụ điện năng.
- Giảm thiểu tắc nghẽn dây (routing congestion).
- Giảm số lượng tầng kim loại cần thiết.
- Hỗ trợ tool EDA tối ưu về diện tích và thời gian xử lý.

Timing Verification – Methods

STA: Static Timing Analysis

Là phương pháp phân tích thời gian dựa trên **mô tả logic** ở mức cổng (gate-level), **không cần mô phỏng** hoặc test pattern.

Cơ chế:

- Xây dựng đồ thị thời gian từ thiết kế → phân tích đường truyền từ **input** → **output**.
- So sánh với các ràng buộc timing (clock period, setup/hold...).

Ưu điểm:

- **Không cần test pattern**
- **Synchronous path** có thể được kiểm tra **100% chính xác**
- Nhanh và phù hợp cho chip lớn

Hạn chế:

- Không kiểm tra được mạch **asynchronous** hoặc mạch đặc biệt (gated clocks, mux path...)
- **Special paths** (false path, multi-cycle path) cần được chỉ định thủ công

DTA: Dynamic Timing Analysis

Là phương pháp dùng **mô phỏng logic** có tính đến **thời gian delay**, thực thi bằng test pattern.

Cơ chế:

- Mô phỏng từng chu kỳ, theo kiểu hoạt động của mạch (event-driven).
- Tính toán sự truyền tín hiệu và delay thật sự tại từng cổng, từng đường truyền.

Ưu điểm:

- Có thể phân tích được **asynchronous circuit** và các mạch khó
- Kiểm tra đúng hành vi thực tế của mạch với delay

Hạn chế:

- Cần rất nhiều test patterns**
- Không thực tế khi áp dụng cho toàn bộ chip → **dùng cục bộ**

Lưu ý:

"Verify synchronous paths with STA, and use DTA for limited paths that cannot be checked by STA."

=> Hãy luôn thiết kế **synchronous** và dùng **STA là chính**, chỉ dùng DTA khi không thể tránh khỏi.

Tóm lại:

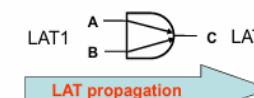
	STA (Static)	DTA (Dynamic)
Test pattern	Không cần	Cần
Tốc độ kiểm tra	Rất nhanh	Chậm do phải mô phỏng
Synchronous	Hỗ trợ hoàn toàn	Hỗ trợ
Asynchronous	Không hỗ trợ	Có thể kiểm tra
Dùng khi nào	Chính yếu (default)	Với các đường đặc biệt hoặc async

Timing Verification – STA Basic Algorithm

STA Basic Algorithm – Static Timing Analysis

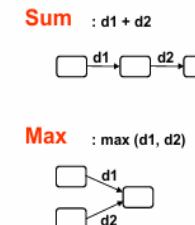
Khái niệm chính:

- LAT (Latest Arrival Time):** Thời điểm muộn nhất mà một tín hiệu đến tại một node trong mạch **mà không vi phạm thời gian clock**.
- STA tool** sẽ truyền LAT dọc theo các đường tín hiệu, tính theo quy tắc **max + delay**.

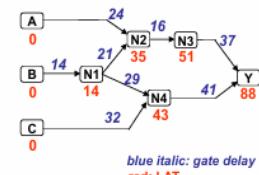


STA tool propagates LAT by selecting the maximum of LATs and adding delay of ARC to the LAT.

LAT: Latest Arrival Time



Example



blue italic: gate delay
red: LAT

Nguyên lý tính toán trong STA

1. **Sum:** Khi các tín hiệu đi nối tiếp, tổng delay được cộng lại:

2. **Max** Khi có **nhiều đường vào** cùng hội tụ tại một điểm (fan-in), ta chọn đường có **LAT lớn nhất** vì nó đến trễ nhất:

Khái niệm cơ bản: Setup Slack & Hold Slack

Trong thiết kế đồng bộ sử dụng **clock edge** để điều khiển việc lưu trữ dữ liệu giữa các flip-flop, có 2 yêu cầu bắt buộc:

Setup Time Requirement

- Dữ liệu phải đến **trước clock edge** một khoảng thời gian nhất định.
- Nếu đến trễ hơn ⇒ **Vi phạm Setup** ⇒ Dữ liệu có thể không được chốt đúng.

Hold Time Requirement

- Dữ liệu phải giữ ổn định ít nhất một khoảng thời gian sau **clock edge**.
- Nếu thay đổi quá sớm ⇒ **Vi phạm Hold** ⇒ Dữ liệu cũ có thể bị ghi nhầm.

Công thức Setup Slack: Setup Slack = Tclk - (Data Path Delay + Tsetup - Clock Skew)

→ Nếu **Slack > 0** ⇒ OK

→ Nếu **Slack < 0** ⇒ Vi phạm Setup

Công thức Hold Slack: Hold Slack = (Data Path Delay + Clock Skew) - Thold

→ Nếu **Slack > 0** ⇒ OK

→ Nếu **Slack < 0** ⇒ Vi phạm Hold

Các đại lượng dùng để tính toán

Tên	Ý nghĩa
Tclk	Chu kỳ clock
LAT	Latest Arrival Time – tín hiệu đến trễ nhất
EAT	Earliest Arrival Time – tín hiệu đến sớm nhất
Tsetup	Thời gian yêu cầu dữ liệu phải ổn định trước clock
Thold	Thời gian yêu cầu dữ liệu phải giữ ổn định sau clock
Clk-Q delay	Delay từ clock đến dữ liệu ra ở flip-flop nguồn
Data path delay	Delay qua logic giữa 2 flip-flop
Clock skew	Chênh lệch thời gian giữa clock đến FF1 và FF2

So sánh LAT và EAT

Thuộc tính	LAT (Latest Arrival Time)	EAT (Earliest Arrival Time)
Ý nghĩa	Thời điểm trễ nhất tín hiệu có thể tới	Thời điểm sớm nhất tín hiệu có thể tới
Dùng để kiểm tra	Setup Timing (tránh đến trễ)	Hold Timing (tránh đến quá sớm)
Tính toán	Dùng phép max trên đường truyền	Dùng phép min trên đường truyền
Nguy hiểm khi	LAT > thời gian cho phép \Rightarrow Setup fail	EAT < thời gian giữ yêu cầu \Rightarrow Hold fail

Timing Verification – Design Process.

Timing Violation – Cách xử lý theo thứ tự ưu tiên

Bước 1: Tối ưu hậu kỳ (post-layout optimization)

"If the timing doesn't meet the requirement, replace cells and re-route wiring for better result."

- ✓ Đây là bước tối ưu thông thường khi **Timing Analysis** báo lỗi:
 - Thay thế cells:** chọn cells nhanh hơn (ví dụ chuyển từ slow NAND sang fast NAND).
 - Tối ưu lại đường đi (re-route):** giảm chiều dài dây để giảm delay.
 - Buffer insertion:** thêm buffer để tránh fanout quá lớn gây delay.

→ Không cần sửa RTL, dễ làm, nhanh, tiết kiệm thời gian thiết kế.

Bước 2: Quay lại sửa RTL (nặng nề hơn)

"When it isn't possible to meet the timing requirement by just replacing and re-routing, then we have to go back to RTL code and have to rewrite the code."

- Khi các tối ưu vật lý không đủ để khắc phục:
 - Quay về RTL (Verilog/VHDL)** để thay đổi logic.
 - Thay đổi thuật toán, chia nhỏ pipeline**, hoặc **đơn giản hóa** logic.

Tốn thời gian hơn vì phải:

- Chạy lại **synthesizable flow**.
- Làm lại STA, simulation...
- Có thể ảnh hưởng đến **functionality** nếu không cẩn thận.

Quy trình tổng quát khi bị Timing Violation

Timing Violation?

└> Try re-routing, replacing cells → Pass? → ✓ Done

→ Fail? → Go back to RTL → Modify → Re-synthesize → Re-check STA → ✓

- If the timing doesn't meet the requirement, replace cells and re-route wiring for better result.
- When it isn't possible to meet the timing requirement by just replacing and re-routing, then we have to go back to RTL code and have to rewrite the code.