101-1 Under-Graduate Project
Digital IC Design Flow

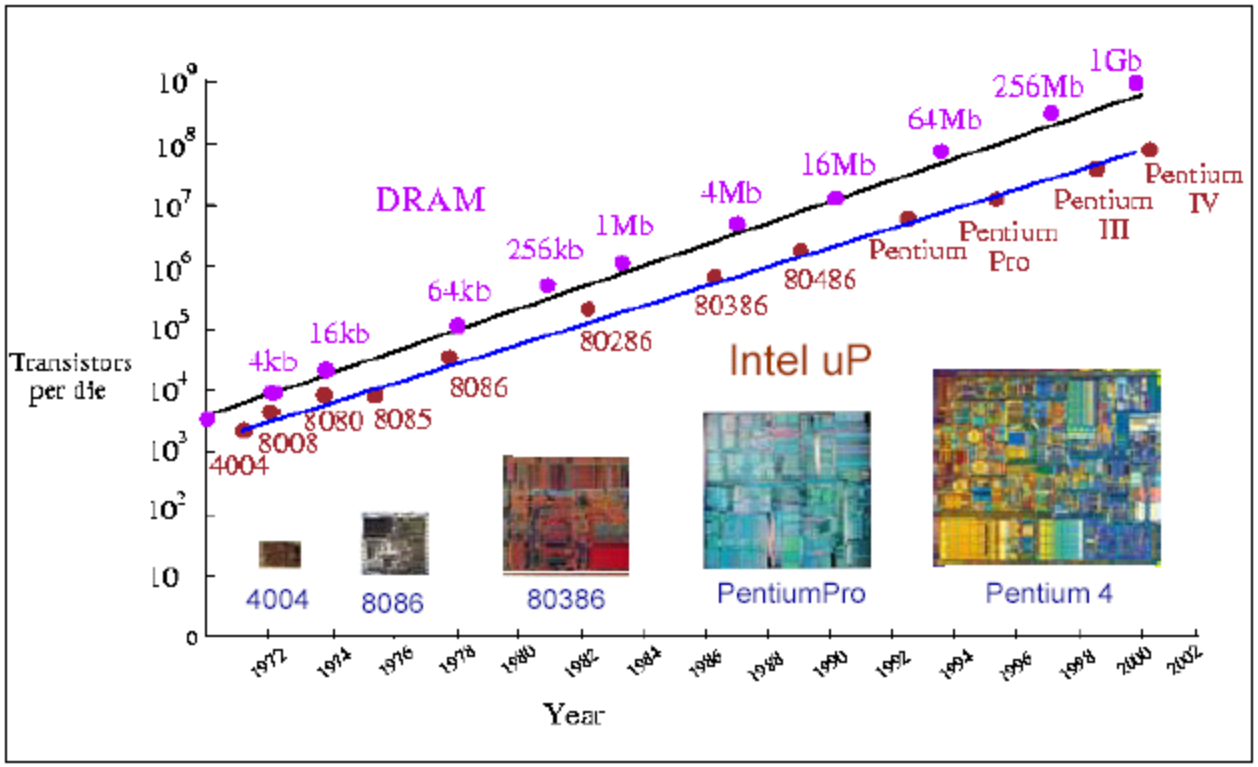
Speaker: Ming-Chun Hsiao
Adviser: Prof. An-Yeu Wu
Date: 2012/9/25

Outline

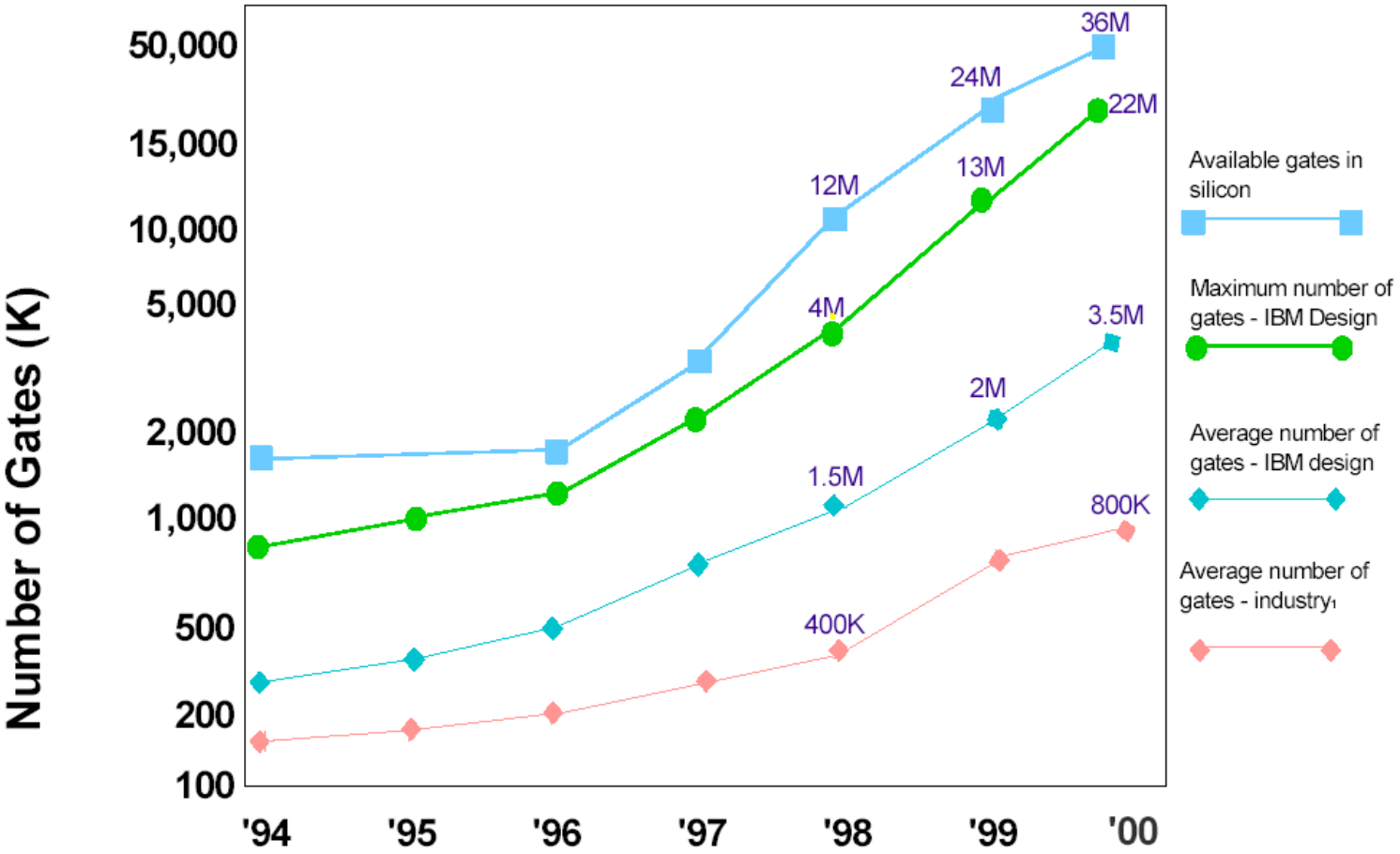
- Introduction to Integrated Circuit
- IC Design Flow
- Verilog HDL concept
- Verilog Simulator

Moore's Law: Driving Technology Advances

- Logic capacity doubles per IC at regular intervals (1965).
- Logic capacity doubles per IC every 18 months (1975).

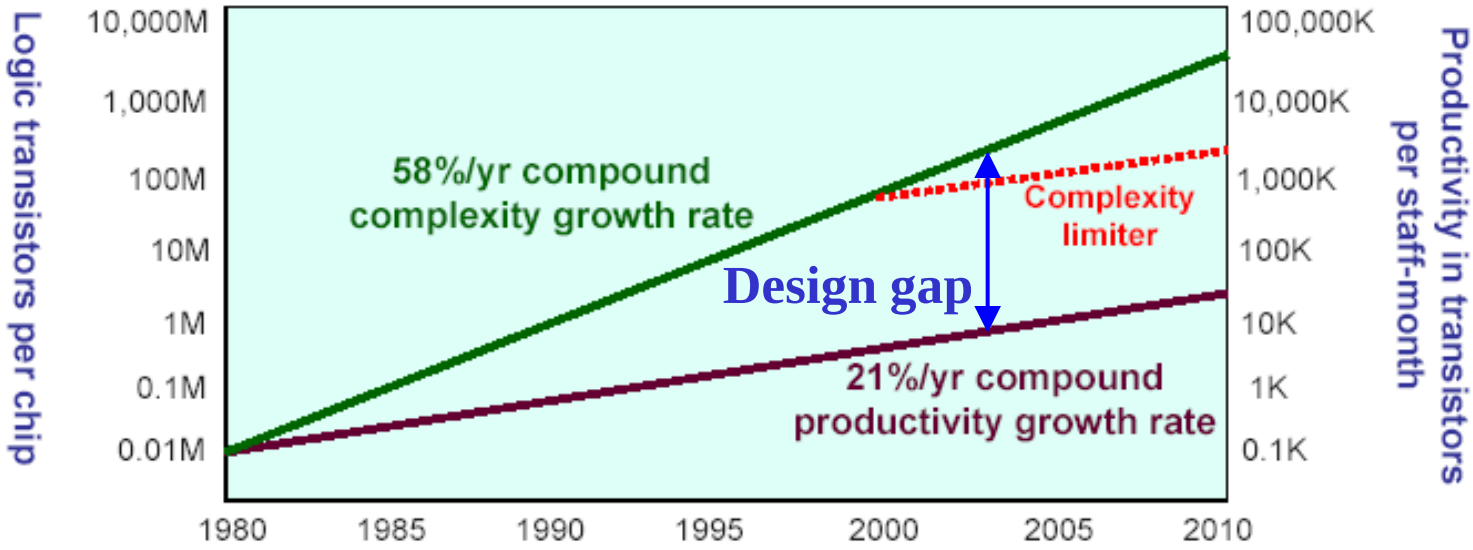


Chips Sizes



Source: IBM and Dataquest

Design Productivity Crisis



- Human factors may limit design more than technology.
- Keys to solve the productivity crisis:
  - Design techniques: hierarchical design, SoC design (IP reuse, platform-based design), etc.
  - CAD: algorithms & methodology

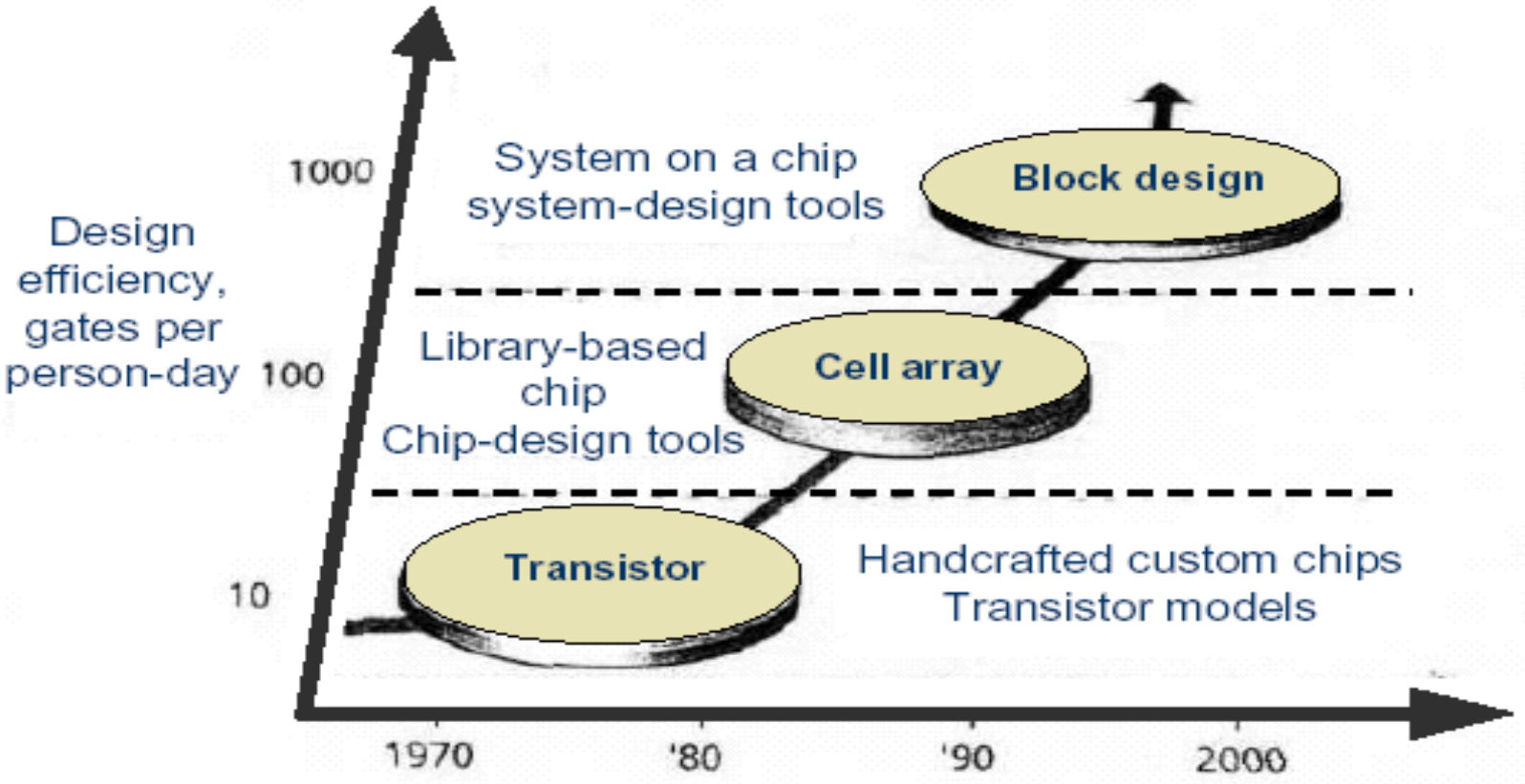
Increasing Processing Power

- Very high performance circuits in today's technologies.
  - Gate delays: ~27ps for a 2-input Nand in CU11
  - Operating frequencies:
    - Up to 500MHz for SoC/Asic
    - Over 1GHz for custom designs
- The increase in speed/performance of circuits allowed blocks to be reused without having to be redesigned and tuned for each application.
- Enhanced Design Tools and Techniques
  - Although not enough to close the "design gap", tools are essential for the design of today's high-performance chips

IP / Cores

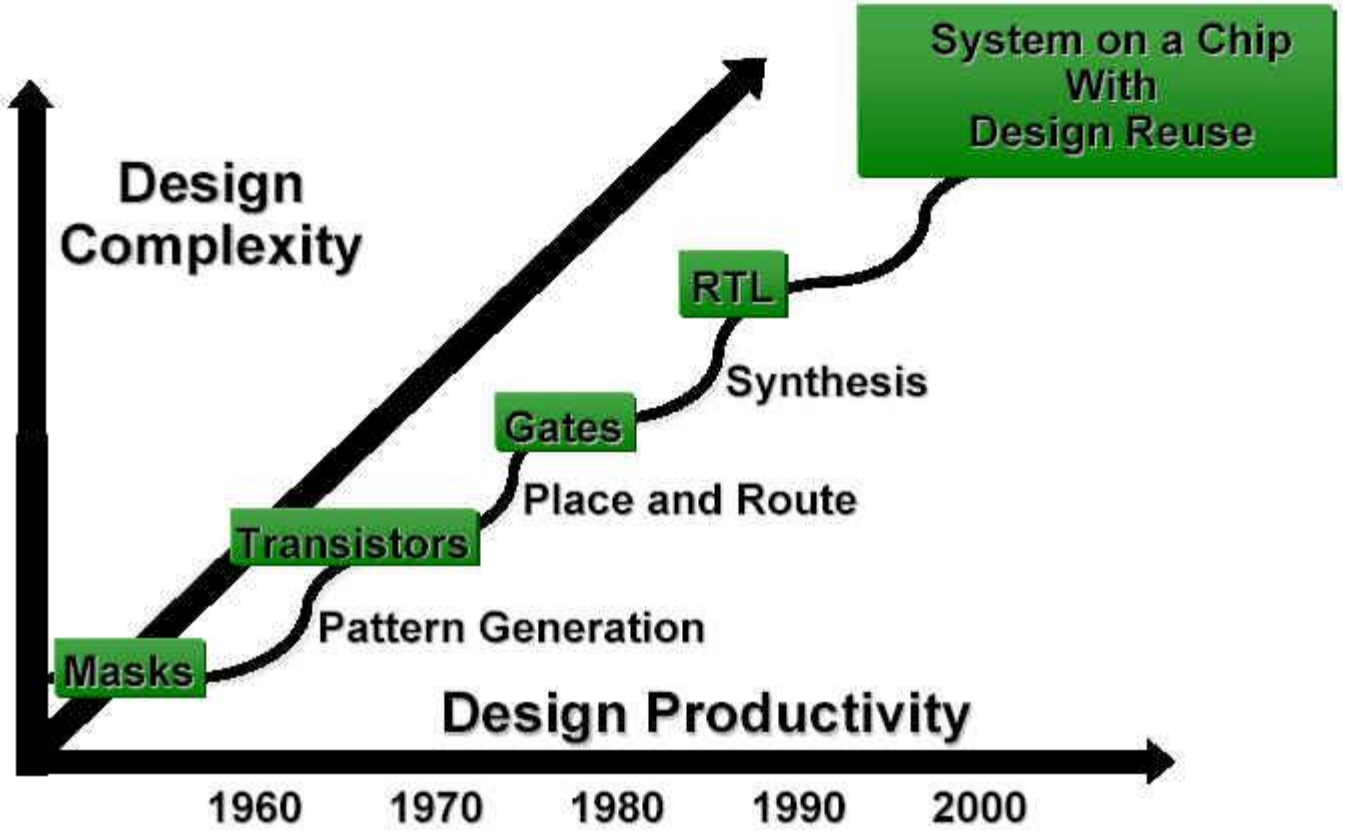
- Soft IP/Core
  - Delivered as RTL verilog or VHDL source code with synthesis script
  - Customers are responsible for synthesis, timing closure, and all front-end processing
- Firm IP/Core
  - Delivered as a netlist to be included in customer's netlist (with don't touch attribute)
  - Possibly with placement information
- Hard IP/Core
  - Due to their complexity, they are provided as a blackbox (GDSII). Ex. Processors, analog cores, PLLs
  - Usually very tight timing constraints. Internal views not be alterable or visible to the customer

Changes in the Nature of IC Design



(IEEE Spectrum Nov,1996)

Chasing the Design Gap

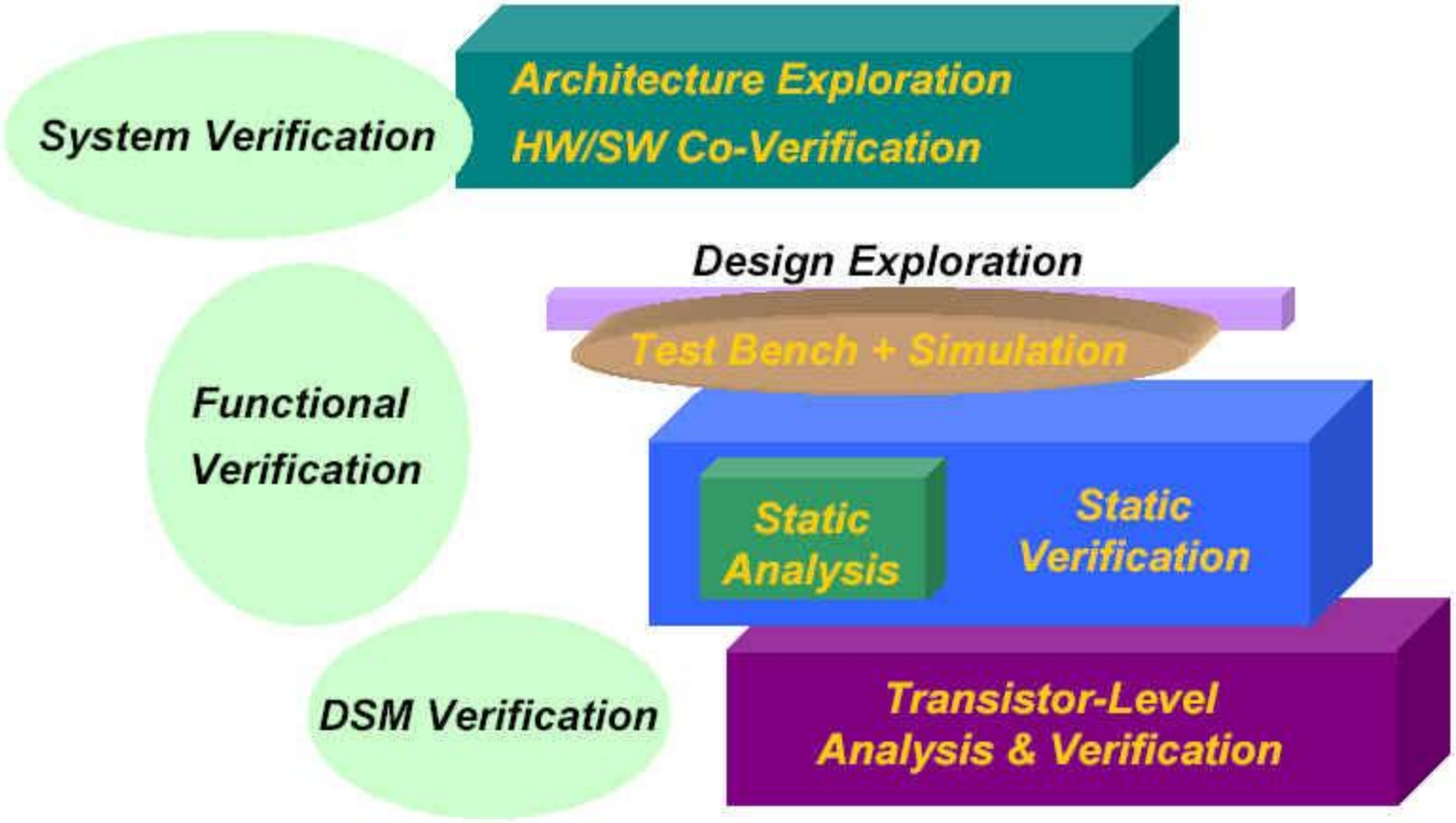


Evolution of Silicon Design

Year	1997	1998	1999	2002
Process Technology	0.35u	0.25u	0.18u	0.13u
Design Cycle (month)	18 ~ 12	12 ~ 10	10 ~ 8	8 ~ 6
Derivative Cycle (month)	8 ~ 6	6 ~ 4	4 ~ 2	3 ~ 2
Silicon Complexity (gate)	200 ~ 500 k	1 ~ 2 M	4 ~ 6 M	10 ~ 25 M
Applications	Cellular, PDAs, DVD	Set-top boxes, Wireless PDA	Internet appliances, Anything portable	Ubiquitous computing Intelligent, inter-connected con-trollers

Source: "*Surviving the SoC revolution – A Guide to Platform-based Design*," Henry Chang et al, Kluwer Academic Publishers, 1999

Methodology – Analysis and Verification



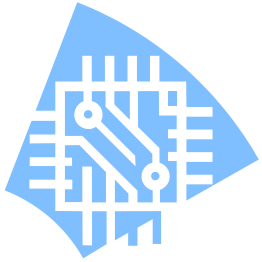
Outline

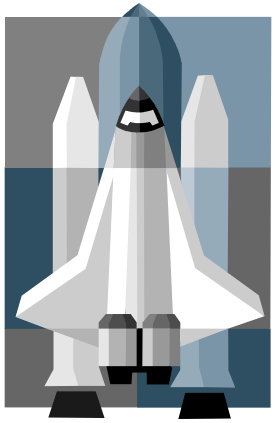
- Introduction to Integrated Circuit
- IC Design Flow
- Verilog HDL concept
- Verilog Simulator

IC Design and Implementation

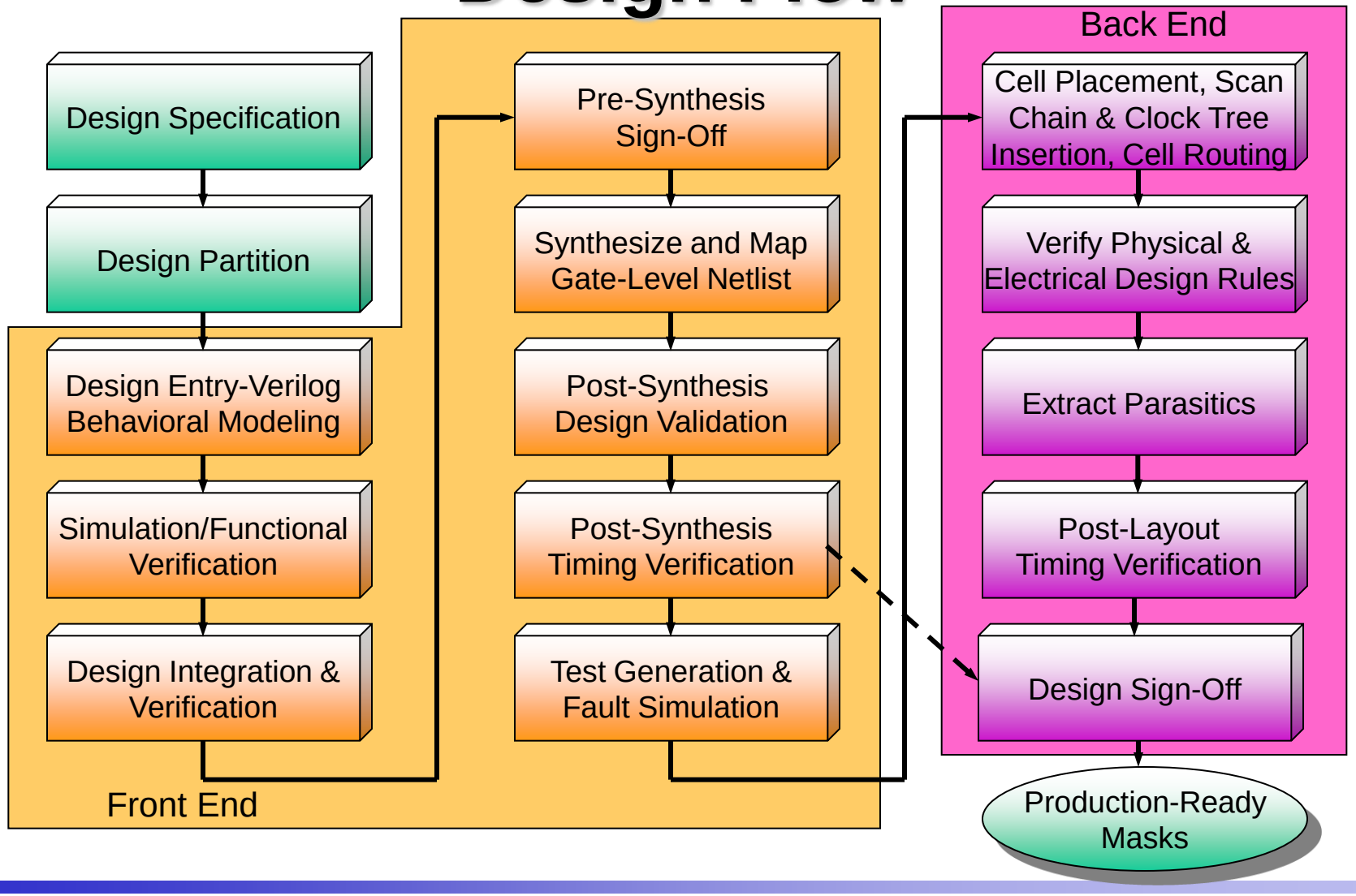
Idea

Design

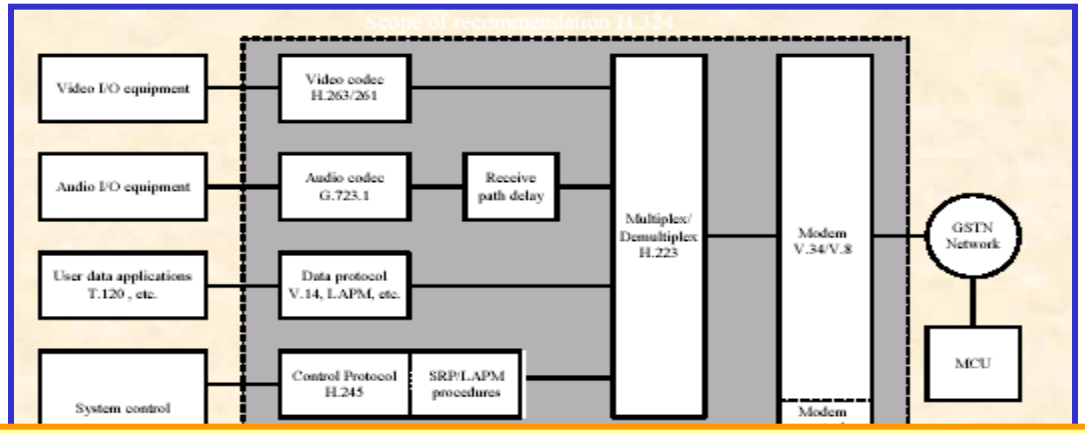




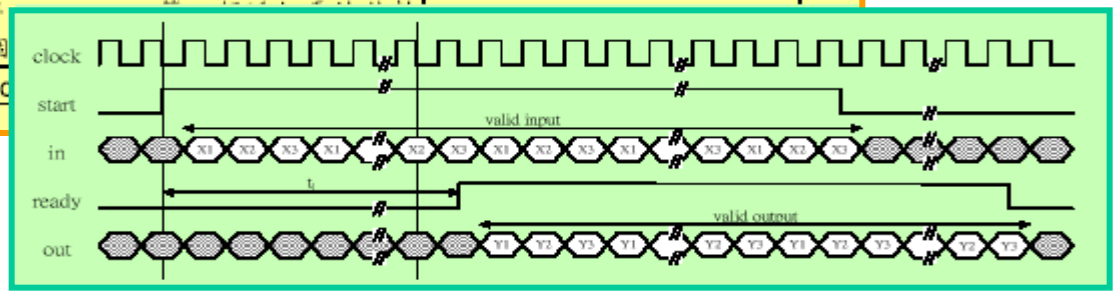
Design Flow


Design Specification
Design Partition
Design Entry-Verilog Behavioral Modeling
Simulation/Functional Verification
Design Integration & Verification
Front End
Pre-Synthesis Sign-Off
Synthesize and Map Gate-Level Netlist
Post-Synthesis Design Validation
Post-Synthesis Timing Verification
Test Generation & Fault Simulation
Back End
Cell Placement, Scan Chain & Clock Tree Insertion, Cell Routing
Verify Physical & Electrical Design Rules
Extract Parasitics
Post-Layout Timing Verification
Design Sign-Off
Production-Ready Masks

System Specification



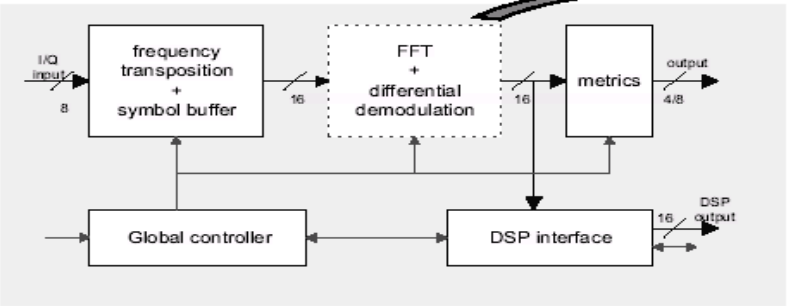
腳位名稱		描述	Drive Strength/Output Load
clk	輸入	系統時脈	assume infinite
reset	輸入	系統重置訊號，high active	1 ns/pf
din	輸入	每個clock cycle輸入一個16-bit 正整數	1 ns/pf
ready	輸出	reset為1時 出前的半個	
dout	輸出	每個clock c	



From CIC

Algorithm Mapping

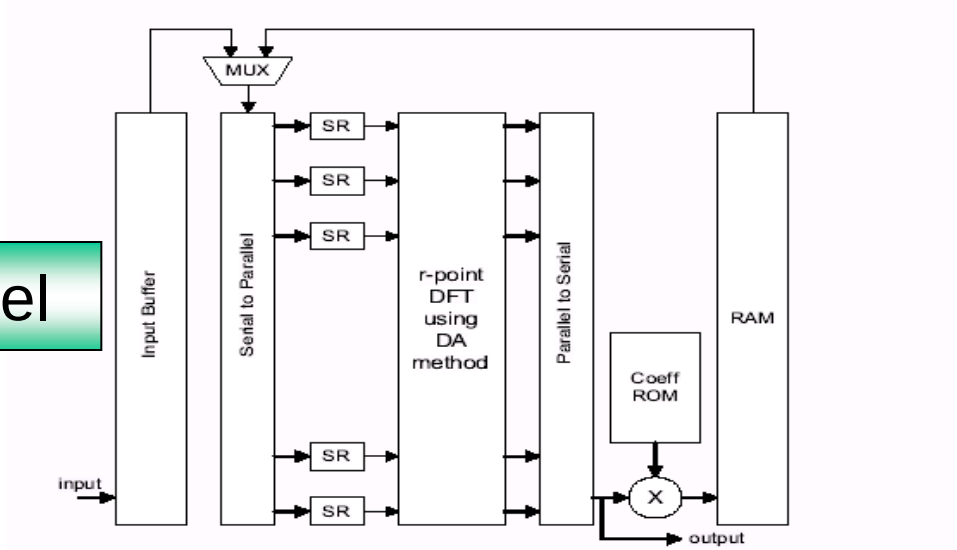
System Level



$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} \qquad k = 0,1,\ldots,N-1$$

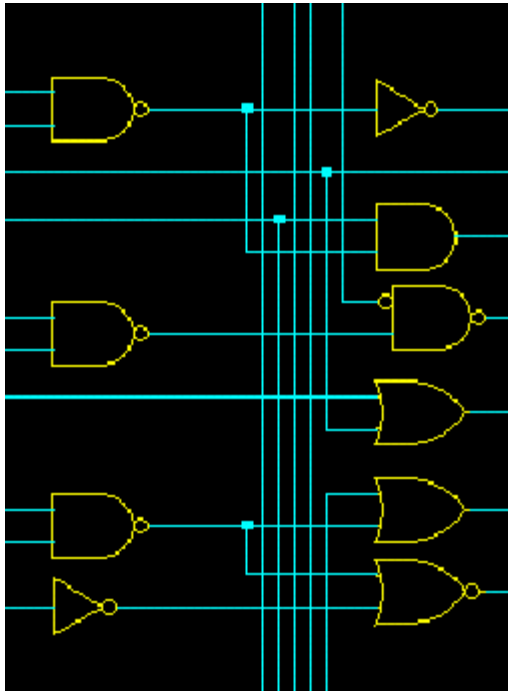
$$X(k) = \sum_{n=0}^{N/2-1} x(2n) W_{N/2}^{nk} + W_N^k \sum_{n=0}^{N/2-1} x(2n+1) W_{N/2}^{nk}$$

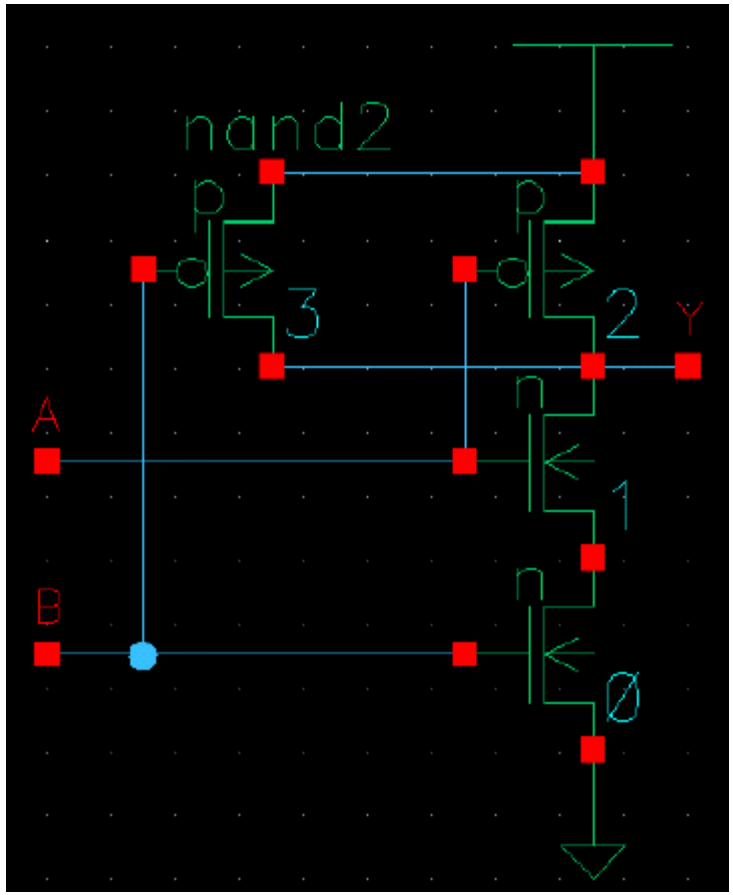
RTL Level



From CIC

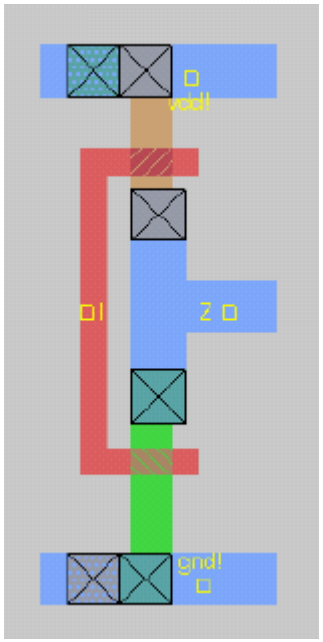
Gate and Circuit Level Design

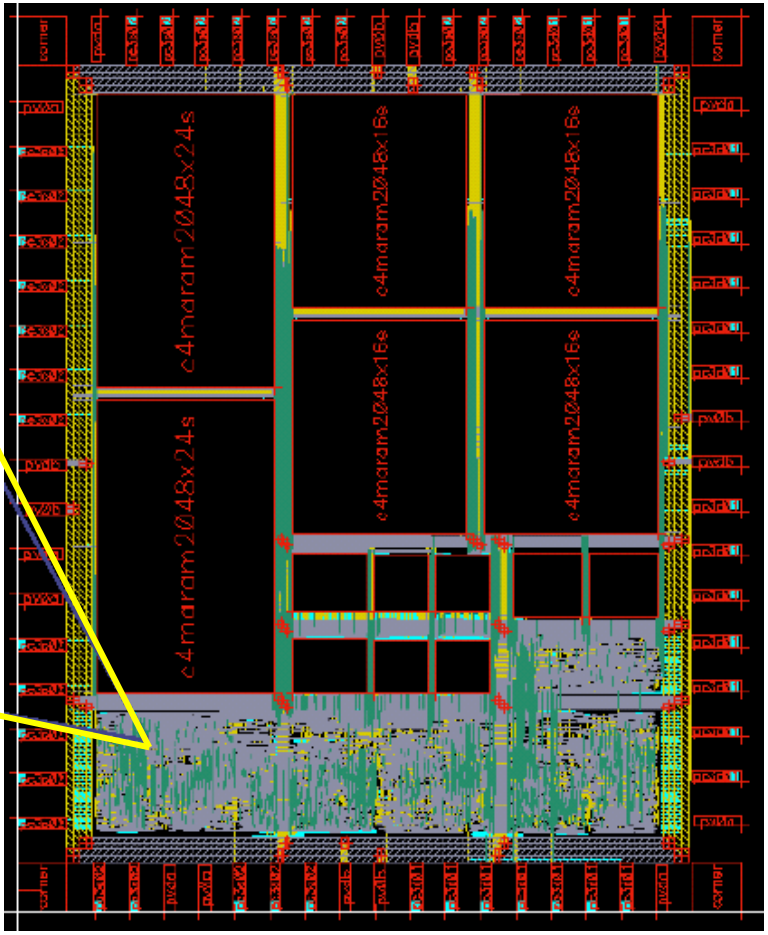




From CIC

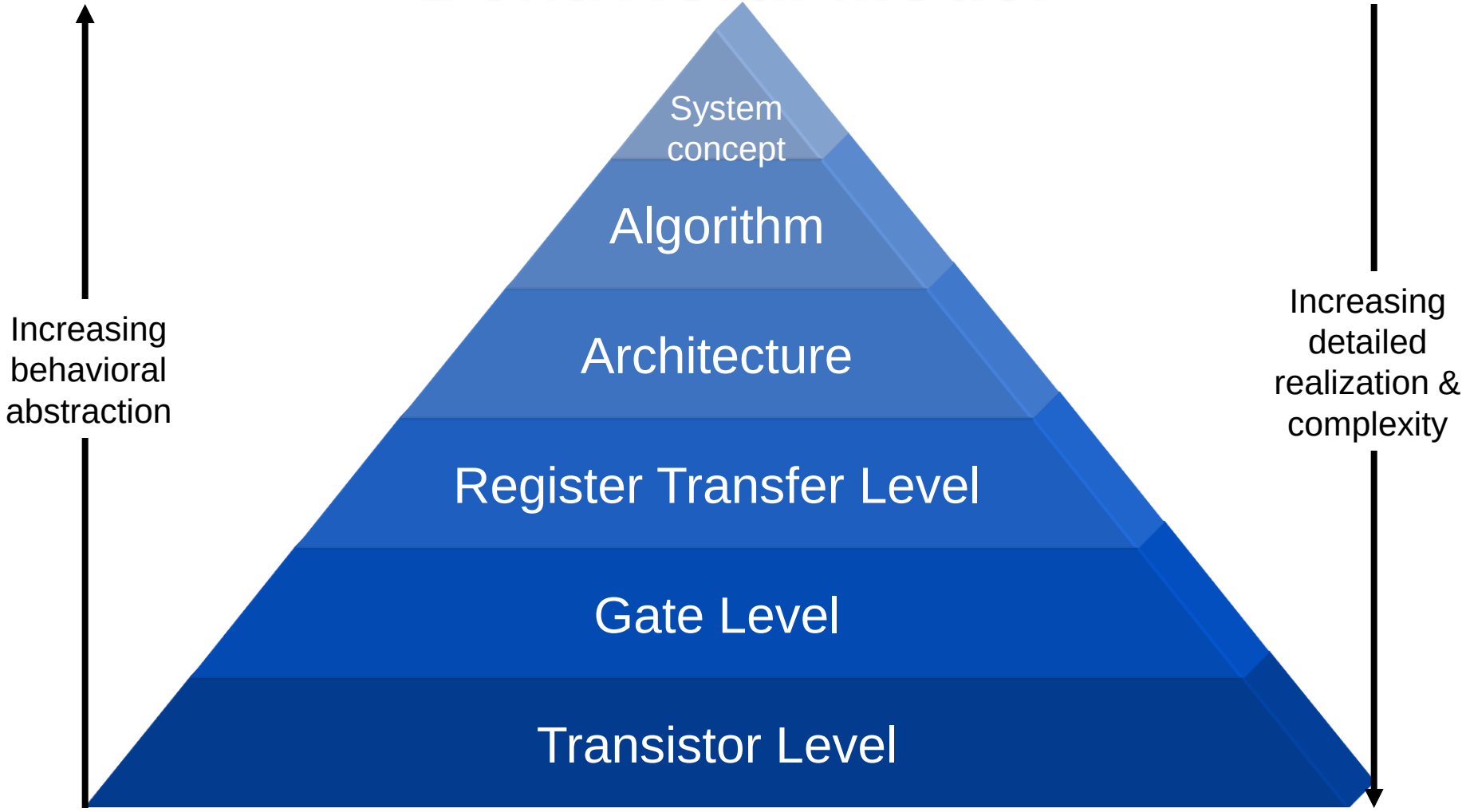
Physical Design



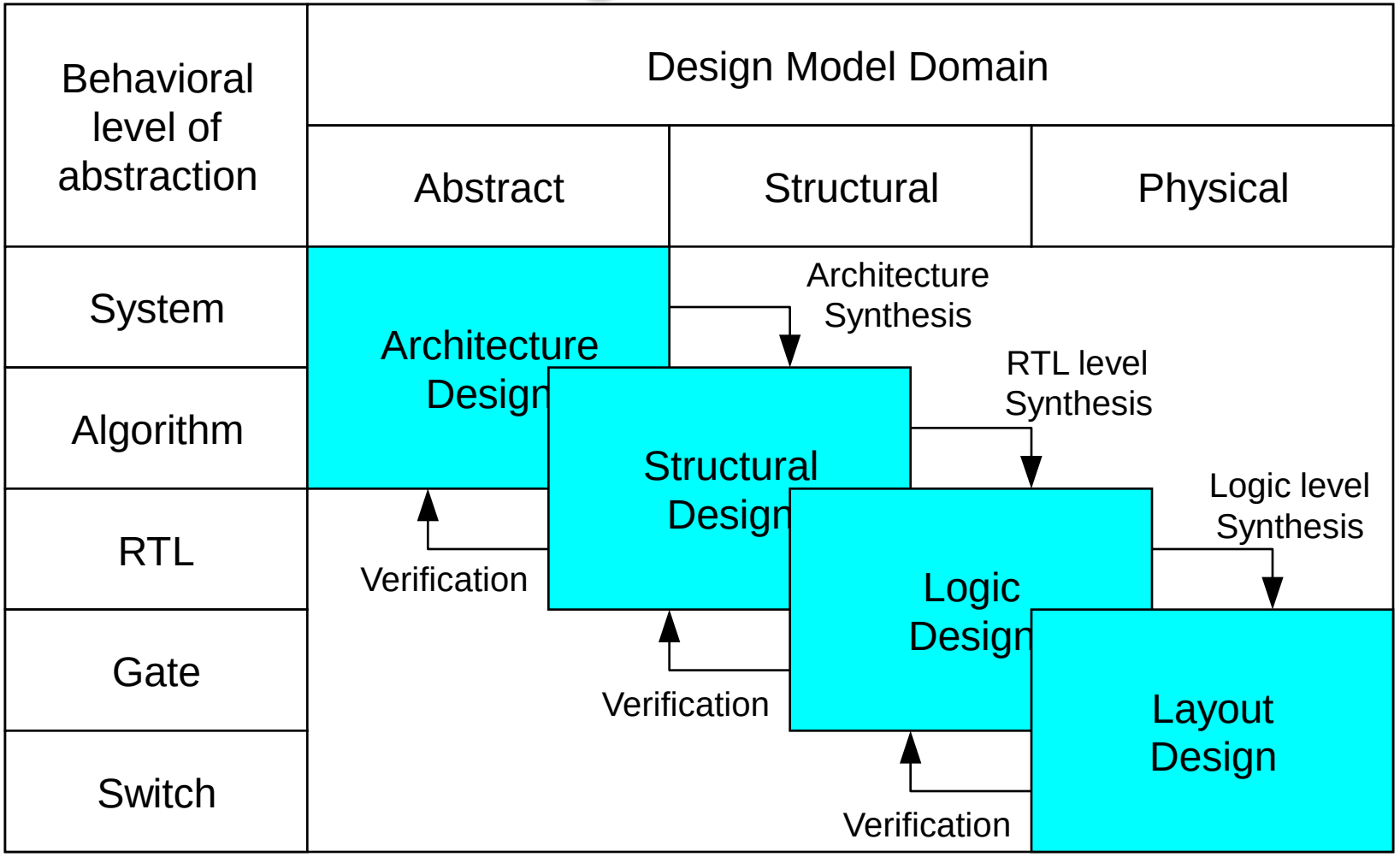


From CIC

Behavioral Model



Design Domain



Outline

- Introduction to Integrated Circuit
- IC Design Flow
- Verilog HDL concept
- Verilog Simulator

Introduction to HDL

- HDL – Hardware Description Language
- Why use an HDL ?
  - Hardware is becoming very difficult to design directly
  - HDL is easier and cheaper to explore different design options
  - Reduce time and cost

Verilog HDL

- **Features**
  - HDL has high-level programming language constructs and constructs to describe the connectivity of your circuit.
  - Ability to mix different levels of abstraction freely
  - One language for all aspects of design, test, and verification
  - Functionality as well as timing
  - Concurrency perform
  - Support timing simulation

Compared to VHDL

- Verilog and VHDL are comparable languages
- VHDL has a slightly wider scope
  - System-level modeling
  - Exposes even more discrete-event machinery
- VHDL is better-behaved
  - Fewer sources of non-determinism (e.g., no shared variables ???)
- VHDL is harder to simulate quickly
- VHDL has fewer built-in facilities for hardware modeling
- VHDL is a much more verbose language
  - Most examples don't fit on slides

The Verilog Language

- Originally a modeling language
  for a very efficient event-driven digital logic simulator
- Later pushed into use as a specification language
  for logic synthesis
- Now, one of the two most commonly-used languages
  in digital hardware design (VHDL is the other)
- Combines structural and behavioral modeling styles

Different Levels of Abstraction

- Architectural / Algorithmic
  - A model that implements a design algorithm in high-level language constructs
- Register Transfer Logic (RTL)
  - A model that describes the flow of data between registers and how a design process these data.
- Gate
  - A model that describe the logic gates and the interconnections between them.
- Switch
  - A model that describes the transistors and the interconnections between them.

Verilog HDL Behavior Language

- Structural and procedural like the C programming language
- Used to describe algorithm level and RTL level Verilog models
- Key features
  - Procedural constructs for conditional, if-else, case, and looping operations.
  - Arithmetic, logical, bit-wise, and reduction operations for expressions.
  - Timing control.

Verilog HDL Structural Language

- Used to describe gate-level and switch-level circuits.
- Key features
  - A complete set set of combinational primitives
  - Support primitive gate delay specification
  - Support primitive gate output strength specification

Language Conventions (1/2)

- Case-sensitivity
  - Verilog is **case-sensitive**.
  - Some simulators are case-insensitive
  - Advice: - Don't use case-sensitive feature!
  - Keywords are **lower case**
- Different names must be used for different items within the same scope
- Identifier alphabet:
  - Upper and lower case alphabeticals: a~z A~Z
  - Decimal digits: 0~9
  - Underscore: _

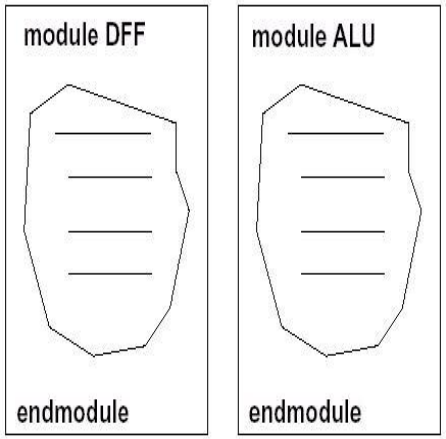
Language Conventions (2/2)

- Maximum of 1024 characters in identifier
- First character not a digit
- Statement terminated by ;
- Free format within statement except for within quotes
- Comments:
  - All characters after // in a line are treated as a comment
  - Multi-line comments begin with /* and end with */
- Compiler directives begin with // synopsys XXX
- Built-in system tasks or functions begin with $
- Strings enclosed in double quotes and must be on a single line "Strings"

Design Encapsulation

- Encapsulate structural and functional details

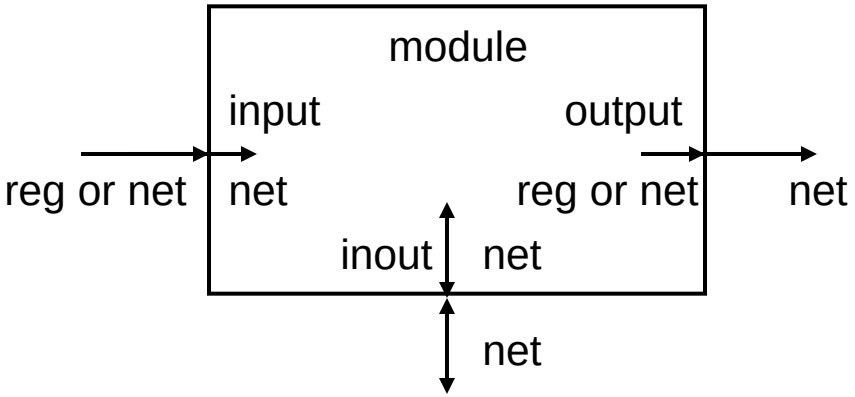
```
module my_design (ports_list);
... // Declarations of ports go here
... // Structural and functional details go here
endmodule
```



- Encapsulation makes the model available for instantiation in other modules

Port Declaration

- Three port types
  - Input port
    - input a;
  - Output port
    - output b;
  - Bi-direction port
    - inout c;



Port list

```
module full_add (S, CO, A, B, CI) ;

  output S, CO ;
  input A, B, CI ;

 --- function description ---

endmodule
```

Port Declaration

Two Main Data Types

- Nets represent connections between things
  - Do not hold their value
  - Take their value from a driver such as a gate or other module
  - Cannot be assigned in an *initial* or *always* block
- Regs represent data storage
  - Behave exactly like memory in a computer
  - Hold their value until explicitly assigned in an *initial* or *always* block
  - Never connected to something
  - Can be used to model latches, flip-flops, etc., but do not correspond exactly
  - Shared variables with all their attendant problems

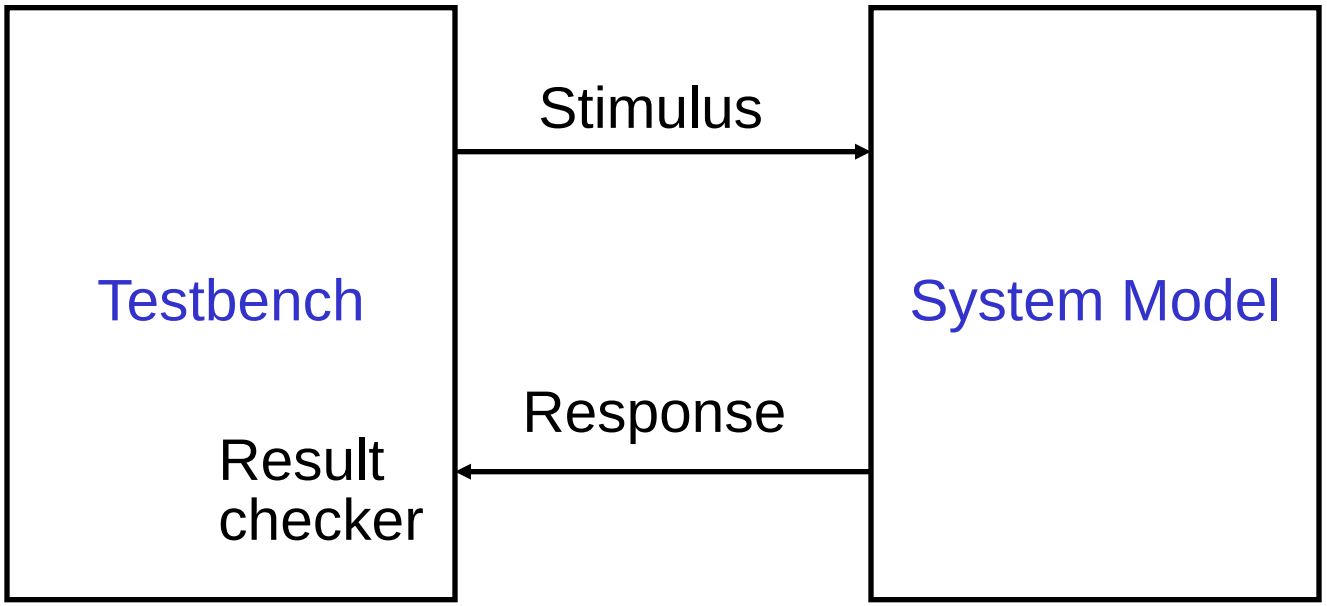
Primitive Cells

- Primitives are simple modules
- Verilog build-in primitive gate
  - *not, buf:*
    - Variable outputs, single input (last port)
  - *and, or, buf, xor, nand, nor, xnor:*
    - Single outputs (first port), variable inputs

```
module full_add (S, CO, A, B, CI) ;
  --- port Declaration ---
  wire    net1, net2, net3;
  xor  U0(S,A,B,CI);
  and  U1(net1, A, B);
  and  U2(net2, B, CI);
  and  U3(net3, CI, A);
  or   U4(CO, net1, net2, net3);
endmodule
```

How Are Simulators Used?

- Testbench generates stimulus and checks response
- Coupled to model of the system
- Pair is run simultaneously



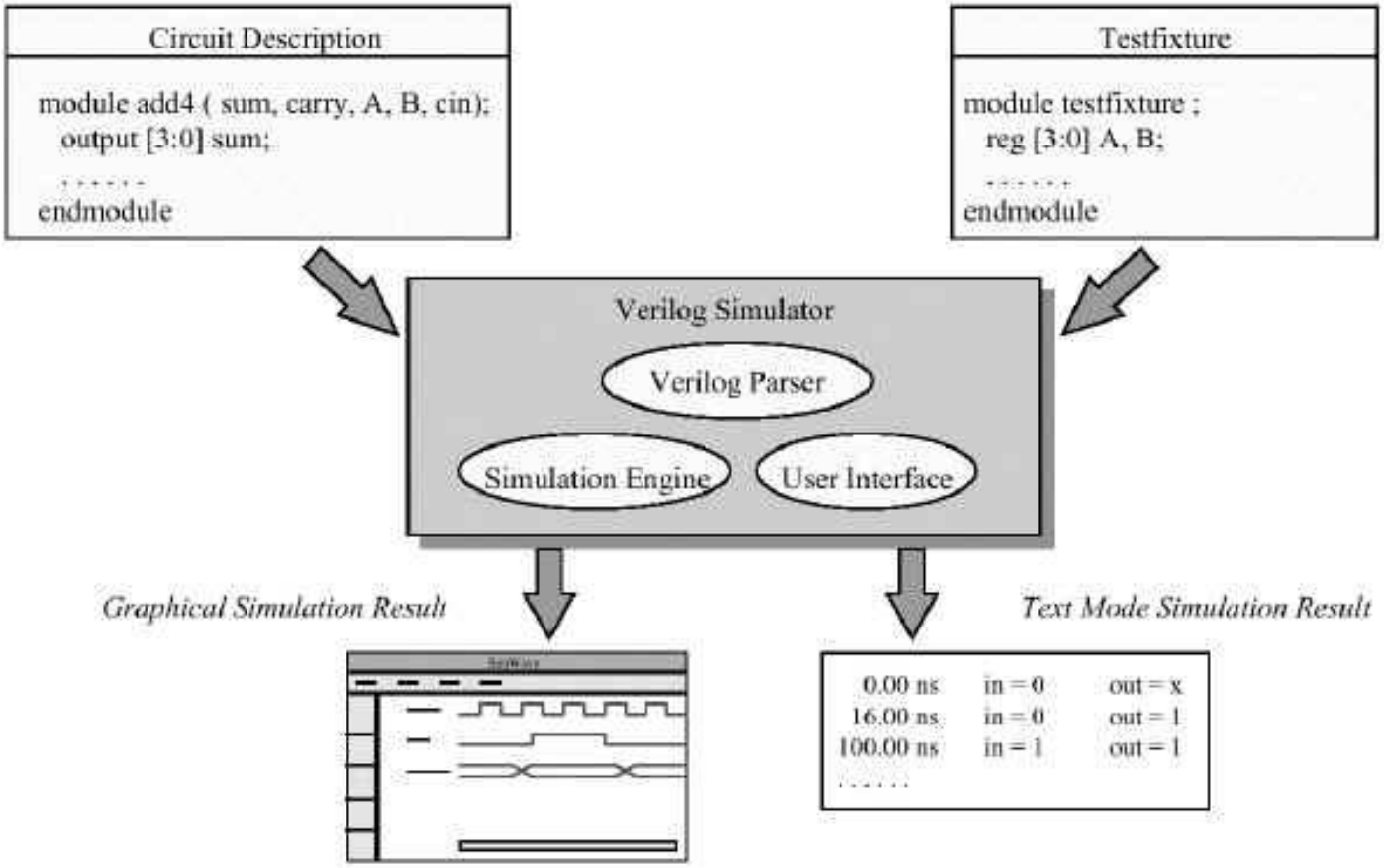
Example: Testbench

```
module t_full_add();
wire sum, c_out;
reg a, b, cin; // Storage containers for stimulus waveforms
full_add M1 (sum, c_out, a, b, cin); //UUT
initial begin // Time Out
#200 $finish; // Stopwatch
end
initial begin // Stimulus patterns
#10 a = 0; b = 0; cin = 0; // Statements execute in sequence
#10 a = 0; b = 1; cin = 0;
#10 a = 1; b = 0; cin = 0;
#10 a = 1; b = 1; cin = 0;
#10 a = 0; b = 0; cin = 1;
#10 a = 0; b = 1; cin = 1;
#10 a = 1; b = 0; cin = 1;
#10 a = 1; b = 1; cin = 1;
end
endmodule
```

Outline

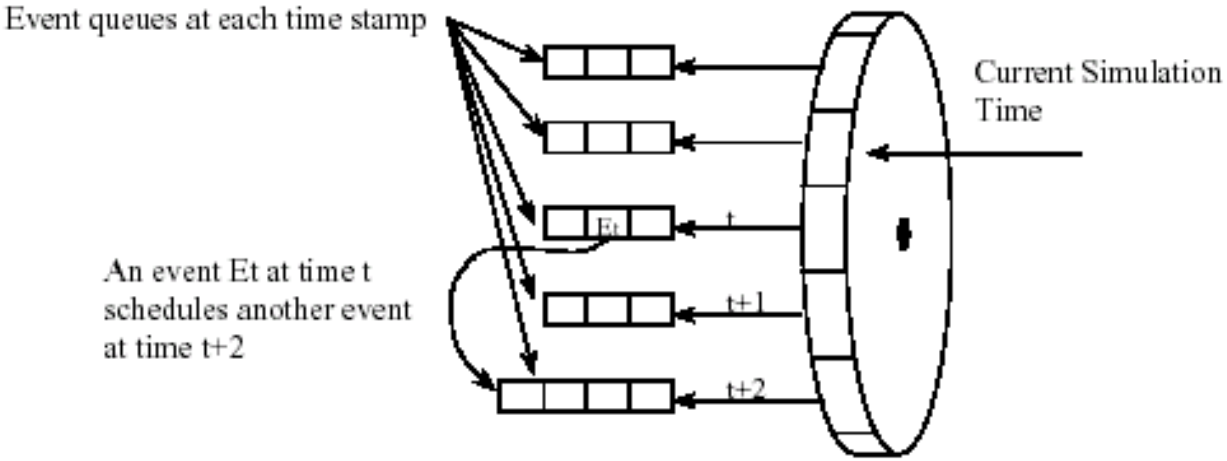
- Introduction to Integrated Circuit
- IC Design Flow
- Verilog HDL concept
- Verilog Simulator

Verilog Simulator



Event-Driven Simulation

- A change in the value of a signal (variable) during simulation is referred to as an *event*
- Spice-like analog simulation is impractical for VLSI circuits
- Event-driven simulators update logic values only when signals change



Styles

- Structural - instantiation of primitives and modules
- RTL/Dataflow - continuous assignments
- Behavioral - procedural assignments

Structural Modeling

- When Verilog was first developed (1984) most logic simulators operated on netlists
- Netlist: list of gates and how they're connected
- A natural representation of a digital logic circuit
- Not the most convenient way to express test benches

Behavioral Modeling

- A much easier way to write testbenches
- Also good for more abstract models of circuits
  - Easier to write
  - Simulates faster
- More flexible
- Provides sequencing
- Verilog succeeded in part because it allowed both the model and the testbench to be described together

Style 1 - Structural

```
module full_add (S, CO, A, B, CI) ;

output S, CO ;
input A, B, CI ;

wire N1, N2, N3;

half_add HA1 (N1, N2, A, B),
         HA2 (S, N3, N1, CI);

or P1 (CO, N3, N2);

endmodule
```

```
module half_add (S, C, X, Y);

output S, C ;
input X, Y ;

xor  (S, X, Y) ;
and (C, X, Y) ;

endmodule
```

Style 2 – Dataflow/RTL

```
module fa_rtl (S, CO, A, B, CI) ;

output S, CO ;
input A, B, CI ;

assign S = A ^  B ^ CI;                  //continuous assignment
assign CO = A & B | A & CI | B & CI;     //continuous assignment

endmodule
```

Style 3 – Behavioral

```
module fa_bhv (S, CO, A, B, CI) ;

output S, CO ;
input A, B, CI ;

reg S, CO;          // required to "hold" values between events.

always@(A or B or CI)                  //;
begin
  S <= A ^ B ^ CI;                     // procedural assignment
  CO <= A & B | A & CI | B & CI;       // procedural assignment
end
endmodule
```

How Verilog Is Used

- Virtually every ASIC is designed using either Verilog or VHDL (a similar language)
- Behavioral modeling with some structural elements
- "Synthesis subset"
  - Can be translated using Synopsys' Design Compiler or others into a netlist
- Design written in Verilog
- Simulated to death to check functionality
- Synthesized (netlist generated)
- Static timing analysis to check timing

Verilog-HDL Simulators

- Mentor Graphics ModelSim
  - http://www.mentor.com/
- Cadence Verilog-XL
- NC-Verilog