

硬體描述語言 Hardware Description Language (HDL)

蕭勝夫

Shen-Fu Hsiao

Room: EC5003

Outlines

- Course Information
- Overview
- Different Modeling Techniques
- Design Flows and Logic Synthesis

This course is **NOT** taught in English.

It is offered in **Chinese (Mandarin)**.

Course Information

Topics

- Verilog (IEEE std. 1364)
- VHDL (IEEE std. 1076)
- Logic Synthesis
- SystemVerilog (IEEE std. 1888, if time allows)

Course Schedule

- Introduction (~1 week)
- Verilog-1, structural and dataflow modeling (~2 weeks)
- Verilog-2, behavioral modeling (~2 weeks)
- Verilog-3, logic synthesis and design flow (~1 week)
- Verilog-4, codes for basic components (~1 week)
- Verilog-5, pipelined THUMB (~1 week)
- Verilog-6: other modeling methods (~0.5 week)
- Verilog-7: Verilog 1995 vs. 2001 (~0.5 week)
- Verilog-8: code for advanced components (~1 week)
- VHDL-1, data types (~1 week)
- VHDL-2, concurrent and sequential codes (~1 week)
- VHDL-3, components, packages, and configuration (~1 week)
- Verilog vs. VHDL (~1 week)

References

Lectures notes available in 中山課程網 (cyber university)
<http://cu.nsysu.edu.tw/>

- Verilog
 - S. Palnitkar, “**Verilog HDL, A Guide to Digital Design and Synthesis**”, 2nd ed., , Sun Microsystems, Inc, 2003.
 - <http://www.sutherland-hdl.com/index.html> (SystemVerilog)
- VHDL (suggested textbook for VHDL)
 - V. A. Pedroni, “**Circuit Design and Simulation with VHDL**”, 2nd ed., MIT Press, 2010.
- Coding Styles
 - M. Keating and P. Bricaud, “**Reuse Methodology Manual, For System-on-a-Chip Designs**”, 3rd ed., Kluwer Academic Pub., 2002.
- Verification
 - W. K. Lam, “**Hardware Design Verification, Simulation and Formal Method-Based Approaches**”, Pearson Education, Inc., 2005.
- IEEE standards (available at IEEEXplore database)
 - Verilog (IEEE standard1364), VHDL (IEEE standard 1076).

Grading

- Homework (60%)
 - 5+1(bonus) homework assignments
 - HW1: three modeling methods
 - HW2: pipelined add/sub-multiply unit
 - HW3: 16-bit RISC CPU debug and APR
 - HW4: image edge detection
 - HW5: acceleration of CNN operations
 - HW6 (bonus): memory compiler for CNN accelerator
 - 2~3 weeks for each homework
- Examinations (40%)
 - mid-term exam
 - final exam

EDA (Electronic Design Automation) Tools

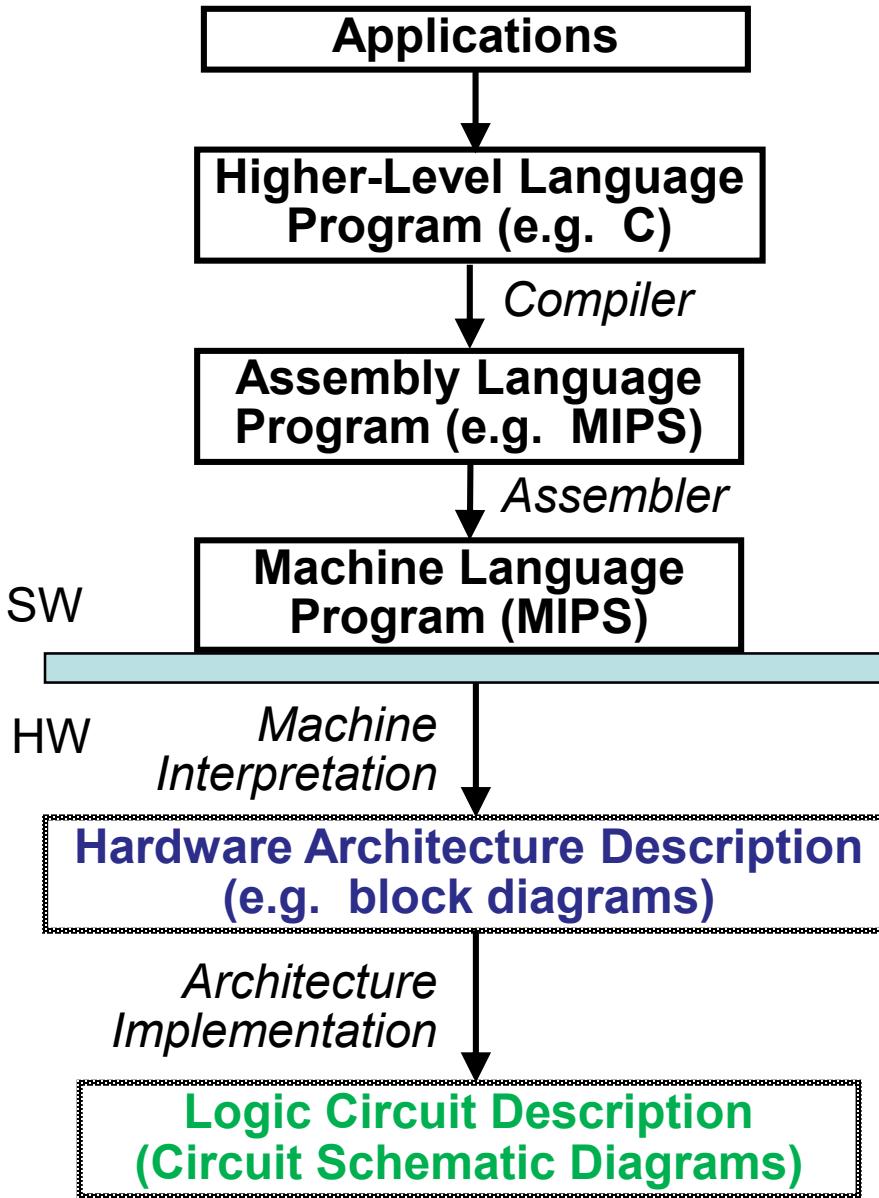
- vcs, NC-Verilog, ModelSim, Verilog-XL, ...
 - simulation for HDLs (Verilog and VHDL)
- Design Compiler (for *front-end* cell-based ASIC design flow)
 - logic synthesis from Register-Transfer Level (RTL) to gate netlists
 - delay/power report from static timing analysis (STA)
- PrimeTime (dynamic timing analysis)
 - power and timing simulation with user provided inputs
- Verdi (for debug)
- Xilinx Vivado (for FPGA design flow)
 - simulation and synthesis for quick prototyping on FPGA chips
- Innovus, IC Compiler (for *back-end* cell-based design flow)
 - automatic placement and routing (P&R) to generate physical layout from gate netlists
- SRAM Memory Compiler

Course website and TA

- Lecture notes available
中山課程網 <http://elearn.nsysu.edu.tw/>
- Teacher: 蕭勝夫 Shen-Fu Hsiao
 - Email: sfhsiao@cse.nsysu.edu.tw
- TA (Teaching Assistant)
 - 丁祥益 m133040045@g-mail.nsysu.edu.tw
 - 邱奕綸 chiu91523@gmail.com
 - 王敬翔 jimmyhsiang93@gmail.com
 - 王鼎睿 read94385@gmail.com
 - Room: EC5015
- on-site course in Chinese 實體授課

Overview

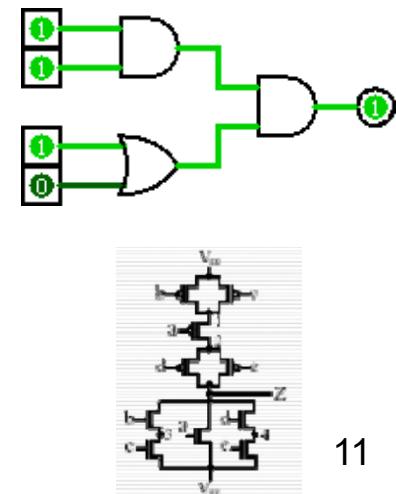
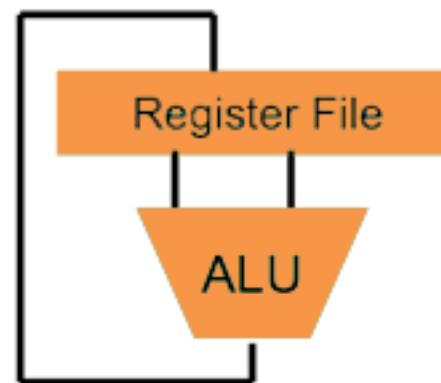
Design Hierarchy



temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

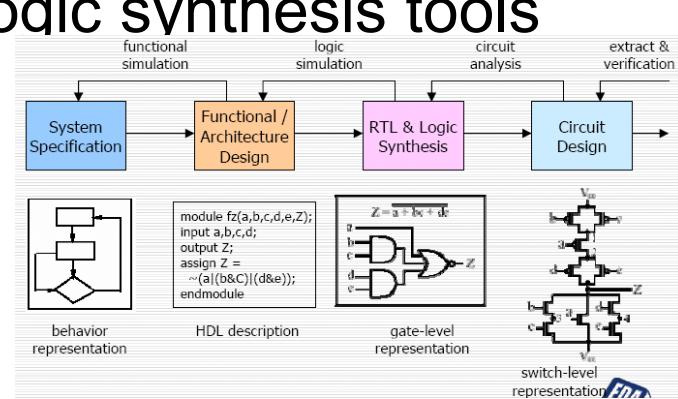
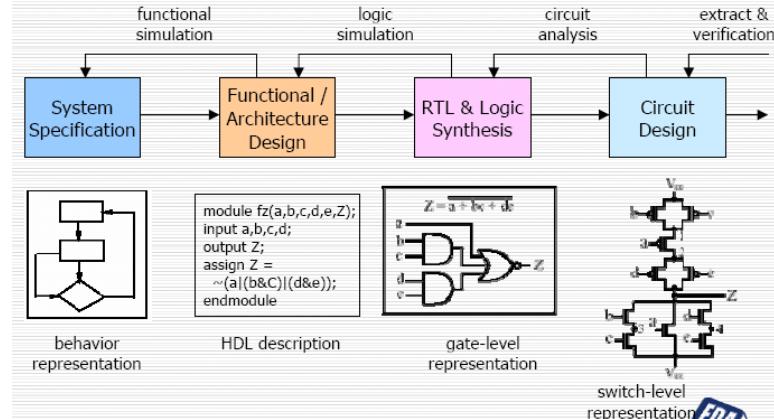
lw \$t0, 0(\$2)
lw \$t1, 4(\$2)
sw \$t1, 0(\$2)
sw \$t0, 4(\$2)

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111



Different Levels of Hardware Design

- system level
 - C, SystemC, SystemVerilog
- Register Transfer Level (RTL)
 - Verilog, VHDL
- gate level
 - usually generated from RTL using logic synthesis tools
 - ✓ e.g., Synopsys Design Compiler (DC)
 - ✓ e.g., Xilinx Vivado synthesis
 - ✓ front-end cell based design flow
- physical level
 - usually generated from gate-level using placement-and-routing (P&R) tools
 - ✓ e.g., Synopsys IC Compiler, Cadence Innovus
 - ✓ back-end cell-based design flow



Digital vs. Analog Design

- **digital** IC design (digital chip design)
 - consider only logic 1 and 0
 - core courses: digital system (logic design), HDL
 - usually start with high level HDL
 - generate final IC using logic synthesis and P&R tools
- **analog** IC (chip) design
 - consider voltage/current
 - core courses: electronics, VLSI design
 - usually in Input/Output (I/O) interface, like sensors
 - Use ADC (Analog-to-Digital Converter) or DAC to pass signal to or receive signals from digital circuits
 - usually start with circuit and layout

Combinational vs. Sequential Logic

- **Combinational Logic**

- Logic circuits with pure feed-forward datapath
 - No flip flops or latches, e.g.,

```
module adder (input wire [7:0], a, b, output wire [7:0] c);  
    assign c = a + b; // dataflow modeling  
endmodule
```

- **Sequential Logic**

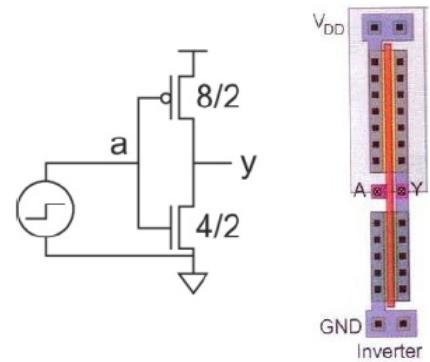
- Logic circuits with storage elements such as latches and/or flip-flops, e.g.,

```
module lateched (input [7:0], a, b, input clk, output reg [7:0] c);  
    always @ (posedge clk) // behavioral modeling  
        c <= a + b;  
endmodule
```

IC Design Flows

- **full-custom**

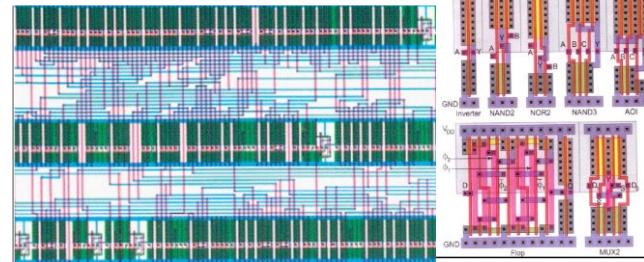
- design entry: circuit schematic and layout with W/L
- for optimized small circuits or analog circuits
- EDA tools: circuit simulator, layout editor, DRC, ERC, LVS
- related course: VLSI Design



- **cell-based (ASIC)**

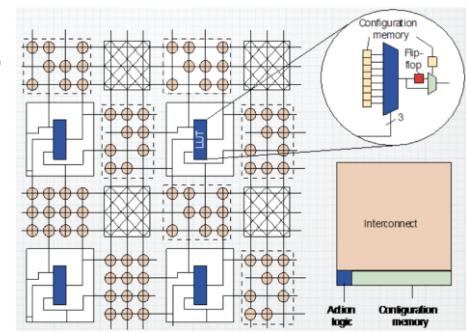
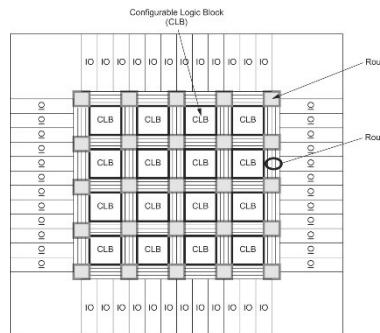
- design entry: **Verilog/VHDL**
- EDA tools: HDL simulator, logic synthesis, P&R, post-layout simulator
- for large digital systems
- related course: HDL

```
Verilog
module flop (input      clk,
              input [3:0] d,
              output reg [3:0] q);
    always @ (posedge clk)
        q <= d;
endmodule
```

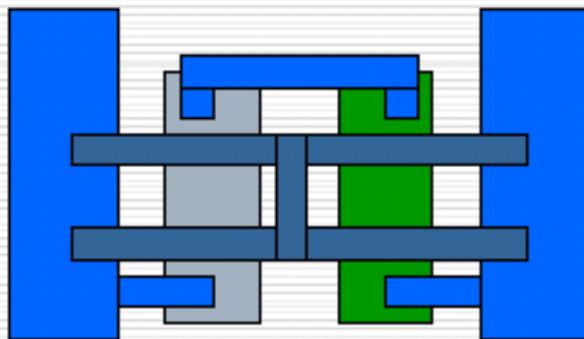


- **FPGA (on pre-fabricated chip)**

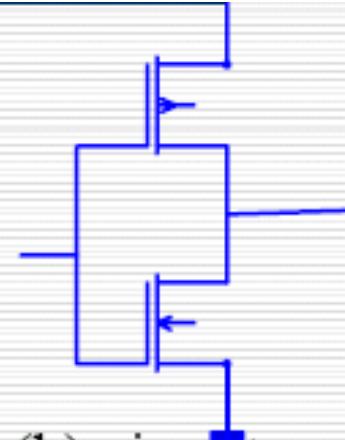
- design entry: **Verilog/VHDL**
- EDA tools: from FPGA company (such as Xilinx)
- for quick prototyping of large digital systems
- re-programmable
- related course: FPGA, SoPC



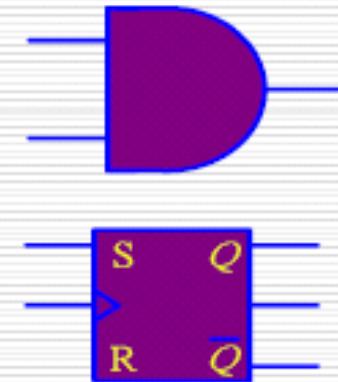
Design on Different Levels



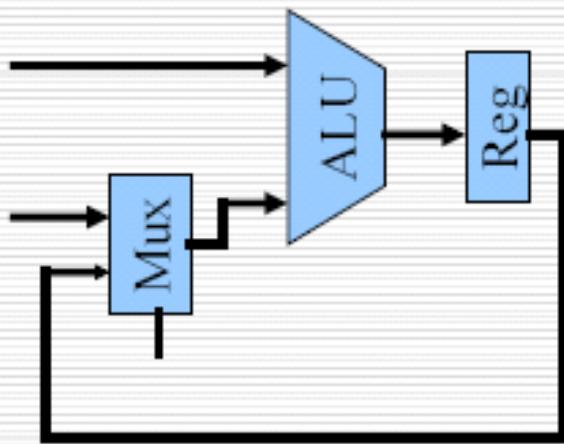
(a) silicon



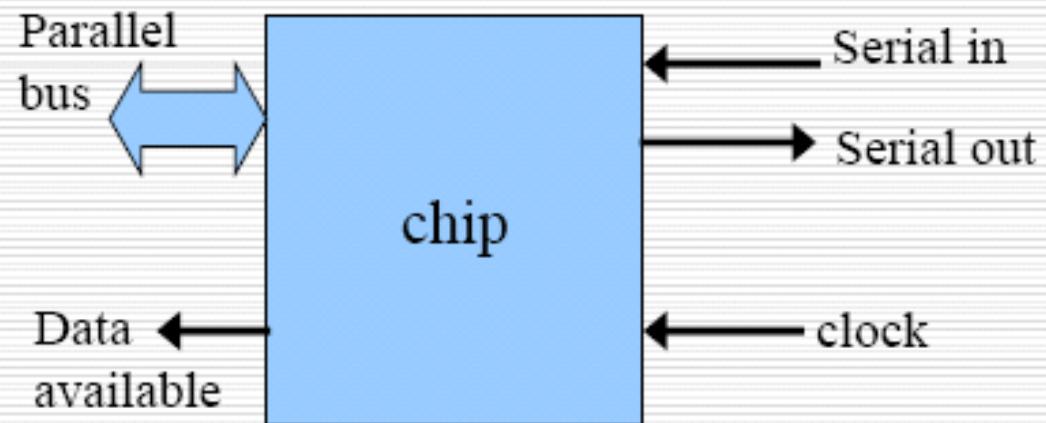
(b) circuit



(c) gate F-F



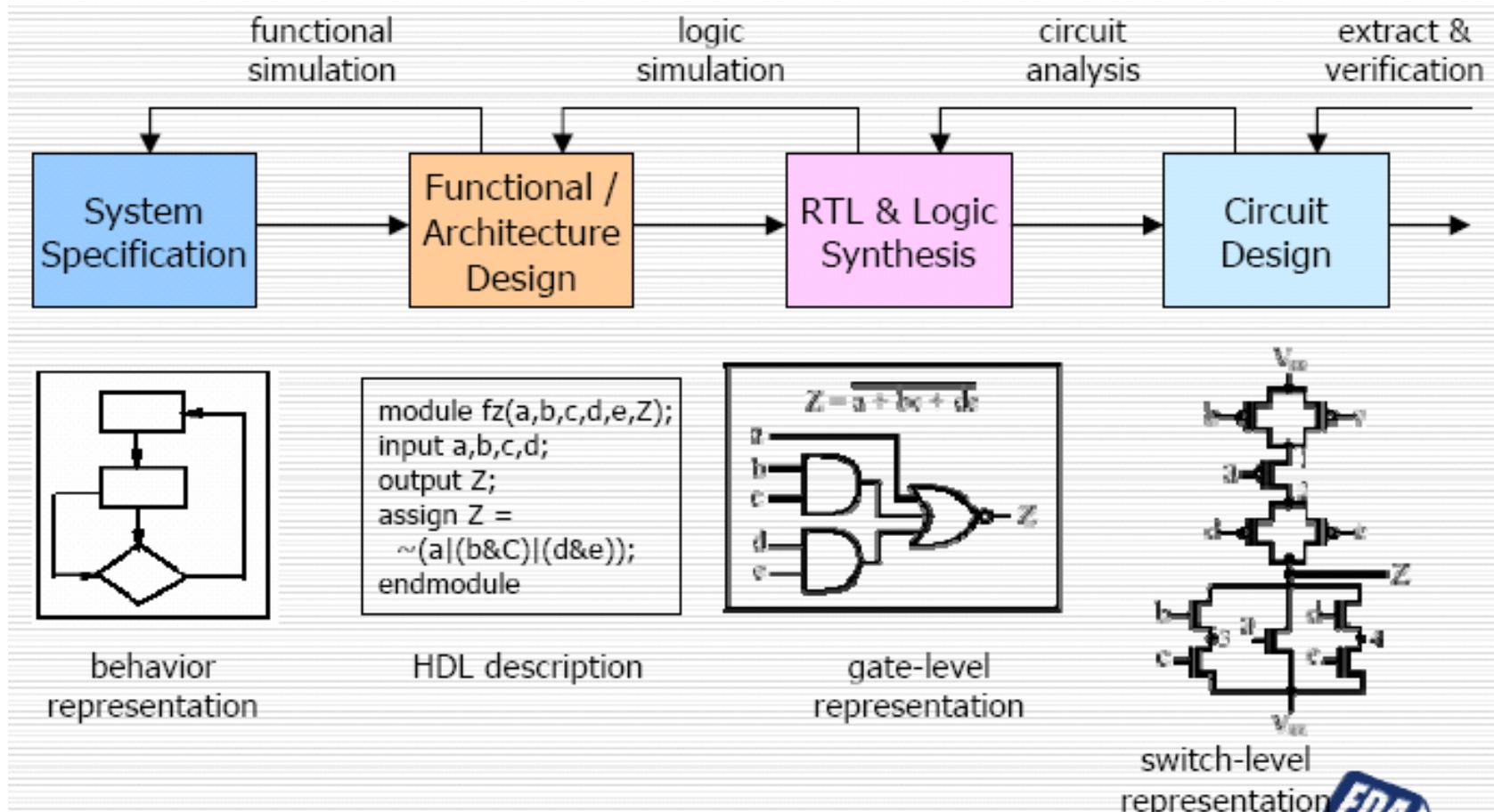
(d) Register



(e) Chip

Typical Cell-Based Design Flow(1/2)

- system -> RTL -> gate -> transistors
 - logic synthesis (e.g., Synopsys Design Compiler) to convert RTL into gate-level



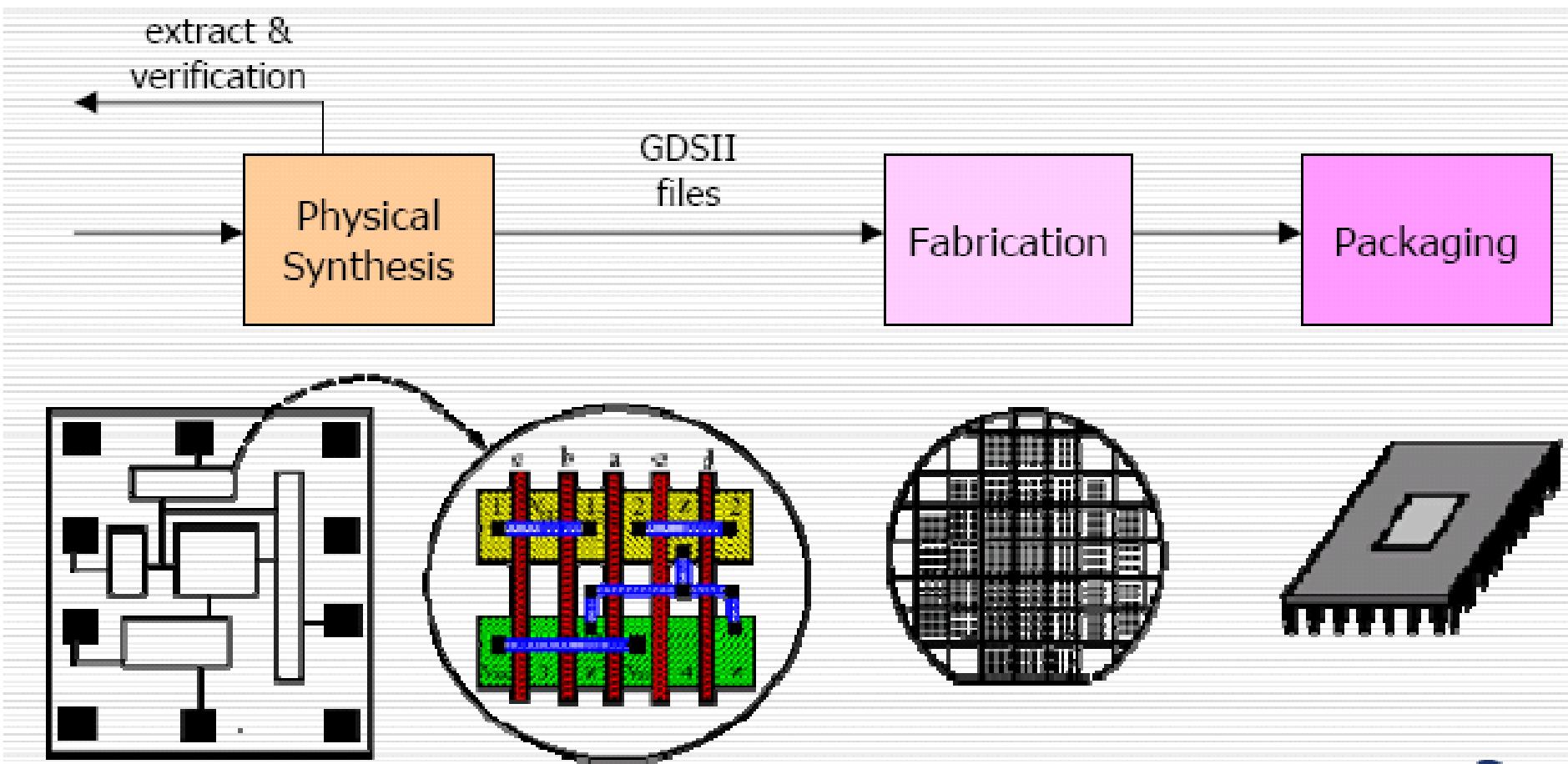
Typical Cell-Based Design Flow (2/2)

placement & routing (P&R)
(Innovus, IC Compiler)

fabrication
(台積電)

packaging
(日月光)

- floorplan -> layout -> chip -> package



Hardware vs. Software

	SW (Software)	HW (Hardware)	courses
Abstraction	Application programs	Digital system design	
Highest level	Flow chart	Function block diagram	
	C (sequential)	Verilog (parallel)	HDL
	Assembly	Logic	Digital system
	Assembly	Circuit	VLSI Design
Lowest level	Binary machine code	Layout	VLSI Design
Tools	Complier e.g., gcc -O optimization level	Logic synthesizer e.g., Synopsys Design Compiler, constraints (delay, power, area)	HDL
Execution	CPU	Tape-out chip (IC), or Pre-fabricated PGA chip	

Always think of hardware architecture when writing HDL codes !!!
Better to have an architecture diagram before coding.

Hardware Description Language (HDL)

- HDL
 - originally for purpose of simulation
 - later for design (only *synthesizable* subset of HDL)
 - allows design to be described at a higher abstract level (such as behavioral level)
 - functional verification of the design can be done early in the design cycle (RTL simulation)
 - logic synthesis tools (such as Synopsys Design Compiler) automatically convert design into gate-level netlist of a selected fabrication technology
- **Verilog** HDL : IEEE standard 1364)
 - U.S.A. (Silicon Valley), Japan, and **Taiwan**
- **VHDL** (VHSIC HDL) : IEEE standard 1076
 - U.S.A. (IBM, Intel, TI), Europe, Korea

Quick History of HDLs

- ISP (circa 1977) - research project at CMU
 - Simulation, but no synthesis
- Abel (circa 1983) - developed by Data-I/O
 - Targeted to programmable logic devices
 - Not good for much more than state machines
- Verilog (circa 1985) - developed by Gateway (now Cadence)
 - Similar to Pascal and C
 - Fairly efficient and easy to write
 - **IEEE standard 1364 (in year 1995)**
- VHDL (circa 1987) - DoD sponsored standard
 - Similar to Ada (emphasis on re-use and maintainability)
 - Very general but verbose
 - **IEEE standard 1076 (in year 1987)**

History of Verilog

- 1983: Gateway release Verilog HDL
- 1989: Cadence bought Gateway
- 1990: Cadence released Verilog to public
- 1993: 85% of ASIC designs using Verilog
- **1995**: IEEE adopted as standard 1364
- **2001**: major updated as IEEE 1364-2001
- 2005: minor revision

*Three largest EDA vendors: Synopsys, Cadence, Mentor Graphics

History of VHDL

- 1983: IBM, TI, DoD promote VHDL
- **1987**: IEEE standard 1076
- **1993**: revised as IEEE standard 1076'93
- 2000, 2002, 2008: new versions
- IEEE 1076.3-1997: VHDL package for synthesis
- IEEE 1076.1-1997: VHDL-AMS
 - support analog, digital and mixed signal simulation

HDL: Verilog, VHDL

- Both are IEEE “standard” HDL languages
- EDA (Electronic Design Automation) tools provide front-ends (simulation and synthesis) to both HDLs
- Verilog is “simpler”
 - Less syntax, fewer constructs, more like C
- VHDL supports large, complex systems
 - Better support for modularization
 - More details
- Both will be taught in this course

* Complete document of all IEEE standards are available from IEEEExplore database <http://ieeexplore.ieee.org/>

VHDL vs. Verilog

- Verilog HDL
 - U.S.A. (Silicon Valley), Japan, and **Taiwan**
- VHDL
 - U.S.A. (IBM, Intel, TI), **Europe**, Korea
- Both Verilog and VHDL (~40%)
 - Legacy code, Intellectual property (IP) block

	VHDL	Verilog
Compilation	Compile	interpretative
Libraries	Yes	No
Resuability	Package	Include
Readability	ADA	C & ADA
Easy to Learn	Less intuitive	EASY

32-bit adder (Verilog vs. VHDL)

- Verilog

```
// case-sensitive
// i.e., a != A
module adder32 (input [31:0] a, b,
                 output [31:0] c);
    assign c = a + b;
endmodule
```

- VHDL

```
-- VHDL is case-insensitive, i.e. a = A
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_signed.all;

entity adder32 is
    port (a, b: in std_logic_vector(31 downto 0);
          c: out std_logic_vector(31 downto 0));
end;

architecture adder32_arch1 of adder32 is
begin
    c <= a + b;
end;
```

RTL vs. System-level Modeling

- **RTL (Register Transfer Level)** modeling
 - **Verilog** (IEEE std.1364)
 - **VHDL** (IEEE std.1076)
 - EDA tools available for automatic synthesis to hardware
- System-level modeling
 - More higher (abstract) level modeling
 - for HW/SW co-simulation/verification in ESL (Embedded System Level)
 - **SystemC** (IEEE-1666)
 - Similar to advanced C++
 - **SystemVerilog** (IEEE-1888)
 - Superset of Verilog
 - Automatic EDA synthesis tool either not available or not mature yet
 - Xilinx HLS (High Level Synthesis) could accept high-level language such as C as input, but still need some manual efforts

Different Modeling Techniques

Parallel Coding in Hardware Coding

- hardware programming is basically parallel coding
 - order of codes does NOT matter
 - e.g., describe two hardware units: adder and multiplier

```
...  
assign c=a+b; // descrbe adder first  
assign out=c*d; // then describe multiplier  
...
```

```
...  
assign out=c*d; // describe multiplier  
assign c=a+b; // then describe adder  
...
```

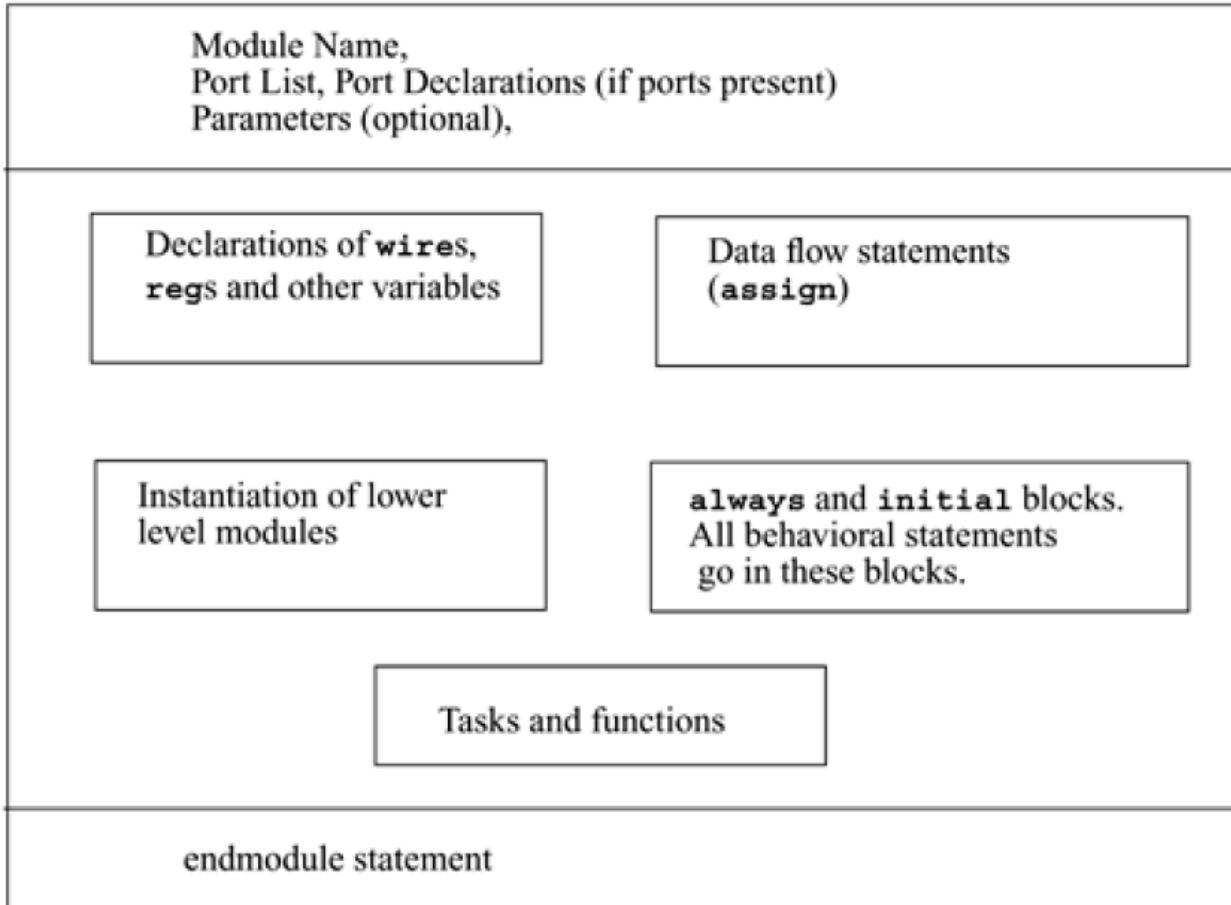
- software programming is usually sequential
 - order of codes matters

```
...  
c=a+b;  
out=c*d; // out = (a+b) *d  
...
```

```
...  
out=c*d; // out = old_c *d  
c=a+b; // new_c = a+b  
...
```

Components of a Verilog Module

- declaration
 - inputs, outputs
 - parameters
 - data types
- ***structure*** model
- module instantiation
- ***dataflow*** model
 - continuous assignment
- ***behavior*** model
 - procedural assignment



```

module M (P1, P2, P3, P4);
input P1, P2;
output [7:0] P3;
inout P4;

reg [7:0] R1, M1[0:1023];
wire W1, W2, W3, W4;
wire [3:0] W5;
parameter C1=const;

// non-synthesizable coding
// used in testbench
initial
begin : blockname
  // statements
end

// behavioral modeling
always
begin
  // statements
end

```

```

// dataflow: continuous assignment
assign W1 = expr ; // use operators

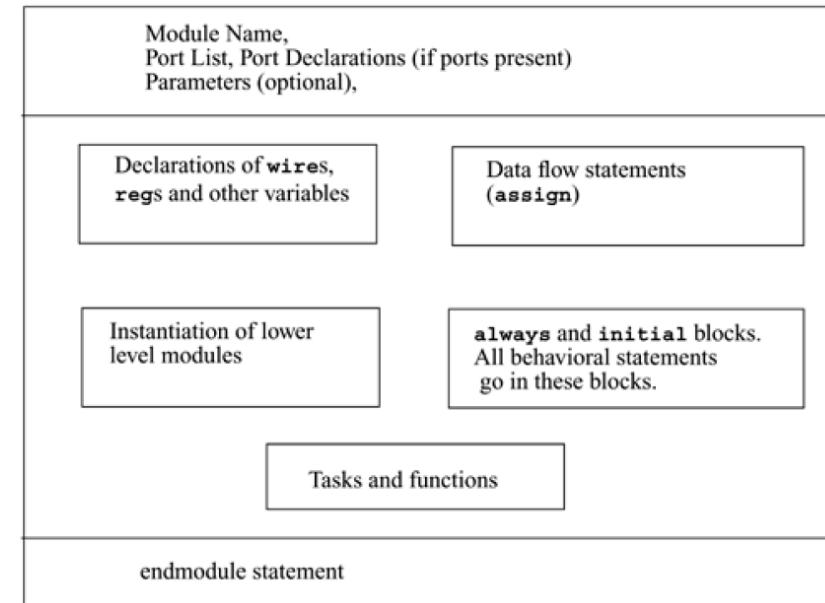
// module instances: structural modeling
COMP U1 (.PP2(W2), .PP1(W1));
COMP U2 (W3, W4);

task T1;
input A1, A2;
output A3, A4;
begin
  // statements
end
endtask

function [7:0] F1;
input A1;
begin
  // statements
end
endfunction

endmodule

```



example 1: Full Adder (FA)

- definition
 - inputs: a, b, cin
 - outputs: s, cout
- Boolean functions
 - $s = a \text{ xor } b \text{ xor } \text{cin}$
 - $\text{cout} = (\text{a and b}) \text{ or } (\text{b and cin}) \text{ or } (\text{a and cin})$
-

input			output	
a	b	cin	s	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Structural Modeling for FA

- describe the structure of FA
 - *instantiate* other lower-level module or gates
 - FA composed of XOR gates, AND gates, and OR gates
 - .e.g., Verilog built-in gate primitives **and**, **or**, **xor**

```
module FA_structure (input a, b, cin output s, cout);
wire w1, w2, w3, w4; // internal nets
xor X1 (w1, a, b); // module instantiation with instance name X1
xor X2 (s, w1, cin);
and A1 (w2, a, b);
and A2 (w3, b, cin);
and A3 (w4, a, cin);
or O1 (cout, w2, w3, w4);
endmodule
```

Dataflow Modeling for FA

- use operators (logical or arithmetic)
 - logical operators: \sim , $\&$, $|$, \wedge
 - arithmetic operators: $+$, $-$, $*$
- *continuous assignment*
 - start with Verilog keyword: **assign**

```
module FA_dataflow (input a, b, cin output s, cout);
  assign s = (a^b)^cin;
  assign cout = (a & b) | (b & cin) | (a & cin); // use logical operators
// assign {cout, s} a + b + cin; // use arithmetic operator, suggested
endmodule
```

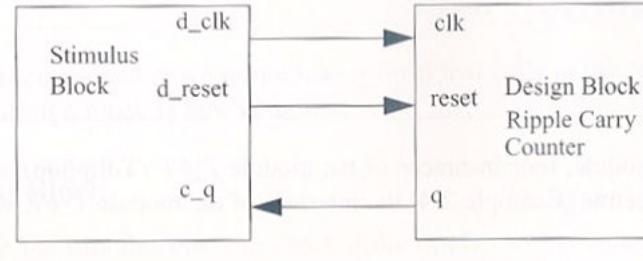
Behavioral Modeling for FA

- describe the high-level behavior (function)
 - use Verilog keyword **always**
 - left-hand-side (LHS) of assignment must be **reg** datatype
- procedural assignments

```
module FA_behavior (input a, b, cin output reg s, cout);
  always @ (a or b or cin)
    {cout, s} = a + b + cin; // suggested coding
    // s = (a^b)^cin;
    // cout = (a & b) | (b & cin) | (a & cin);
endmodule
```

Simulation

- Verify the hardware design
- Design block
 - must be described with ***synthesizable*** HDL codes
- Stimulus block
 - apply inputs to the hardware and compare hardware output with the expected results
 - could be in ***non-synthesizable*** HDL codes



```

`timescale 1ns/10ps // time unit / time precision
module testbench; ...
FA_structure FA_s (a, b, cin, s, cout); // instantiate hardware under test
...
for (i=1; i<=100; i=i+1) begin // test 100 times
  #10 a={random}%2; b={random}%2; cin={random}%2; // after 10 units, apply test patterns
  {expected_cout, expected} = a + b + cin;
  if (s != expected | cout != expect_count) error_count = error_count + 1; ...
end
...
endmodule
  
```

example 2: two-to-one multiplexer (MUX2)

- definition
 - inputs: in0, in1, s
 - outputs: cout

input			output
s	in0	in1	out
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

- Boolean functions
 - $s' = \text{not}(s)$
 - $\text{out} = (\text{in0 and } s') \text{ or } (\text{in1 and } s)$

Verilog codes for MUX2

```
module MUX2_structure (input in0, in1, s, output out);
wire sb, w0, w1;
not NOT1 (sb, s)
and A0 (w0, in0, sb);
and A1 (w1, in1, s);
or O1 (out, w0, w1);
endmodule
```

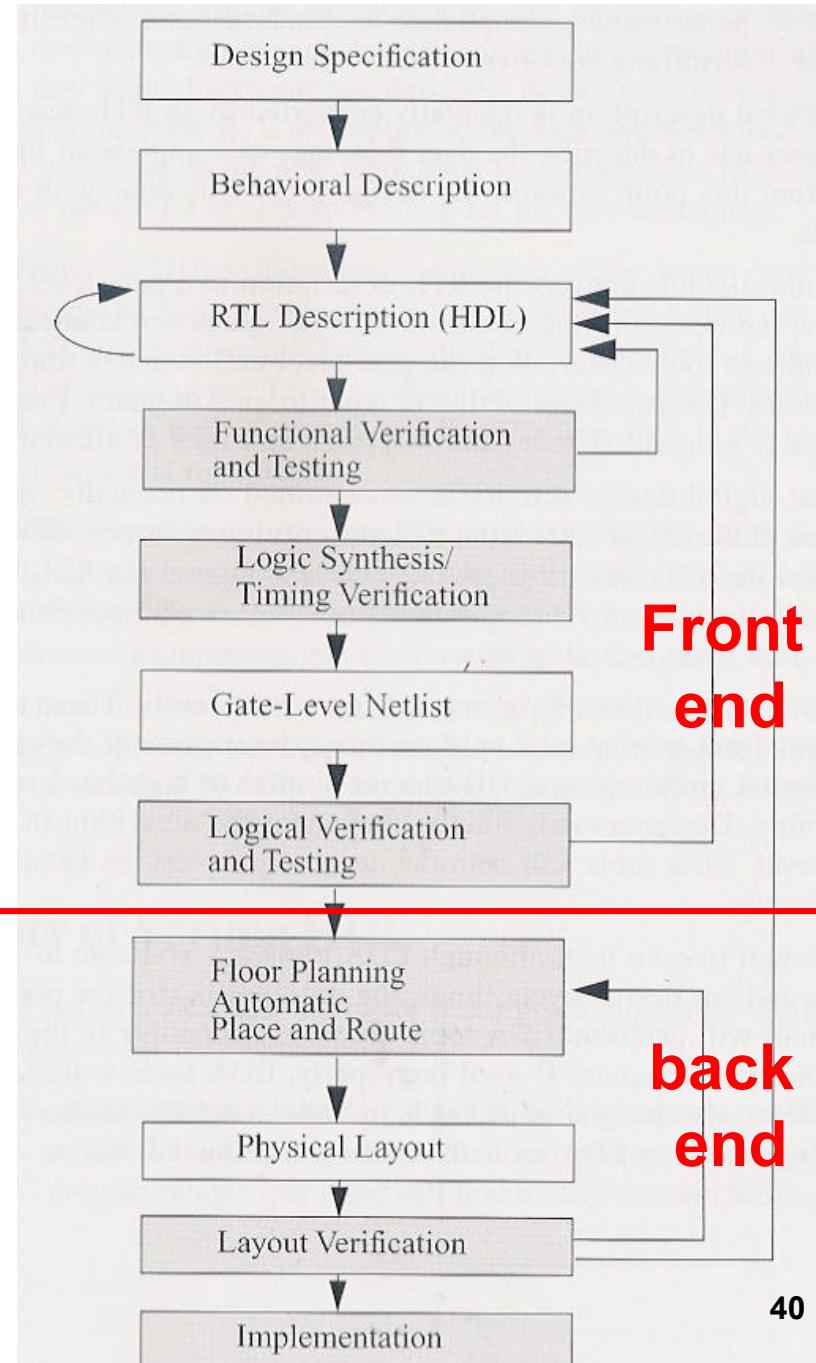
```
module MUX2_dataflow (input in0, in1, s, output out);
assign s = (in0 & (~s)) | (in1 & s);
// assign out = s ? in1 : in0; // conditional operator ? : , suggested
endmodule
```

```
module MUX2_behavior (input in0, in1, s, output reg out);
always @ (in0 or in1 or s)
  if ( s == 1) out = in1 else out = in0; // suggested
endmodule
```

Design Flows and Logic Synthesis

Typical Design Flow

- Front-end
 - RTL coding/simulation
 - Synthesis
 - Gate-level simulation (post-synthesis simulation)
- Back-end (ASIC)
 - Place and route (P&R)
 - Post-layout simulation
 - tapeout to foundry (e.g. TSMC)
- Back-end (FPGA)
 - Mapping and implementation on pre-fabricated FPGA chip

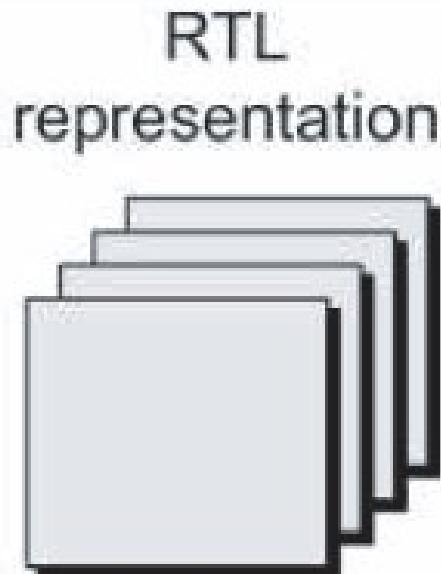


Importance of HDL

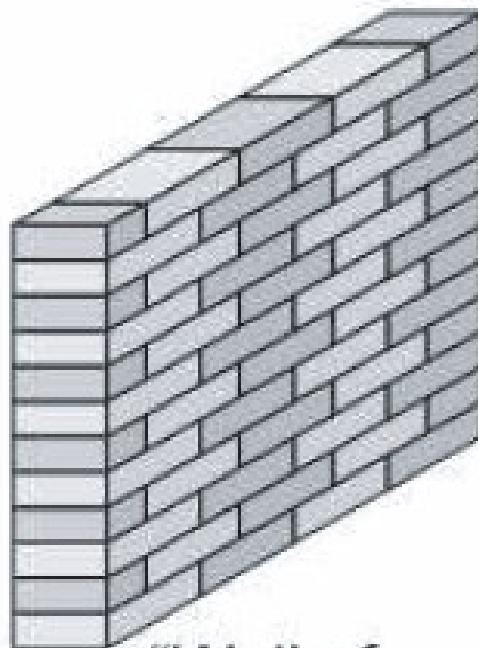
- designs can be described at register transfer level (RTL)
- logic synthesis tools automatically convert the RTL into gate-level netlist (logic gates and their interconnections)
- SystemVerilog, SystemC
 - used for system-level design in SOC
 - push the design entry higher than RTL
 - efficient tools are still emerging to convert system-level codes into RTL for HW implementation

RTL vs. gate-level

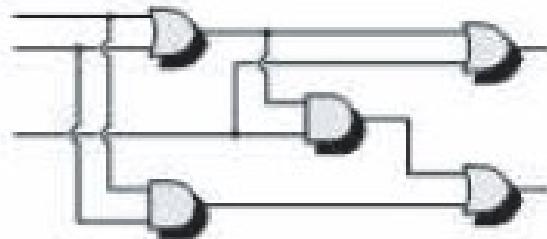
- RTL → synthesizer → gate-level
(C program → compiler → assembly program)



(Higher level
of abstraction)



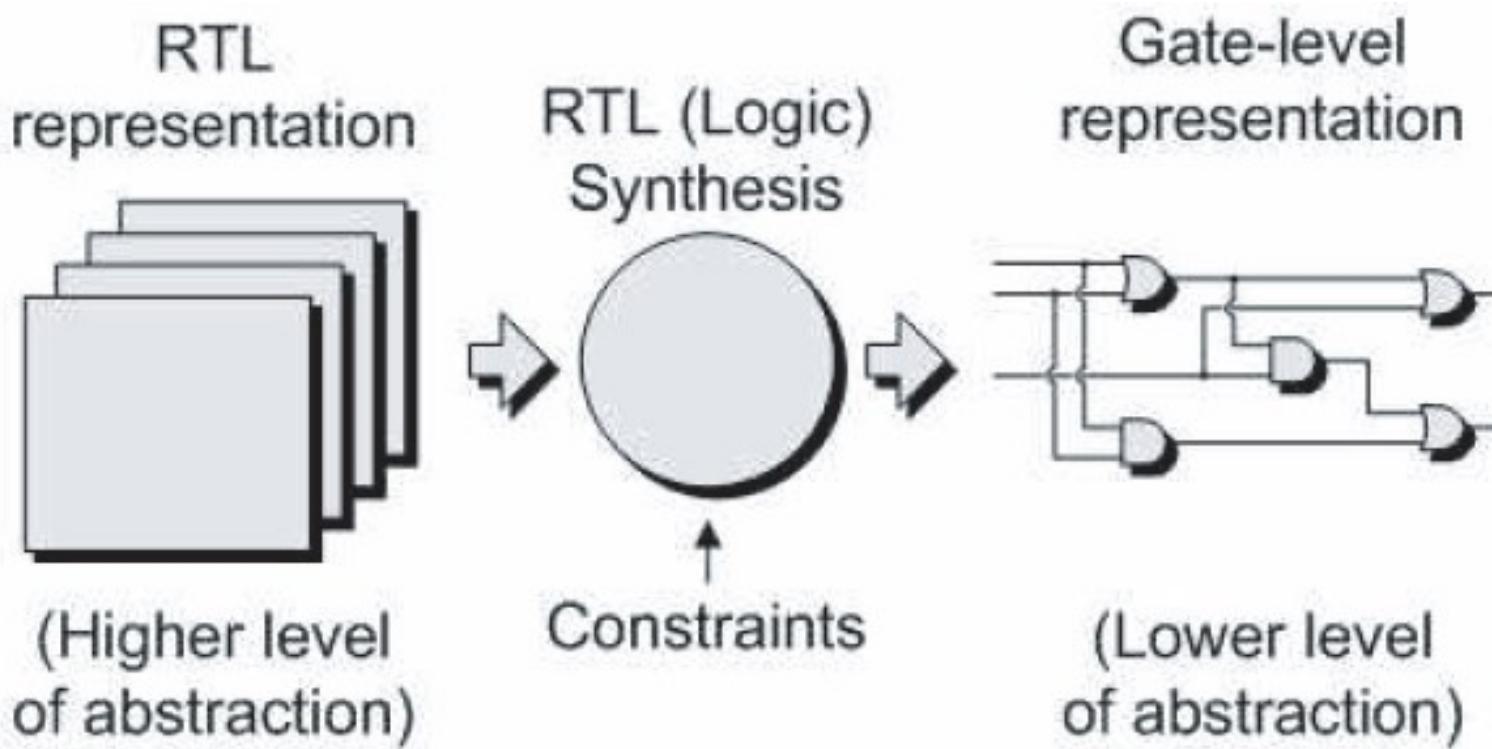
Gate-level
representation



(Lower level
of abstraction)

RTL Logic Synthesis

- Hardware design in RTL (similar to SW design)
- Logic synthesis tool converts RTL to gate-level
- Function verification in RTL simulation
- Timing verification in gate-level simulation



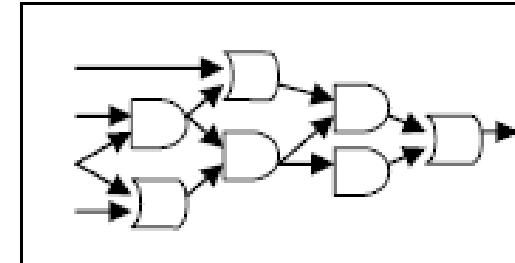
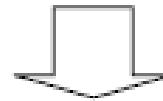
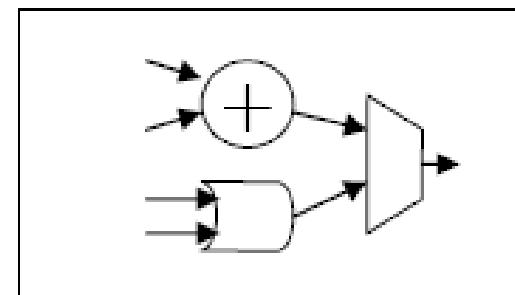
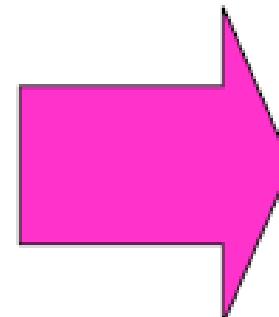
RTL synthesis

- RTL to gate-level
 - Might synthesize into different gate-level results, depending on user's area/speed constraints

```
--VHDL  
if(A='1') then  
  Y<=C + D;  
elseif (B='1') then  
  Y<=C or D;  
else Y<=C;  
endif
```

```
//Verilog  
if(A==1)  
  Y=C + D;  
else if(B==1)  
  Y=C | D;  
else Y=C;
```

RTL
synthesis



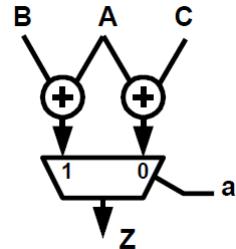
ASIC vs. FPGA Design Flows

- both ASIC (Application Specific Integrated Circuits) and FPGA (Field Programmable Gate Array) use HDL as design entries
- ASIC
 - based on process-technology dependent standard cell library
 - uses logic synthesis tools (such as Synopsys Design Compiler) to synthesize gate-level netlist from HDL
 - need back-end Placement-and-Routing (P&R) to generate final physical layout
 - need IC foundry to fabricate the chip
 - cannot re-program to other hardware function once tape-out
- FPGA
 - use synthesis tool from FPGA toolkit to map HDL to a particular pre-fabricated FPGA chip for quick prototyping
 - do not need IC foundry
 - re-programmable to different hardware with different HDL

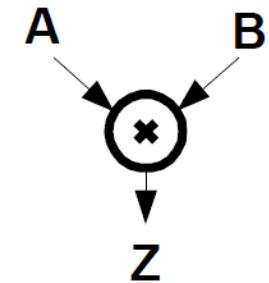
RTL coding vs. synthesized results

- different codings result in different hardware
 - always guess what hardware is synthesized during coding

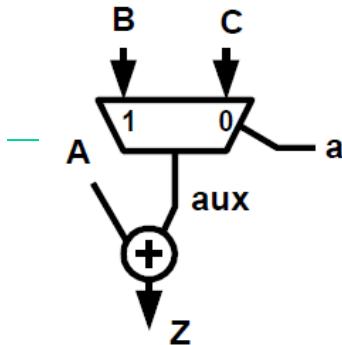
```
If(a == 1)  
  Z= A + B;  
Else  
  Z= A + C;
```



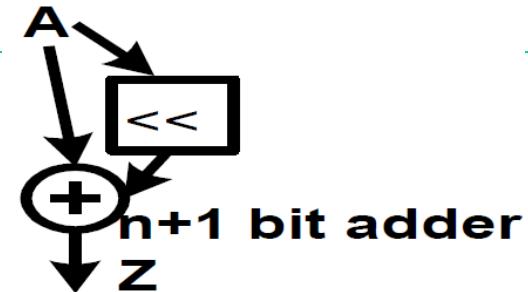
```
assign B = 3;  
assign Z = A * B;
```



```
if(a == 1)  
  aux = B;  
else  
  aux = C;  
Z = A + aux;
```



```
assign Z = A + 2 * A;
```



What you will learn in this course

- use Verilog/VHDL to describe “large” hardware
 - e.g., CPU, AI accelerators, ...
 - Homework 3: pipelined RISC CPU
 - Homework 4: image edge detector
 - Homework 5: convolutional neural network accelerator