EE/CSE 371
Design of Digital Circuits and Systems

Autumn 2023

SystemVerilog Review & Tips

Department of Electrical and Computer Engineering
University of Washington, Seattle WA

Mahmood A. Hameed

Lecture 1

The Teaching Team

- **Instructor**: Prof. Mahmood A. Hameed
  - Contact: hameedm@uw.edu, Microsoft Teams
  - Office Hours: Thursdays 1:00 pm to 3:00 pm in ECE 230
- **Teaching Assistants**:
  - K Gupta, kshitijg@uw.edu
  - Matthew Hung, mhung20@uw.edu
  - Ting Jones, jonesshu@uw.edu
- **Grader:**
  - Atharva Mattam, am262@uw.edu
- **Lab office hours and demo slot link:**
  - Click here

Lecture Outline

- **Course Introduction**
- Course Policies
- Hardware Description Language
- SystemVerilog Review & Tips

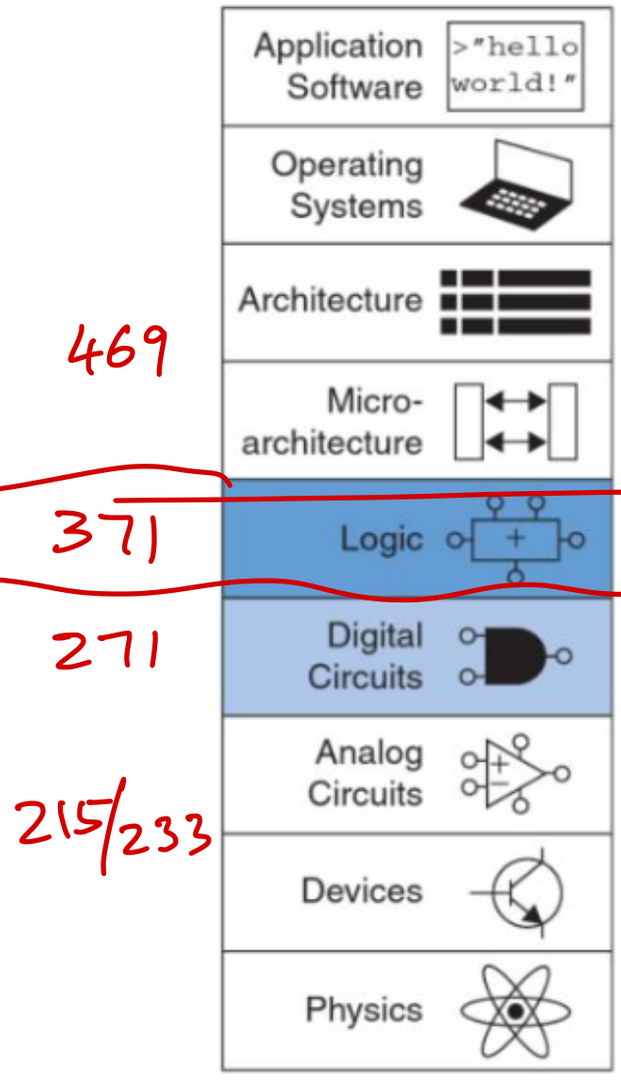
Introductions: Students

- 59 students registered, 56 ECE and 3 CSE
- Expected background
  - **Prereq:** EE271 or CSE369 – construction of synchronous digital systems (combinational + sequential logic), timing introduction, SystemVerilog introduction
  - **Prereq:** EE205 or EE215 – familiarity with circuits

Course Motivation

- More advanced digital logic design
  - Higher-level circuit design and analysis techniques
  - Interfacing with various devices/peripherals
  - How to implement algorithms *in hardware*
  - Practical timing analysis
  - "Verilog finishing school"

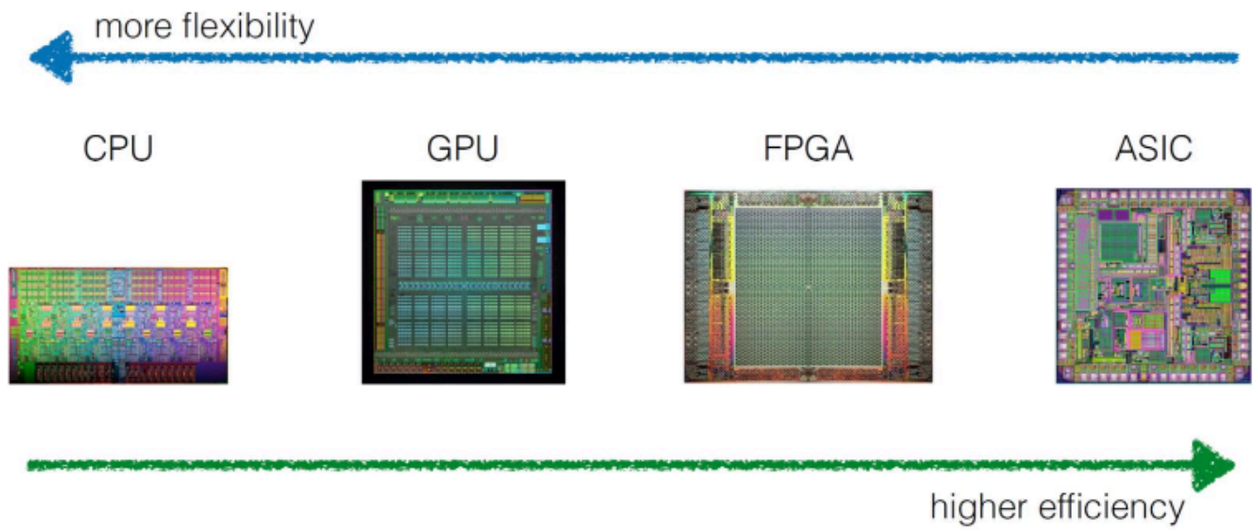
Harris and Harris. "Digital Design and Computer Architecture" 2nd ed.



469
371
271
215/233
280

Course Motivation

- Heavy computational loads have put us in a golden age of **hardware specialization**
  - Tailor your chip architecture to the characteristics of a **stable** workload
  - More and more companies are building specialized chips



Course Motivation

- Example: FPGAs could replace GPUs in many deep learning applications
  - https://bdtechtalks.com/2020/11/09/fpga-vs-gpu-deep-learning/
  - FPGAs have advantages in terms of power consumption, environmental robustness, and lifespan
  - "FPGA is a type of processor that can be customized after manufacturing, which makes it more efficient than generic processors."
  - "But the problem with FPGAs is that they are very hard to program."

Technical Communication

- Many of you are graduating soon!
  - Going off to do and build wonderful things
  - Try as you might, you can't really avoid society and other people
- It's no longer good enough to just get something working – you have to be able to explain it
  - Working in a group/team
  - Someone taking over as maintainer
  - Convince a client or venture capitalist or funding agency or students
- The lab reports and demos might seem tedious at first, but are valuable practice!

Lecture Outline

- Course Introduction
- **Course Policies**
- Hardware Description Language
- SystemVerilog Review & Tips

Course Meetings

- Lectures
  - Lecture slides found on Piazza → Resources → Lectures
  - Group work most lectures
- Office Hours and Lab Demos
  - Live demo using remote FPGA lab (LabsLand) and answering TA questions
  - Will support in-person demos (ECE 361/365)
  - When a TA is available, lab demo times are also office hours

Grading

- Labs (50%) Groups of 2
  - 5 regular labs and 1 "final project"
  - Combination of lab reports, code submission, and lab demos
- Homework (20%) Groups of 4
  - Combination of theoretical and coding questions
  - Interspersed between the labs
- Exams (30%)
  - Midterm exam (15%), in class, ~~Thursday~~ Wednesday November ~~22nd~~ 1st
  - Final Exam (15%), Wednesday December 13th, 2:30 pm to 4:20 pm

Collaboration Policy

- Complete your work as permitted (either individually or with partner/ group)
  - Don’t attempt to gain credit for something you didn’t do and don’t help others do so either
- **OK:**
  - Discussing/studying lectures and/or course material
  - *High-level* discussion of general approaches
  - Help with error messages, tools peculiarities, etc.
- **Not OK:**
  - Work in a larger group than permitted
  - Giving away solutions or having someone else (human or AI) do your assignment for you

Collaboration Notes

- Make sure to add your groupmate(s) to your submissions in Gradescope
- Working in a group isn't a guarantee for success
  - Don't just split up the work: work simultaneously or at least communicate frequently (two brains are better than one!)
- See the "Collaboration Policies and Tips" section of the syllabus for more info

Deadlines

- Labs reports & code are submitted via Gradescope Fridays before 11:59 pm
- Lateness is counted in *days* and cannot be submitted more than 2 days late (Sunday 11:59 pm)
- 5 late day tokens; 10% penalty for each late day after that
- 15-min lab demos within a week of report deadline
  - Usually in assigned slot (sign-up process)
- Homework cannot be submitted late
  - Solutions released immediately to help you study for exams

To-do List

- Go through Piazza, resources, syllabus
  - Read over course policies and Assignment Info PDF
- Install Quartus & ModelSim on your machine
- Register on LabsLand
- Look for partner(s) for homework and lab
  - Figure this out *before* signing up for a lab demo slot
- Homework 1 (10/4) & Lab 1 (10/6) posted today

Lecture Outline

- Course Introduction
- Course Policies
- **Hardware Description Language**
- SystemVerilog Review & Tips

Hardware Description Language (HDL)

- HDL is a specialized computer language used to describe the structure and behavior of digital logic circuits
  - Useful for rapid prototyping of digital hardware
- Comparison with programming language (*e.g.*, C)
  - Describes *hardware* instead of software
  - Intrinsically parallel instead of sequential
  - Includes explicit notion of time instead of just instructions
  - Compiled to target FPGAs and ASICs via netlists (more specific) instead of compiled to target CPUs via machine code (more general)

Electronic Design Automation (EDA)

- EDA is a type of computer-aided design (CAD) specific to designing electronic systems
  - EDA tools allow for *synthesis* and *simulation*
- **Synthesis:** Compile HDL code into a netlist (*i.e.*, list of electronic components and wires that connect them)
- **Simulation:** Apply inputs to the simulated circuit so we can check output for correctness without involving or endangering hardware
  - Tracks the state of each structural element, handles the interactions between concurrent elements, and models the passage of time

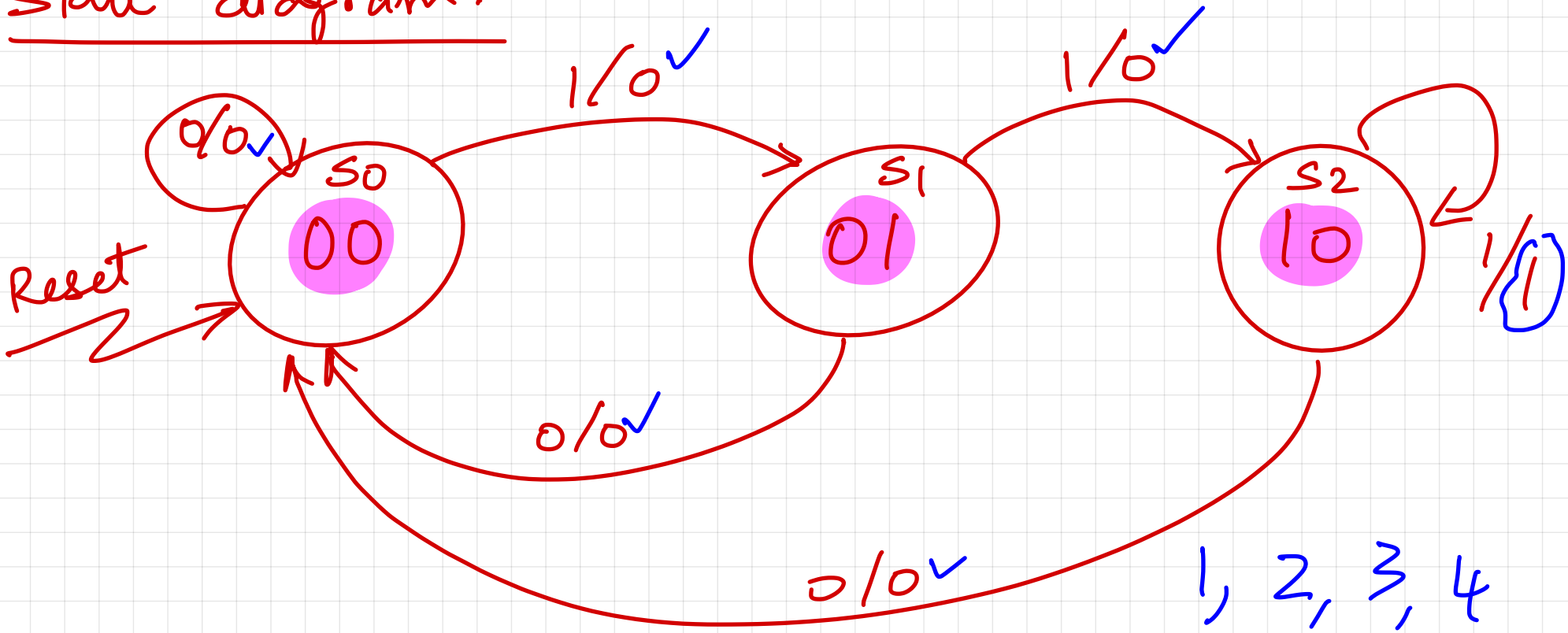
Design Flow Using Gates

- Design of a synchronous digital system using gates:
  - Write a specification
  - If necessary, partition the design into smaller parts and write a specification for each part
  - From the specification, draw a state machine diagram
    - Minimize the number of states when possible
  - Assign a binary encoding to represent each state
  - Derive the next state and output logic
    - Optimize the logic to minimize the number of gates needed
  - Choose a suitable placement for the gates to optimize integrated circuit reuse and distance on printed circuit board
  - Design the routing between the integrated circuits

three Ones FSM

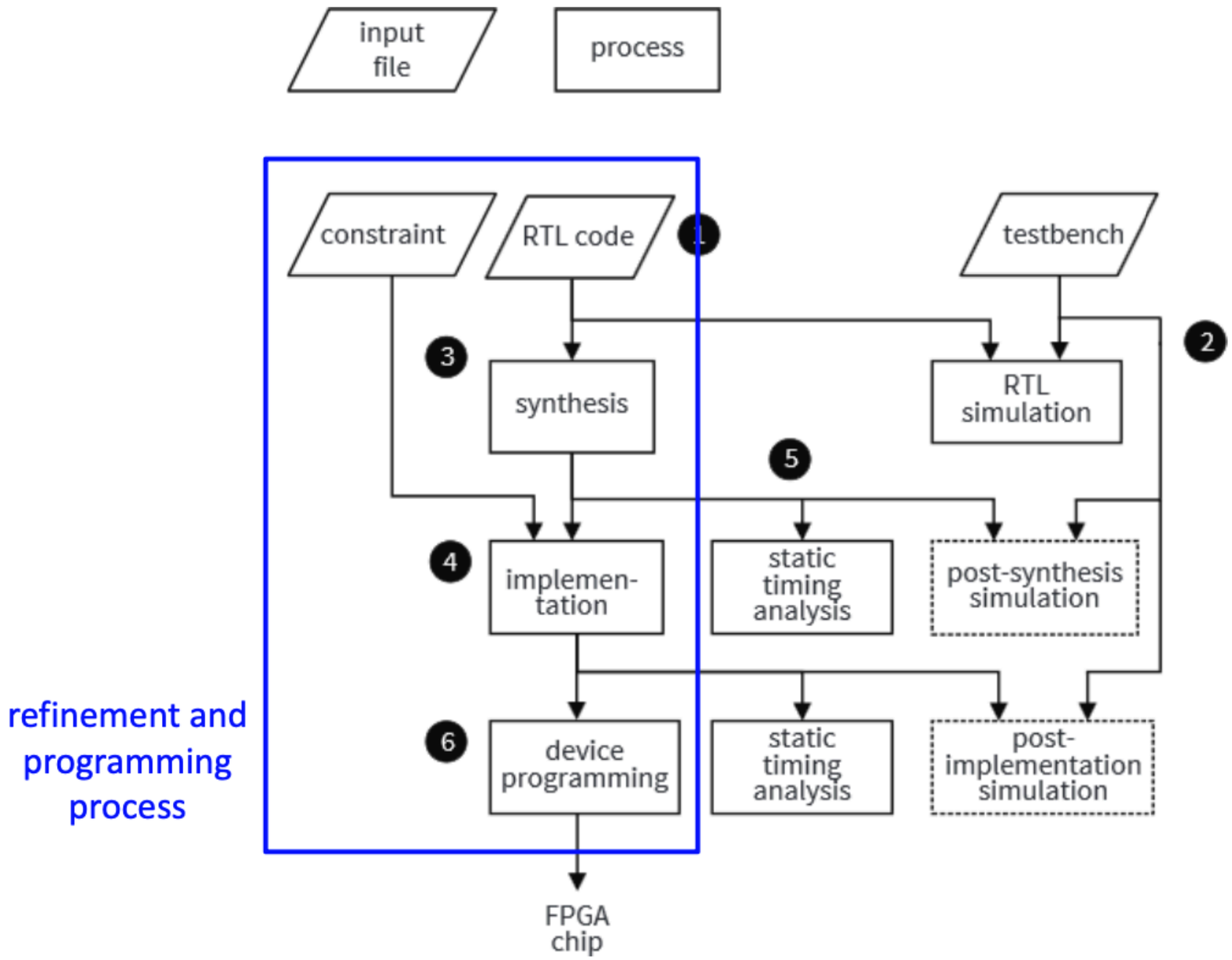
Mealy. Output z ps, input

State diagram.



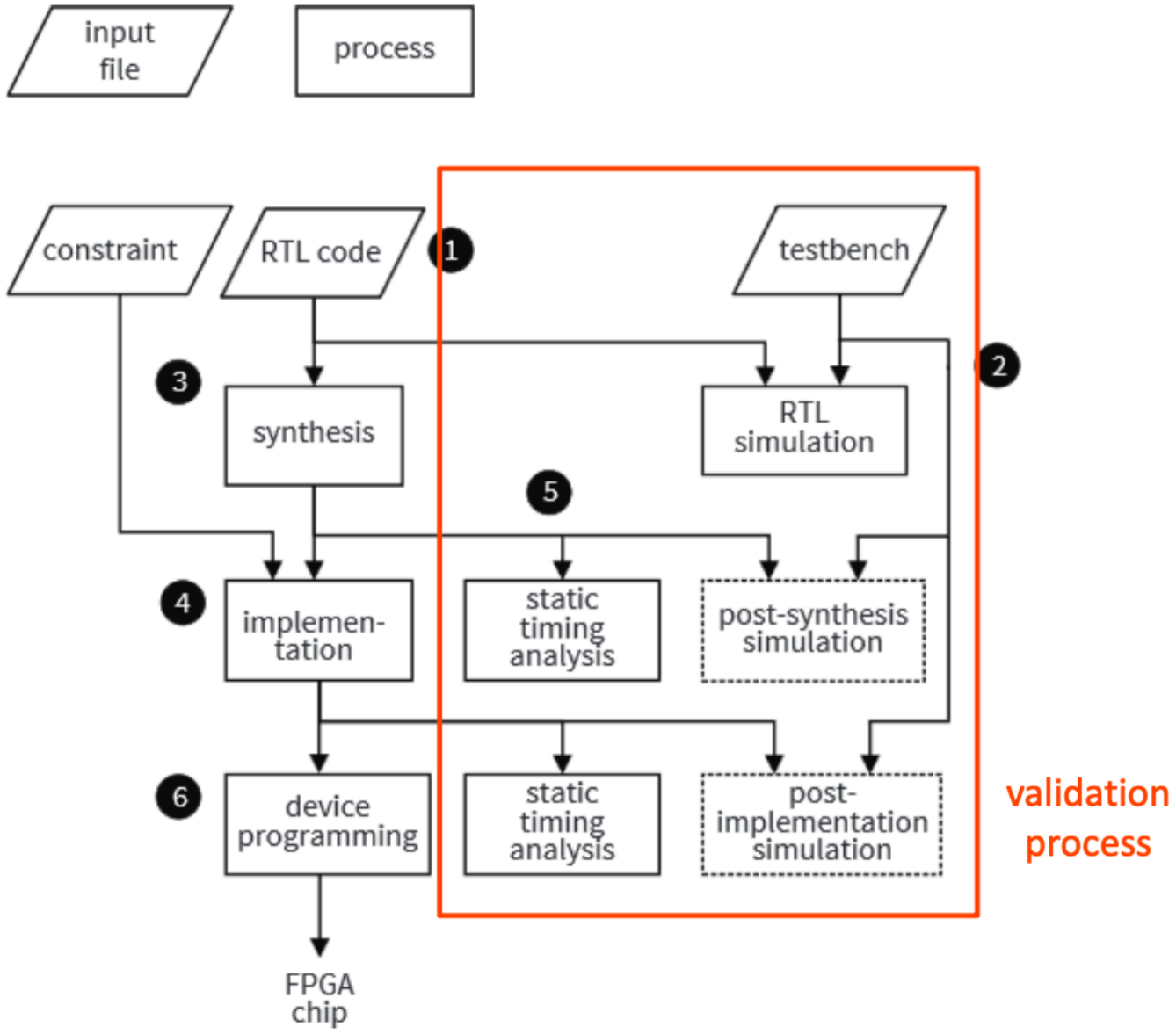
Least number of clock cycles for output to be '1'. 2

FPGA-Based Design Flow



Source: Figure 2.3 from "FPGA Prototyping" by P. Chu

FPGA-Based Design Flow



Source: Figure 2.3 from "FPGA Prototyping" by P. Chu

Lecture Outline

- Course Introduction
- Course Policies
- Hardware Description Language
- **SystemVerilog Review & Tips**

  - One of the two leading HDLs along with VHDL-2019

SystemVerilog Tips

- Always plan out your design before you ever touch your keyboard
  - Block diagram naturally maps into module port lists
- Always think of the underlying hardware when you write your code
  - **Structural:** lower-level modeling of gate-level connections
  - **Behavioral:** higher-level modeling of logic behavior
- Most modules are organized similarly, so develop and use coding patterns
- Comment your code and test thoroughly as you go
- Pay attention to compiler warnings and errors

SystemVerilog Review Question

- Will the following compile? Is there a difference?

```
logic v, w, x, y, z;
and g1 (x, v, w);
xor g2 (y, v, w);
nor g3 (z, x, y);
```

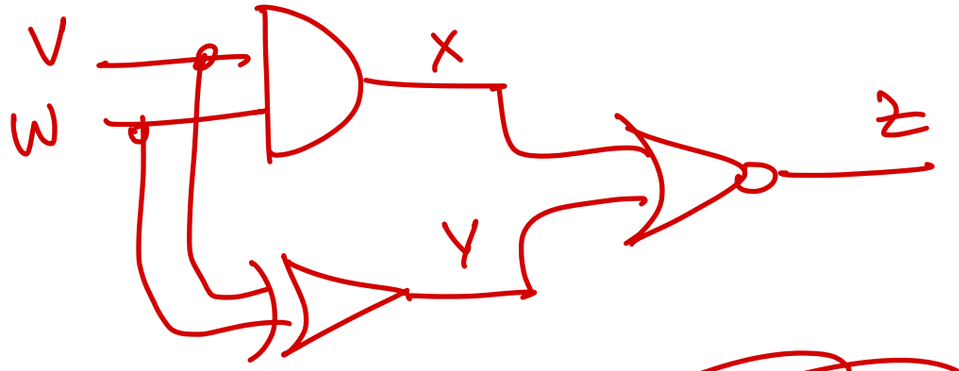
output inputs

```
logic v, w, x, y, z;
nor g3 (z, x, y);
and g1 (x, v, w);
xor g2 (y, v, w);
```

```
and g1 (x, v, w);
logic v, w, x, y, z;
xor g2 (y, v, w);
nor g3 (z, x, y);
```

Fix!

Hardware:



12:44

271/369 SystemVerilog Refresher

- Combinational logic:
  - Usually in `assign` or `always_comb` blocks and use blocking assignment (=)
  - Implicit sensitivity list – updates anytime any input changes
  - Testbenches for CL should just cycle through all input combinations at fixed time intervals (*e.g.*, `#20;`)
- Sequential logic:
  - Usually in `always_ff` blocks and use non-blocking assignment (<=)
  - Explicit sensitivity list – usually `@(posedge clk)`
  - Testbenches for SL should take every transition that you care about, achieved by changing inputs every clock cycle

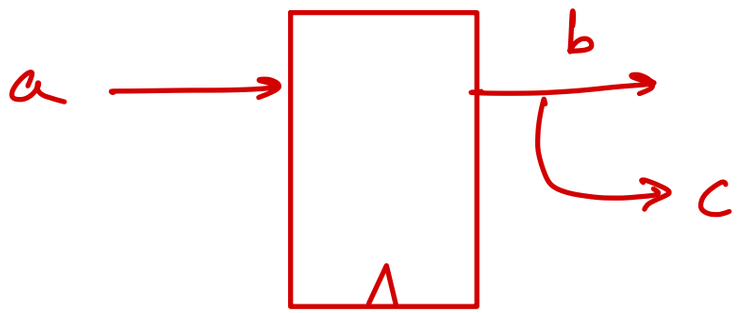
Blocking vs. Nonblocking Assignment

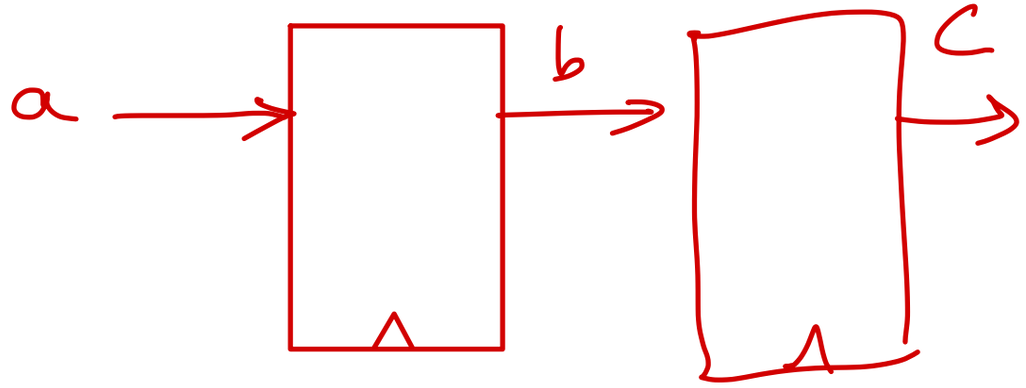
- **Blocking** statement (=): the effects of these statements are "evaluated" sequentially
- **Nonblocking** statement (<=): the effects of these statements "evaluated" in parallel
- Exercise: draw out implied hardware

```
always_ff @ (posedge clk)
begin
   b  = a;
   c  = b;
end
```

(new)

```
always_ff @ (posedge clk)
begin
   b <= a;
   c <= b;
end
```





271/369 SystemVerilog Refresher

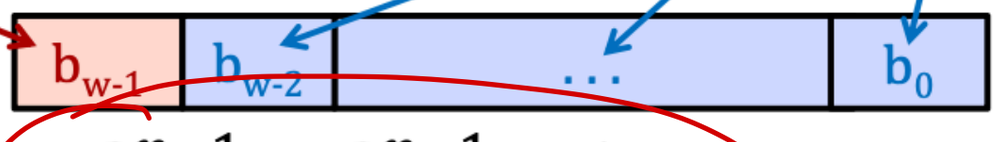
- We generally default to using `logic` for all signals, which is a 4-state data type that can be used as either a *net* or *variable*
  - Possible states (`logic` defaults to `X`):
    - `0` = zero, low, FALSE
    - `1` = one, high, TRUE
    - `X` = unknown, uninitialized, contention (conflict)
    - `Z` = floating (disconnected), high impedance
  - **Nets** (*e.g.*, `wire`) transmit values and **variables** (*e.g.*, `reg`) store data

271/369 SystemVerilog Refresher

- Multiple signals can be organized under the same name as a bus or array
  - A *bus*, also known as a *vector* or *packed array*, is a collection of a single data type
    - *e.g.*, `logic [31:0] divided_clocks;` (type)
  - "Regular" array syntax is known as an *unpacked array*
    - *e.g.*, `logic an_unpacked_array[4:0];`
  - *Multidimensional arrays* can be combinations of packed and unpacked dimensions
    - *e.g.*, `logic [3:0] two_D_array[4:0];`
    - Dimensions accessed left to right, starting with unpacked
  - Numbering matters (*i.e.*, `[`n`-1:0]` ≠ `[0:`n`-1]`)

Review: Integers in Computing

- **Unsigned integers** follow the standard base 2 system
  - $b_7b_6b_5b_4b_3b_2b_1b_0 = b_7 2^7 + b_6 2^6 + \cdots + b_1 2^1 + b_0 2^0$
  - In n bits, represent integers 0 to $2^n - 1$
- **Signed integers** use *Two's Complement representation*
  - b_{w-1} has weight -2^{w-1}, other bits have usual weights $+2^i$



  - In n bits, represent integers -2^{n-1} to $2^{n-1} - 1$
  - Most significant bit acts as a *sign bit* (0 = pos, 1 = neg)
  - Handy negation procedure: take the bitwise complement and then add one (`~x + 1 == -x`)
- ⚠ The choice affects the behavior of operations such as bit extension, shifting, and comparisons

4-bit

1000

-8

271/369 SystemVerilog Refresher

- Multi-bit constants: <n>'<s><b>#…#
  - <n> is width (*unsized* by default)
  - <s> is signed designation (omit or 's')
  - <b> is radix/base specifier (decimal by default)
  - All letters are case-*in*sensitive, _ can be used to add spaces

1000_0111

32 bits.

42 = 32+8+2

Literal	Width	Base	Bits
3'd6	3	10	110
6'o42	6	8	100_010
8'hAB	8	16	1010_1011

Literal	Width	Base	Bits
42	32	10	00…0101010
'b101	32	2	0……00101
-3'd5	3	10	011

- Compiler will usually warn you if there is a size mismatch
  - Can "cast" using #'(<sig>) syntax

$\overline{101} + 1 = 010 + 1$

$= 011$

1 → 000....01

'1 → 1111....1

1'b1 → 1

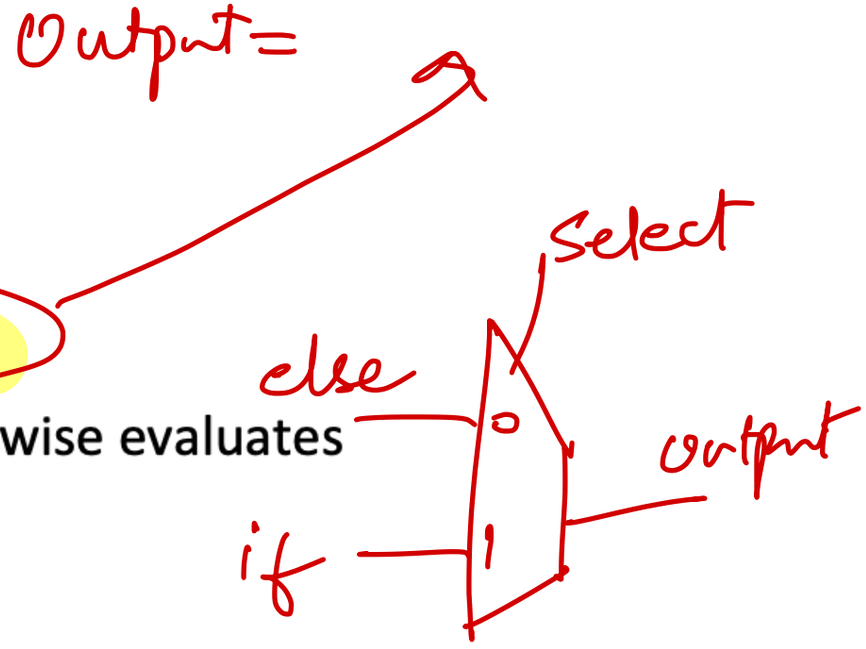
Basic Operators

- Possibly new:
  - `371 % 271` → `100` (remainder)
  - `2**3` → `8` (pow)

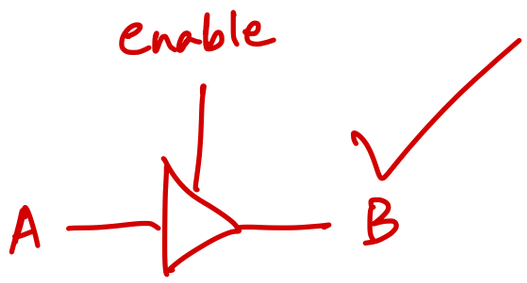
Type	Symbol	Description
Arithmetic	+	addition
	−	subtraction
	*	multiplication
	/	division
	%	modulus
	**	exponentiation
Shift	>>	logical right shift
	<<	logical left shift
	>>>	arithmetic right shift
	<<<	logical left shift
Relational	>	greater than
	<	less than
	>=	greater than or equal to
	<=	less than or equal to
Equality	==	equality
	!=	inequality
	===	case equality
	!==	case inequality
Bitwise	~	bitwise negation
	&	bitwise and
	|	bitwise or
	^	bitwise xor
Logical	!	logical negation
	&&	logical and
	||	logical or

Ternary Operator

- Conditional assignment
  - `select ? <if_expr> : <else_expr>`
    - If `select` is true, then evaluates to `<if_expr>`, otherwise evaluates to `<else_expr>`
  - What does this look like in hardware?



- **Example:** tristate buffer
  - `enable ? in : 'bZ`
    - When enabled, pass the input to the output, otherwise be high impedance



enable	A	B
0	0	Z
0	1	Z
1	0	0
1	1	1

B = enable ? A : 'bz