ECE 327/627
Digital Hardware Systems

Lecture 1: Introduction

Andrew Boutros

andrew.boutros@uwaterloo.ca

Know your Instructors & TAs

- **Course Instructor: Andrew Boutros**
  - Joined UWaterloo in January 2025
  - BSc @ German University in Cairo, Egypt
  - MASc and PhD @ UToronto – FPGA architecture & CAD
  - Led the Toronto Office of MangoBoost
  - Research Scientist @ Intel Labs & PSG (now Altera)

- **Lab Instructor: Maran Ma**
  - Joined UWaterloo in 2022
  - PhD @ McGill University
  - Research interest in brain-machine interfacing (providing capstone consultation in this area)
  - 8 years industry experience in display and imaging

- **TAs:**
  - Ahmed Elsousy – PhD student
  - Marwan Mekhemer – MASc student

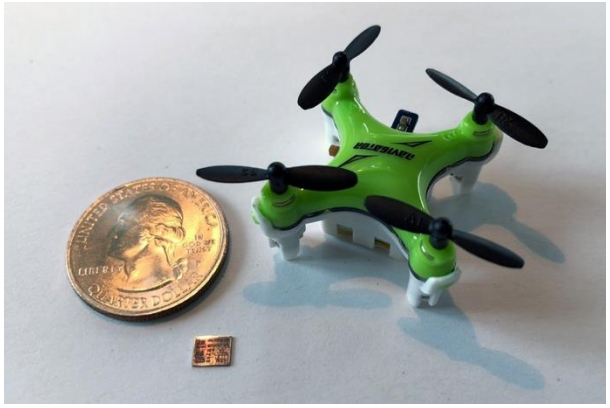
What is this course all about?

Fundamentals of the **design** and **optimization** of digital computer hardware

Source: apple.com

Source: nytimes.com

Source: apple.com



Source: rle.mit.edu

Source: waymo.com

What is this course all about?

Fundamentals of the **design** and **optimization** of digital computer hardware

Source: a

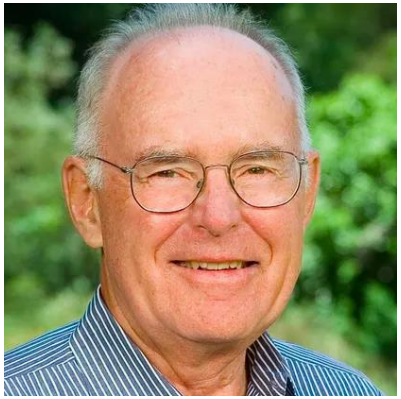
But why bother about hardware design? Just develop software and run it on general-purpose programmable processors

Source: rle.mit.edu

Source: waymo.com

Process Technology Improvements

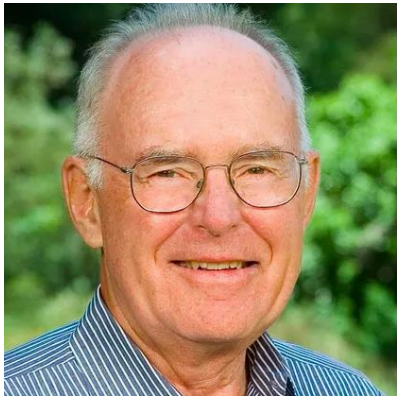
- Gordon Moore (1929-2023)



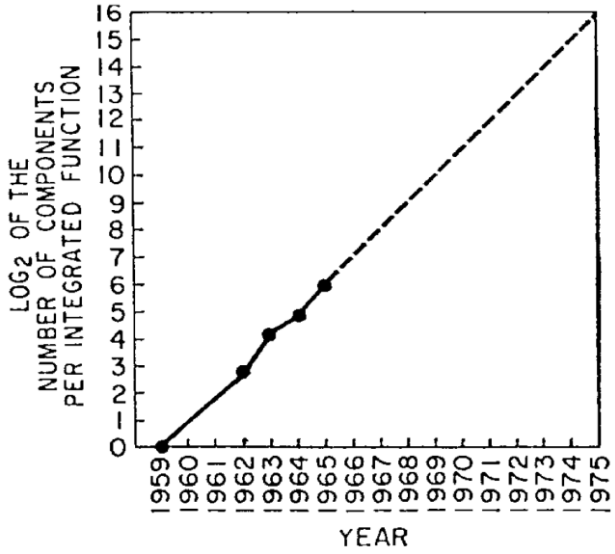
Source: forbes.com

Process Technology Improvements

- Gordon Moore (1929-2023)
  - Co-founder of Intel Corp.
  - Proposed **Moore's (empirical) Law** in 1965
  - Number of components in an integrated circuit doubles every generation (1 year in 1965, revised to 2 years in 1975)



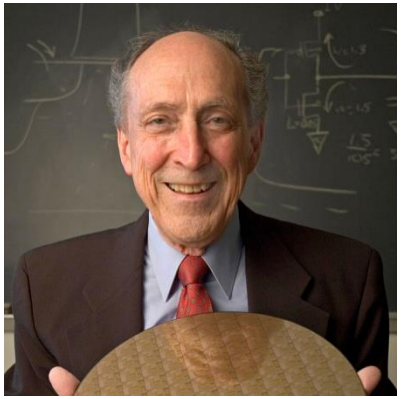
Source: forbes.com



Gordon E. Moore, "Cramming More Components onto Integrated Circuits," Electronics, pp. 114–117, April 19, 1965.

Process Technology Improvements

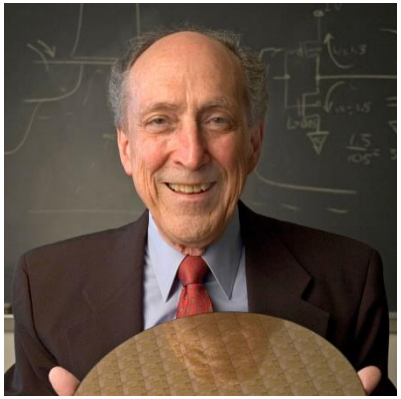
- Robert Dennard (1932-2024)



Source: nytimes.com

Process Technology Improvements

- Robert Dennard (1932-2024)
  - Researcher at IBM (invented DRAM)
  - Formulated **Dennard's Scaling Law**
  - As transistors get smaller, power density (power per unit area) stays constant



Source: nytimes.com

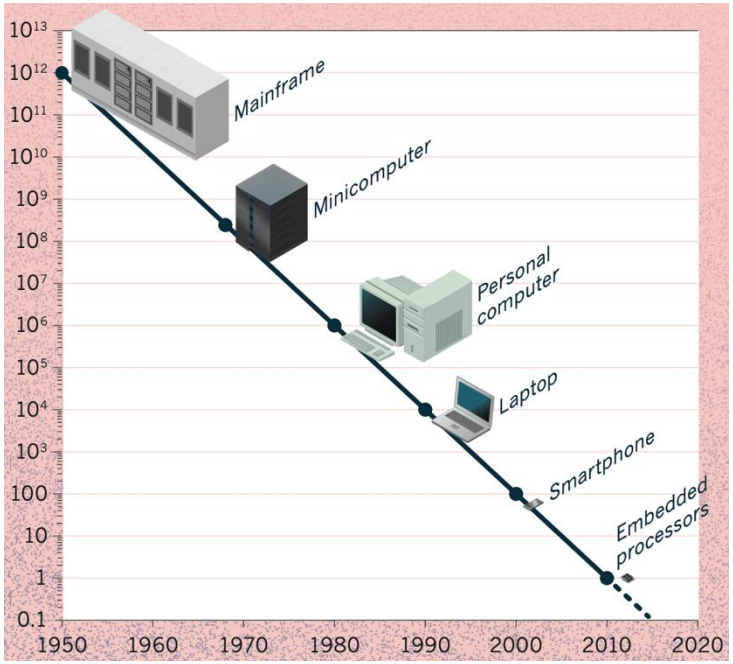
SCALING RESULTS FOR CIRCUIT PERFORMANCE

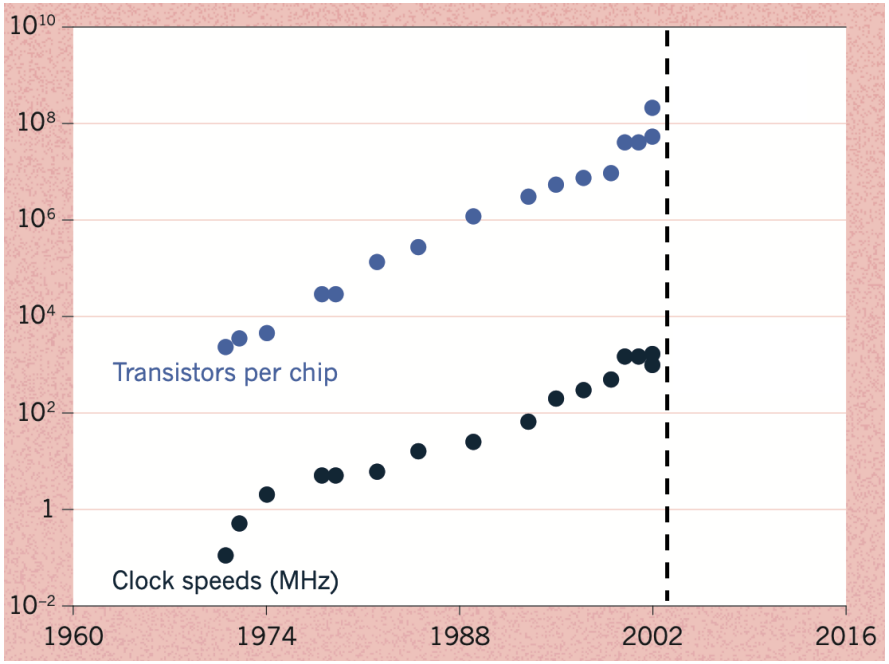
Device or Circuit Parameter	Scaling Factor
Device dimension t_{ox}, L, W	$1/\kappa$
Doping concentration N_a	κ
Voltage V	$1/\kappa$
Current I	$1/\kappa$
Capacitance $\epsilon A/t$	$1/\kappa$
Delay time/circuit VC/I	$1/\kappa$
Power dissipation/circuit VI	$1/\kappa^2$
Power density VI/A	1

Dennard, Robert H., et al. "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions." *IEEE Journal of solid-state circuits* 9.5 (1974): 256-268.

That's a Pretty Good Deal

- Based on these observations, every generation …
  - 2x number of transistors per chip → Capacity improves
  - Run these transistors faster → Clock frequency improves
  - At the same power budget → Power/transistor improves

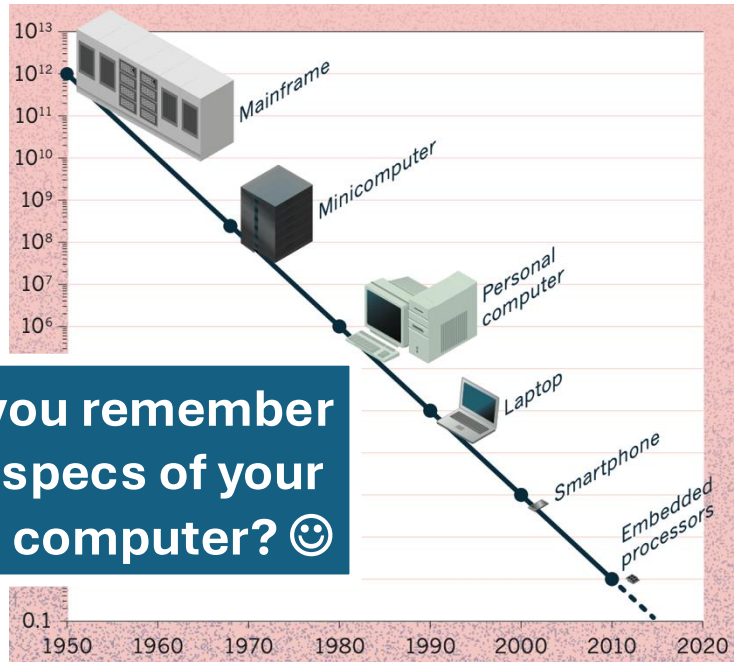




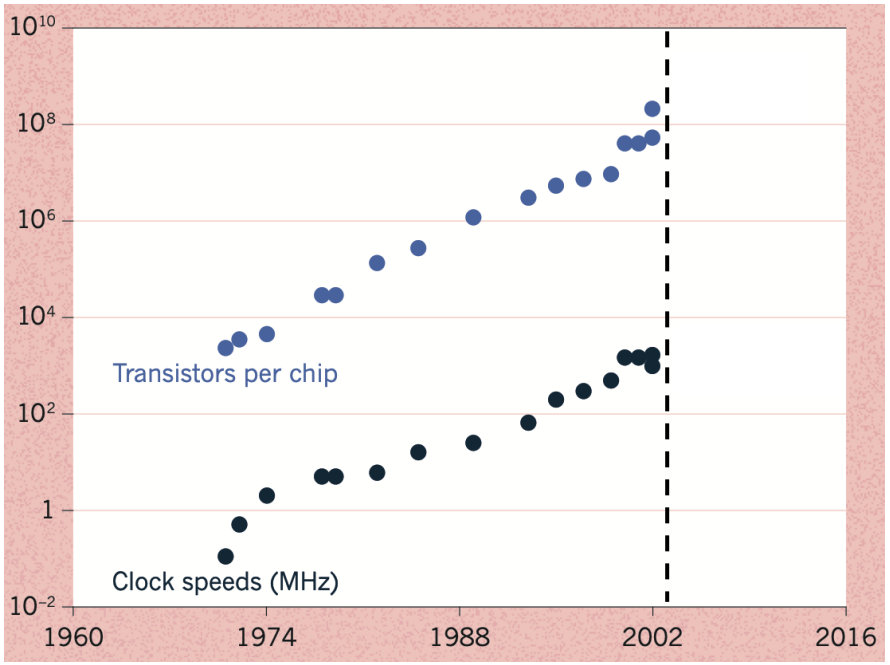
Waldrop, M. Mitchell. "The Chips are Down for Moore's Law." *Nature News* 530.7589 (2016): 144.

That's a Pretty Good Deal

- Based on these observations, every generation …
  - 2x number of transistors per chip → Capacity improves
  - Run these transistors faster → Clock frequency improves
  - At the same power budget → Power/transistor improves



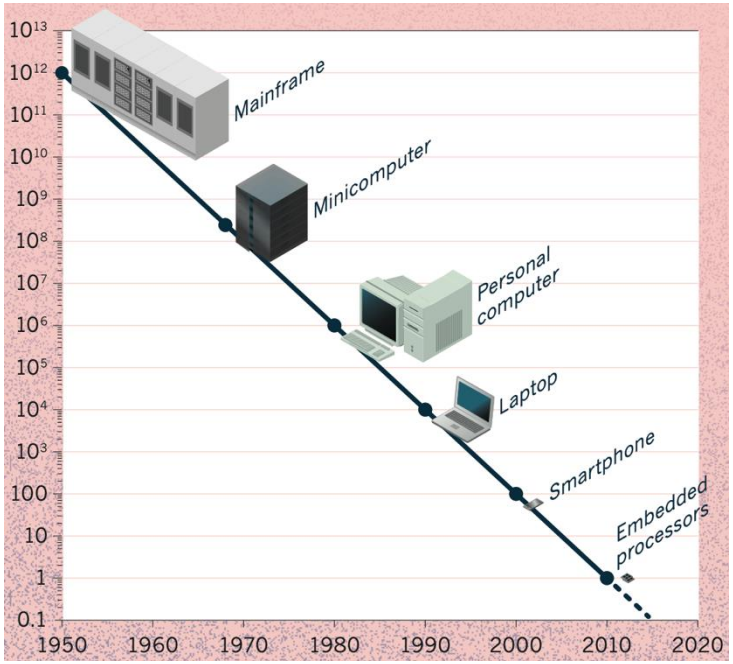
Do you remember the specs of your first computer? ☺

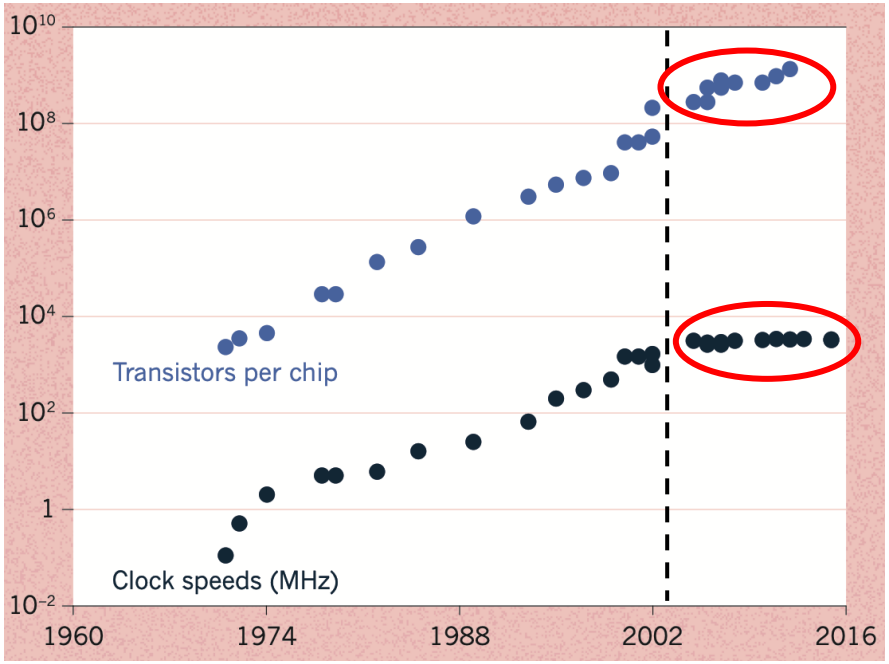


Waldrop, M. Mitchell. "The Chips are Down for Moore's Law." *Nature News* 530.7589 (2016): 144.

But Nothing Good (or Bad) Lasts Forever

- Process technology improvements are slowing down
  - Transistors cannot shrink beyond a few atoms thick
  - Power consumption not scaling well → leakage current
- Need to find other ways to improve performance …



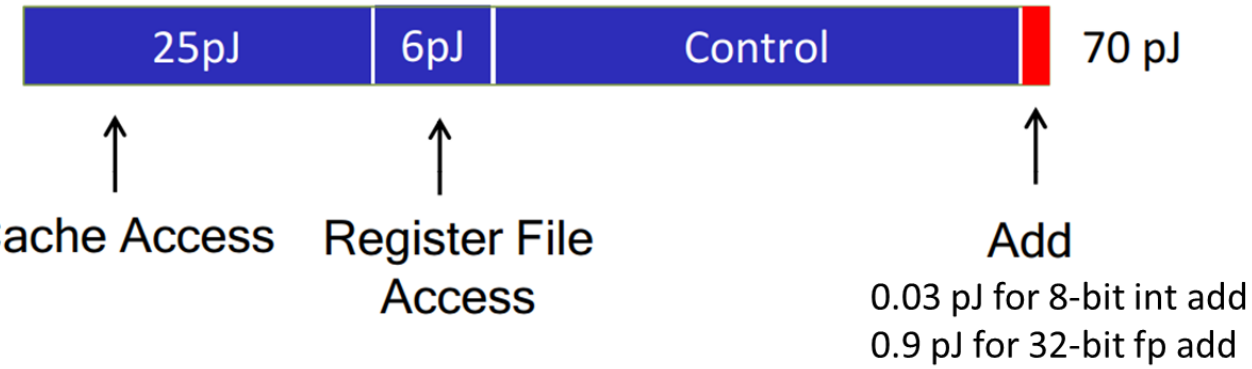


Waldrop, M. Mitchell. "The Chips are Down for Moore's Law." *Nature News* 530.7589 (2016): 144.

Thinking Outside the Box

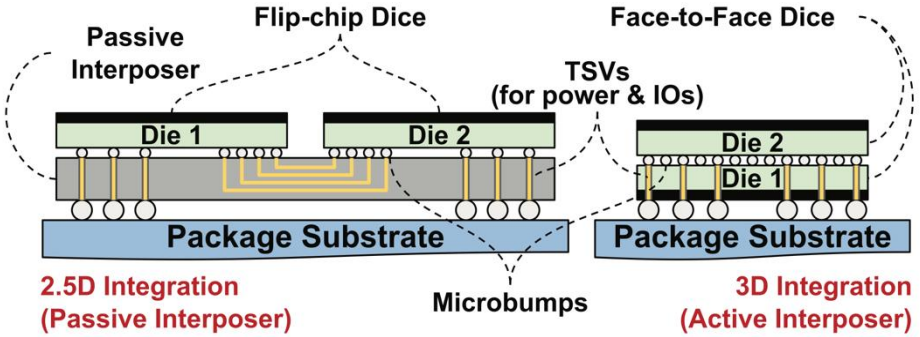
- Architecture innovations
  - Find more efficient ways to use the available transistors
  - Examples:
    - Heterogeneous architectures (big/little cores)
    - Single-Instruction Multiple-Data (SIMD) for parallel workloads

45 nm CPU energy breakdown [from M. Horowitz, "Computing's Energy Problem," *ISSCC,* 2014].



Thinking Outside the Box

- New technologies to face different challenges
  - System scalability challenges
    - Multi-die system-in-package (e.g., chiplets, 3D stacking)

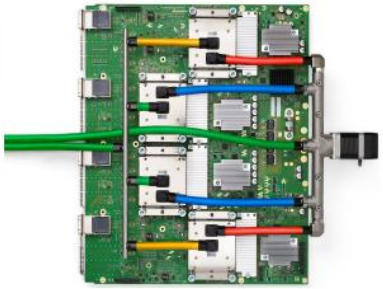


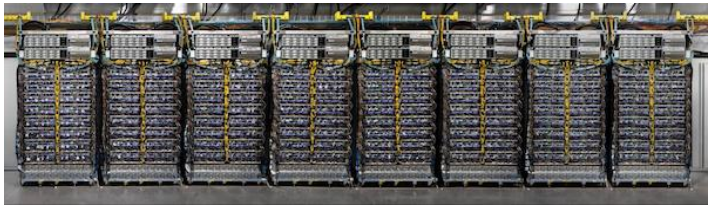
  - Data transport energy challenges
    - Near-Memory or In-Memory compute
  - New compute paradigms
    - Neuromorphic computing (event-driven processing)
    - Quantum computing
    - Optical computing

Thinking Outside the Box

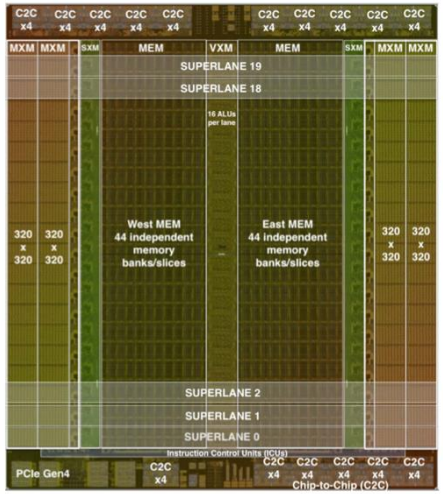
- Domain-specific computers
  - Deliver performance by specialization (e.g., AI chips)

Google TPU

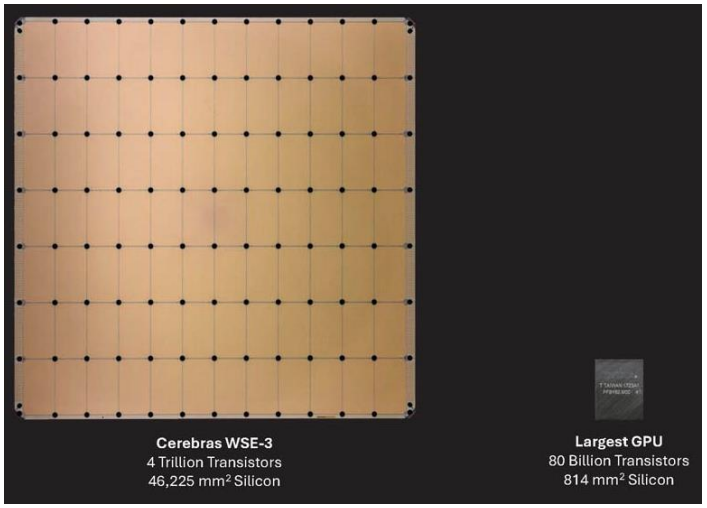




Groq TSP



Cerebras WSE



Jouppi, Norm, et al. “TPU v4: An Optically Reconfigurable Supercomputer for Machine Learning with Hardware Support for Embeddings”, ISCA 2023
Abts, Dennis, et al. “Think Fast: A Tensor Streaming Processor (TSP) for Accelerating Deep Learning Workloads”, ISCA 2020
Lie, Sean, et al. “Cerebras Wafer-Scale AI: GPU Impossible Performance”, Hot Chips 2024

Golden Time for Computer Hardware Designers

- Many AI hardware startups (based on publicly available funding info)

groq™ $3B+

cerebras $1.9B

$1.18B

Tenstorrent

SambaNova® SYSTEMS $1.1B

etched $125M

- Many software companies are now building their HW (e.g., OpenAI, Amazon, Google, Meta, Microsoft)

Median Package

OpenAI Hardware Engineer San Francisco, CA	Total per year	Level
	$1.67M	L6

Base	Stock (/yr)	Bonus	Years at company	Years exp
$452K	$1.22M	$0	0 Years	5 Years

Source: levels.fyi
May 4, 2025

Future Prospects of Taking ECE 327/627

- This class will cover the basic concepts of computer hardware design and optimization.

- It is **an introduction** that opens the door for:
  - Full-time or Co-op roles in semiconductor companies
  - Advanced research in computer architecture, embedded systems, VLSI design, domain-specific acceleration, compiler stacks, etc.

Now some logistics ...

Class Schedule

Course	Meet Days	Meet Time	Location
ECE 327 001 [LEC]	**Mon, Thu** *Jan 5 - Apr 6*	10:00AM - 11:20AM	E7 4053
	Mondays *Jan 5, Jan 19, Feb 2, Feb 23, Mar 9, Mar 23*	11:30AM - 12:20PM	E7 4053
ECE 327 101 [TUT]	**Wednesdays** *Jan 5 - Apr 6*	12:30PM - 01:20PM	E7 4053
ECE 627 001 [LEC]	**Mon, Thu** *Jan 5 - Apr 6*	10:00AM - 11:20AM	E7 4053

Class Schedule

Makeup lectures: Not used until I announce otherwise beforehand

Course	Meet Days	Meet Time	Location
ECE 327 001 [LEC]	**Mon, Thu** *Jan 5 - Apr 6*	10:00AM - 11:20AM	E7 4053
	Mondays *Jan 5, Jan 19, Feb 2, Feb 23, Mar 9, Mar 23*	11:30AM - 12:20PM	E7 4053
ECE 327 101 [TUT]	**Wednesdays** *Jan 5 - Apr 6*	12:30PM - 01:20PM	E7 4053
ECE 627 001 [LEC]	**Mon, Thu** *Jan 5 - Apr 6*	10:00AM - 11:20AM	E7 4053

No tutorial session this week!

Class Schedule

Makeup lectures: Not used until I announce otherwise beforehand

Course	Meet Days	Meet Time	Location
ECE 327 001 [LEC]	**Mon, Thu** *Jan 5 - Apr 6*	10:00AM - 11:20AM	E7 4053
	Mondays *Jan 5, Jan 19, Feb 2, Feb 23, Mar 9, Mar 23*	11:30AM - 12:20PM	E7 4053
ECE 327 101 [TUT]	**Wednesdays** *Jan 5 - Apr 6*	12:30PM - 01:20PM	E7 4053
ECE 627 001 [LEC]	**Mon, Thu** *Jan 5 - Apr 6*	10:00AM - 11:20AM	E7 4053

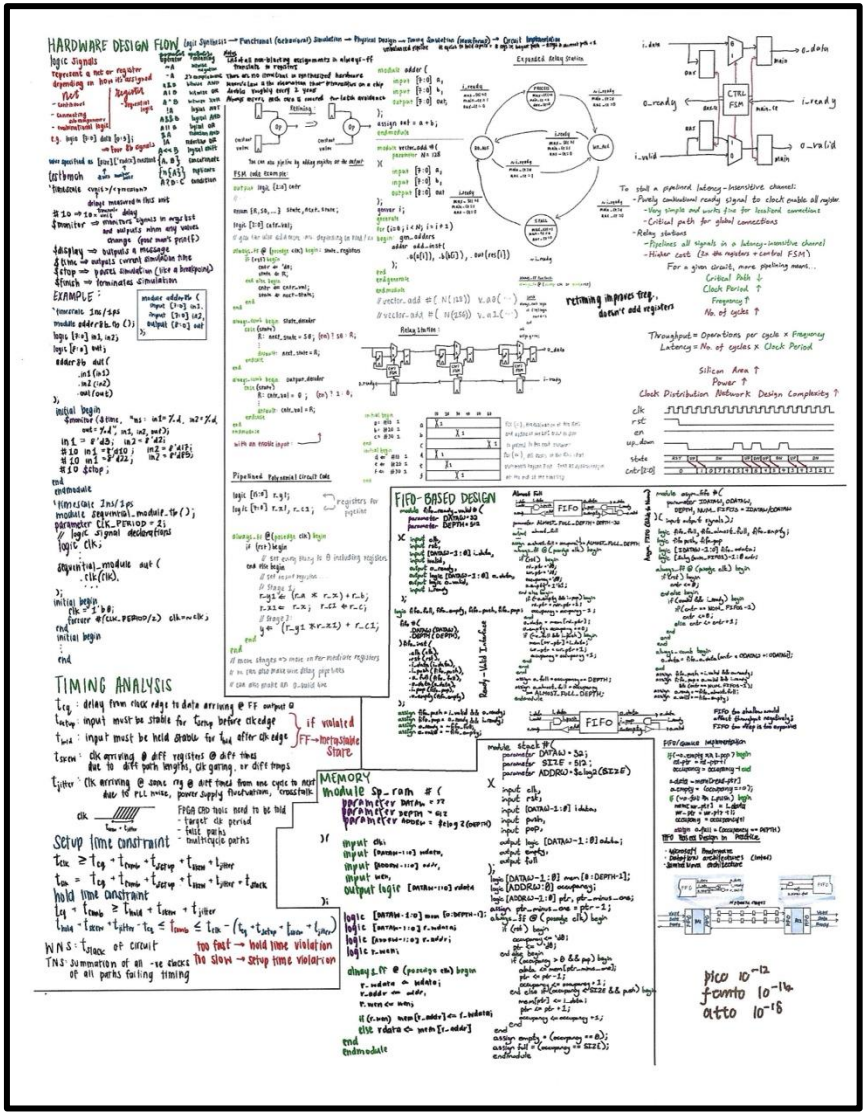
Course Overview: Grade Breakdown

Component	ECE 327	ECE 627
Labs (x4)	35%	25%
Project	-	25%
Midterm Exam	25%	20%
Final Exam	40%	30%

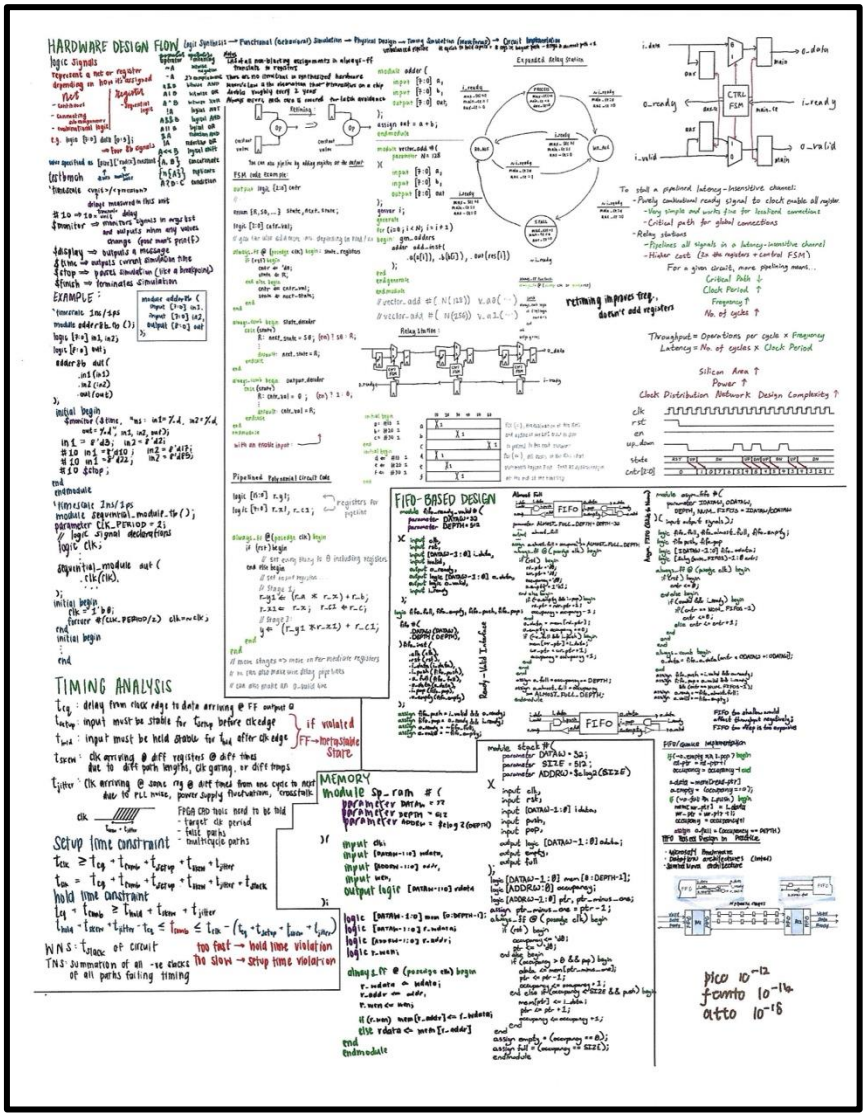
- Labs are done in groups of two – stay tuned for team registration instructions!
- **[ECE 327]** Individual lab assessment ~50% lab grade
- Paper-based exams, closed notes, no internet access, 1 letter-sized double-sided cheat sheet

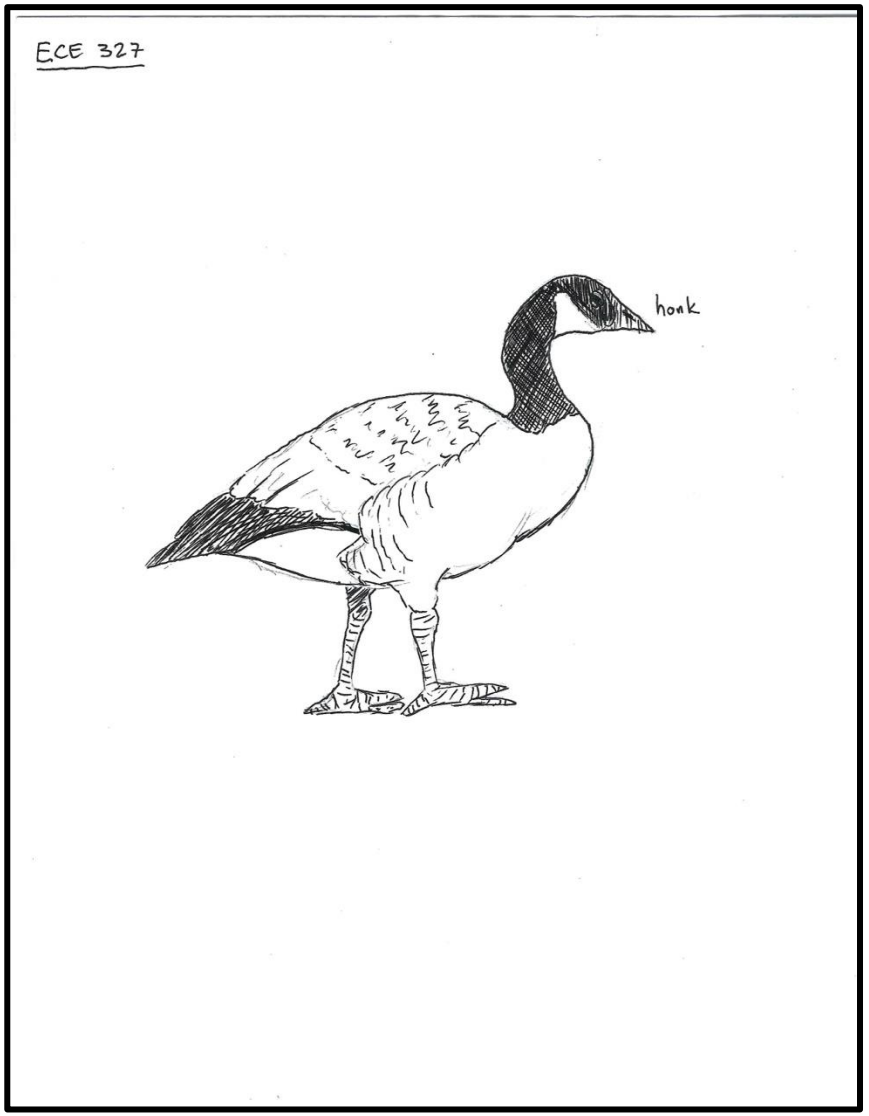
More details: https://outline.uwaterloo.ca/viewer/view/n7u3wu

Cheat Sheet Samples from Previous Years



Cheat Sheet Samples from Previous Years





Academic Integrity

- General discussion is encouraged
- Copying code is a BIG problem!
- Policy 71 is strictly enforced in this course
  - Both copying and allowing copying will be penalized
- “Integrity is doing the right thing, even when no one is watching.” – C. S. Lewis

Using GenAI (e.g., ChatGPT, Claude, Gemini)

- It can be a helpful resource for learning, but you cannot rely mainly on it.

- Even if it can solve an entire lab for you, …
  - The goal of the labs is for you to apply what you learn in this course in practice
  - 50% of the lab grade on the individual assessment of the lab's learning objectives (cannot use GenAI for this)

- You cannot use GenAI for midterm & final exams (no internet access allowed)

Resources

- Piazza Discussion Forum
  - Most effective way to ask questions about the lecture contents, labs, and/or exams
  - Registration Link
  - Access Code: ece327627w26@uwat

- LEARN
  - Where you can find all course materials (lecture slides, lab handouts, etc.)
  - Link: https://learn.uwaterloo.ca/d2l/home/1222924

(Constructive) Feedback

- Tell me about any issues or things to improve
  - In-person or via email
  - Office hours: Monday 12:30 – 1:30 @ E5-4003
  - Anonymous feedback on course LEARN page: *Submit* > *Surveys* > *Anonymous Course Feedback*

- Participation is useful … for everyone!
  - Point out a (non-trivial) mistake
  - Ask a really good question
  - Answer a “Bounty Question”

- 3 (non-trivial) mistakes in 1 lecture
  - Candy for everyone next lecture!

Let's get to the more important stuff ...

Hardware Design Layers of Abstraction

- Most computing devices today are silicon chips
- Can design hardware at different levels
  - **Semiconductor Physics:** electron/hole movement in transistors
  - **Physical Implementation:** layout of N/P-wells, gates, source/drain
  - **Circuit Implementation:** voltages, currents, transistors = switches
  - **Digital Logic:** gates with binary (0 or 1) inputs/outputs
  - **Register Transfer Level (RTL):** logic/arithmetic operations performed on data signals as they flow between registers
  - **System Integration:** blocks or IP cores interconnected to implement a complete system
- Layers of abstraction boost productivity!
  - Very challenging to design a chip with ~100B transistors if you reason about atoms or layout of independent transistors.

Hardware Design Layers of Abstraction

- Most computing devices today are silicon chips
- Can design hardware at different levels
  - **Semiconductor Physics:** electron/hole movement in transistors
  - **Physical Implementation:** layout of N/P-wells, gates, source/drain
  - **Circuit Implementation:** voltages, currents, transistors = switches
  - **Digital Logic:** gates with binary (0 or 1) inputs/outputs
  - **Register Transfer Level (RTL):** logic/arithmetic operations performed on data signals as they flow between registers
  - **System Integration:** blocks or IP cores interconnected to implement a complete system
- Layers of abstraction boost productivity!
  - Very challenging to design a chip with ~100B transistors if you reason about atoms or layout of independent transistors.

Hardware Description Languages (HDLs)

- Describe behavior and structure of hardware circuits
- **Verilog** and **VHDL** are two IEEE standardized HDLs
  - The syntax of the two languages differs in many ways
  - Concepts learned for one can be applied to the other
  - We will focus on SystemVerilog (extension of Verilog)
- Two main objectives:
  - Describe HW to automatically synthesize & implement circuits using physical design CAD tools
  - Verify HW functionality and/or timing using simulators
- NOT programming languages!
  - Explicitly include the notion of time (cycle-level behavior) and concurrency

One Important Rule: Think in Hardware!

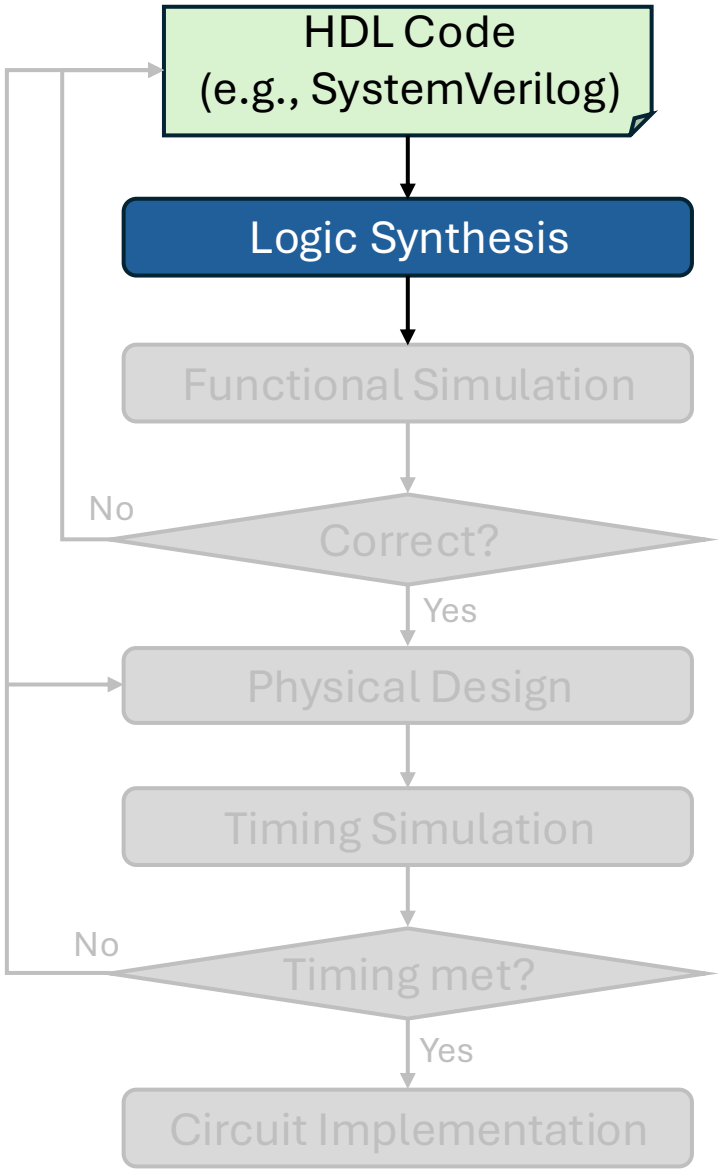
- When first learning HDLs, people tend to write "software-like" code
- Difficult to determine what hardware circuit this code generates when synthesized using CAD tools
- When writing HDL, think of hardware components (e.g., registers, logic gates, memories, adders, multipliers, MUXs) and not software instructions
- One good general guideline …

If you cannot determine the hardware circuit described by an HDL, then you should re-think your implementation!

Hardware Design Flow

Logic Synthesis:

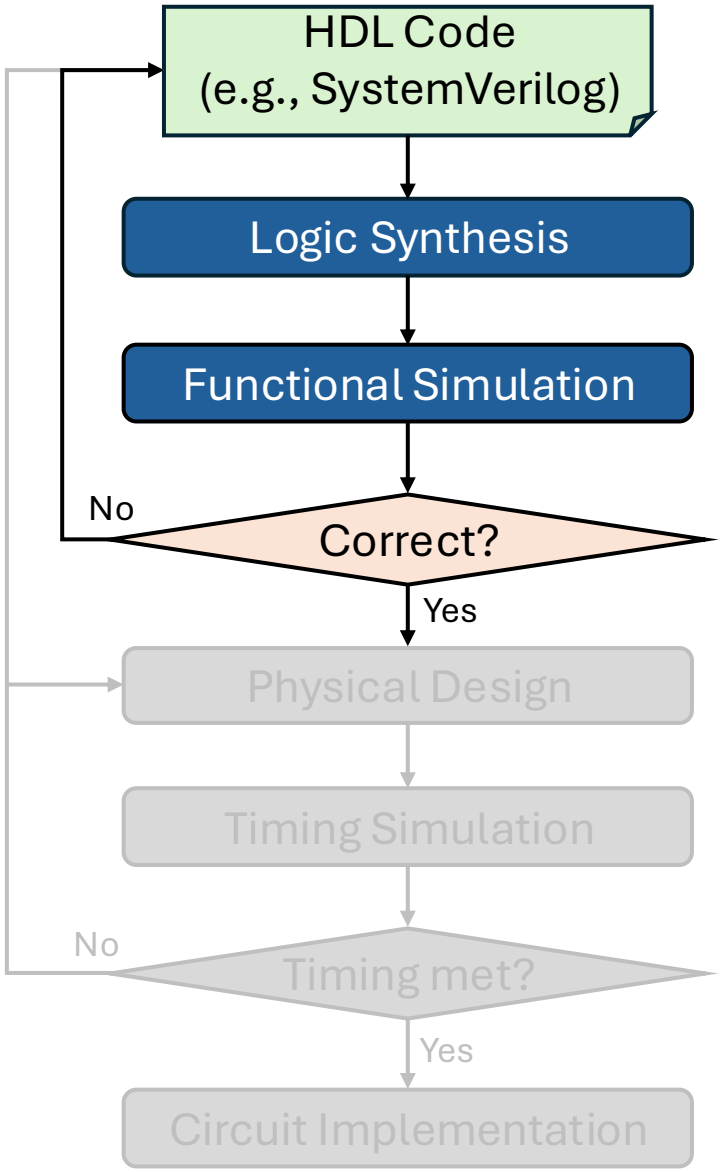
- Translate (or "compile") the HDL code into a circuit netlist (i.e., logic gates + their connectivity)
- Manipulate the generated netlist to produce an equivalent, but more optimized circuit
- Optimization goal differs based on project & implementation target:
  - Minimize area
  - Maximize speed



Hardware Design Flow

Functional (Behavioral) Simulation:

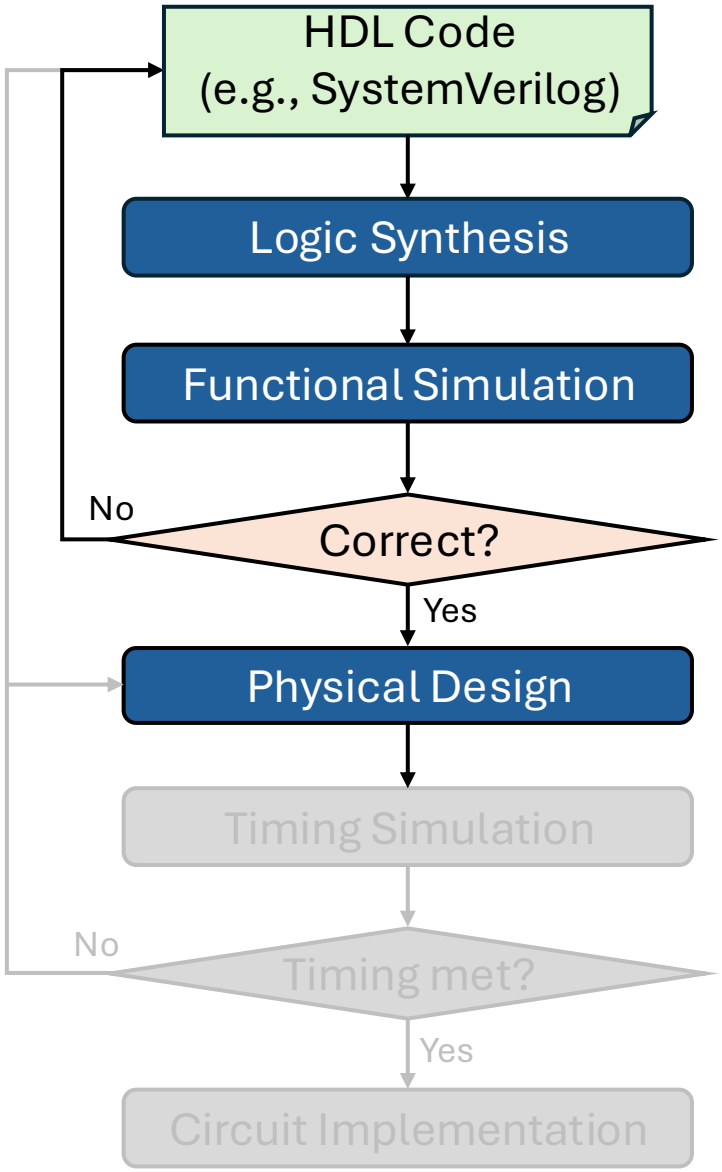
- Verifies that generated circuit netlist will function as intended
- Assumes zero-delay gates & wires (signals propagate instantaneously)
- User provide values to be applied to the netlist inputs by the simulator
- **Simulation testbenches** typically written in non-synthesizable HDL (verification superset of language)
- Results typically provided as timing diagram or "*waveforms*"



Hardware Design Flow

Physical Design:

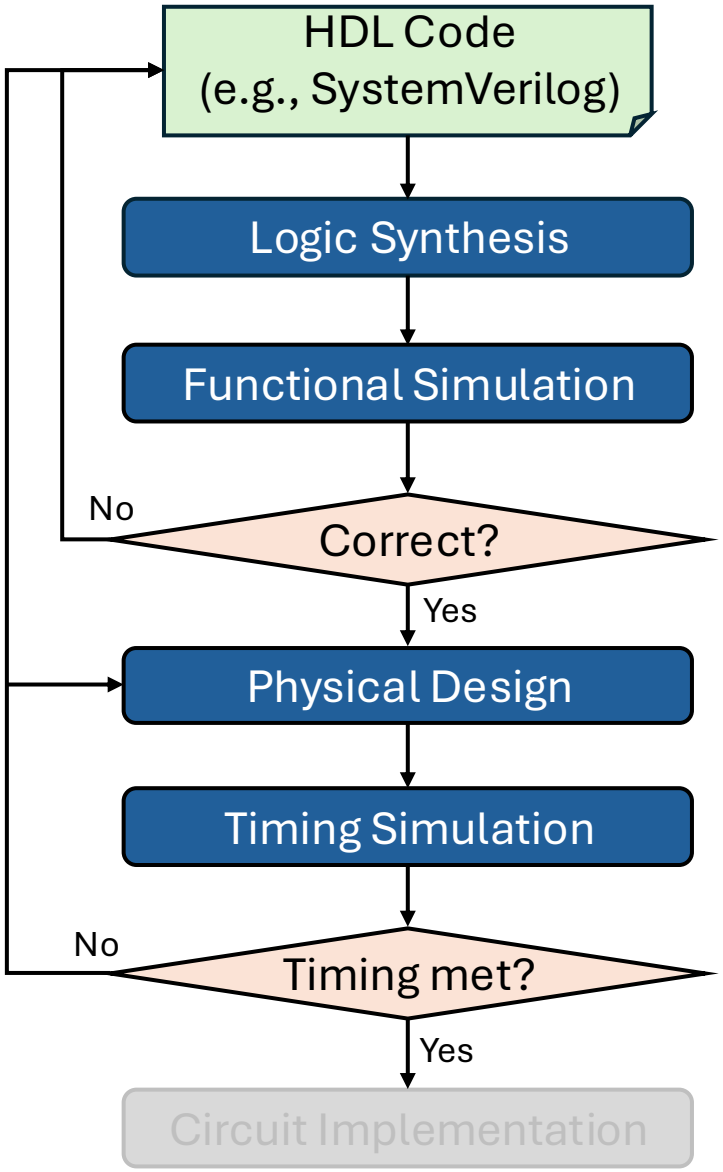
- Map synthesized circuit netlist to available components based on the target implementation technology (sometimes considered part of the synthesis stage)
  - Standard cells for custom chips (ASIC)
  - Lookup tables & other blocks for reconfigurable hardware (FPGA)
- Determine the placement of these components
- Route wires between them to realize the desired circuit



Hardware Design Flow

Timing Simulation:

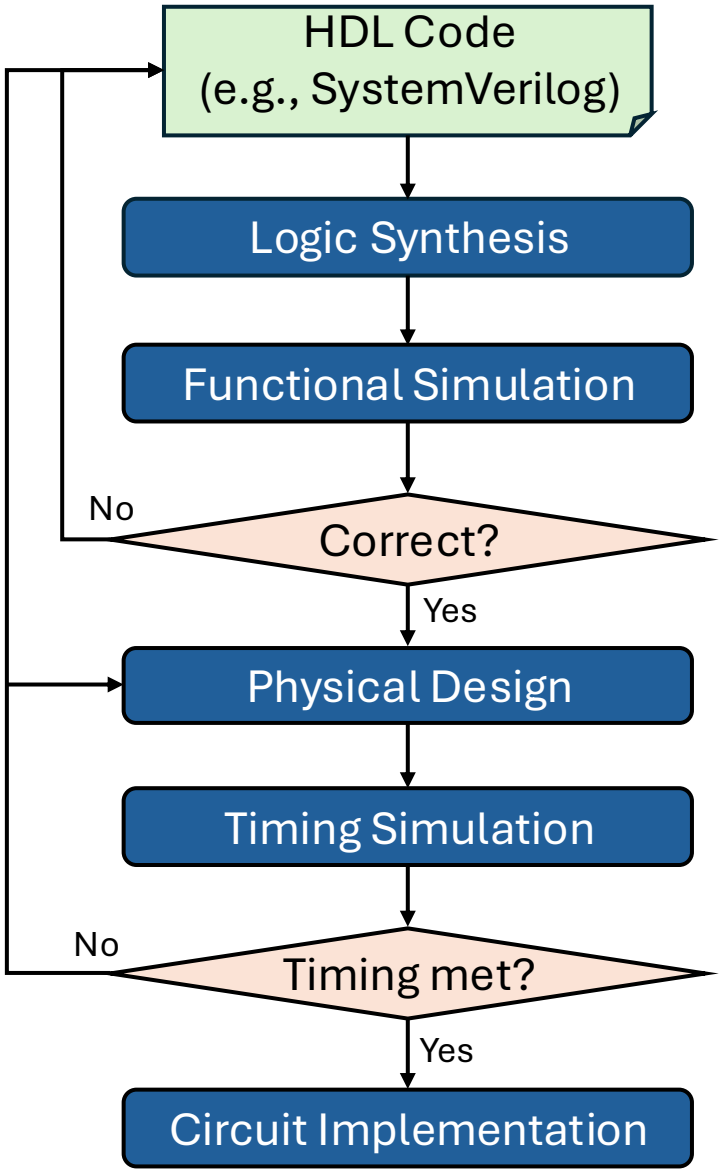
- After physical design, simulate the post-implementation netlist with accurate component & wire delays
  - Component delay is the time to produce an output when inputs change
  - Wire delay is the time for a signal to propagate between components
- Verify that the implemented circuit can run at the desired speed (i.e., *meets timing*)
  - If not, change synthesis or physical design optimization goals or rethink circuit implementation



Hardware Design Flow

Circuit Implementation:

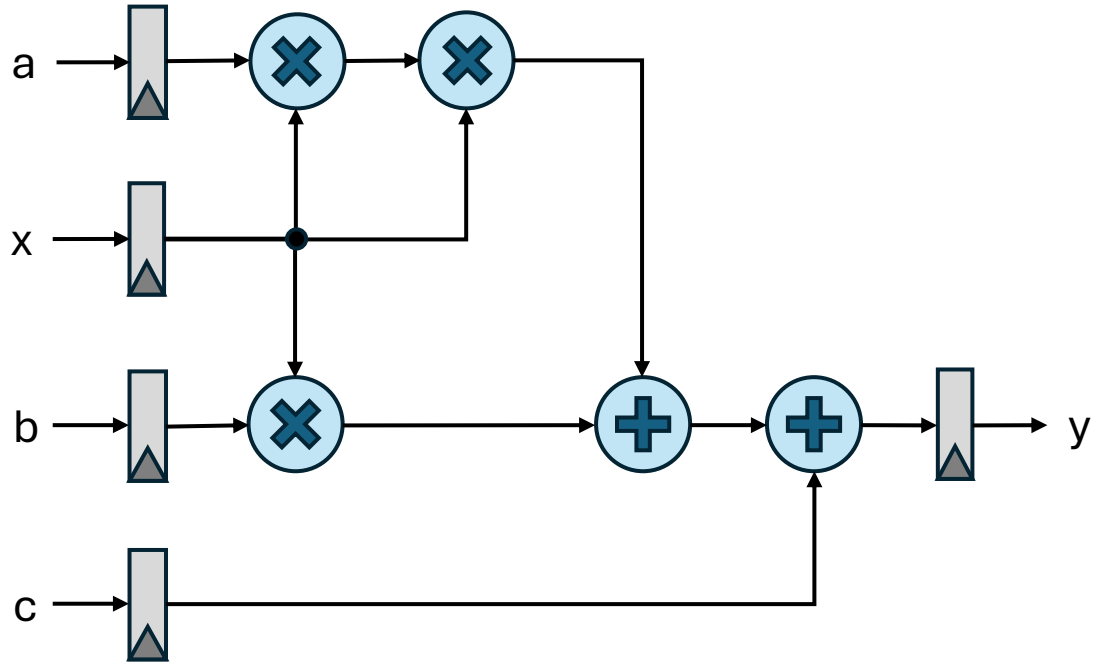
- After making sure the circuit meets all specifications, it is implemented on an actual chip
  - Chip fabrication for ASIC:
    - Takes several weeks or months and costs tens or hundreds of millions of dollars
    - As efficient as it gets!
  - Chip configuration for FPGA:
    - Takes several seconds and each device costs hundreds or thousands of dollars
    - Around 8x bigger in area and 4x slower



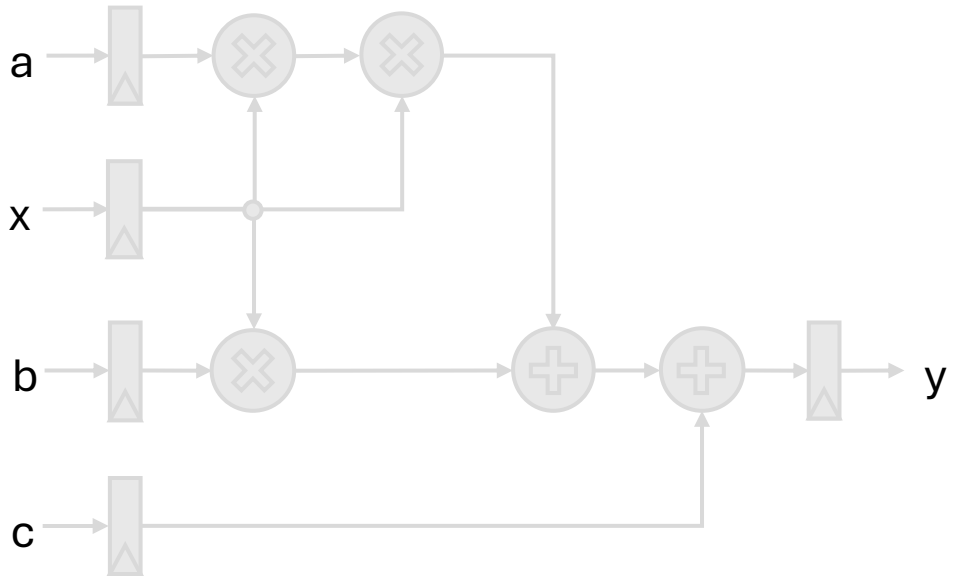
Example: Polynomial Circuit

We want to implement a synchronous circuit that takes as inputs a value x and 3 coefficients (a, b, c) every clock cycle and computes the value

$$y = ax^2 + bx + c$$

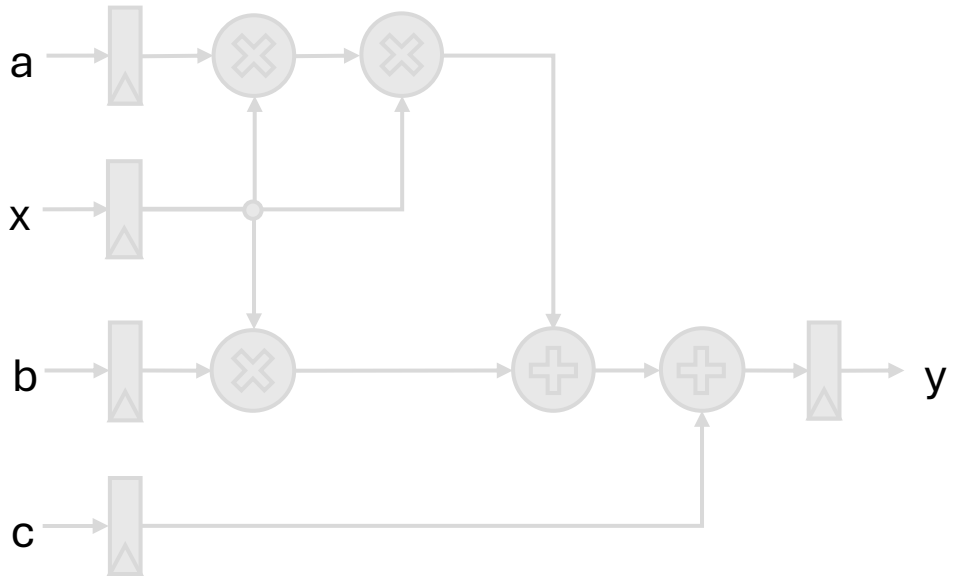


Example: Polynomial Circuit




```
module poly (
     input clk,
     input rst,
     input [7:0] x,
     input [7:0] a,
     input [7:0] b,
     input [7:0] c,
     output [23:0] y
);

endmodule
```

Example: Polynomial Circuit

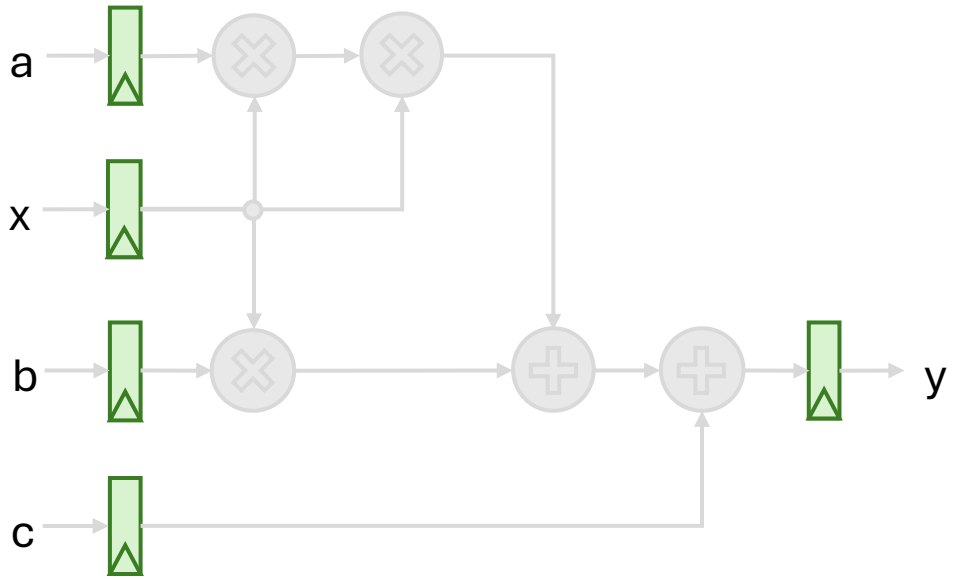


A hardware circuit is defined as a `module`

```
module poly (
    input clk,
    input rst,
    input [7:0] x,
    input [7:0] a,
    input [7:0] b,
    input [7:0] c,
    output [23:0] y
);

endmodule
```

All inputs and outputs have defined bitwidths

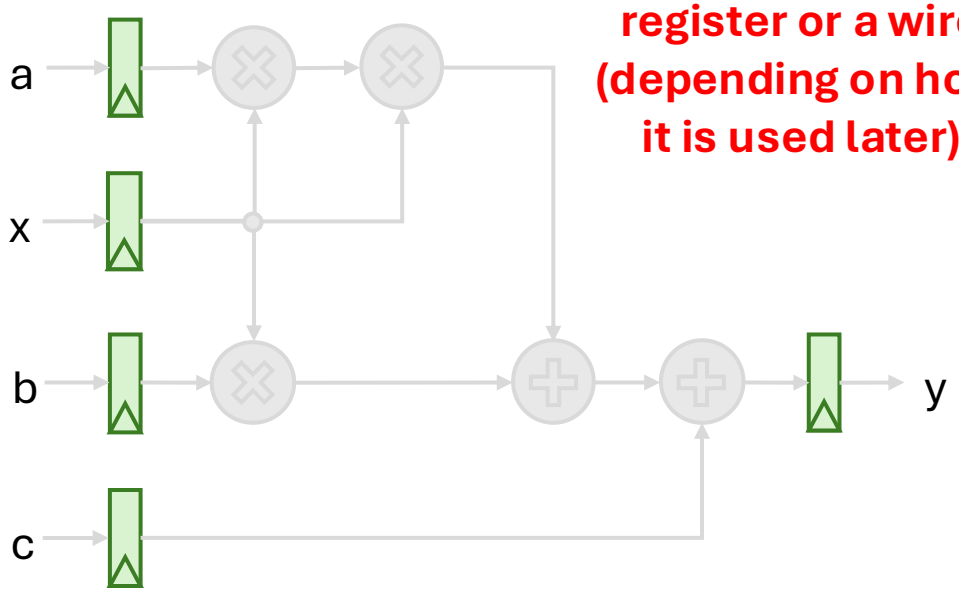
Example: Polynomial Circuit




```
module poly (
     input clk,
     input rst,
     input [7:0] x,
     input [7:0] a,
     input [7:0] b,
     input [7:0] c,
     output [23:0] y
);

logic [7:0] r_x, r_a, r_b, r_c;
logic [23:0] r_y;
```

```
endmodule
```

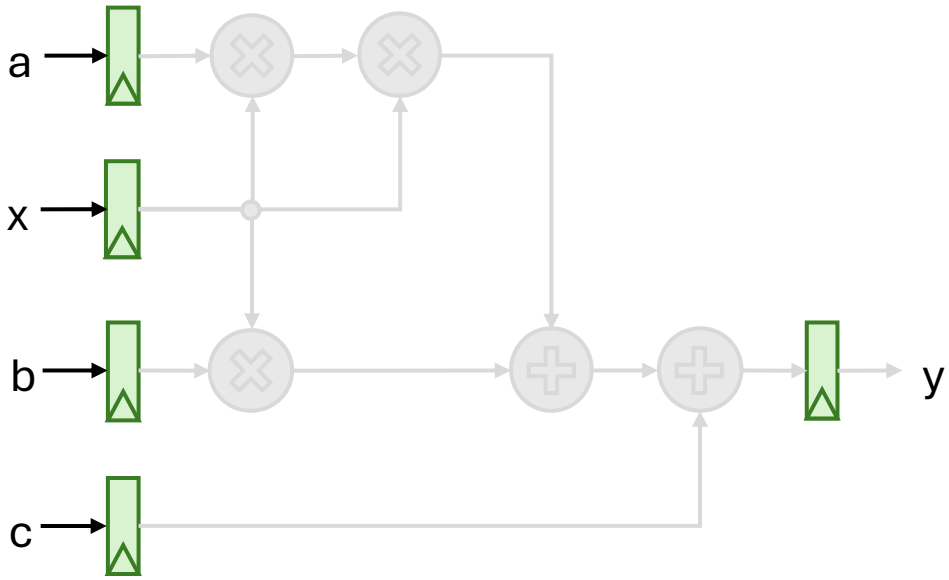
Example: Polynomial Circuit



The keyword `logic` defines a register or a wire (depending on how it is used later)

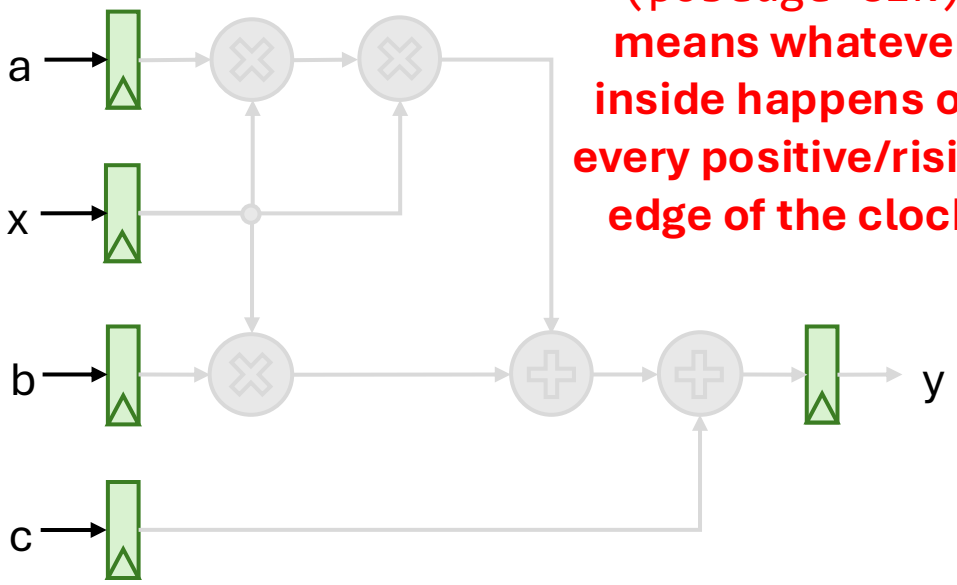
```
module poly (
      input clk,
      input rst,
      input [7:0] x,
      input [7:0] a,
      input [7:0] b,
      input [7:0] c,
      output [23:0] y
);

logic [7:0] r_x, r_a, r_b, r_c;
logic [23:0] r_y;

endmodule
```

Example: Polynomial Circuit




```
module poly (
     input clk,
     input rst,
     input [7:0] x,
     input [7:0] a,
     input [7:0] b,
     input [7:0] c,
     output [23:0] y
);

logic [7:0] r_x, r_a, r_b, r_c;
logic [23:0] r_y;

always_ff @(posedge clk) begin
 if(rst) begin
  r_x <= 0; r_a <= 0;
  r_b <= 0; r_c <= 0;
 end else begin
  // register inputs
  r_x <= x; r_a <= a;
  r_b <= b; r_c <= c;


 end
end


endmodule
```

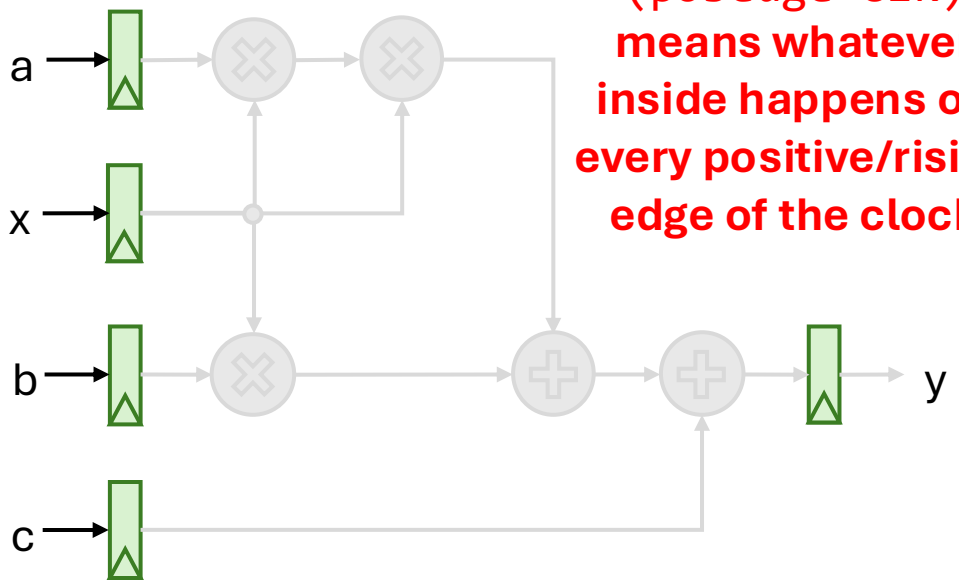
Example: Polynomial Circuit



always_ff @ (posedge clk) **means whatever inside happens on every positive/rising edge of the clock**

```
module poly (
     input clk,
     input rst,
     input [7:0] x,
     input [7:0] a,
     input [7:0] b,
     input [7:0] c,
     output [23:0] y
);

logic [7:0] r_x, r_a, r_b, r_c;
logic [23:0] r_y;

always_ff @(posedge clk) begin
 if(rst) begin
  r_x <= 0; r_a <= 0;
  r_b <= 0; r_c <= 0;
 end else begin
  // register inputs
  r_x <= x; r_a <= a;
  r_b <= b; r_c <= c;


 end
end


endmodule
```

Example: Polynomial Circuit

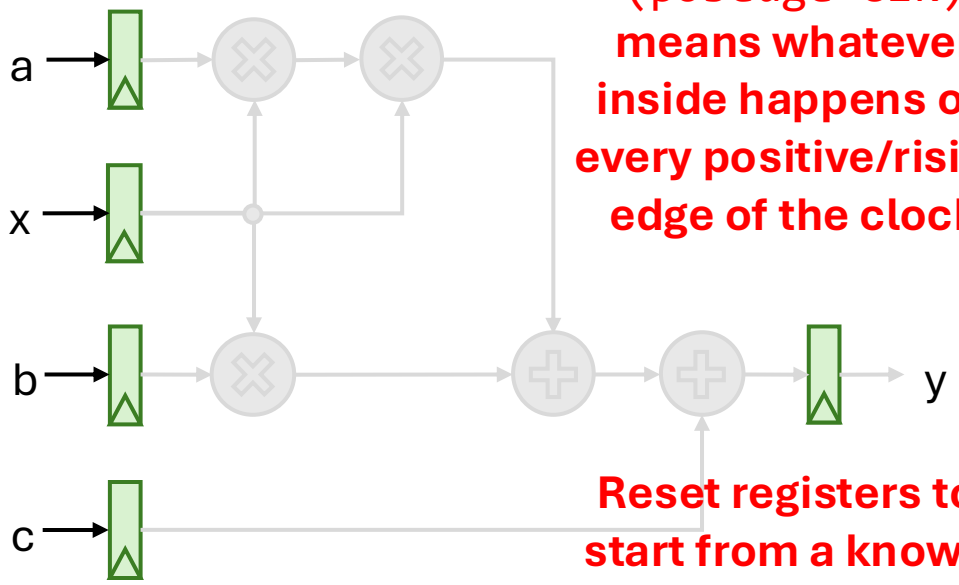


always_ff @ (posedge clk) **means whatever inside happens on every positive/rising edge of the clock**

```
module poly (
      input clk,
      input rst,
      input [7:0] x,
      input [7:0] a,
      input [7:0] b,
      input [7:0] c,
      output [23:0] y
);

logic [7:0] r_x, r_a, r_b, r_c;
logic [23:0] r_y;

always_ff @(posedge clk) begin
 if(rst) begin
  r_x <= 0; r_a <= 0;
  r_b <= 0; r_c <= 0;
 end else begin
  // register inputs
  r_x <= x; r_a <= a;
  r_b <= b; r_c <= c;


 end
end


endmodule
```

Non-blocking assignments mean they all happen concurrently (more on this in next lecture)

Example: Polynomial Circuit



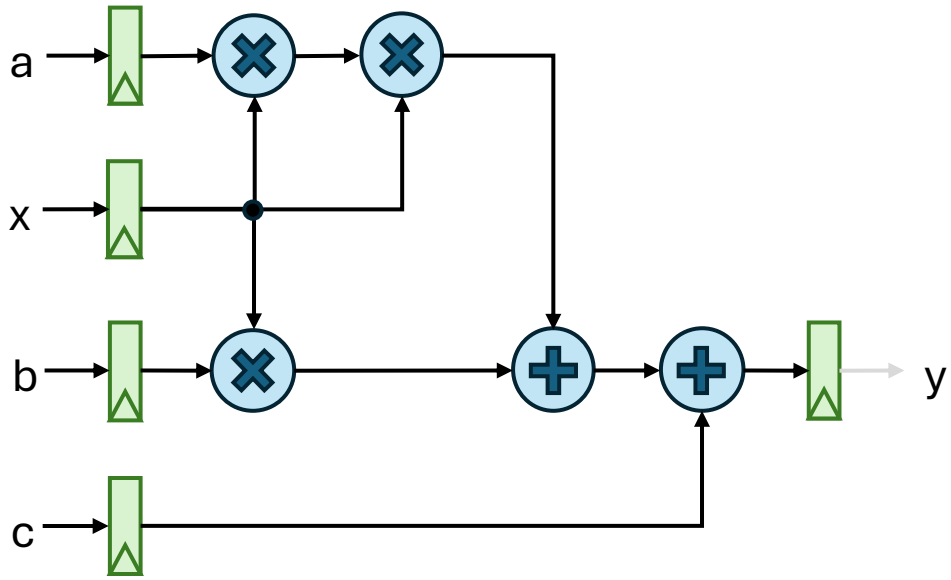
always_ff @ (posedge clk) **means whatever inside happens on every positive/rising edge of the clock**

Reset registers to start from a known state

```
module poly (
     input clk,
     input rst,
     input [7:0] x,
     input [7:0] a,
     input [7:0] b,
     input [7:0] c,
     output [23:0] y
);

logic [7:0] r_x, r_a, r_b, r_c;
logic [23:0] r_y;

always_ff @(posedge clk) begin
 if(rst) begin
  r_x <= 0; r_a <= 0;
  r_b <= 0; r_c <= 0;
 end else begin
  // register inputs
  r_x <= x; r_a <= a;
  r_b <= b; r_c <= c;


 end
end


endmodule
```

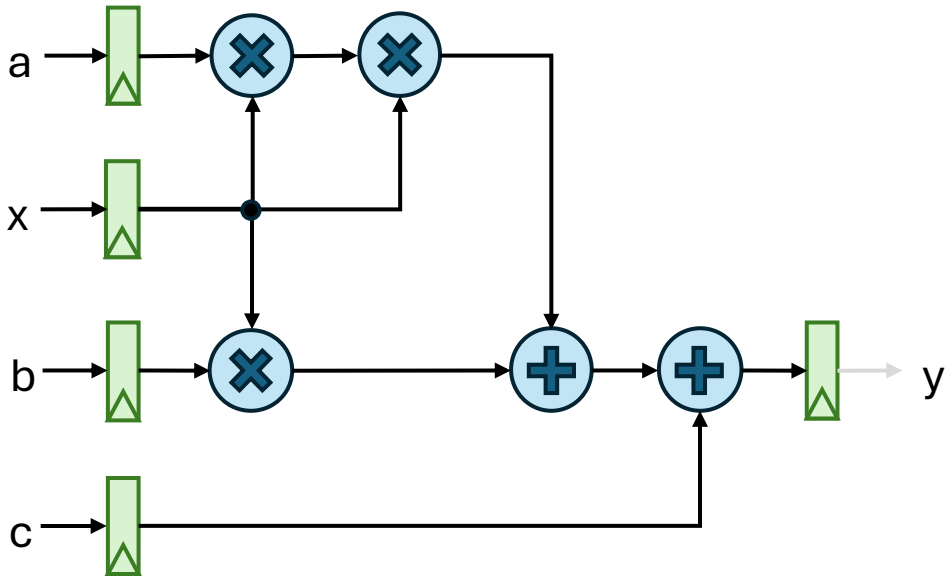
Non-blocking assignments mean they all happen concurrently (more on this in next lecture)

Example: Polynomial Circuit




```
module poly (
     input clk,
     input rst,
     input [7:0] x,
     input [7:0] a,
     input [7:0] b,
     input [7:0] c,
     output [23:0] y
);

logic [7:0] r_x, r_a, r_b, r_c;
logic [23:0] r_y;

always_ff @(posedge clk) begin
 if(rst) begin
  r_x <= 0; r_a <= 0;
  r_b <= 0; r_c <= 0; r_y <= 0;
 end else begin
  // register inputs
  r_x <= x; r_a <= a;
  r_b <= b; r_c <= c;
  // register output
  r_y <= r_a*r_x*r_x + r_b*r_x + r_c;
 end
end


endmodule
```

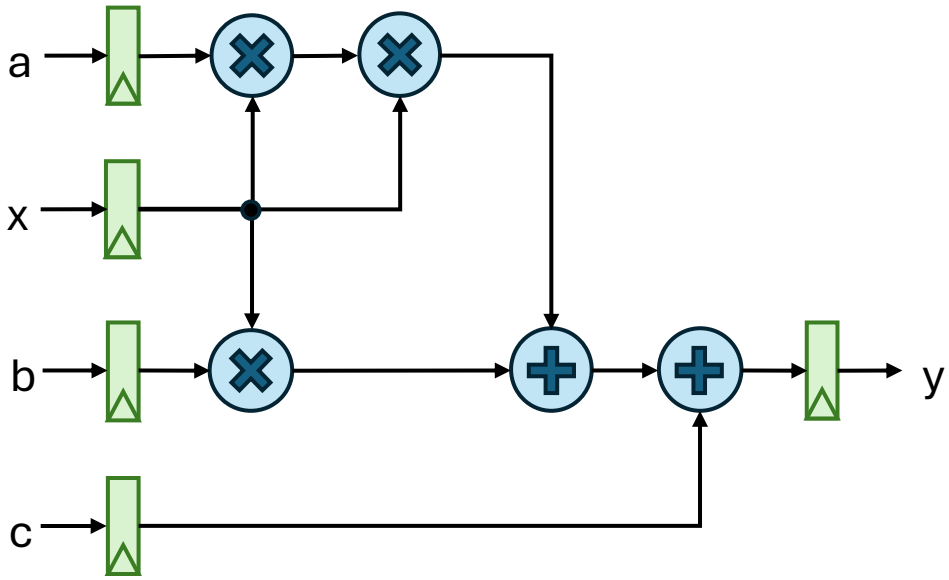
Example: Polynomial Circuit



Behavioral description of circuit functionality

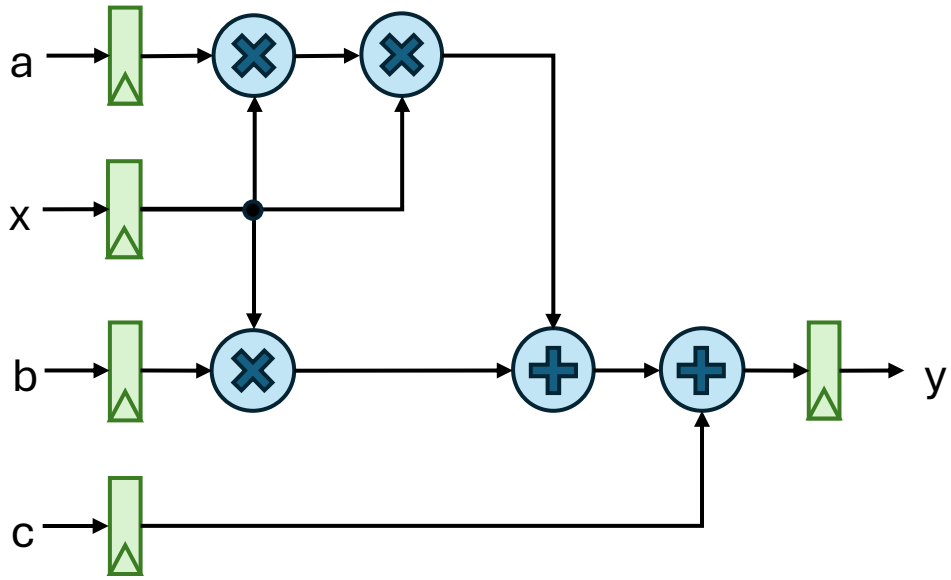
```
module poly (
     input clk,
     input rst,
     input [7:0] x,
     input [7:0] a,
     input [7:0] b,
     input [7:0] c,
     output [23:0] y
);

logic [7:0] r_x, r_a, r_b, r_c;
logic [23:0] r_y;

always_ff @(posedge clk) begin
 if(rst) begin
  r_x <= 0; r_a <= 0;
  r_b <= 0; r_c <= 0; r_y <= 0;
 end else begin
  // register inputs
  r_x <= x; r_a <= a;
  r_b <= b; r_c <= c;
  // register output
  r_y <= r_a*r_x*r_x + r_b*r_x + r_c;
 end
end


endmodule
```

Example: Polynomial Circuit




```
module poly (
     input clk,
     input rst,
     input [7:0] x,
     input [7:0] a,
     input [7:0] b,
     input [7:0] c,
     output [23:0] y
);

logic [7:0] r_x, r_a, r_b, r_c;
logic [23:0] r_y;

always_ff @(posedge clk) begin
 if(rst) begin
  r_x <= 0; r_a <= 0;
  r_b <= 0; r_c <= 0; r_y <= 0;
 end else begin
  // register inputs
  r_x <= x; r_a <= a;
  r_b <= b; r_c <= c;
  // register output
  r_y <= r_a*r_x*r_x + r_b*r_x + r_c;
 end
end

assign y = r_y;

endmodule
```

Example: Polynomial Circuit

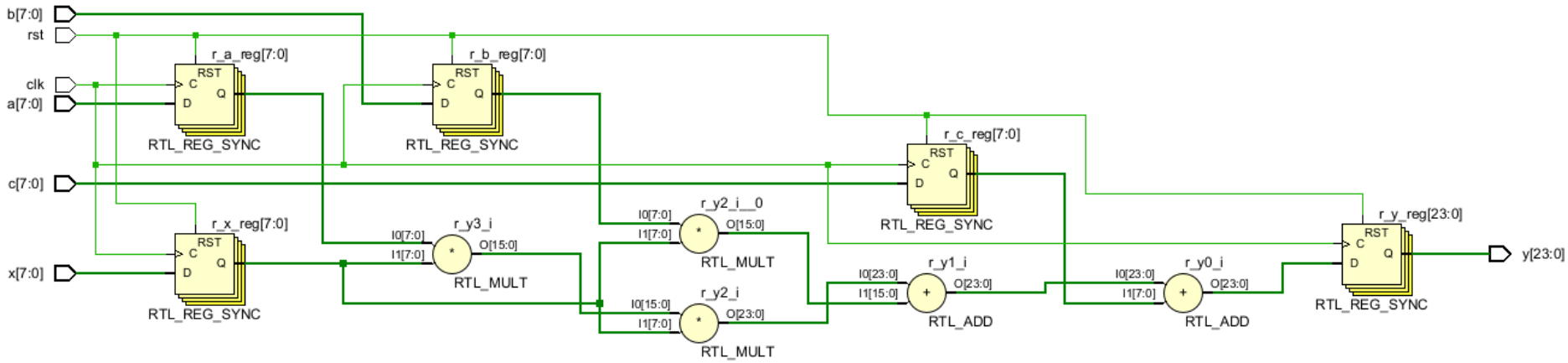



```
module poly (
     input clk,
     input rst,
     input [7:0] x,
     input [7:0] a,
     input [7:0] b,
     input [7:0] c,
     output [23:0] y
);

logic [7:0] r_x, r_a, r_b, r_c;
logic [23:0] r_y;

always_ff @(posedge clk) begin
 if(rst) begin
  r_x <= 0; r_a <= 0;
  r_b <= 0; r_c <= 0; r_y <= 0;
 end else begin
  // register inputs
  r_x <= x; r_a <= a;
  r_b <= b; r_c <= c;
  // register output
  r_y <= r_a*r_x*r_x + r_b*r_x + r_c;
 end
end

assign y = r_y;

endmodule
```

Continuous assignment of output y to the value of register r_y →

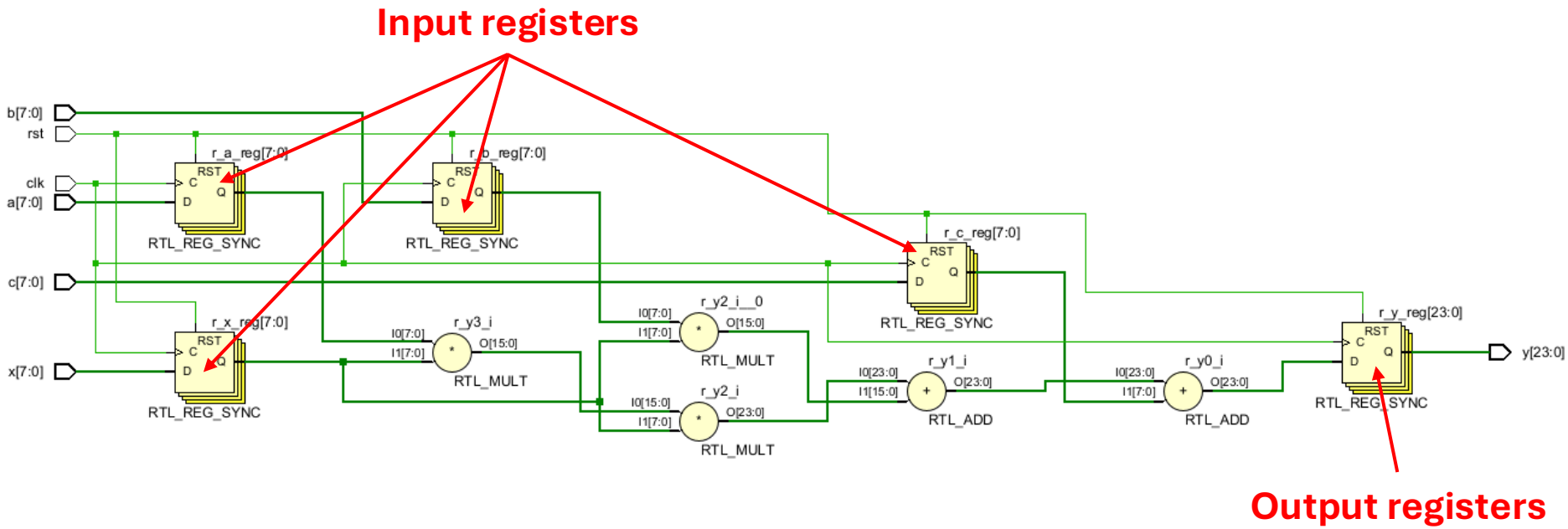
Example: Polynomial Circuit

Synthesized netlist (before technology mapping):



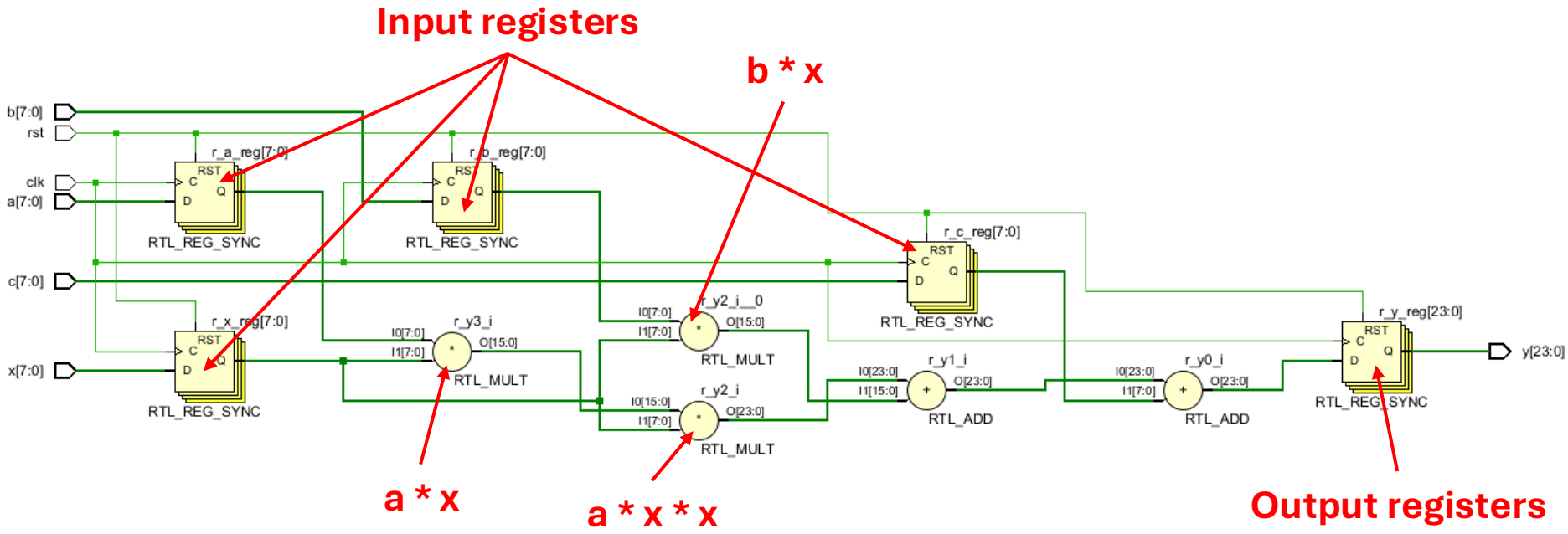
Example: Polynomial Circuit

Synthesized netlist (before technology mapping):



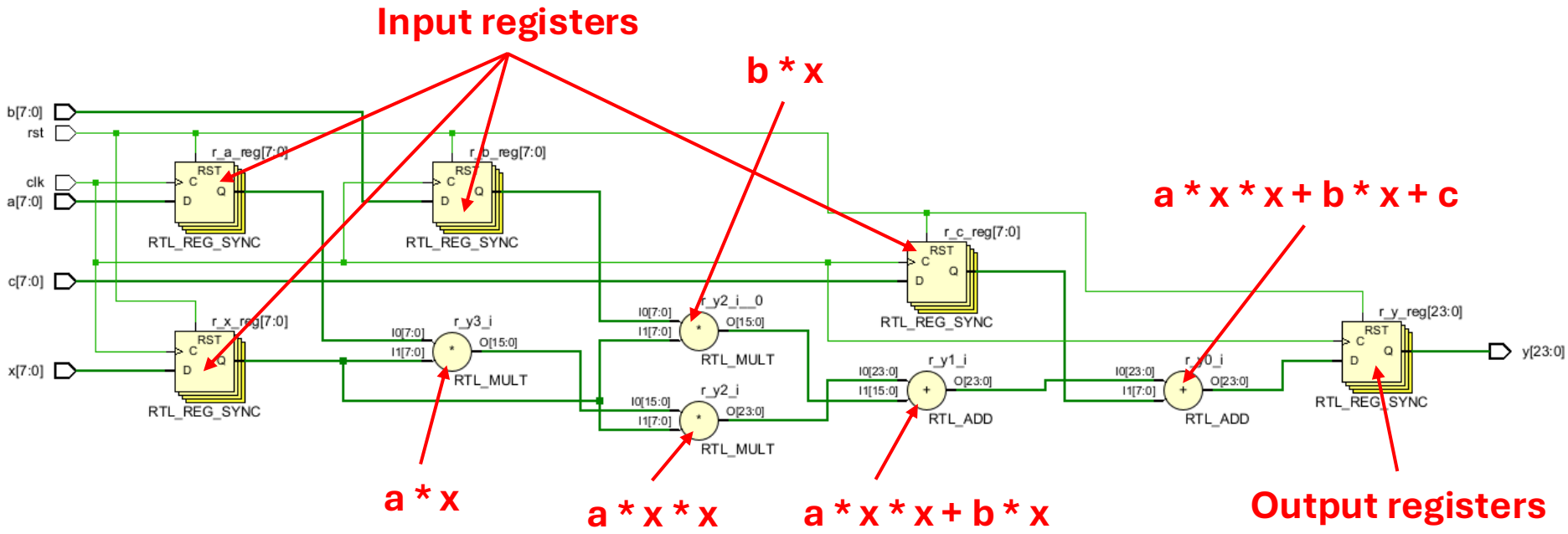
Example: Polynomial Circuit

Synthesized netlist (before technology mapping):



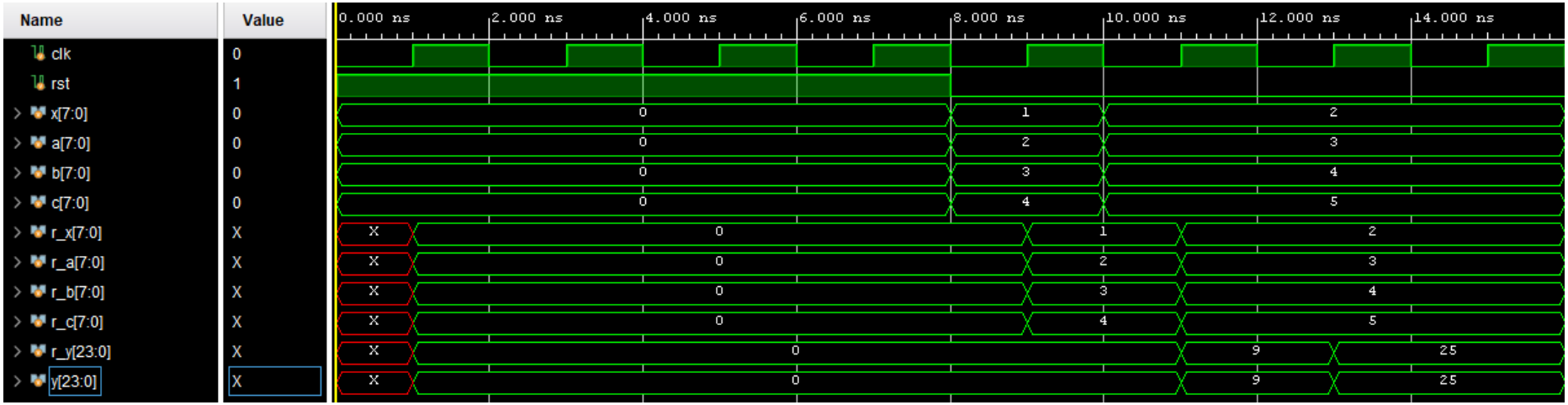
Example: Polynomial Circuit

Synthesized netlist (before technology mapping):



Example: Polynomial Circuit

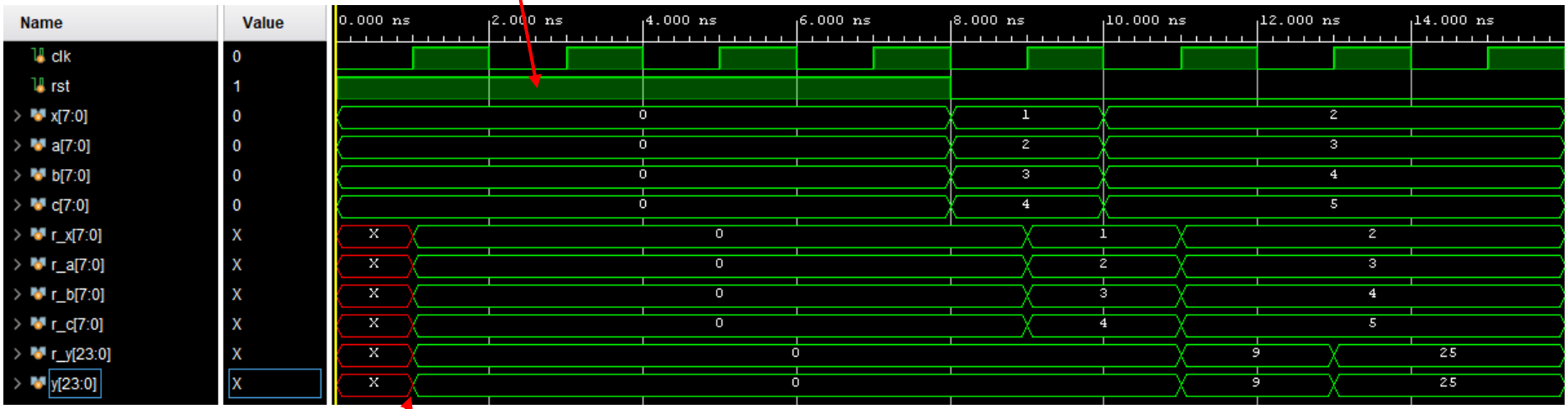
Functional (Behavioral) simulation:



Example: Polynomial Circuit

Functional (Behavioral) simulation:

`rst` **signal asserted for 4 cycles**



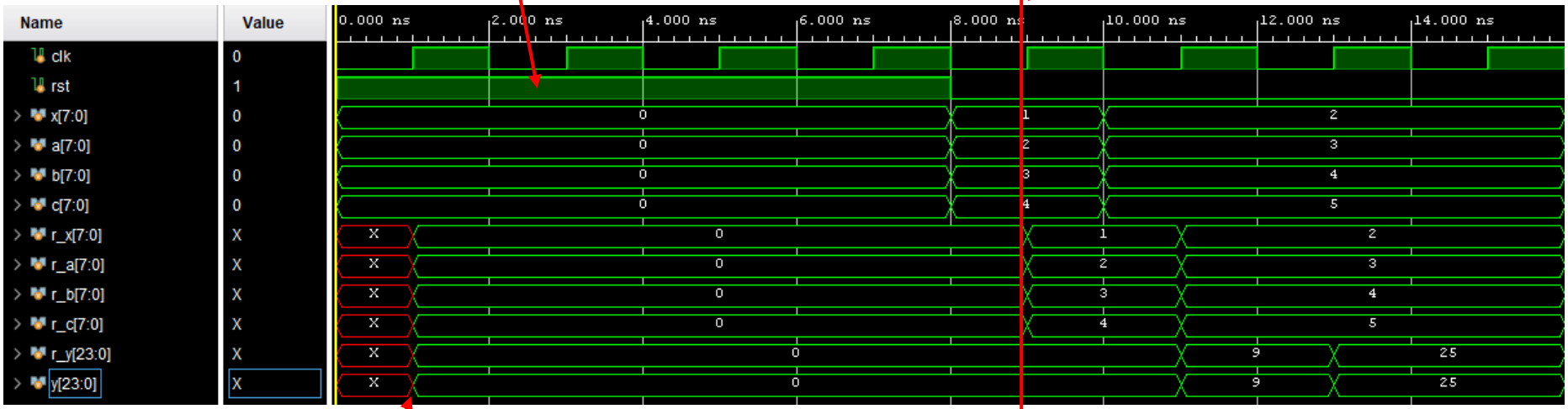
Input & Output register values are reset to zero on rising clock edge. Before that, they are undefined (X)

Example: Polynomial Circuit

Functional (Behavioral) simulation:

Input registers capture first values on first rising clock edge after reset is low. Output register is concurrently set to result of input register values (zeros)

`rst` signal asserted for 4 cycles



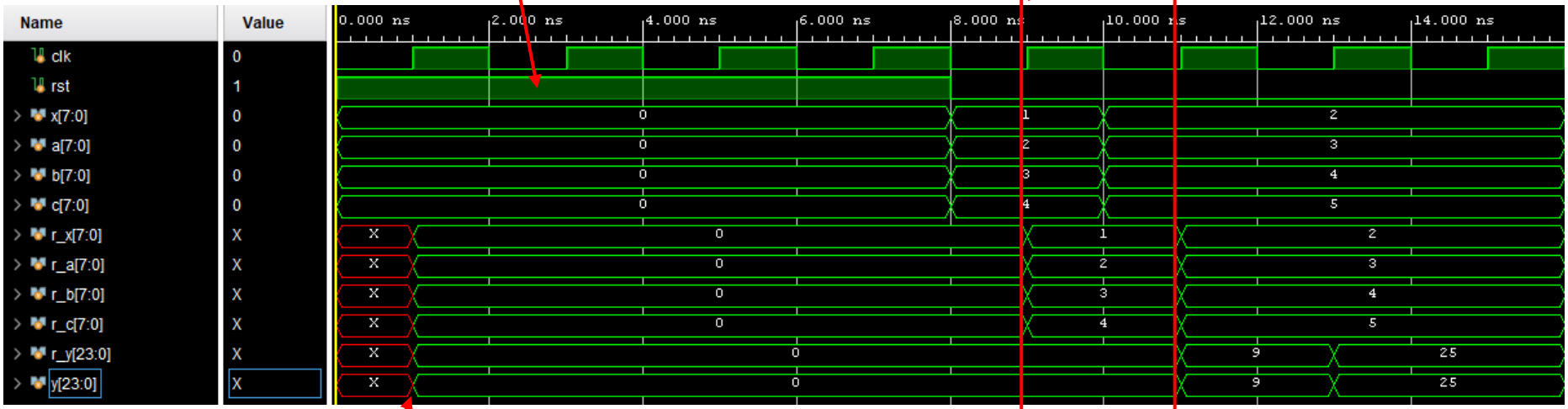
Input & Output register values are reset to zero on rising clock edge. Before that, they are undefined (X)

Example: Polynomial Circuit

Input registers capture first values on first rising clock edge after reset is low. Output register is concurrently set to result of input register values (zeros)

Functional (Behavioral) simulation:

`rst` signal asserted for 4 cycles



Input & Output register values are reset to zero on rising clock edge. Before that, they are undefined (X)

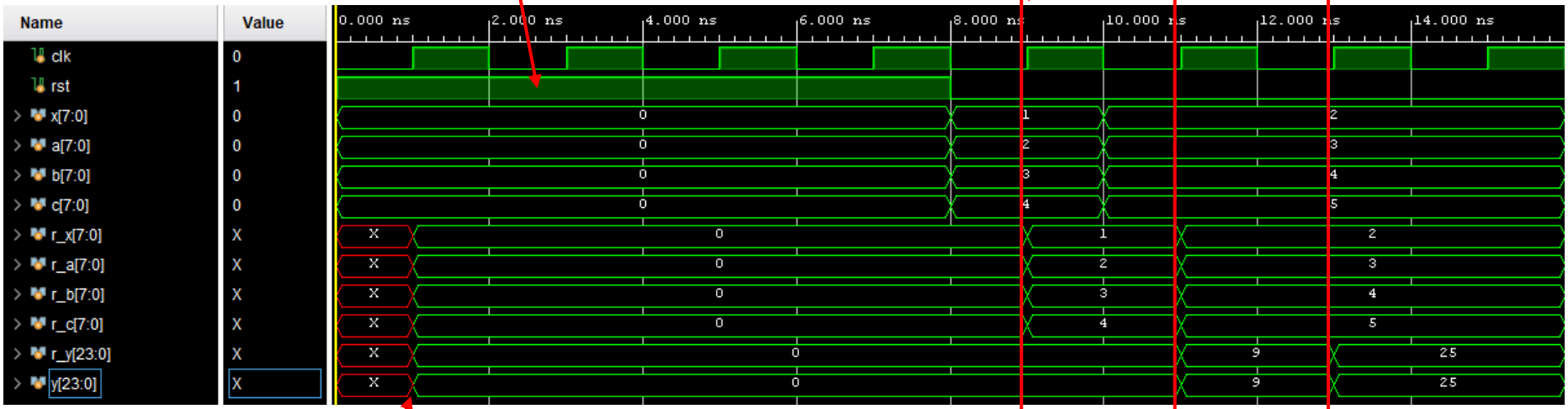
Input registers capture new values on second rising clock edge. Output register is concurrently set to result of input register values (2*1*1+3*1+4)

Example: Polynomial Circuit

Functional (Behavioral) simulation:

Input registers capture first values on first rising clock edge after reset is low. Output register is concurrently set to result of input register values (zeros)

`rst` signal asserted for 4 cycles



Input & Output register values are reset to zero on rising clock edge. Before that, they are undefined (X)

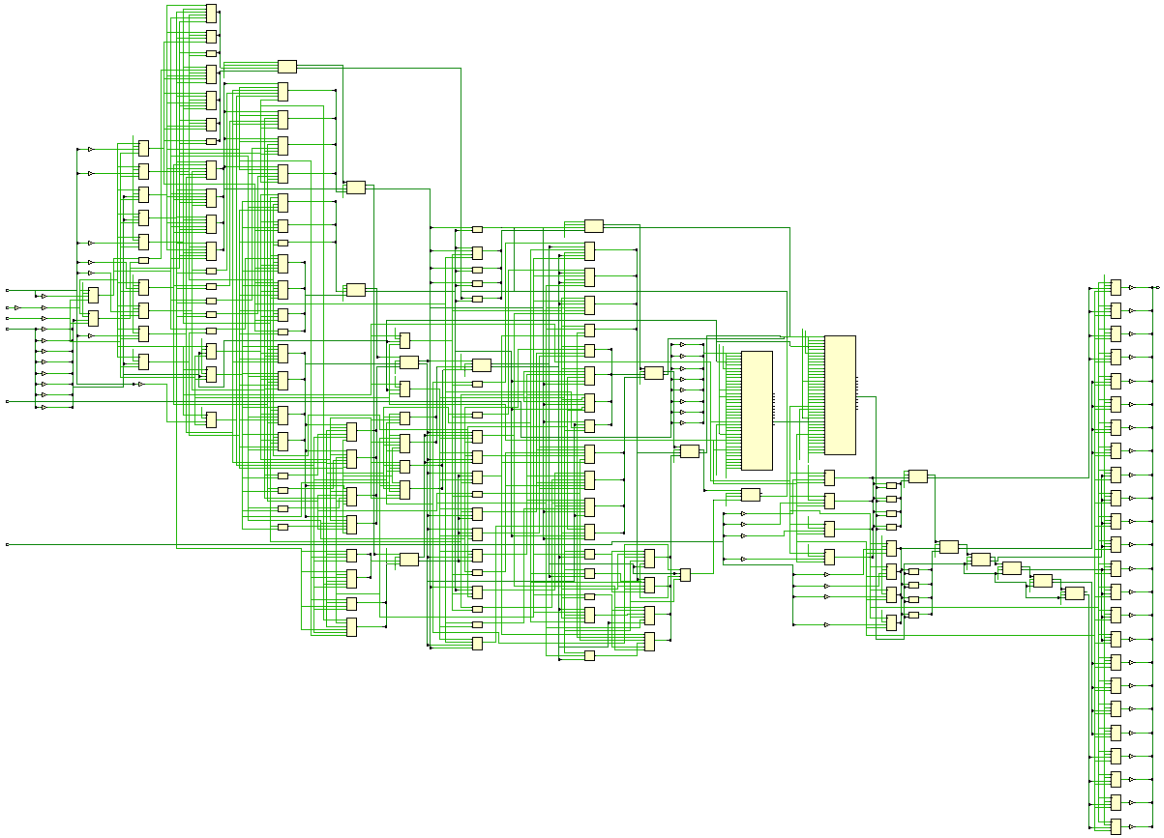
Input registers capture new values on second rising clock edge. Output register is concurrently set to result of input register values (2*1*1+3*1+4)

Output register is set to result for input register values (3*2*2+4*2+5)

Example: Polynomial Circuit

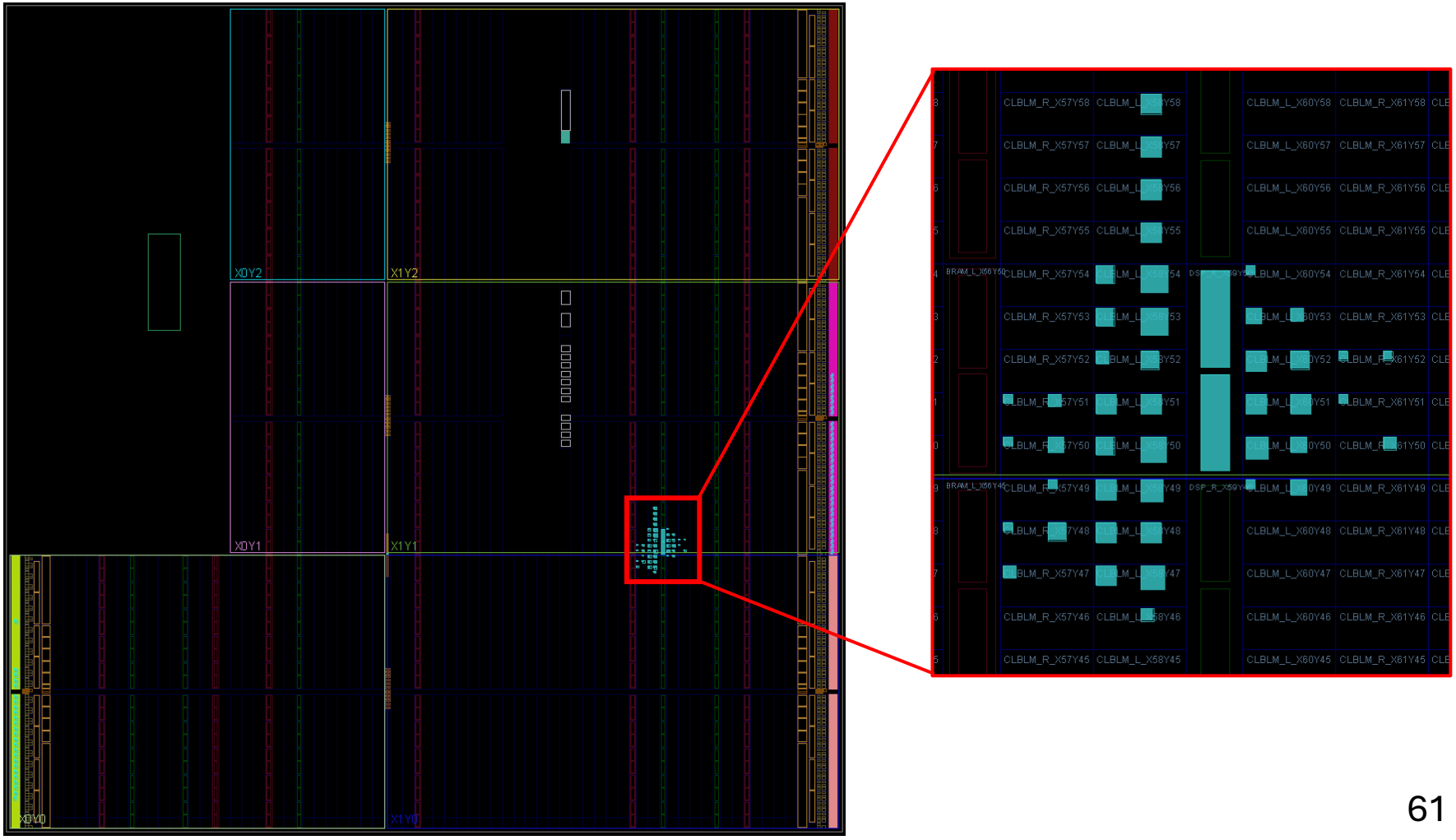
Technology mapped netlist (FPGA):

Complicated but if you investigate carefully, it consists of flip-flops (FFs), lookup tables (LUTs) & 2 arithmetic blocks (DSPs)



Example: Polynomial Circuit

Implementation (FPGA):



Next Lecture …

Dive deeper into the SystemVerilog language features and understand more how RTL simulation works!