Lectures 3:
Introduction to SystemVerilog

UCSD ECE 111

Prof. Farinaz Koushanfar

Fall 2024

Some slides are courtesy of Prof. Lin Prof. Turakia, or Prof. Karna ECE111 courses

Weekly Announcement(s)

Week 1 (oct 3 2024)

Important announcements (Details in Canvas)

- **TA office hours:** are now MWF 9-11am for Fall'24
  - Zoom Meeting ID: 948 6397 0932; Passcode 004453
  - https://ucsd.zoom.us/j/94863970932?pwd=iXcQXbLYjOaAQTdOJlrmglCYMSyeir.1
- **TA Discussion session:** Wed 4-4:50PM (in person Oct 9) Location: CNTRE 214
  - **Oct 2:** The TAs held a Zoom discussion section yesterday and the recording will be available
- **September 30:** The #FinAid survey has been added to the Quizzes section on Canvas for the commencement of academic activity reporting.
  - Due date this coming Friday, **10/4/24**.
- **Oct 1:** Homework 1 was posted on Canvas
  - Due in 1 week from now on Tuesday, **10/8/24**

Homework 1 overview

- This assignment will help you familiarize with digital design at three different abstraction levels
  - Behavioral, dataflow and gate
  - You will also learn to synthesize models and review resource
- Learn how to find usage summary for different models
  - Such as the number of Adaptive Look Up Tables (ALUTs) and inputs that are used to implement Boolean functions).
- Learn to simulate all the different models separately in Modelsim-Altera using the testbenches we have provided
- Learn to review the simulation results

More on FPGA and ASICs

FPGA Synthesis

SystemVerilog .sv source files

```
1   // Full Adder SystemVerilog Module
2   module fulladder(input logic a, b, cin,
3                    output logic s, cout);
4     logic p, g;    // internal nodes
5
6     assign p = a ^ b;
7     assign g = a & b;
8
9     assign s = p ^ cin;
10    assign cout = g | (p & cin);
11  endmodule
12
13
```

FPGA Synthesis 3-Step Flow

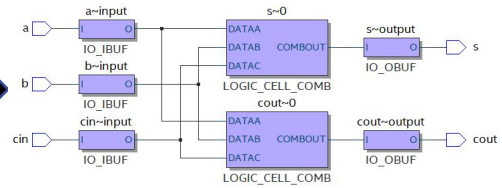
Step 1 : Design check and resource association
- Check for syntax errors in design source files
- Check if design provided can be synthesized
- Associate design to logic cell and blocks

Step 2 : Optimization
- Reduce logic
- Eliminate redundant logic
- Make design smaller and faster (best Fmax)

Step 3 : Technology Mapping
- Connect design to logic
- Predict and add timing estimates
- Create output reports and netlists

Credit : FPGA 101, G.R. Smith

Synthesized output netlist



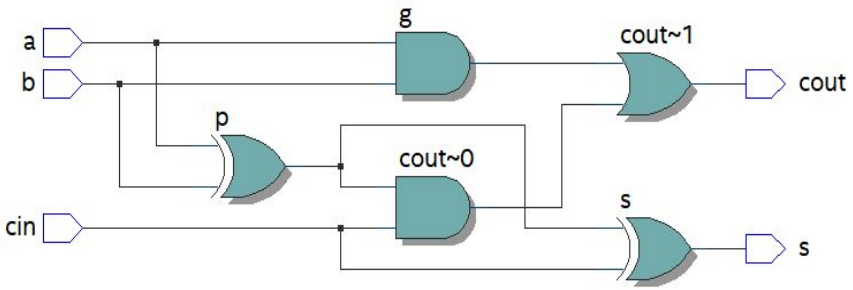
FPGA Netlist Viewer

- FPGA Tools provide netlist viewer post synthesis and implementation :
  - RTL netlist viewer
  - Post-fitting netlist viewer
  - Post-mapping netlist viewer

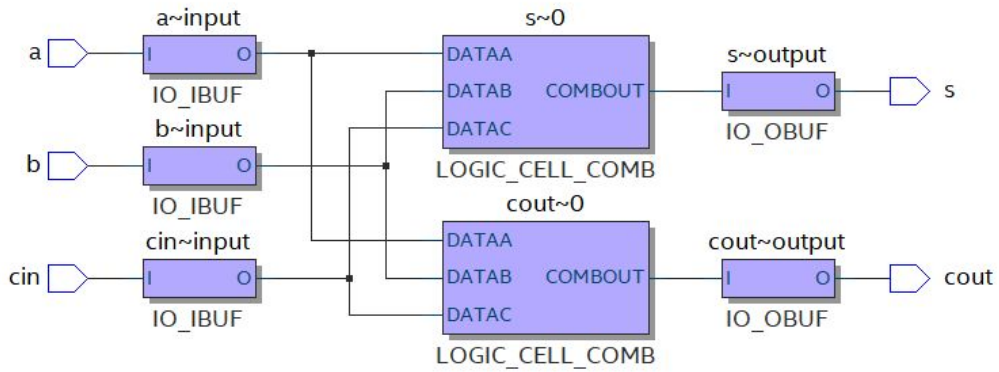
SystemVerilog .sv source files

```
1   // Full Adder SystemVerilog Module
2   module fulladder(input logic a, b, cin,
3                    output logic s, cout);
4     logic p, g;    // internal nodes
5
6     assign p = a ^ b;
7     assign g = a & b;
8
9     assign s = p ^ cin;
10    assign cout = g | (p & cin);
11  endmodule
12
13
```

RTL Netlist Viewer



Post-Fitting Netlist Viewer



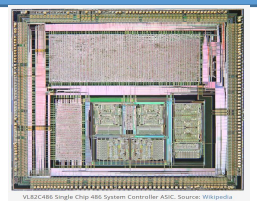
FPGA and ASIC Comparison

FPGA

- ✔ Low time-to-market
- ✔ Flexible (reprogrammable logic)
- ✔ Low NRE (non-recurring engineering) cost
- ✔ Available off-the-shelf

- x Low performance
- x Higher power consumption
- x Cannot implement analog blocks

ASIC

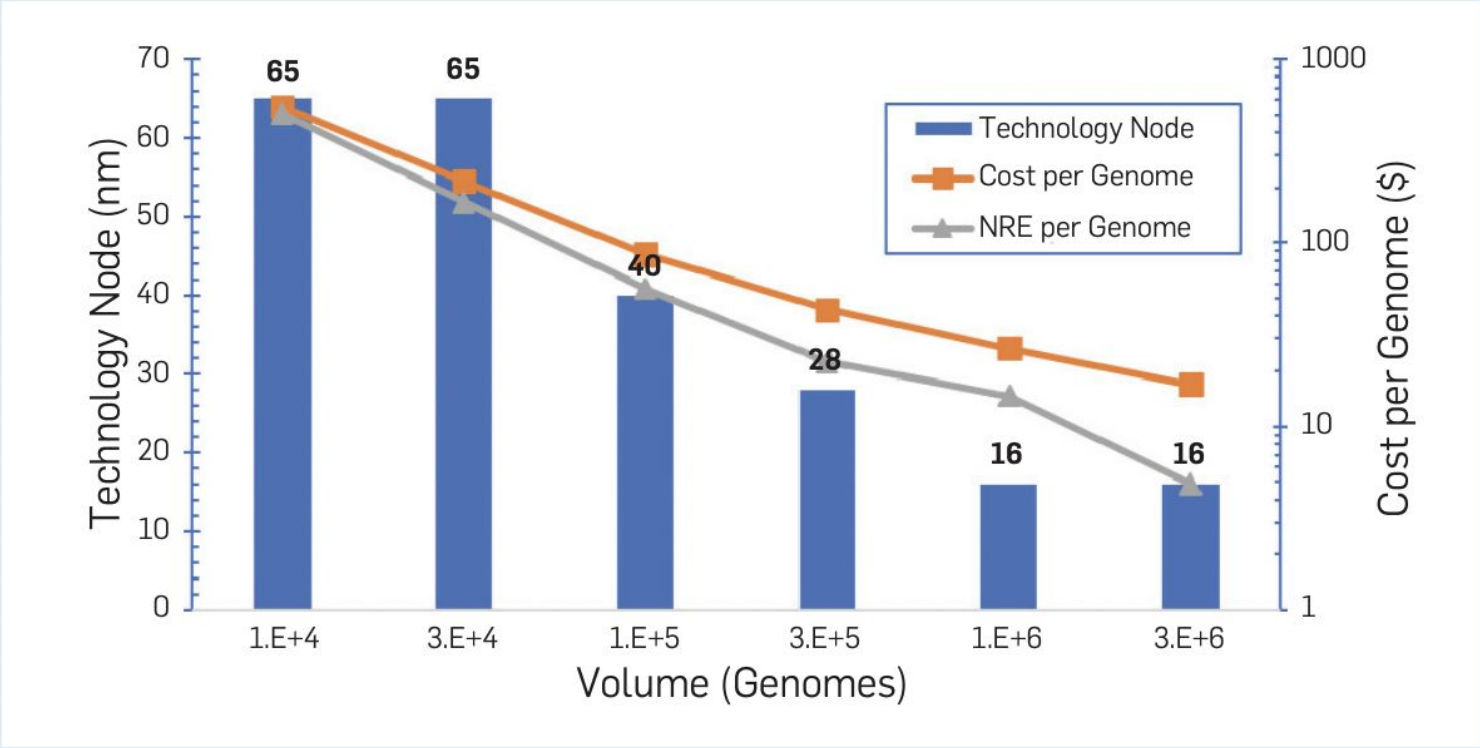


- x High time-to-market
- x Low flexibility (fixed logic)
- x High NRE (non-recurring engineering) cost
- x Need fabrication

- ✔ High performance
- ✔ Low power consumption
- ✔ Can implement analog blocks

Total cost of ownership with ASIC technology node

Figure 1. TCO for a genomics accelerator as a function of volume. At low volumes, older technology nodes give a lower TCO because of their lower nonrecurring costs (NRE).



(Dally, Turakhia & Han, CACM 2020)

Simulation of Digital Designs

What is simulation?

- Simulation aims at verifying the functionality and timing of a design against its original specifications
  - Example : Does 32-bit ALU logic implemented using SystemVerilog perform operations such as addition, subtraction, multiplication, comparison etc. correctly as per the specification?

- Digital Designer engineer performs functional simulation using a simulator at various stages of design development flow:
  - Prior to logic synthesis, simulation is performed to verify functionality of the HDL (SystemVerilog) description of the design
  - After synthesis, gate-level simulation is performed on the gate-netlist design generated by synthesis to verify design timing and re-check functionality of design

What is a simulator?

- Simulator is a **software** that mimics or **imitates** the operation of a digital circuit specified by the HDL code
- Simulators help the designer verify the **correct operation** of the implemented circuit
- Specifically, simulators can generate **waveforms** that model the behavior of implemented design

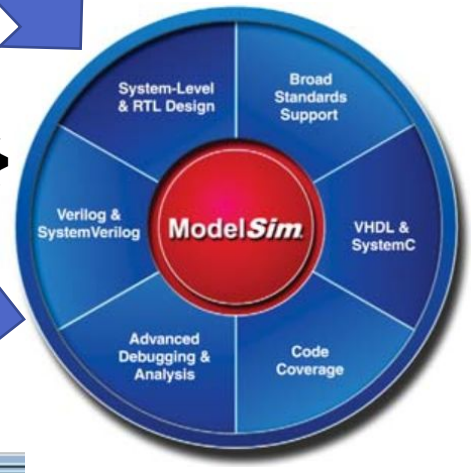
What is the waveform?

- Waveform is a plot of a signal over period of time
- Waveform is generated by the simulator at the end of RTL or gate-level simulation
- Designer and Verification engineers review waveforms to analyze the behavior of a design and also to perform debugging of potential bug in the HDL code

SystemVerilog (SV) RTL Code &

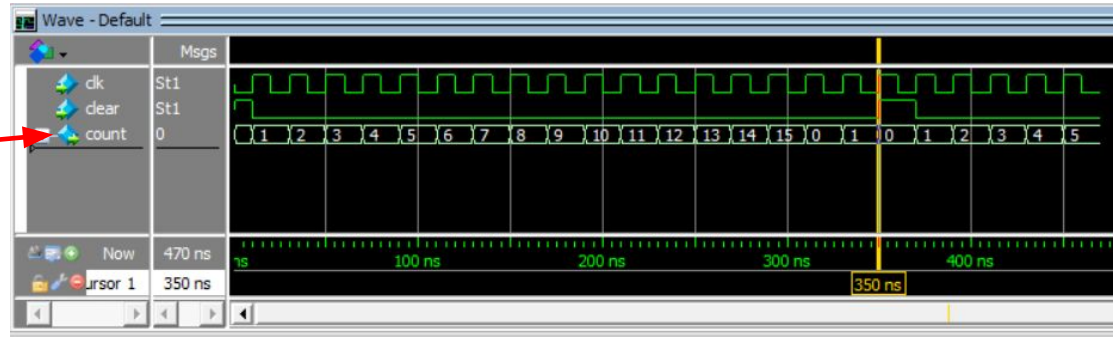
```
// SystemVerilog code for 4-bit counter
module counter (  input clk, input clear, output [3:0] count);
  always @ (posedge clk) begin
    if (clear==1)
      count <= 0;
    else
      count <= count + 1;
  end
endmodule
```

ModelSim Simulator



Waveform generated by Modelsim Simulator at end of simulation

Lets review waveform to see if SV design working like a 4-bit counter



RTL = register transfer level (next lecture)

Simulation and logic synthesis for FPGAs

Modelsim Simulator

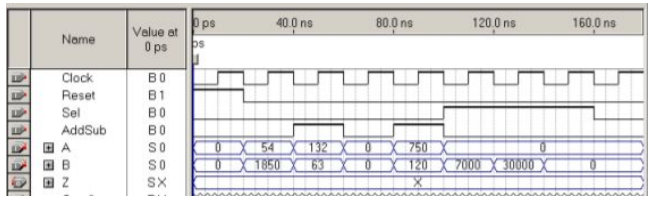
Launch compilation and simulation commands

Testbench (SystemVerilog Code)

Simulator

DUT (SystemVerilog RTL Code)

Waveform trace file



Both Simulator and Logic Synthesis uses the same SystemVerilog (SV) Code

- Simulator is used to verify intended behavior of SV code
- Synthesis is used to generate hardware from SV code

```
1   // Full Adder SystemVerilog Module
2   module fulladder(input logic a, b, cin,
3                    output logic s, cout);
4     logic p, g;    // internal nodes
5
6     assign p = a ^ b;
7     assign g = a & b;
8
9     assign s = p ^ cin;
10    assign cout = g | (p & cin);
11  endmodule
12
13
```

SystemVerilog .sv RTL source code

Altera Quartus Prime

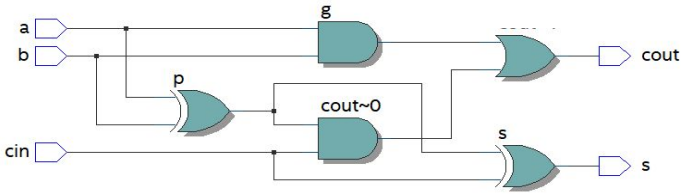
Launch Synthesis commands

Constraints (frequency, area, power)

Logic Synthesis

DUT (SystemVerilog RTL Code)

Gate Netlist



Additional learning resources on FPGA architecture

- Modern FPGA Architecture (Intel Altera)
  - https://youtu.be/EVy4KEj9kZg?si=A6RLPiGacFoB54gA

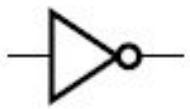
- Altera FPGA Architecture :
  - https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01003.pdf

Back to digital design basics

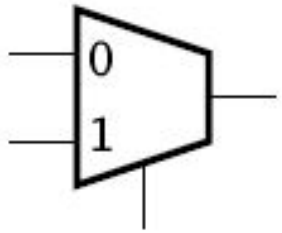
Basic digital circuit design components

AND gate

OR gate



inverter

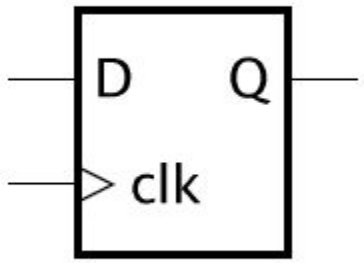


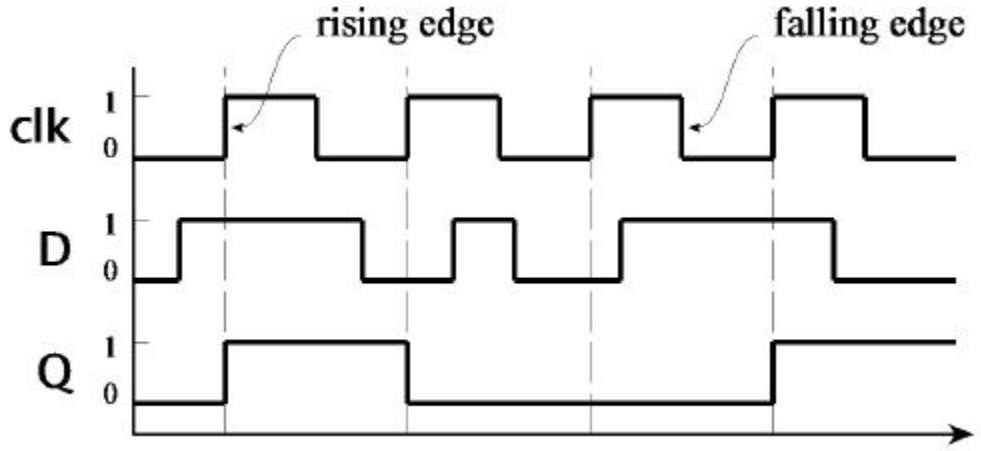
multiplexer

Sequential circuits

- Circuit whose output values depend on current *and previous* input values
  - Include some form of storage of values
- Nearly all digital systems are sequential
  - Mixture of gates and storage components
  - Combinational parts transform inputs and stored values

Flip flops and clocks





- Edge-triggered D-flipflop
  - stores one bit of information at a time
- Timing diagram
  - Graph of signal values versus time

How to represent hardware?

- If you're going to design a computer, you need to write down the design so that:
  - You can read it again later
  - Someone else can read and understand it
  - It can be simulated and verified
  - Even software people may read it!
  - It can be synthesized into specific gates
  - It can be built and shipped and make money

Ways to represent hardware

- Draw schematics
  - Hand-drawn
  - Machine-drawn
- Write a netlist
    - Z52BH I1234 (N123, N234, N4567);
- Write primitive Boolean equations
    - AAA = abc DEF + ABC def
- Use a Hardware Description Language (HDL)
    - assign overflow = c31 ^ c32;

Slide courtesy of Prof. Milo Martin, Upenn

HDLs

- Hardware description language (HDL):
  - allows designer to specify logic function only. Then a computer-aided design (CAD) tool produces or synthesizes the optimized gates.
- Most commercial designs built using HDLs
- Two leading HDLs:

– Verilog
  - developed in 1984 by Gateway Design Automation
  - became an IEEE standard (1364) in 1995

– VHDL
  - Developed in 1981 by the Department of Defense
  - Became an IEEE standard (1076) in 1987

HDLs

- Textual representation of a digital logic design
  - Can represent gates, like a netlist, or abstract logic
- HDLs are not “programming languages”
  - No, really. Even if they look like it, they are not.
  - For many people, a difficult conceptual leap
- Similar development chain
  - Compiler: source code assembly code binary machine code
  - Synthesis tool: HDL source gate-level specification hardware

Pros and cons

Why using an HDL?

- Easy to write and edit
- Compact
- Don’t have to follow a maze of lines
- Easy to analyze with various tools

Why not to use an HDL?

- You still need to visualize the flow of logic
- A schematic can be a work of art
  - But often isn’t!

Synthesis vs. Simulation

- Extremely important to understand that SystemVerilog is BOTH a “Synthesis” language and a “Simulation” language
  - Small subset of the language is “synthesizable”, meaning that it can be translated to logic gates and flip-flops
  - SystemVerilog also includes many features for “simulation” or “verification”, features that have no meaning in hardware!
- Although SystemVerilog syntactically looks like “C”, it is a Hardware Description Language (HDL), NOT a software programming language
  - Every line of synthesizable SystemVerilog MUST have a direct translation into hardware (logic gates and flip flops)
  - Very important to think of the hardware each line of SystemVerilog produce
- Discussed in detail later

SystemVerilog

Hello World!

Simple logic gate in SystemVerilog

- Module

```
module not_gate(in, out); // module name+ports
    // comments: declaring port type
   input in;
   output out;

    // Defining circuit functionality
   assign out = ~in;

endmodule
```

Synthesis

SystemVerilog:

```
module example(input  logic a, b, c,
               output logic y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b &  c;
endmodule
```

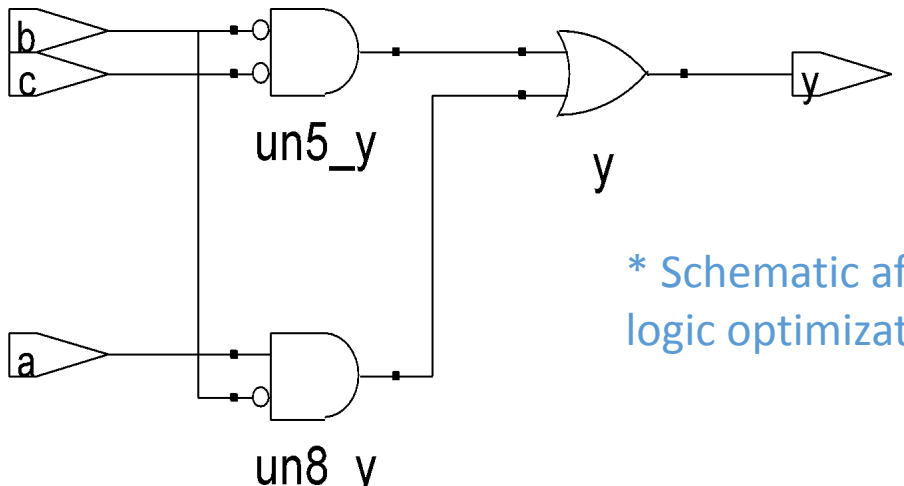
Module Abstraction:

Synthesis

SystemVerilog:

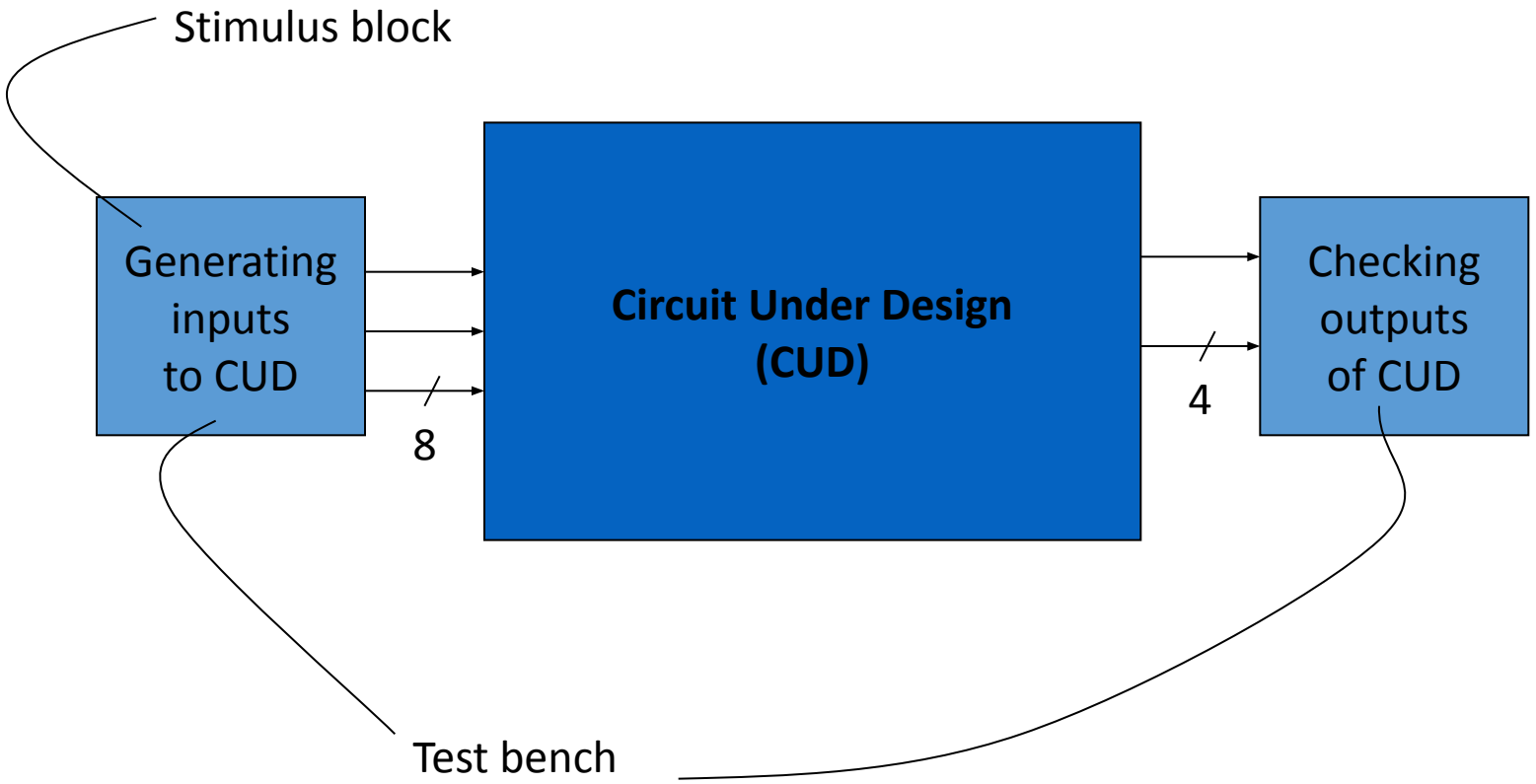
```
module example(input  logic a, b, c,
               output logic y);
  assign y = ~a & ~b & ~c | a & ~b & ~c | a & ~b &  c;
endmodule
```

Synthesis: translates into a netlist (i.e., a list of gates and flip-flops, and their wiring connections)



* Schematic after some logic optimization

Basics of digital design with HDL



Useless SystemVerilog Example

```
module useless;

    initial
        $display("Hello World!");

endmodule
```

- Note the message-display statement
  - Compare to printf() in C

SystemVerilog Syntax

- Case sensitive
  - **Example:** `reset` and `Reset` are not the same signal.
- No names that start with numbers
  - **Example:** `2mux` is an invalid name
- Whitespace ignored
- Comments:
  - `// single line comment`
  - `/* multiline`
    `comment */`