Lecture 1: Introduction

UCSD ECE 111

Prof. Bill Lin

Winter 2019

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Electrical and Computer Engineering

Course Information

- Professor Bill Lin
  - Office hours: Mon 1:00-1:50p, 4310 Atkinson Hall

- Lectures:
  - Section A00: MW 2:00-3:20p, EBU1-2315
  - Section B00: MW 3:30p-4:50p, EBU1-2315

- Teaching Assistants:
  - Jianling Liu Justin Law, Dylan Vizcarra, Yu Huang and Ping Yin
  - Office hours: TBD
  - Note: You may get help from any TA during their office hours.

Course Information

- Course webpage

  http://cwcserv.ucsd.edu/~billlin/classes/ECE111/index.php

- Also linked from ECE courses page:

  http://www.ece.ucsd.edu/courses

- All announcements will be done through Piazza. Sign up here:

  http://piazza.com/ucsd/winter2019/ece111

Course Information

- Course webpage

  http://cwcserv.ucsd.edu/~billlin/classes/ECE111/index.php

- Also linked from ECE courses page:

  http://www.ece.ucsd.edu/courses

- Professor Bill Lin
  - Office hours: Mon 1:00-1:50p, 4310 Atkinson Hall

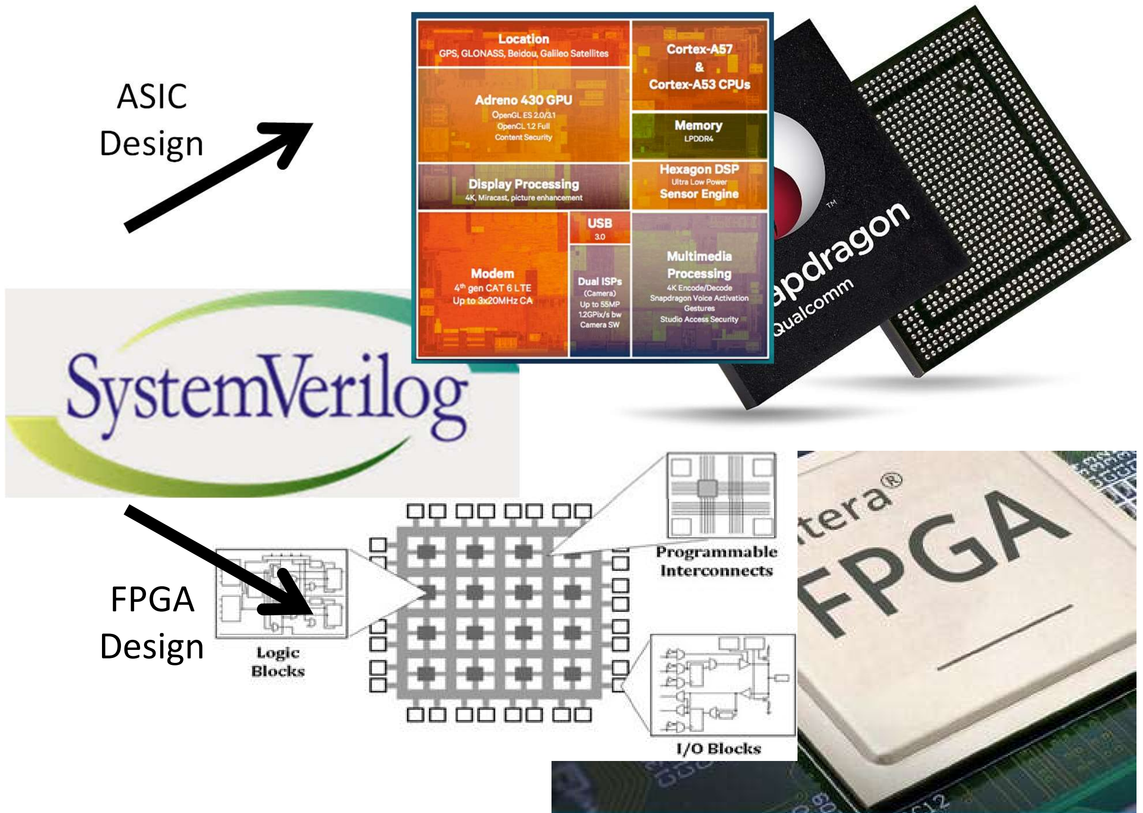
- Lectures:
  - Section A00: MW 2:00-3:20p, EBU1-2315
  - Section B00: MW 3:30p-4:50p, EBU1-2315

Introduction

- Goal: Learn Verilog-based chip design
- In particular, we will be using the Hardware Description Language (HDL) SystemVerilog, which is a “superset” of Verilog:
  - Verilog, IEEE standard (1364) in 1995
  - SystemVerilog, extended in 2005, current version is IEEE Standard 1800-2012
- The name “SystemVerilog” is confusing because it still describes hardware at the same level as “Verilog”, but SystemVerilog adds a number of enhancements and improved syntax.
- SystemVerilog files have a “.sv” extension so that the compiler knows that the file is in SystemVerilog rather than Verilog.

Why Learn Verilog/SystemVerilog

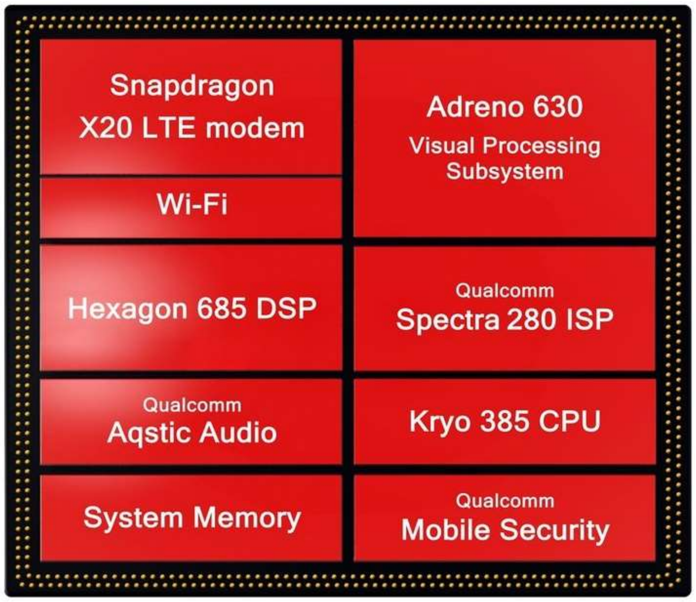
- Most EE jobs are Verilog/SystemVerilog based chip designs



Why Learn Verilog/SystemVerilog

- Example modern Systems-on-Chip (SoC)

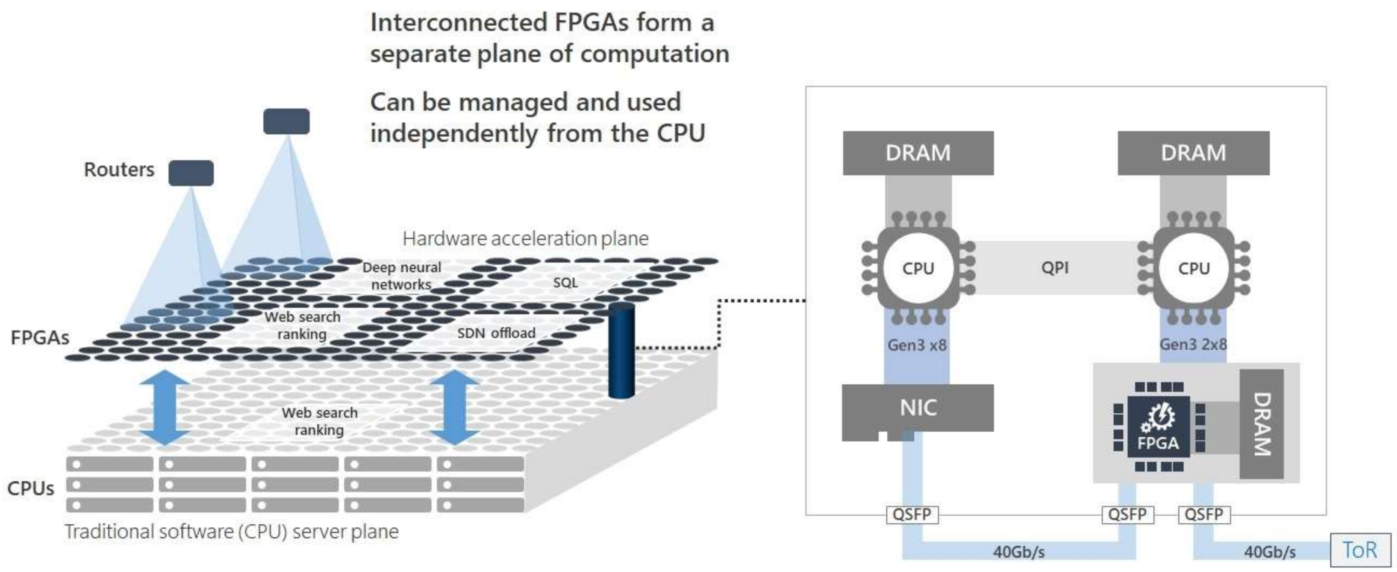
Qualcomm Snapdragon 845 Mobile Processor



Why Learn Verilog/SystemVerilog

- Emergence of the FPGA Cloud

Example: Microsoft's Catapult Project deployed worldwide

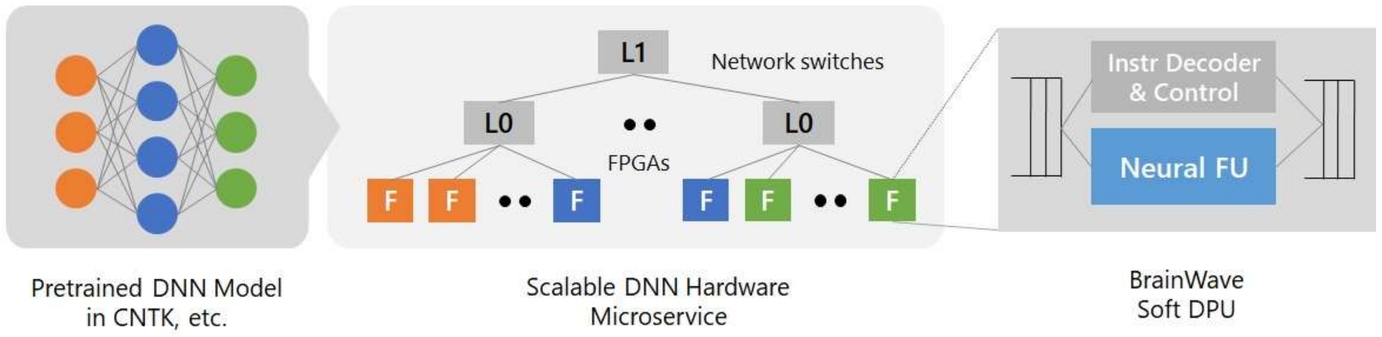

Interconnected FPGAs form a separate plane of computation
Can be managed and used independently from the CPU
Routers
Hardware acceleration plane
Deep neural networks
SQL
Web search ranking
SDN offload
FPGAs
Web search ranking
CPUs
Traditional software (CPU) server plane
DRAM
DRAM
CPU
QPI
CPU
Gen3 x8
Gen3 2x8
NIC
FPGA
DRAM
QSFP
QSFP
QSFP
40Gb/s
40Gb/s
ToR

[Credit: Microsoft, MICRO'16]

Why Learn Verilog/SystemVerilog

- Emergence of the FPGA Cloud

Example: Microsoft’s Project BrainWave



Each FPGA implements many Soft DPUs

[Credit: Microsoft, Hot Chips’17]

Other FPGA Clouds

amazon web services

Alibaba.com

Baidu

FPGA Cloud Applications

- Bing search engine implemented in Microsoft's FPGA cloud
- Machine learning/AI
- High-speed frequency trading
- Bioinformatics (e.g. DNA sequencing)

Class Project

- Final project on Bitcoin mining
- Great deal of interest in cryptocurrencies

Class Project

- Blockchain is the underlying technology for cryptocurrencies, which provides authenticated global ledger (tamper-proof global transaction record)

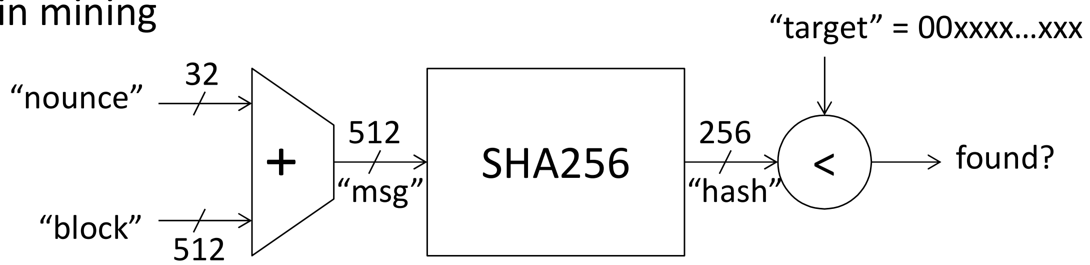
- Blockchain is finding many applications: e.g.,

energy web FOUNDATION

IBM Blockchain

Class Project

- Bitcoin mining



- Every "msg" will produce different 256-bit hash. Changing "nounce" will change "msg" and produce different 256-bit hash.

- Find "nounce" such that SHA256(nounce + block) < "target"

- If "target" has 1 leading 0, then chances of success every 2 tries. If 2 leading 0's, every 4 tries, 30 leading 0's, every billion tries, etc.

- Bitcoin by design makes "target" increasingly difficult after certain number of bitcoins have been mined.

Class Project

- Final project based on how fast can your design evaluate “nonces” (equivalent to how fast you can mine a Bitcoin). i.e., final project grade based on performance only.
- You can use the entire FPGA to create as many instancs of SHA256 as you like, and you can greatly improve the performance of each SHA256 unit using techniques like pipelining, etc.
- Intermediate project: Design of a SHA256 unit.
- Projects done in teams of 2 (you have the option of working alone). Your partner can be in the other section.

Software

- See Software Downloads Page

http://cwcserv.ucsd.edu/~billlin/classes/ECE111/software.php
which links to this:
http://fpgasoftware.intel.com/18.1/?edition=lite

- Quartus Prime Lite Edition
  - Quartus Prime (earlier versions were called Quartus II)
  - ModelSim-Intel FPGA Edition
- Arria II device support
- Available for Windows and Linux
- For Macs, you can use Bootcamp to dual-boot Windows
- Windows Machines with software setup also available in **EBU1-4309**. You should be able to get the door code from here:

https://sdacs.ucsd.edu/~icc/index.php

Software

- Class website has a tutorial page on Quartus and ModelSim

http://cwcserv.ucsd.edu/~billlin/classes/ECE111/Quartus_ModelSim_Tutorial/quartus_modelsim_tutorial.html

More Information

- Recommended textbook
  - Digital Design and Computer Architecture, Second Edition, by David Harris and Sarah Harris
  - We will only be using Chapter 4 of this book, which provides a good overview of SystemVerilog with good examples.
  - Make sure you get the 2nd Edition since the 1st Edition uses Verilog instead of SystemVerilog
  - Book recommended, but not required.

Honor Code

- The UCSD Student Conduct Code

  https://students.ucsd.edu/sponsor/student-conduct/regulations/22.00.html

- Violations will be reported to the Student Conduct Office (as well as failing the class)

Let’s Get Started with SystemVerilog

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!

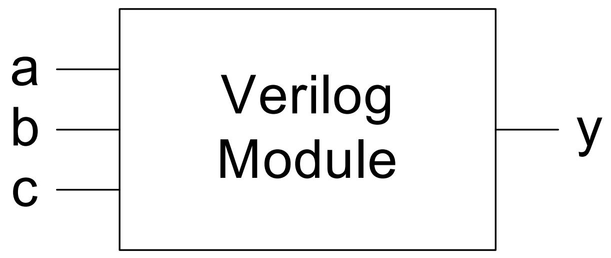
- Although SystemVerilog syntactically looks like “C”, it is a Hardware Description Language (HDL), NOT a software programming language
  - Every line of synthesizable SystemVerilog MUST have a direct translation into hardware (logic gates and flip flops).
  - Very important to think of the hardware that each line of SystemVerilog will produce.

SystemVerilog Modules

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:

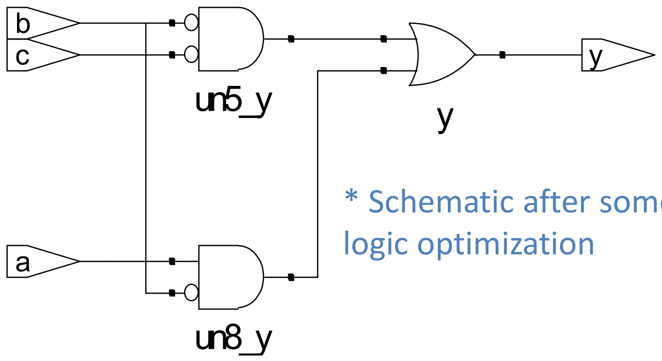


HDL Synthesis

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

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`

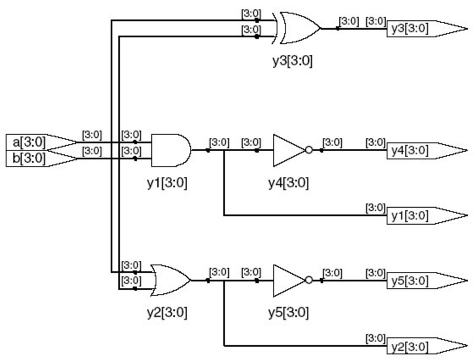
  -
    ```
    /* multiline
         comment */
    ```

Structural Modeling - Hierarchy

```
module and3(input  logic a, b, c,
            output logic y);
  assign y = a & b & c;
endmodule


module inv(input  logic a,
           output logic y);
  assign y = ~a;
endmodule


module nand3(input  logic a, b, c
             output logic y);
  logic n1;                   // internal signal

  and3 andgate(a, b, c, n1);  // instance of and3
  inv  inverter(n1, y);       // instance of inverter
endmodule
```

Bitwise Operators

```
module gates(input  logic [3:0]  a, b,
             output logic [3:0] y1, y2, y3, y4, y5);
   /* Five different two-input logic
      gates acting on 4 bit busses */
   assign y1 = a & b;     // AND
   assign y2 = a | b;     // OR
   assign y3 = a ^ b;     // XOR
   assign y4 = ~(a & b); // NAND
   assign y5 = ~(a | b); // NOR
endmodule
```

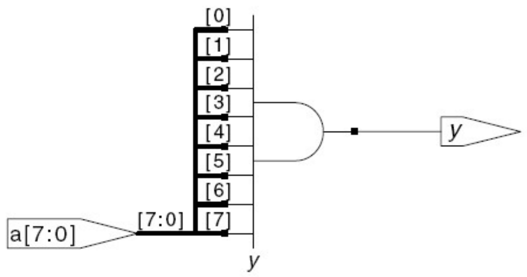


// single line comment

/*…*/ multiline comment

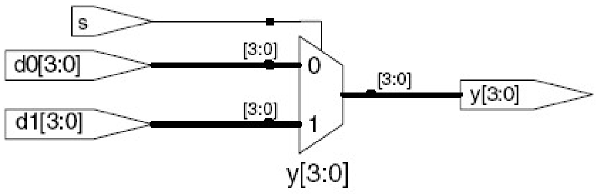
Reduction Operators

```
module and8(input  logic [7:0] a,
            output logic       y);
   assign y = &a;
   // &a is much easier to write than
   // assign y = a[7] & a[6] & a[5] & a[4] &
   //            a[3] & a[2] & a[1] & a[0];
endmodule
```



Conditional Assignment

```
module mux2(input  logic [3:0] d0, d1,
            input  logic       s,
            output logic [3:0] y);
   assign y = s ? d1 : d0;
endmodule
```



`? :` is also called a *ternary operator* because it operates on 3 inputs: `s`, `d1`, and `d0`.

Precedence

Order of operations

Highest	~	NOT
	*, /, %	mult, div, mod
	+, -	add,sub
	<<, >>	shift
	<<<, >>>	arithmetic shift
	<, <=, >, >=	comparison
	==, !=	equal, not equal
	&, ~&	AND, NAND
	^, ~^	XOR, XNOR
	|, ~|	OR, NOR
Lowest	?:	ternary operator