EE-421: Digital System Design

Introduction to Verilog

Instructor: Dr. Rehan Ahmed [rehan.ahmed@seecs.edu.pk]

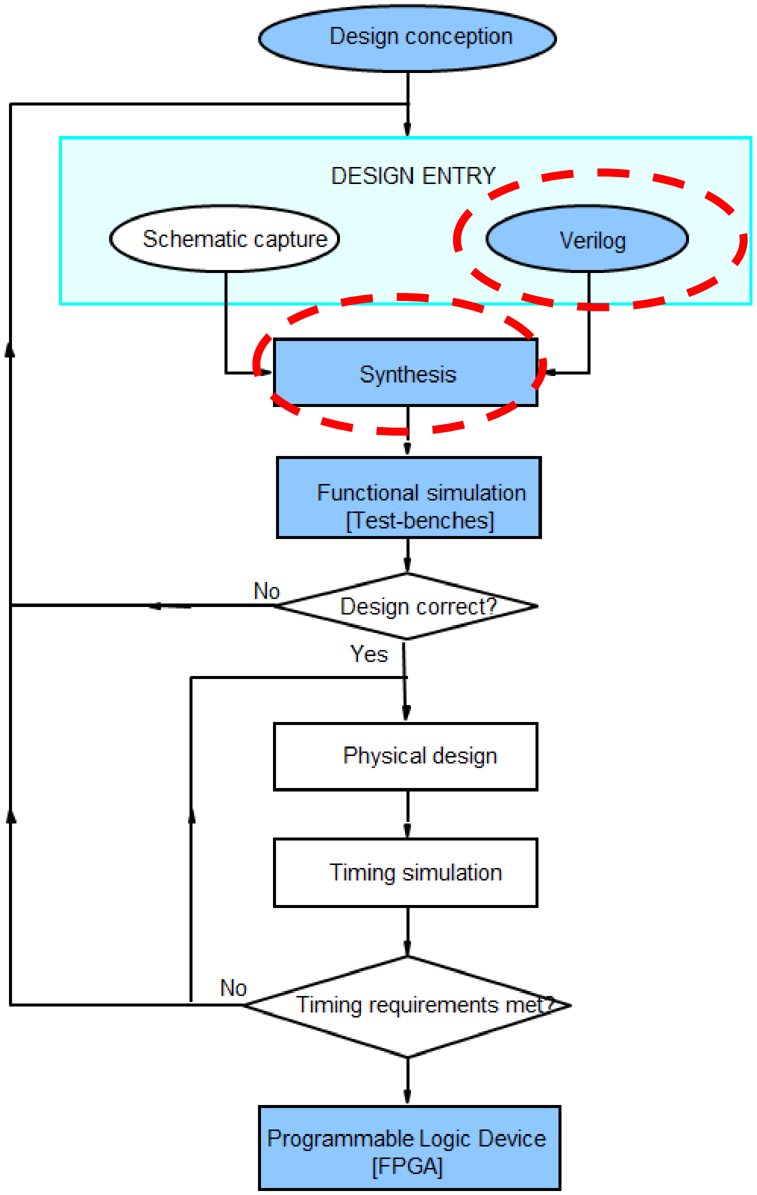
NUST

SCHOOL OF ELECTRICAL ENGINEERING &
COMPUTER SCIENCE (SEECS)

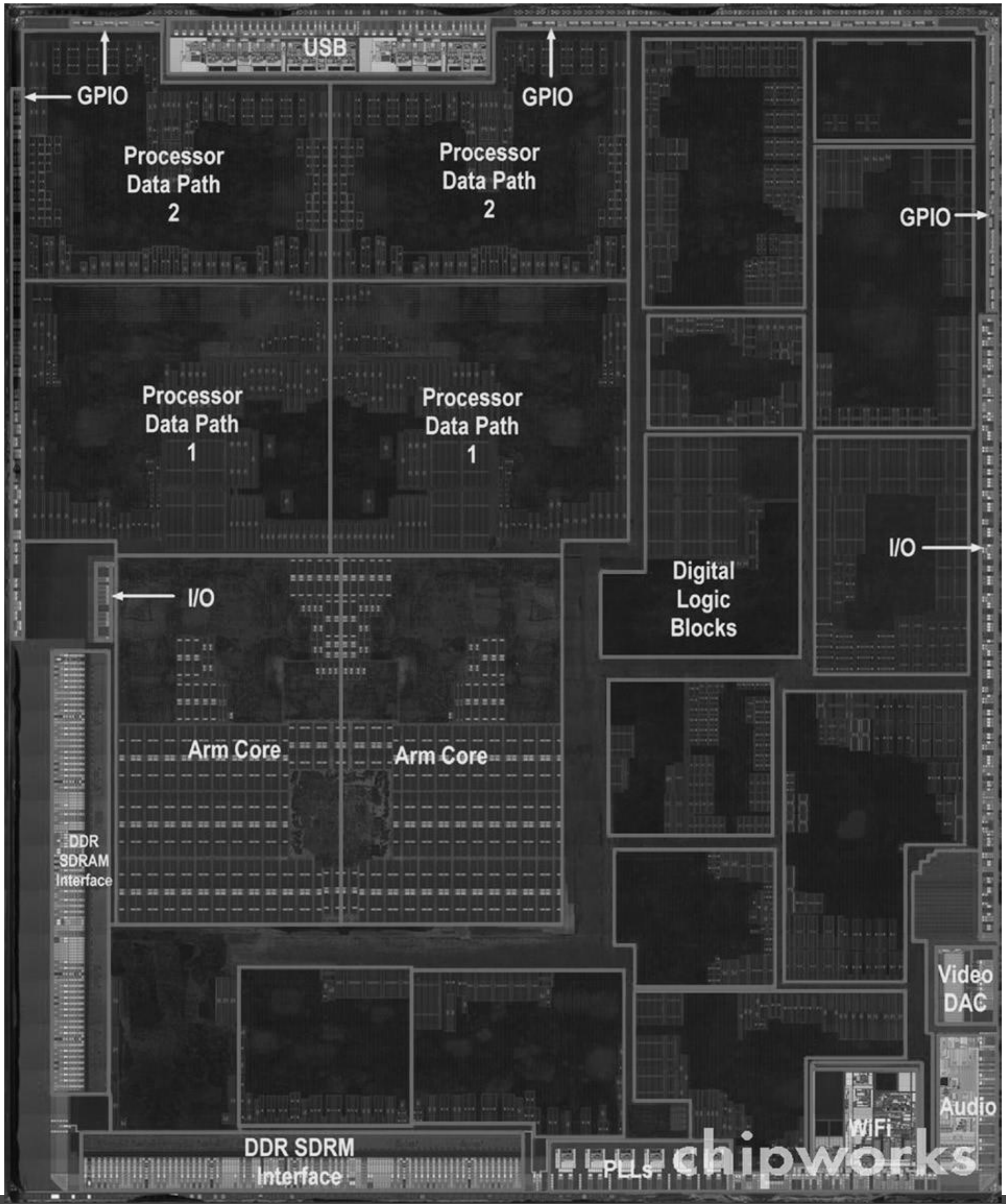
Acknowledgements

- The material used in this slide-set contain material/illustrations from Prof. Milo Martin, Andy Phelps, Prof. Stephen brown, Prof. Steve Wilton and Prof. Onur Mutlu.

Where are we Heading?



Inside your iPhone



How would you design this?

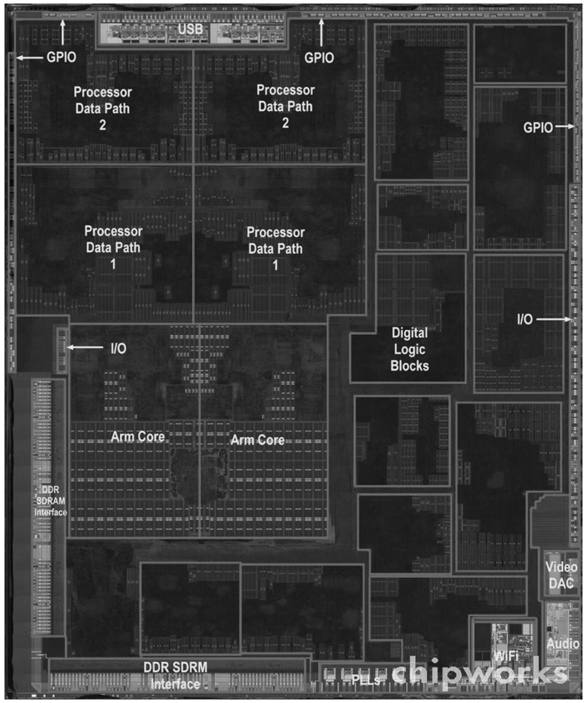
How would you deal with this complexity?

How to Deal with This Complexity?

- A fact of life in computer engineering
  - Need to be able to specify complex designs
    - communicate with others in your design group
  - … and to simulate their behavior
    - *yes, it's what I want to build*
  - … and to synthesize (automatically design) portions of it
    - have an error-free path to implementation

How to Describe a Design?

- Two Ways:
  - Schematic – Draw a picture
  - Hardware Description Language – Standardized text-based description



- Why text is better?
  - More scalable
  - Easier to store
  - Easier to search
  - Easier for revision control
  - Easier to standardize
  - Can leverage automated tools (more on this later)

- All these benefits of HDLs have lead to higher engineer productivity
- Practically **ALL** digital design is now done with HDLs

Hardware Description Languages

- **Two well-known and well-used hardware description languages**

- **Verilog**
  - Developed in 1984 by Gateway Design Automation
  - Became an IEEE standard (1364) in 1995
  - More popular in US

- **VHDL (VHSIC Hardware Description Language)**
  - Developed in 1981 by the US Department of Defense
  - Became an IEEE standard (1076) in 1987
  - More popular in Europe

- In this course we will use Verilog

So, about Verilog

- Verilog is a (surprisingly) big language
  - Lots of features for synthesis and simulation of hardware
  - Can represent low-level features, e.g. individual transistors
  - Can act like a programming language, with “for” loops etc.
  - *Daunting task to learn all of its features*

- We’re going to learn a focused *subset* of Verilog
  - We will use it at a level (behavioral) appropriate for *digital design*
  - Focus on synthesizable constructs
  - Focus on avoiding subtle synthesis errors
  - Initially restrict some features just because they aren’t necessary

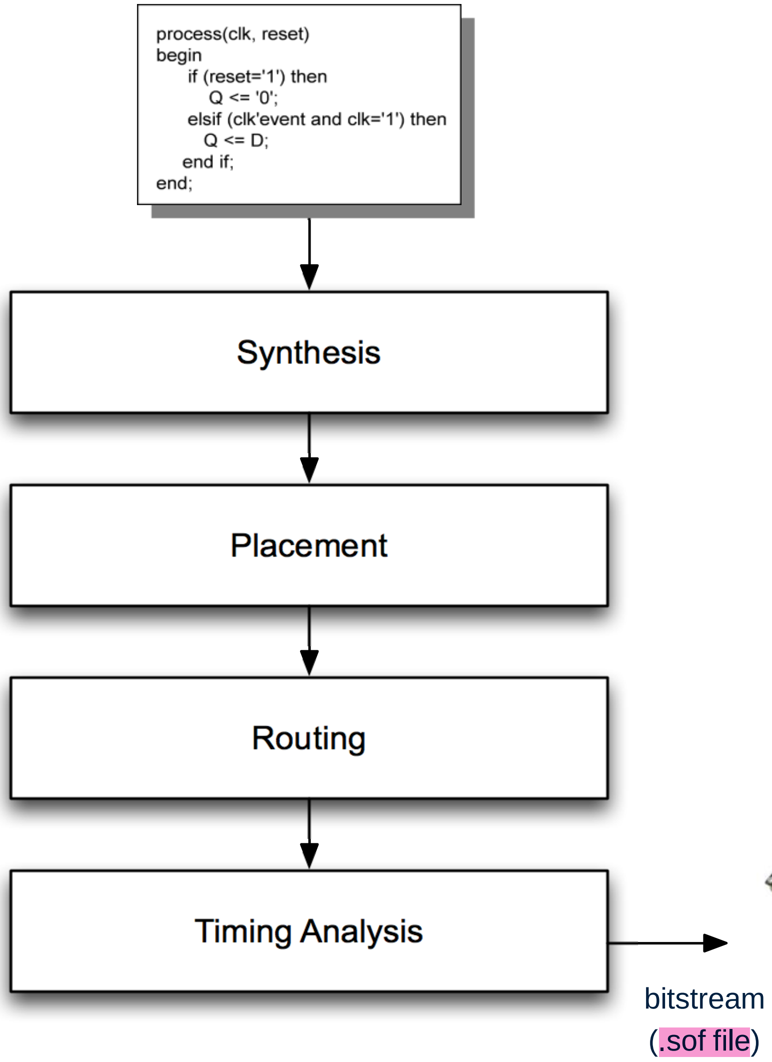
Before we start, Remember one lesson

- HDLs are **NOT** *“programming languages”*
  - No, really. Even if they look like it, they are not.
  - For many people, a difficult conceptual leap
- Hardware is not Software
  - Software is sequential
  - In a program, we start at the beginning (e.g. “main”), and we proceed sequentially through the code as directed
  - The program represents an algorithm, a step-by-step sequence of actions to solve some problem

```
for (i = 0; i<10; i=i+1) {
        if (newPattern == oldPattern[i]) match = i;
}
```

  - Hardware is all active at once; there is no starting point
- The magic is NOT in the language;
  - It’s in your express-i-bility!
- Verilog is case sensitive
  - K is not the same as k

FPGA CAD



```
process(clk, reset)
begin
    if (reset='1') then
       Q <= '0';
    elsif (clk'event and clk='1') then
       Q <= D;
   end if;
end;
```

What Happens with HDL Code?

- **Synthesis**
  - Modern tools are able to map ***synthesizable*** *HDL code* into low-level *cell libraries* → *netlist describing gates and wires*
  - They can perform many optimizations
  - … however they can **not** guarantee that a solution is optimal
    - Mainly due to computationally expensive placement and routing algorithms
  - Most common way of Digital Design these days

- **Simulation**
  - Allows the behavior of the circuit to be verified without actually manufacturing the circuit
  - Simulators can work on *structural* or *behavioral* HDL

Verilog Module

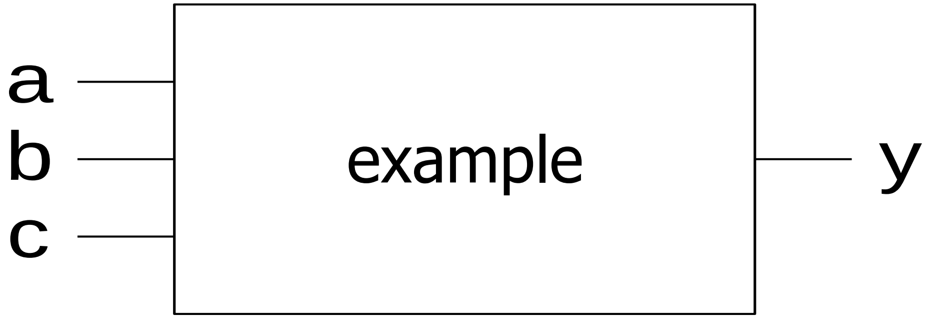
Defining a Module in Verilog

- A module is the main building block in Verilog

- We first need to define:
  - Name of the module
  - Directions of its ports (e.g., input, output)
  - Names of its ports
- Then:
  - Describe the functionality of the module

inputs
output

Implementing a Module in Verilog



name of module

Port list (inputs and outputs)

ports have a declared type

a module definition

```
module example (a, b, c, y);
        input a;
        input b;
        input c;
        output y;

// here comes the circuit description

endmodule
```

A Question of Style

- **The following two codes are functionally identical**

```
module test ( a, b, y );
       input a;
       input b;
       output y;

endmodule
```

```
module test ( input a,
              input b,
              output y );

endmodule
```

port name and direction declaration can be combined

What If We Have Multi-bit Input/Output?

- **You can also define multi-bit Input/Output (Bus)**
  - [range_end : range_start]
  - **Number of bits:** range_end – range_start + 1
- **Example**:

```
input  [31:0] a;   // a[31], a[30] .. a[0]
output [15:8] b1;  // b1[15], b1[14] .. b1[8]
output [7:0]  b2;  // b2[7], b2[6] .. b2[0]
input         c;   // single signal
```

- **a** represents a 32-bit value, so we prefer to define it as:
  `[31:0] a`
- It is preferred over `[0:31] a` which resembles *array* definition
- It is good practice to be consistent with the representation of multi-bit signals, i.e., always [31:0] or always [0:31]

Basic Syntax

- Verilog is case sensitive
  - `SomeName` and `somename` are not the same!
- Names cannot start with numbers:
  - `2good` is not a valid name
- Whitespaces are ignored

```
// Single line comments start with a //

/* Multiline comments
   are defined like this */
```