EE 421 / CS 425
Digital System Design
Laboratory 1

Fall 2024

Shahid Masud

LUMS
A Not-for-Profit University

Lab Course Learning Objectives

1. Describe, Simulate and Debug combinational and sequential digital systems using SystemVerilog hardware description language (Design Capture).
2. Understand Design Flow and EDA tools for Simulation and Synthesis of FPGA design of digital systems.
3. Implement and test digital systems on FPGA platforms (hardware boards).

Course Outline and Grading Scheme

- **Lab Completion (10 to 11 Labs): 50%**
  - Lab Attendance (1% each lab)
  - TA grades the Task Completion (2% each lab)
  - Submitting Reports and Observations, as required (2% each lab)
- **Lab Projects (1 to 2): 30%**
  - SystemVerilog coding and Testbench
  - Simulation
  - Synthesis
  - FPGA prototyping
  - Project report
  - Presentation
- **Lab (Midterm) Exam: 20%**

MS Grading: Final Grade is Combined (80% Theory, 20% Lab)

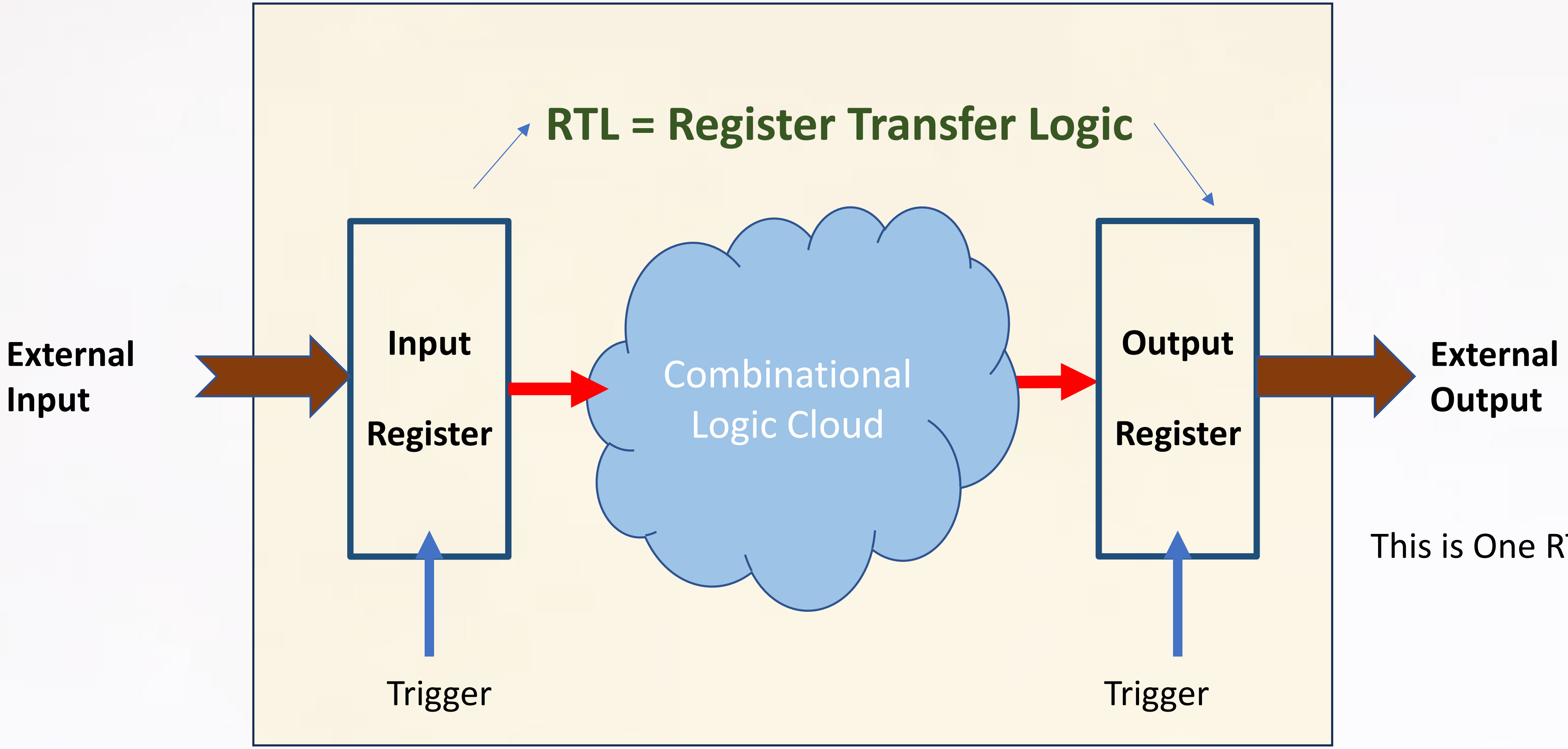
LUMS
A Not-for-Profit University

Today's Topics

- Learning Objectives
- Lab Course Outline and Grading
- RTL Design - Introduction
- Design Flow – from idea to product
- Step through EDA tools
- Introduction to SystemVerilog and Design Methodology
- Tools Installation

RTL View of Digital System Design

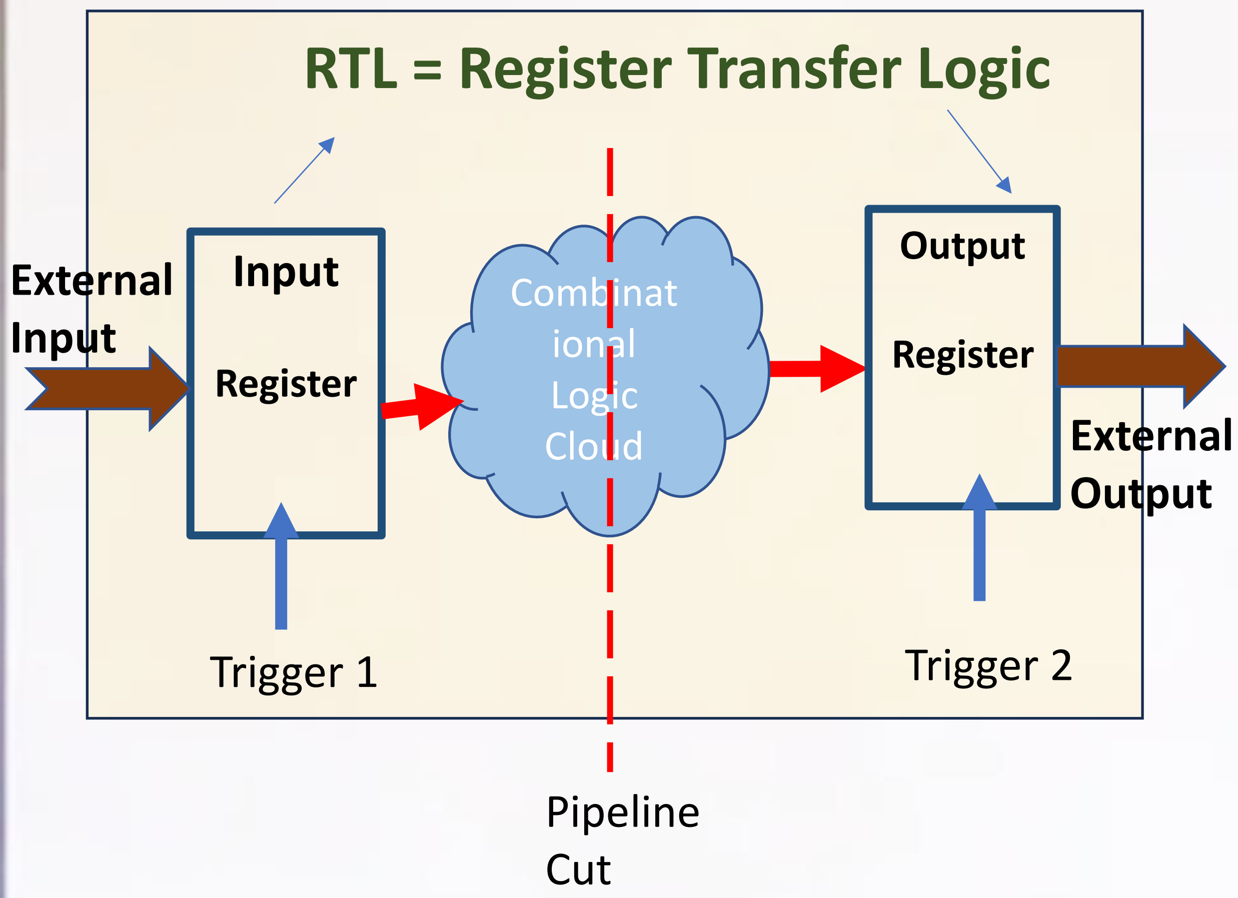
Engineering System has Inputs and Outputs

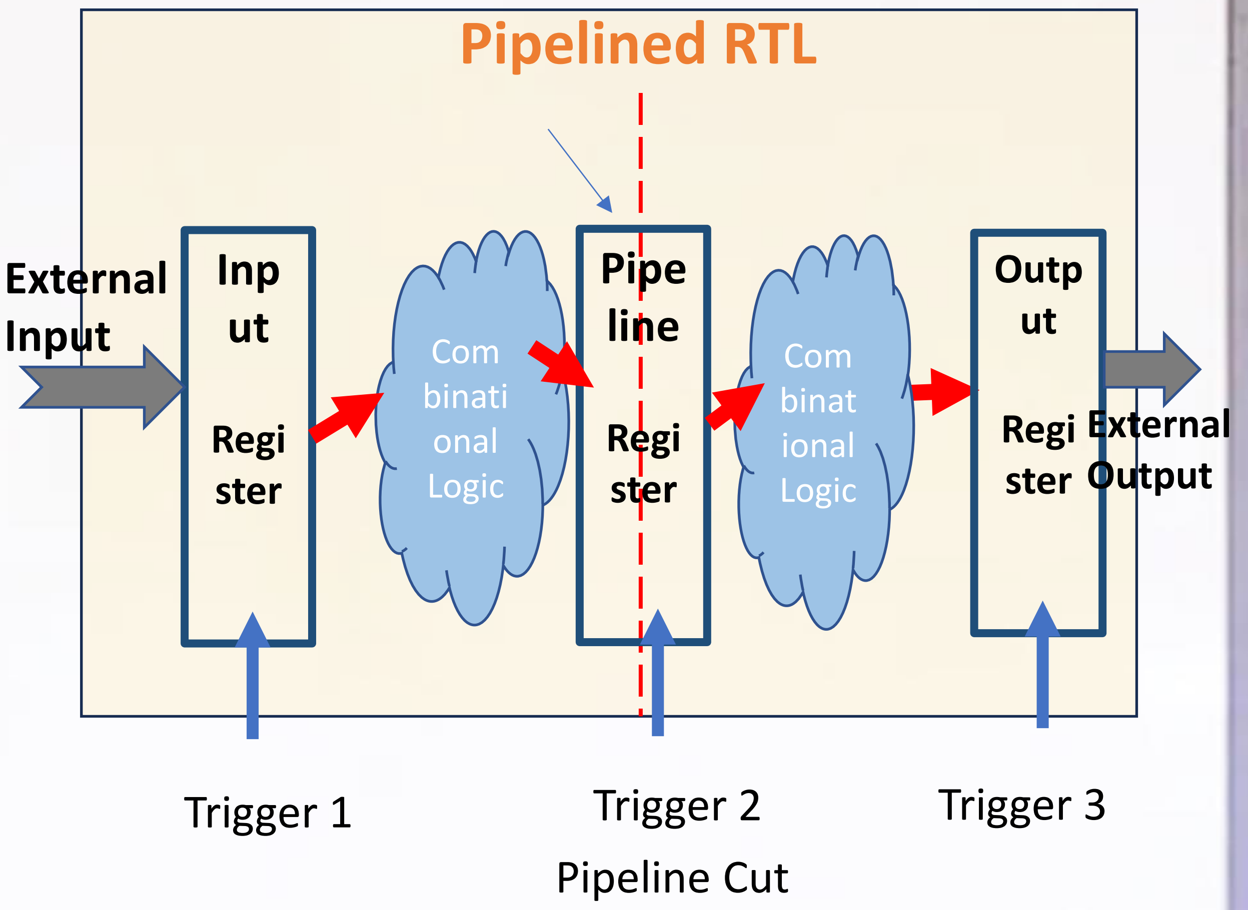


This is One RTL Module

A Complex Digital System has many RTL Modules

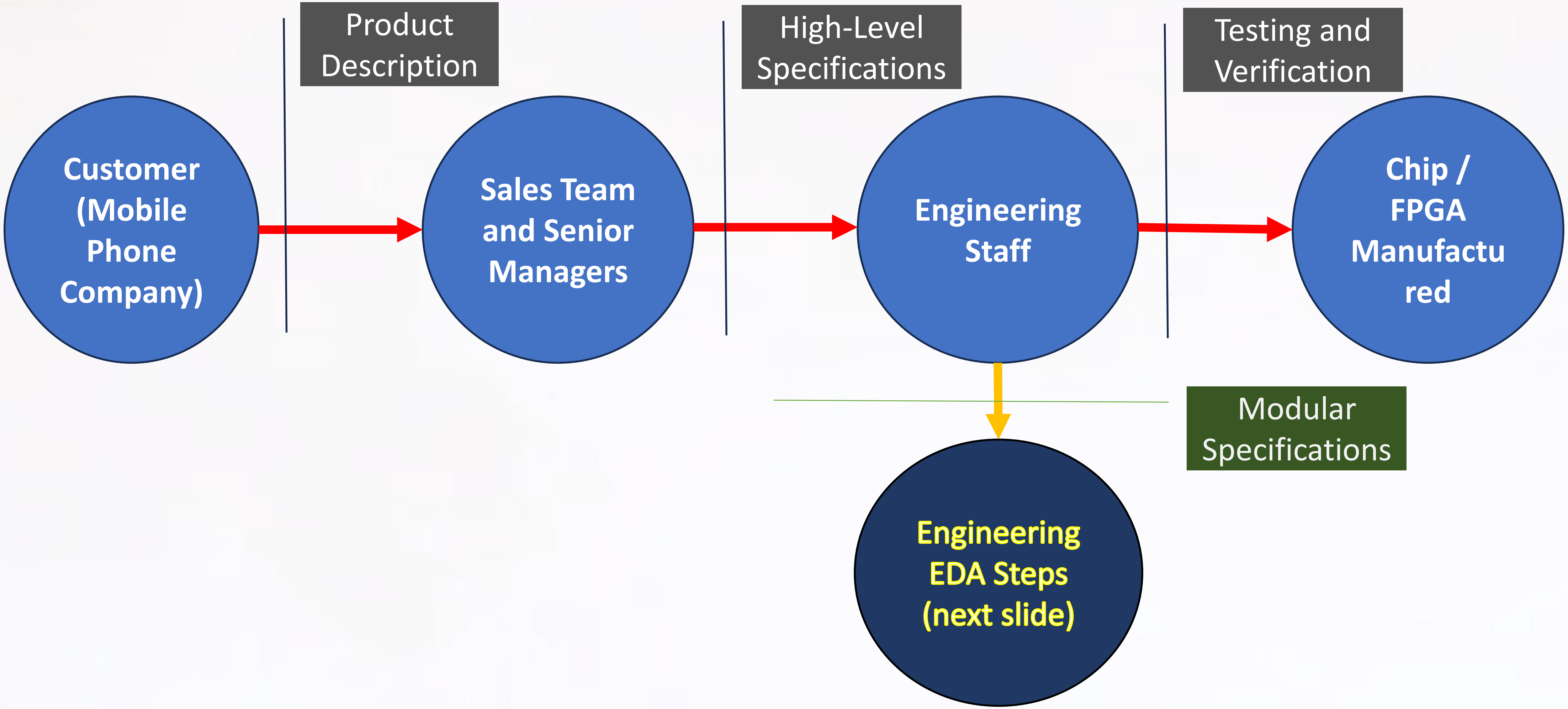
RTL Design with a Pipeline Cut



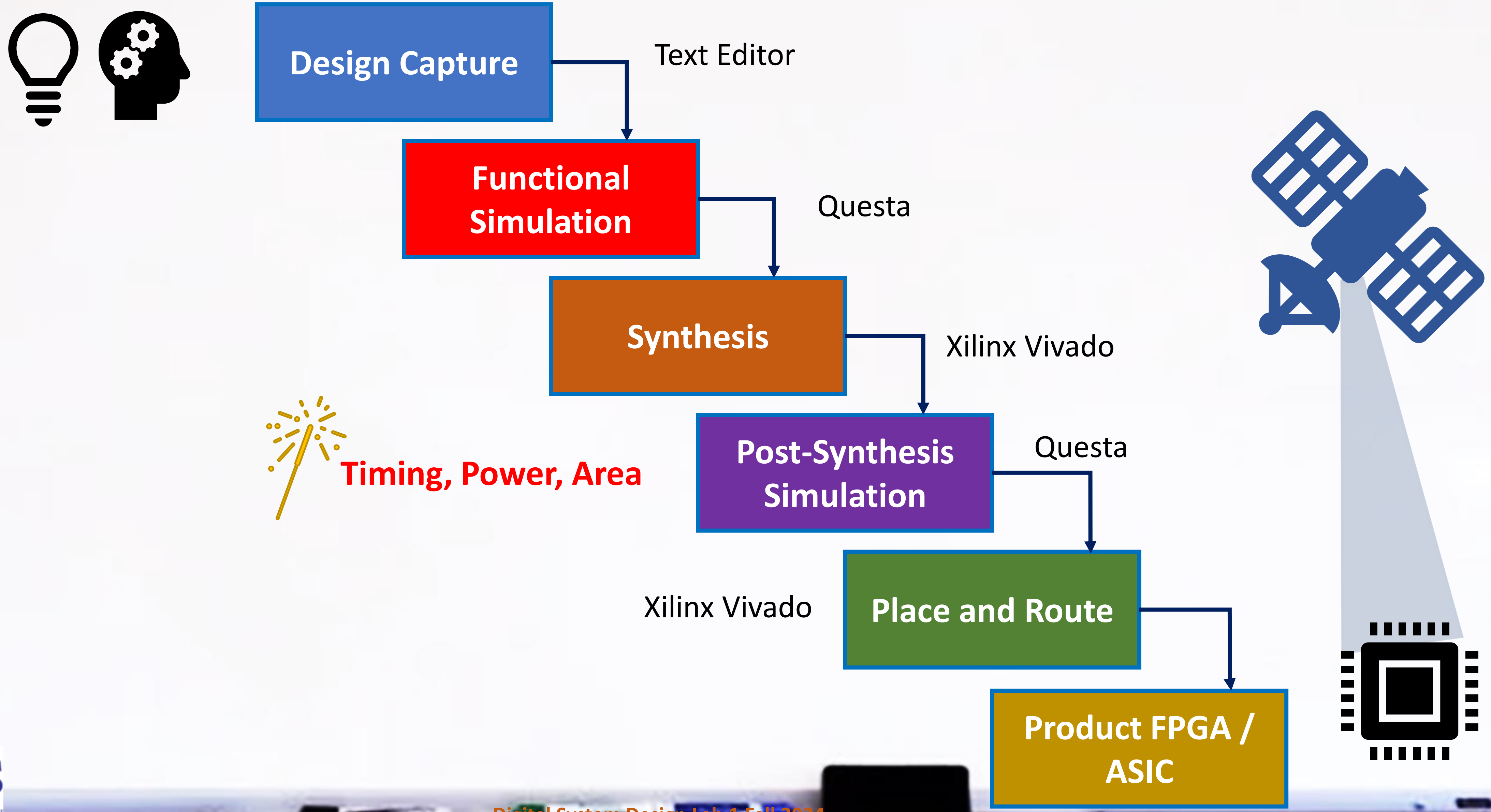


What is achieved?

Design Flow – from idea to product



Step Through EDA Tools



Simulation Tool Questa

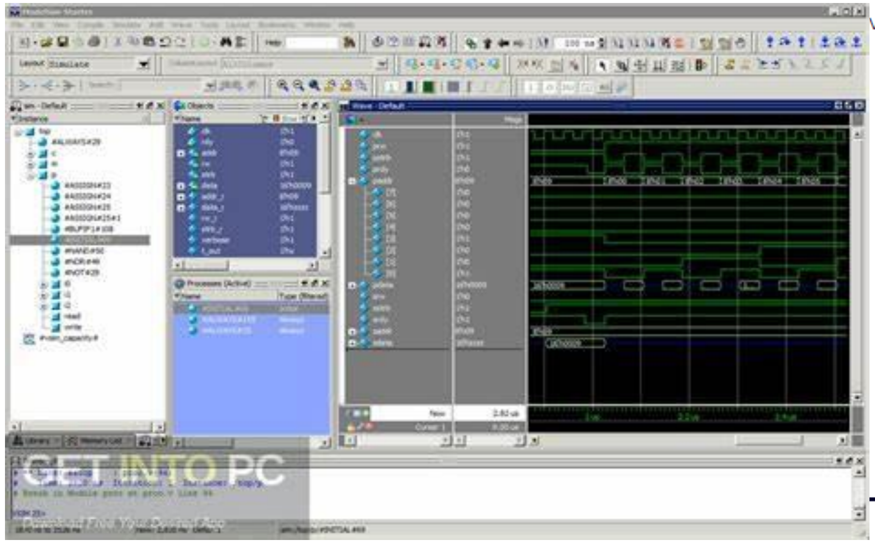
https://resources.sw.siemens.com/en-US/fact-sheet-questasim?bc=eyJwYWdlIjoiM3piMjROVWxuaUJDTGlmR0RXdkhlaylsInNpdG

SIEMENS Digital Industries Software

Software & Products | Solutions & Services | Industries | Training & Support

Questa Advanced Simulator

Verification > Questa Simulatic



In addition to the sheer size of designs and the inclusion of multiple embedded processors and advanced interconnect systems, the increase in software content and the configurability required by multi-platform design require a functional verification solution that unifies a broad arsenal of verification features. This places tremendous importance on having a verification plan informed by the collection of coverage metrics that track progress against the plan throughout the verification process. This intelligent verification plan enables engineers to allocate and manage resources efficiently and identify trends as the project progresses.

The QuestaSim verification solution delivers on these requirements for complex SoC designs. QuestaSim achieves industry-leading performance and capacity through aggressive, global compile and simulation optimization algorithms for

https://resources.sw.siemens.com/en-US/fact-sheet-questasim?bc=eyJwYWdlIjoiM3piMjROVWxuaUJDTGlmR0RXdkhlaylsInNpd

Software & Products | Solutions & Services | Industries | Training & Support

Questa Advanced Simulator

SystemVerilog, VHDL, and SystemC. Meanwhile, its Questa Visualizer debug environment provides high-performance, high-capacity debugging for dramatic regression throughput improvements when running large test suites.

QuestaSim Benefits

- Industry-leading high performance multi-language simulator
- High-performance, high-capacity unified debug
- Reference simulator for LRM compatibility
- UVM, SystemVerilog, VHDL, SystemC, and mixed language support
- Native compiled, single kernel simulator technology
- Next generation Visualizer debug environment
- Code coverage and functional coverage
- SVA and PSL assertions
- Intelligent coverage closure
- Integrated verification management and analysis
- Simulate in advanced optimization mode
- Best-in-class power-aware verification technology
- Profiling for hotspot analysis
- C code debug
- X-propagation dynamic simulation
- Real number modeling
- Common coverage database and flows

Share

Synthesis Tool Xilinx Vivado

Documentation Portal

Search in all documents

Sign In

Vivado Design Suite User Guide: Synthesis (UG901) UG901 2023-06-09 2023.1 English

Search in document

Keywords

- Vivado Synthesis
  - Introduction
  - Synthesis Methodology
  - Using Synthesis
  - RTL Linter
  - Running Synthesis
  - Setting a Bottom-Up, Out-of-Context Flow
  - Incremental Synthesis
  - Using Third-Party Synthesis Tools with Vivado IP
  - Moving Processes to the

Terms and Conditions | Privacy | Trademarks | Statement on Forced Labor | Fair and Open Competition | UK Tax Strategy | Cookie Policy | Cookie Settings

Vivado Synthesis

Introduction

Synthesis is the process of transforming a Register Transfer Level (RTL) specified design into a gate-level representation. AMD Vivado™ synthesis is timing-driven and optimized for memory usage and performance. Vivado synthesis supports a synthesizeable subset of:

- SystemVerilog: IEEE Standard for SystemVerilog-Unified Hardware Design Specification, and Verification Language (IEEE Std 1800-2012)
- Verilog: IEEE Standard for Verilog Hardware Description Language (IEEE Std 1364-2005)
- VHDL: IEEE Standard for VHDL Language (IEEE Std 1076-2002)
- VHDL 2008
- Mixed languages: Vivado supports a mix of VHDL, Verilog, and SystemVerilog.

In most instances, the Vivado tools also support Xilinx design constraints (XDC), which is based on the industry-standard Synopsys design constraints (SDC).

!! **Important:** Vivado synthesis does not support UCF constraints. Migrate UCF constraints to XDC constraints. For more information, see *ISE to Vivado Design Suite Migration Guide* (UG911).

Place and Route – Xilinx (FPGA) Technology

https://docs.xilinx.com/r/2021.2-English/ug892-vivado-design-flows-overview/Placement-and-Routing

Documentation Portal

Search in all documents | Sign In

ado Design Suite User Guide: Design Flows Overview (UG892) | UG892 | 2021-11-17 | 2021.2 English

rch in document

ywords

Placement and Routing

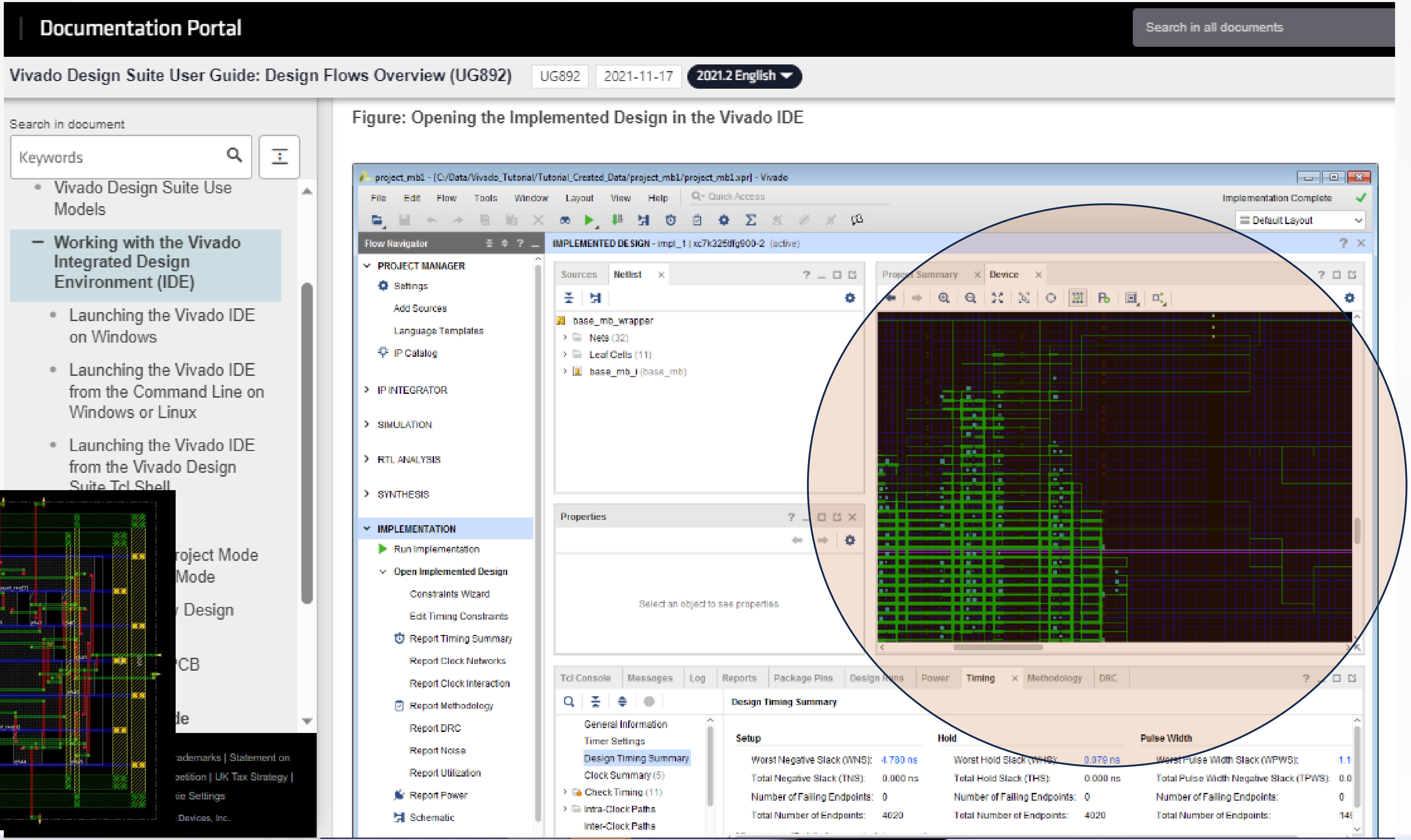
When the synthesized netlist is available, Vivado implementation provides all the features necessary to optimize, place and route the netlist onto the available device resources of the target part. Vivado implementation works to satisfy the logical, physical, and timing constraints of the design.

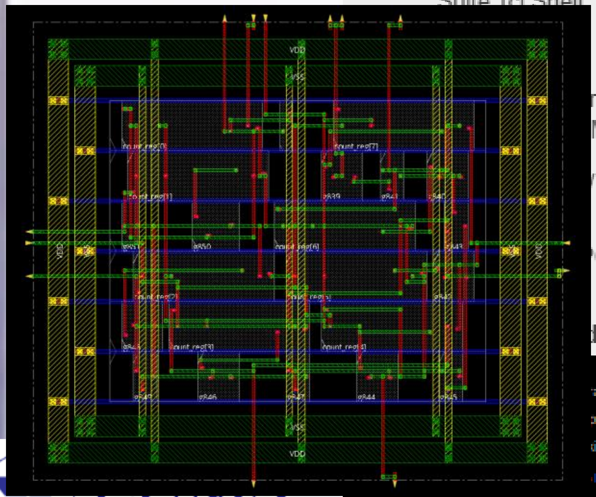
For challenging designs the Vivado IDE also provides advanced floorplanning capabilities to help drive improved implementation results. These include the ability to constrain specific logic into a particular area, or manually placing specific design elements and fixing them for subsequent implementation runs. For more information, see the *Vivado Design Suite User Guide: Design Analysis and Closure Techniques* (UG906).

Hardware Debug and Validation

After implementation, the device can be programmed and then analyzed with the Vivado logic analyzer, or within the standalone Vivado Lab Edition environment. Debug signals can be identified in the RTL design, or inserted after synthesis and are processed throughout the flow. You can add debug cores to the RTL source files, to the synthesized netlist, or in an implemented design using the using the Engineering Change Order (ECO) flow. You can also modify the nets connected to a debug probe, or route internal signals to a package pin for external probing using the ECO flow. For more information, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908).

Post Layout Design on Xilinx FPGA





Hardware Description Languages - SystemVerilog

- Specialized Computer Language for Capturing Hardware Design Idea
- Can be used to describe structure and **behavior** of digital circuits
- HDL include notion of hardware specific **time or delay**
- They have support for **Concurrency** which is peculiar to hardware

Remember 3 distinct features of HDL not available in other Programming languages

SystemVerilog

- Verilog is a Hardware Description Language that was Standardized by IEEE through Standard IEEE-1364-2001, first proposed in 1995
- Most Verilog constructs can be Simulated and Synthesized
- System Verilog includes some extensions for high-level object-oriented design and first became IEEE standard in 2008
- VHDL is another HDL (Very High-Speed Integrated Circuit HDL), 1987
- VHDL is strictly typed language and allows complex data types
- Both Verilog and VHDL are used worldwide
- SystemVerilog is becoming popular as it incorporates UVM features for verification; latest IEEE Standard 1800-**2023, previous 2017, 2012**

Design Capture in SystemVerilog HDL

SystemVerilog Allows Design Capture at Various Hierarchy Levels:

1. Switch Level (NMOS and PMOS transistors)
2. Gate Level (Describing Circuit as Logic Gates)
   1. Basic Functions of Gates are available as Primitives
3. Data Flow (RTL) Level Design
4. Behaviour Level Design

CMOS VLSI Design

Structural Design like Schematic

The Design is captured in a text file or it can be obtained from schematic diagram

SystemVerilog Syntax

```
// comments
module ExampleOne( output f,
                   input  a, b);

   wire Ax, Bx;    // Internal wires

   /* Behavioral description */
   assign Ax = a && b;
   assign Bx = a || b;
   assign f = (Ax && Bx) || a;

endmodule

```

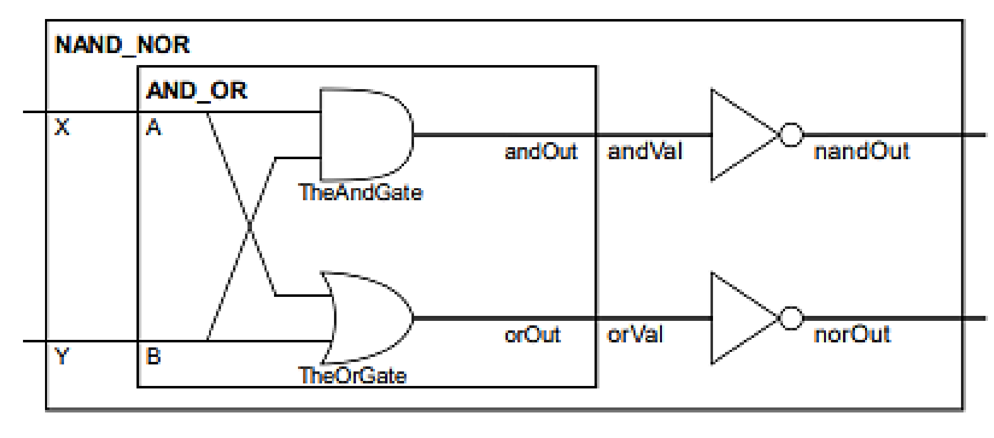
Post-2001 Verilog allows the port name, direction, and type to be declared together.

- Each port needs to have a user-defined name.
- The port directions are declared to be one of the three types: **input**, **output**, or **inout**.
- A port can take on any of the data types, but only **wires**, **registers**, and **integers** are synthesizable.

Structural SystemVerilog (port mapping)

Components available in library are port mapped

```
// Compute the logical AND and OR of inputs A and B (ANSI-style)
module AND_OR (output logic andOut, orOut,
               input  logic A, B);
  and TheAndGate (andOut, A, B);
  or  TheOrGate (orOut, A, B);
endmodule
```

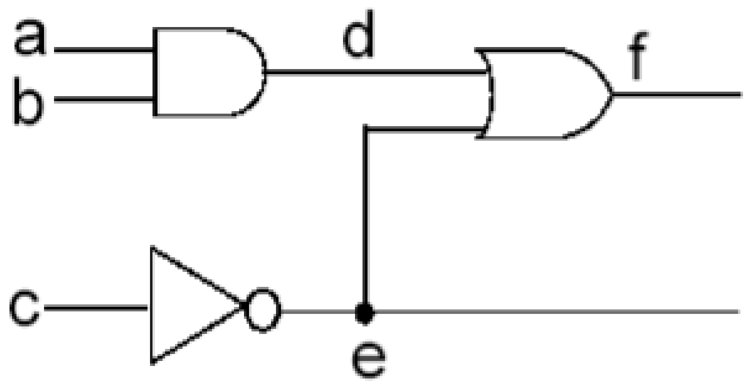



```
// Compute the logical NAND and NOR of inputs X and Y.
// The AND_OR module definition can be in the same file or
//   in a separate file in the same project.
module NAND_NOR (nandOut, norOut, X, Y);
  output logic nandOut, norOut;
  input  logic X, Y;
  logic andVal, orVal;  // local signals (not ports)

  AND_OR aoSubmodule (.andOut(andVal), .orOut(orVal), .A(X), .B(Y));
  not n1 (nandOut, andVal);
  not n2 (norOut, orVal);
endmodule
```

Also an example of Hierarchical Design

Behavioral SystemVerilog (assign statement)

- This is the Highest Level of Abstraction in HDL
- Different Circuit blocks can be represented as Functions, Tasks and Hierarchical Modules
- ‘Assign’ and ‘Always’ are relevant constructs

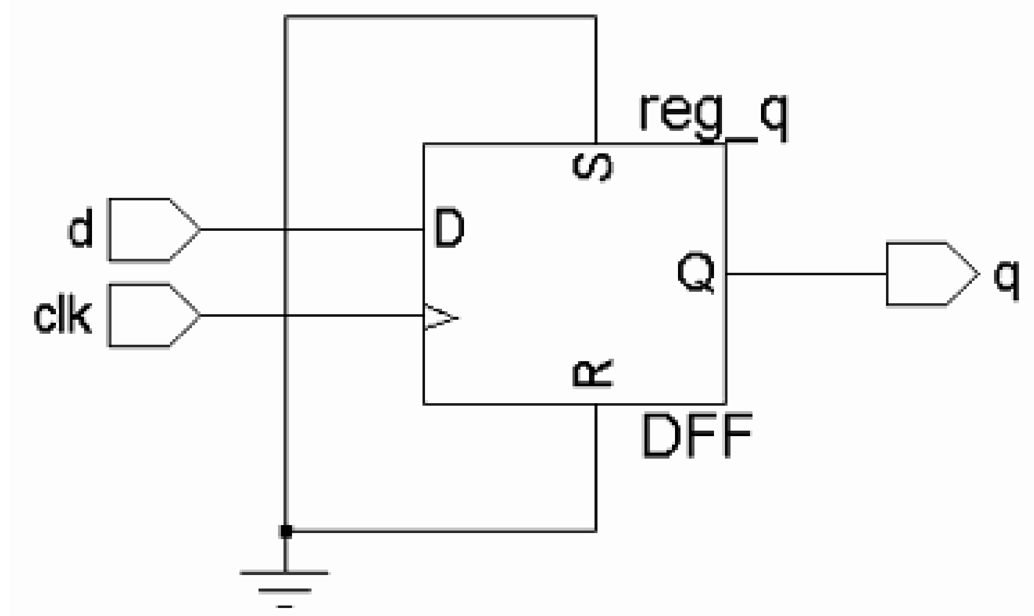



```
module example(a, b, c, f, e);
   input a, b, c;
   output f, e;
   assign d = a & b;
   assign e = ~c;
   assign f = d | e;
endmodule
```

Youpyo Hong, Dongguk University

RTL in SystemVerilog (always blocks)

- RTL =Register Transfer Level
- Module is described in Verilog in the form of Data Flow
- Signals are assigned by the data operation
- Design is Implemented using Concurrent Assignments
- ‘Always’ and ‘Clk’ are relevant constructs




```
module ff (output reg q, input d,
           input clk);

  always @(posedge clk)
    q <= d;

endmodule
```

Modular Approach in SystemVerilog

- Module is a basic building block in SystemVerilog
- Module can be a collection of multiple low level logic blocks
- Modules are interconnected through Port Interface
- Internal functionality of any Module can be altered without affecting any other Module

```
module mod_param
    #(parameter int             WIDTH=8,
      parameter logic[7:0]      VALUE='h0)
     (input     logic           clk,
      input     logic           rst_n,
      input     logic[WIDTH-1:0] in,
      output    logic[WIDTH-1:0] out);


endmodule
```

```
module top_level (
  input logic clock,
  input logic reset,
  output logic [7:0] count_8,
  output logic [11:0] count_12
);

  // Instantiation of the 8 bit counter
  // In this instance we can use the default
  // value fo the parameter
  counter 8bit_count (
    .clock (clock),
    .reset (reset),
    .count (count_8)
  );

  // Instantiation of the 12 bit counter
  // In this instance we must override the
  // value of the WIDTH parameter
  counter # (
    .WIDTH (12)
  ) 12bit_count (
    .clock (clock),
    .reset (reset),
    .count (count_12)
  );

endmodule : top_level
```

LUMS
A Not-for-Profit University

Use of Parameter and Generate in SystemVerilog

```
* A simple generate example. This paramerter OPERATION_TYPE,
* passed when this module is instantiated, is used to select
* the operation between inputs `a` and `b`.
*/
module conditional_generate
    #(parameter OPERATION_TYPE = 0)
    (
        input  logic [31:0] a,
        input  logic [31:0] b,
        output logic [63:0] z
    );

    // The generate-endgenerate keywords are optional.
    // It is the act of doing a conditional operation
    // on a parameter that makes this a generate block.
    generate
        if (OPERATION_TYPE == 0) begin
            assign z = a + b;
        end
        else if (OPERATION_TYPE == 1) begin
            assign z = a - b;
        end
        else if (OPERATION_TYPE == 2) begin
            assign z = (a << 1) + b; // 2a+b
        end
        else begin
            assign z = b - a;
        end
    endgenerate
endmodule: conditional_generate
```

```
// Design for a half-adder
module ha ( input   a, b,
            output  sum, cout);

  assign sum  = a ^ b;
  assign cout = a & b;
endmodule

// A top level design that contains N instances of half adder
module my_design
    #(parameter N=4)
        (  input [N-1:0] a, b,
           output [N-1:0] sum, cout);

    // Declare a temporary loop variable to be used during
    // generation and won't be available during simulation
    genvar i;

    // Generate for loop to instantiate N times
    generate
        for (i = 0; i < N; i = i + 1) begin
          ha u0 (a[i], b[i], sum[i], cout[i]);
        end
    endgenerate
endmodule
```

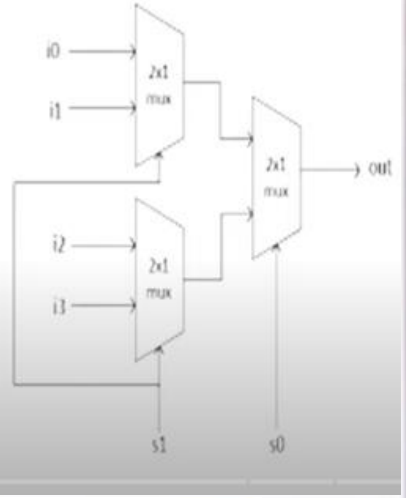
Hierarchical Approach in SystemVerilog

- Instantiation allows creation of **Hierarchy** in SystemVerilog.
- The Modules that are used in another Module are called **Instances** or **Instantiation**.
- Nesting of Modules is not permitted as it is Hardware Design.
- One Module can be used in many other Modules through **Instantiation**

Design Reuse

4 to 1 MUX using 2 to 1 MUX

```
module mux_2to1 (i0,i1,sel,out);
    input i0,i1,sel;
    output out;
    always@(i0,i1,sel)
    begin
        if(sel)
            out = i1;
        else
            out = i0;
    end
endmodule
```




```
module mux_4to1 (i0,i1,i2,i3,s1,s0,out);
    input i0,i1,i2,i3,s1,s0;
    output out;
    wire x1,x2;

    mux_2to1 m1 (i0,i1,s1,x1);
    mux_2to1 m2 (i2,i3,s1,x2);
    mux_2to1 m3 (x1,x2,s0,out);

endmodule
```

Design Abstraction in SystemVerilog

Two Design Methodologies

- **Top-Down Methodology**
  - We define the top-level block and identify the Sub Blocks needed to build the top-level block
  - These Sub Blocks are further divided until we come to Leaf Cells which cannot be further Sub Divided
- **Bottom-Up Methodology**
  - We first Identify the Building Blocks (Leaf Cells) that are available to us in a library
  - We build bigger blocks using these Building Blocks
  - The process continues until we reach the Top-Level block

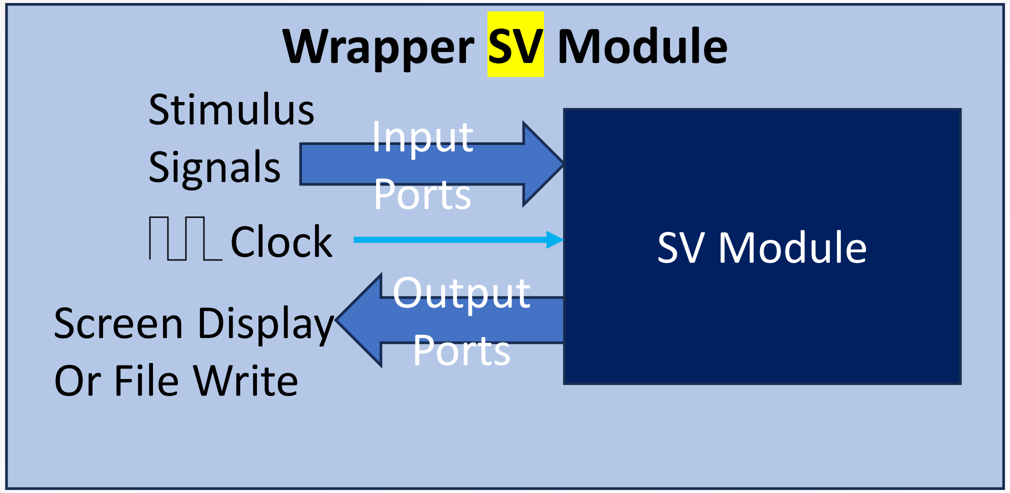
Test Bench in HDL

- Used for Functional as well as Post-Synthesis Testing
- Timing Information can be added through Gate Models

LUMS
A Not-for-Profit University

Writing Test Benches in SystemVerilog

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

  assign y = ~b & ~c | a & ~b;
endmodule
```

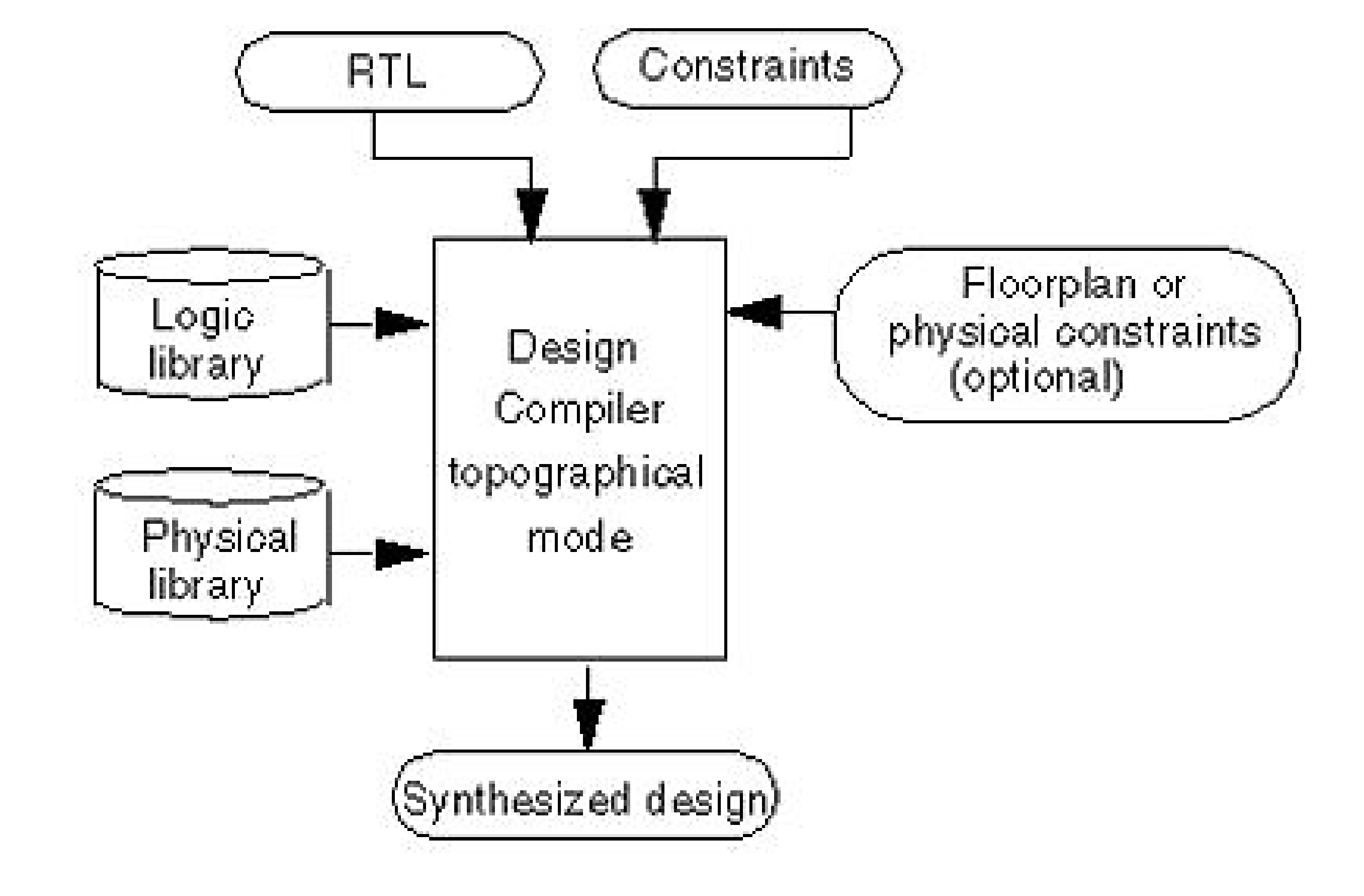


SV = SystemVerilog file extension

Instantiation

```
module tb ();
  reg a, b, c;
  wire y;

 design dut (.a(a), .b(b), .c(c), y(y));

 // apply input sequence

  initial
    begin

      a = 0; b = 0; c = 0 ; # 10;
      a = 0; b = 0; c = 1 ; # 10;
      a = 0; b = 1; c = 0 ; # 10;
      a = 0; b = 1; c = 1 ; # 10;
      a = 1; b = 0; c = 0 ; # 10;
      a = 1; b = 0; c = 1 ; # 10;
      a = 1; b = 1; c = 0 ; # 10;
      a = 1; b = 1; c = 1 ; # 10;


    end
endmodule
```

RTL Synthesis



LUMS
A Not-for-Profit University

Lab 1 Task – EDA Tools Installation

- **Refer to Lab Manual 1 and Appendix A**

- --------------- for own learning ---------------------
- SystemVerilog language
  https://www.chipverify.com/tutorials/systemverilog

- Online EDA tools (iverilog) for simulation and testing

https://courses.edaplayground.com/home