香港中文大學
The Chinese University of Hong Kong

CENG3430 Rapid Prototyping of Digital Systems

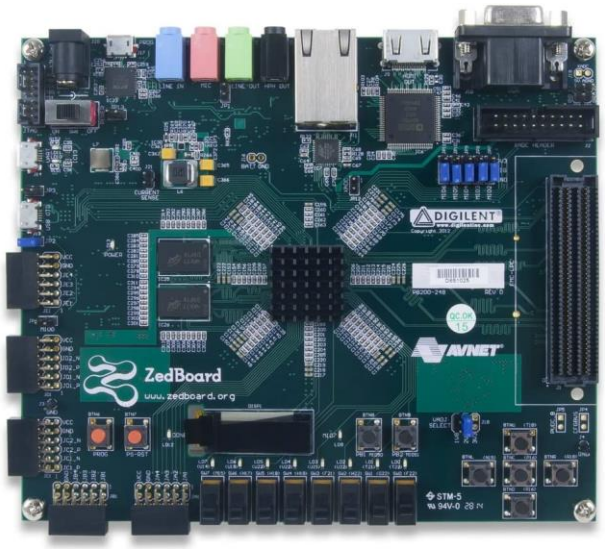
Lecture 02: Digital System Design Basics

Ming-Chang YANG

mcyang@cse.cuhk.edu.hk

Outline

- Design Flow of Digital Systems
  - ① Requirements, Design Spec., and Design Formulation
  - ② Design Entry
  - ③ Behavioral Simulation
  - ④ Logic Synthesis
  - ⑤ Post-Synthesis Simulation
  - ⑥ Mapping, Placement, Routing
  - ⑦ Design Target
- Our Design Target: **Zynq ZedBoard**
  - ZedBoard Layout and Interfaces
  - Specifying ZedBoard in Vivado
  - Hardware Setup for ZedBoard
  - Programming the ZedBoard
  - Xilinx Design Constraints (XDC) File

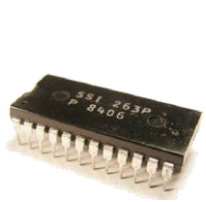


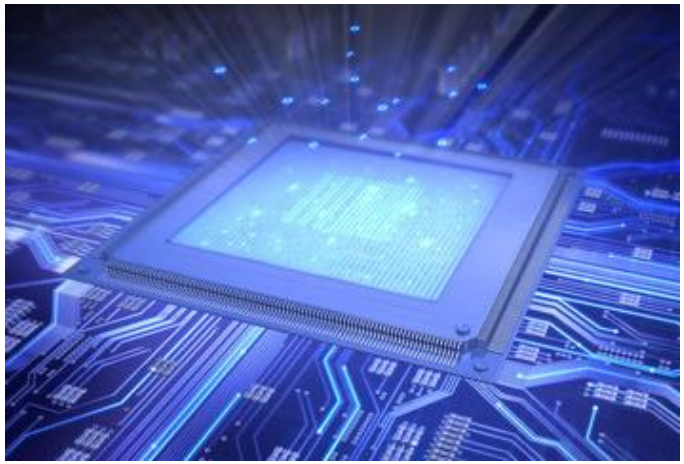
Integrated Circuit Technology

- Integrated circuit (IC) technology has improved to allow more and more components on a chip.
  - Small Scale Integration (SSI): 1 to 20 gates
  - Medium Scale Integration (MSI): 20 to 200 gates
  - Large Scale Integration (LSI): 200 to few thousands gates
  - Very Large Scale Integration (VLSI): More than 10,000 gates
  - Ultra Large Scale Integration (ULSI): 100 million transistors

Gates

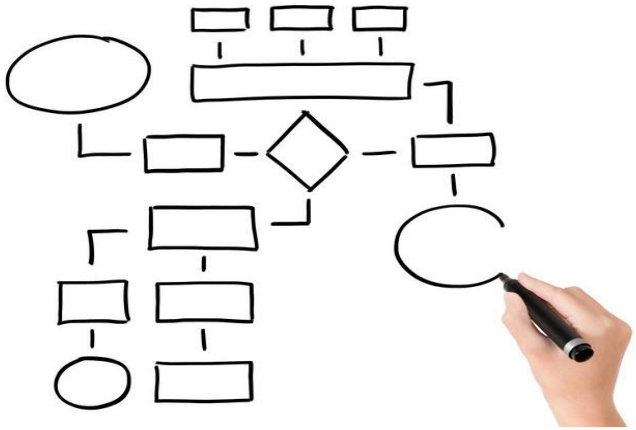
- Digital system design have become more complex.

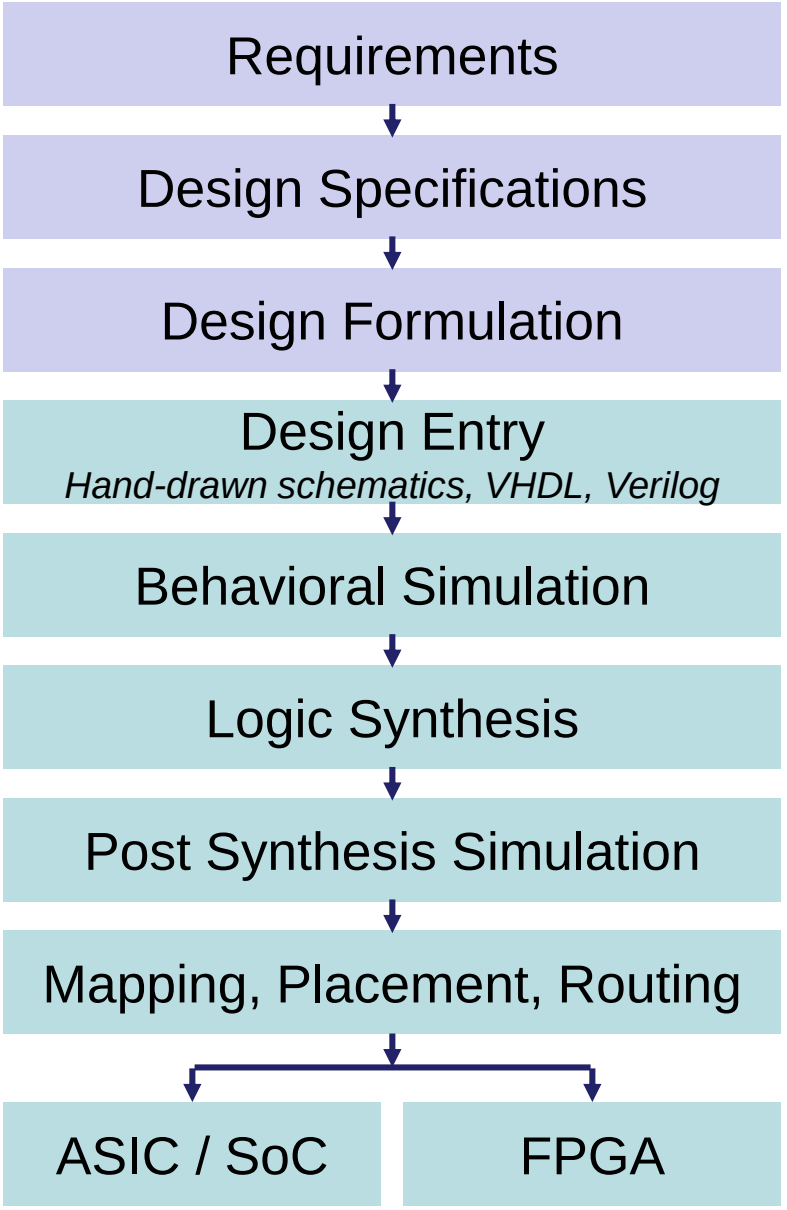




Design Flow of Digital Systems (1/7)

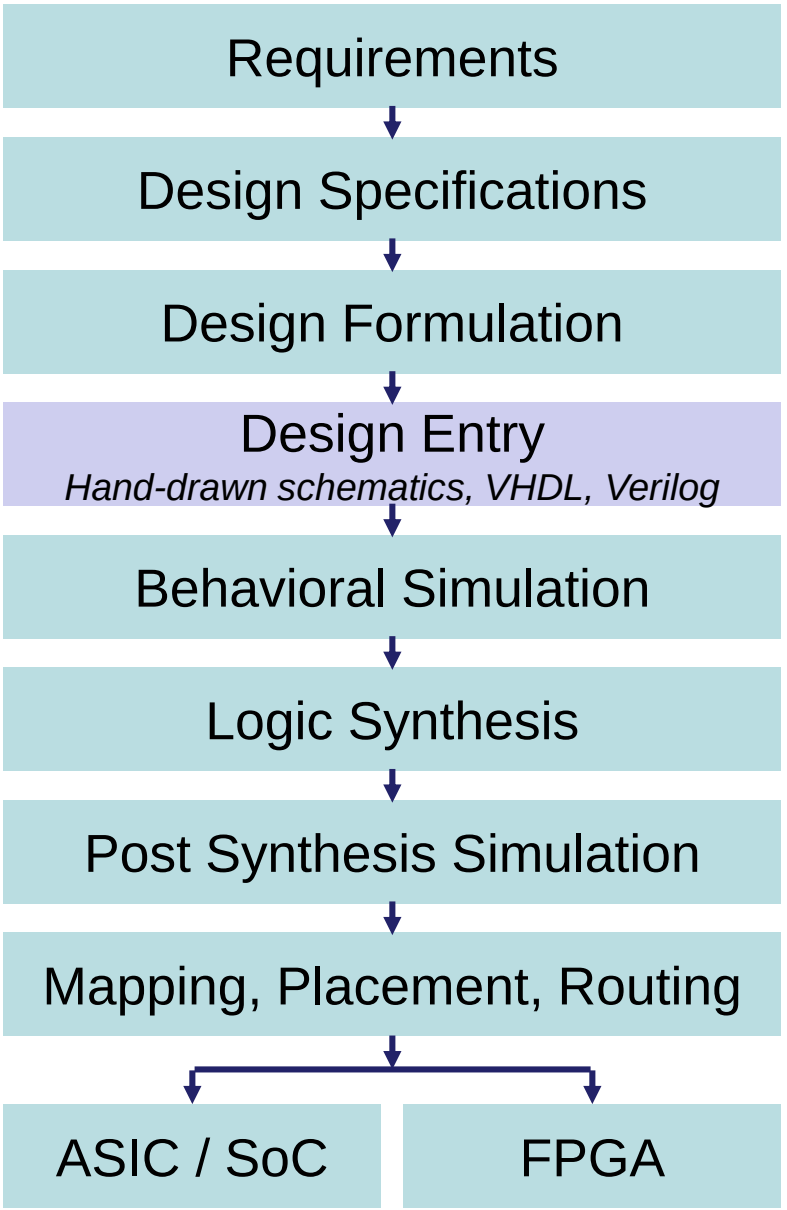
- Requirements, Design Spec., and Design Formulation:
  - All the designs start with design requirements and design specifications.
  - The next step is to formulate the design conceptually.
    - Either at a block diagram level or at an algorithmic level.





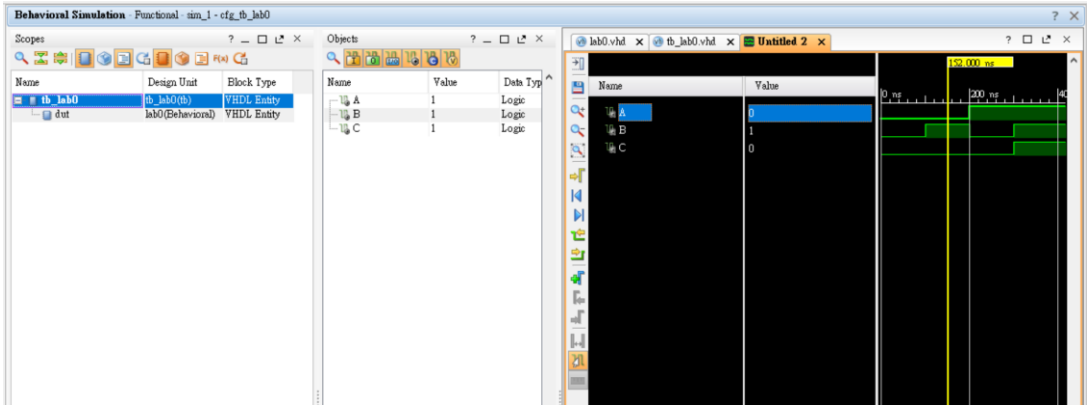
Design Flow of Digital Systems (2/7)

- Design Entry:
  - *Olden days*: Hand-drawn schematic or blueprint
  - *Now*: Computer-aided design (CAD) tools
    - **Schematic Capture**: Design with gates, flip-flops, and standard building blocks.
      - E.g., ORCAD
    - **Hardware Descriptions Languages (HDLs)**: Design and debug at higher level
      - E.g., VHDL and Verilog

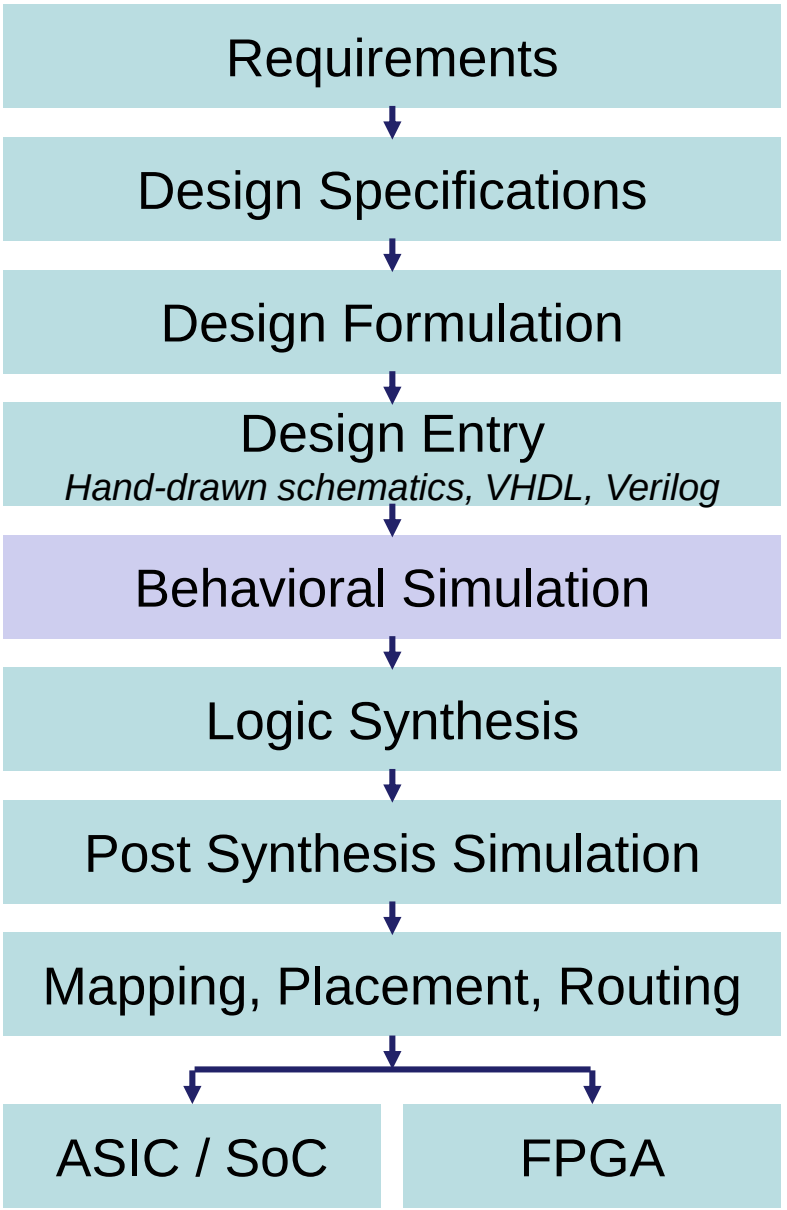


Design Flow of Digital Systems (3/7)

- Behavioral Simulation:
  - The entered design then should be simulated;
  - To ensure it function correctly at high-level behavioral model;
  - To unveil problems in the design.



Note: Behavior simulation is a software-based simulation, which does not account for timing or physical constraints.

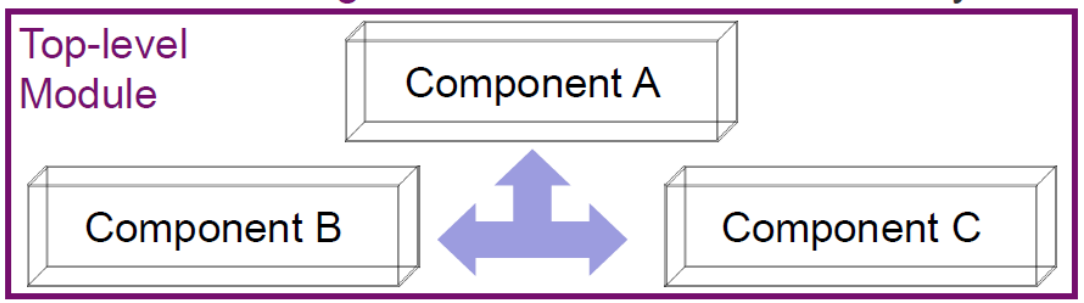


How does Testbench work? (3-1)

- In Lab01, we use "Online VHDL Testbench Template Generator" to generate a testbench template.

① The template uses "structural design" to create a "top-level" **tb_AND** module to test the developed "**AND**" entity.

- Structural Design: Like a circuit but describe it by text.



Connected by `port map` in the architecture body of the top-level design module

- **Design Steps**:
  - **Step 1**: Create `entities`
  - **Step 2**: Create `components` from `entities`
  - **Step 3**: Use "`port map`" to relate the components

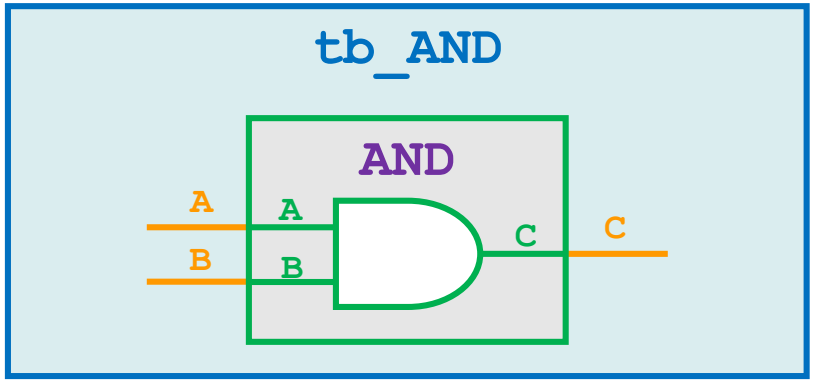
Testbench Template

```
...
architecture tb of tb_AND is
  component AND
    port (A : in std_logic;
          B : in std_logic;
          C : out std_logic);
  end component;
  signal A : std_logic;
  signal B : std_logic;
  signal C : std_logic;
  begin
    dut : AND
    port map ( A => A,
               B => B,
               C => C);
    stimuli : process
    begin
      A <= '0'; B <=  '0' ;
      -- EDIT Add stimuli here
    end process;
end tb;
```

How does Testbench work? (3-2)

② The template "plugs in" the developed **AND** entity via "component declaration" and "port map".

- It also declares internal signals (i.e., **A**, **B**, & **C**) to interconnect external signals (i.e., **A**, **B**, & **C**).
- "Nominal" instead of "positional" port map is used.



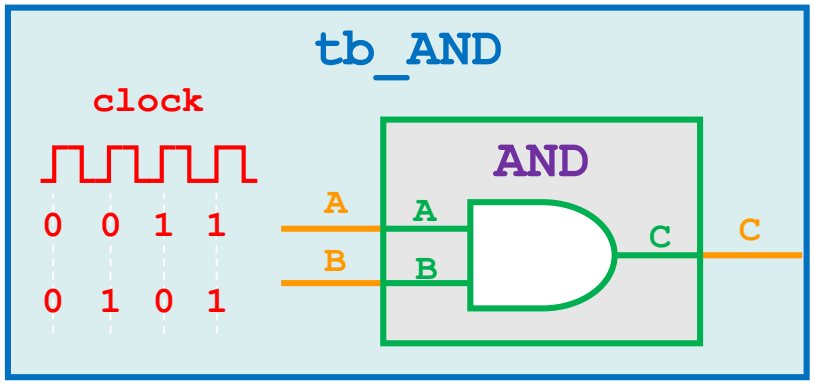
Testbench Template

```
...
architecture tb of tb_AND is
  component AND
    port (A : in std_logic;
          B : in std_logic;
          C : out std_logic);
  end component;
  signal A : std_logic;
  signal B : std_logic;
  signal C : std_logic;
  begin
    dut : AND
    port map ( A => A,
               B => B,
               C => C);
    stimuli : process
    begin
      A <= '0'; B <=  '0' ;
      -- EDIT Add stimuli here
    end process;
end tb;
```

How does Testbench work? (3-3)

③ The template typically uses a **process** to validate the developed entity.

- A **process** represents a sequential block of code, allowing us to feed all test cases “sequentially”.
- However, the template does not come with sufficient “test cases”.



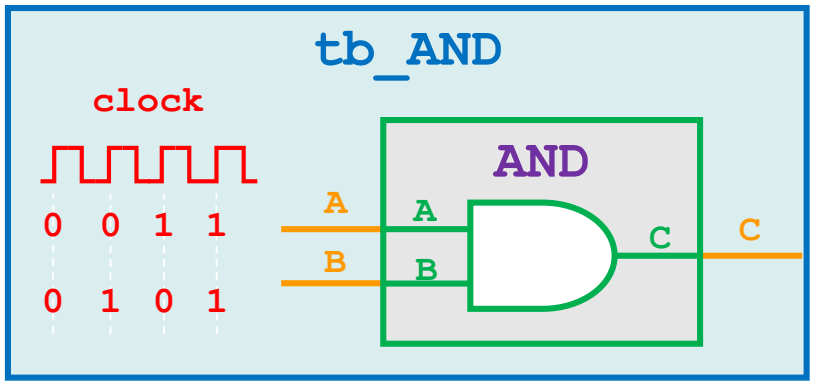
- Users need to complete this part themselves!

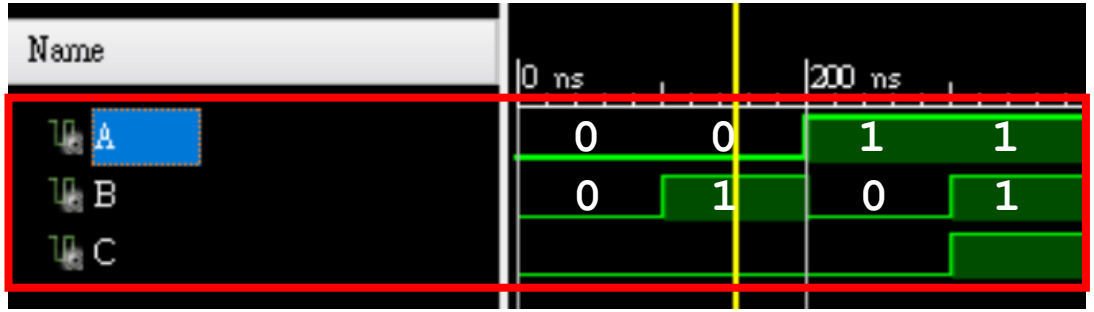
Testbench Template

```
...
architecture tb of tb_AND is
  component AND
    port (A : in std_logic;
          B : in std_logic;
          C : out std_logic);
  end component;
  signal A : std_logic;
  signal B : std_logic;
  signal C : std_logic;
  begin
    dut : AND
    port map ( A => A,
               B => B,
               C => C);
    stimuli : process
    begin
      A <= '0'; B <=  '0' ;
      -- EDIT Add stimuli here
    end process;
end tb;
```

Class Exercise 2.1

- Complete the Testbench to feed all test cases as follows:





Testbench Template

```
...
architecture tb of tb_AND is
  component AND
    port (A : in std_logic;
          B : in std_logic;
          C : out std_logic);
  end component;
  signal A : std_logic;
  signal B : std_logic;
  signal C : std_logic;
  begin
    dut : AND
    port map ( A => A,
               B => B,
               C => C);
    stimuli : process
    begin
      A <= '0'; B <=  '0' ;
      -- EDIT Add stimuli here
    end process;
end tb;
```

Class Exercise 2.1 (Answer)

- Style 1 (Simpler)

```
26      port map (A => A,
27                B => B,
28                C => C);
29
30      stimuli : process
31      begin
32          -- EDIT Adapt initia
33          A <= '0';
34          B <= '0';
35          wait for 100ns;
36          A <= '0';
37          B <= '1';
38          wait for 100ns;
39          A <= '1';
40          B <= '0';
41          wait for 100ns;
42          A <= '1';
43          B <= '1';
44          wait for 100ns;
```

Sequential

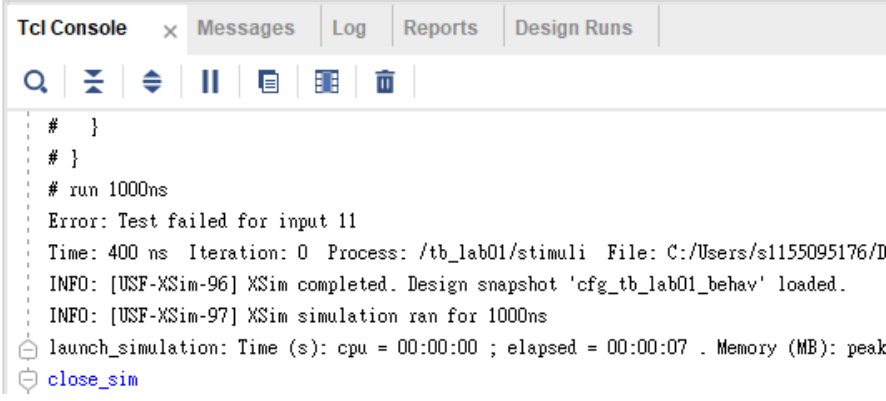
The inputted values will "last for" **100ns**.

- Style 2 (With assertion)

```
30      stimuli : process
31      begin
32          -- EDIT Adapt initializ
33          A <= '0';
34          B <= '0';
35          wait for 100
36          assert(C = '0')
37          report "Test failed for input 00" severity error;
38
39          A <= '0';
40          B <= '1';
41          wait for 100ns;
42          assert(C = '0')
43          report "Test failed for input 01" severity error;
44
45          A <= '1';
46          B <= '0';
47          wait for 100ns;
48          assert(C = '0')
49          report "Test failed for input 10" severity error;
```

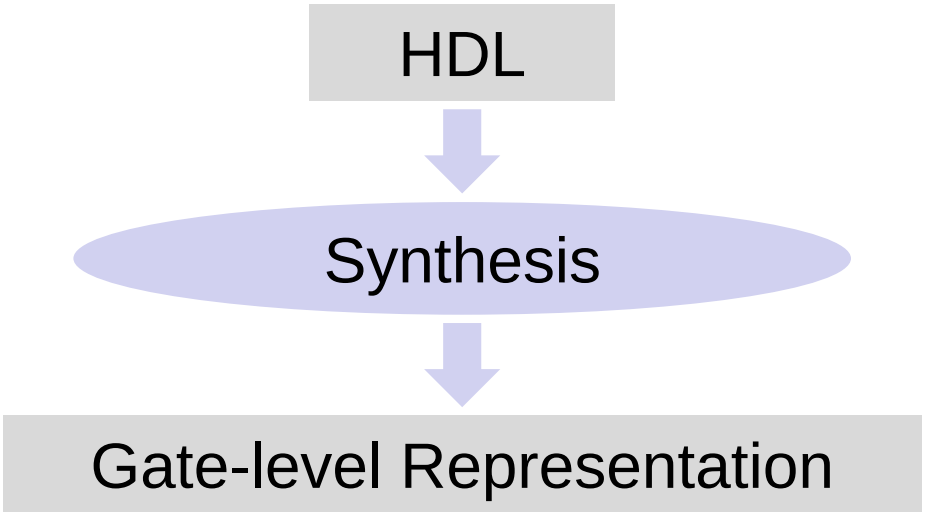
Sequential

It can print something in the console if anything went wrong!

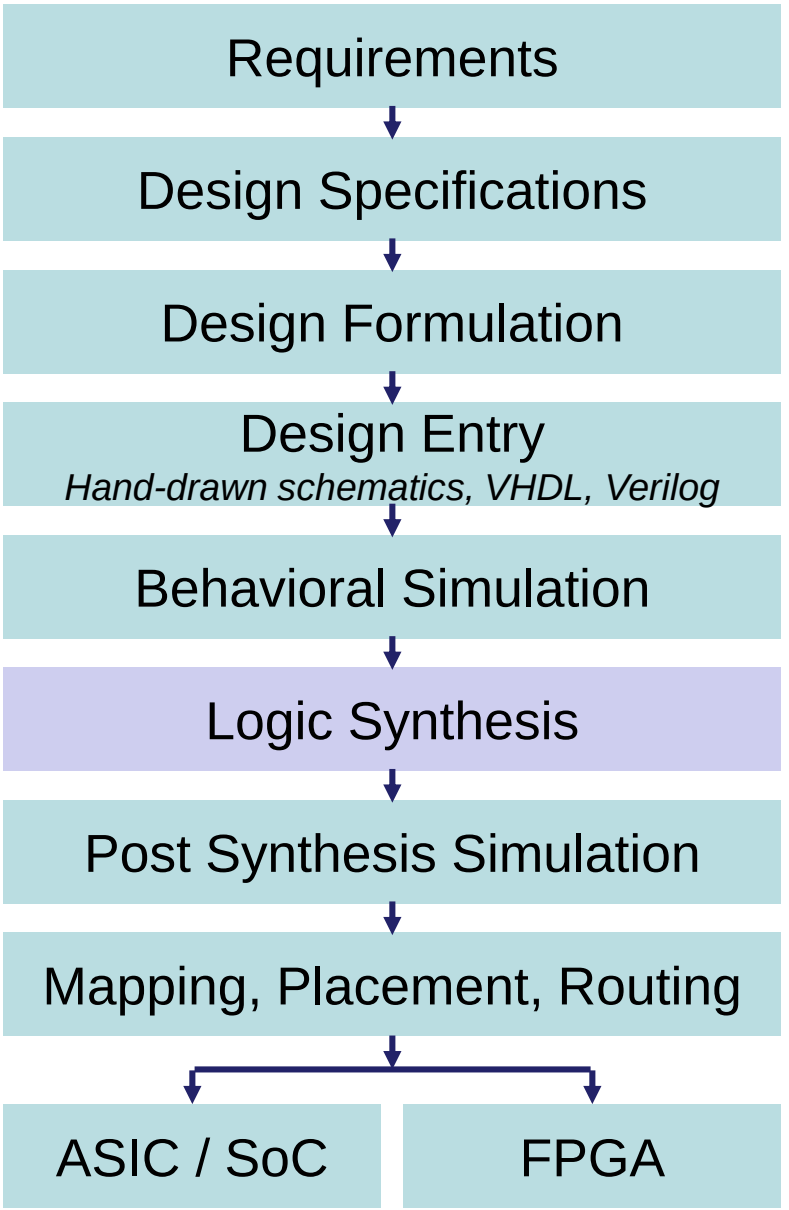


Design Flow of Digital Systems (4/7)

- Logic Synthesis



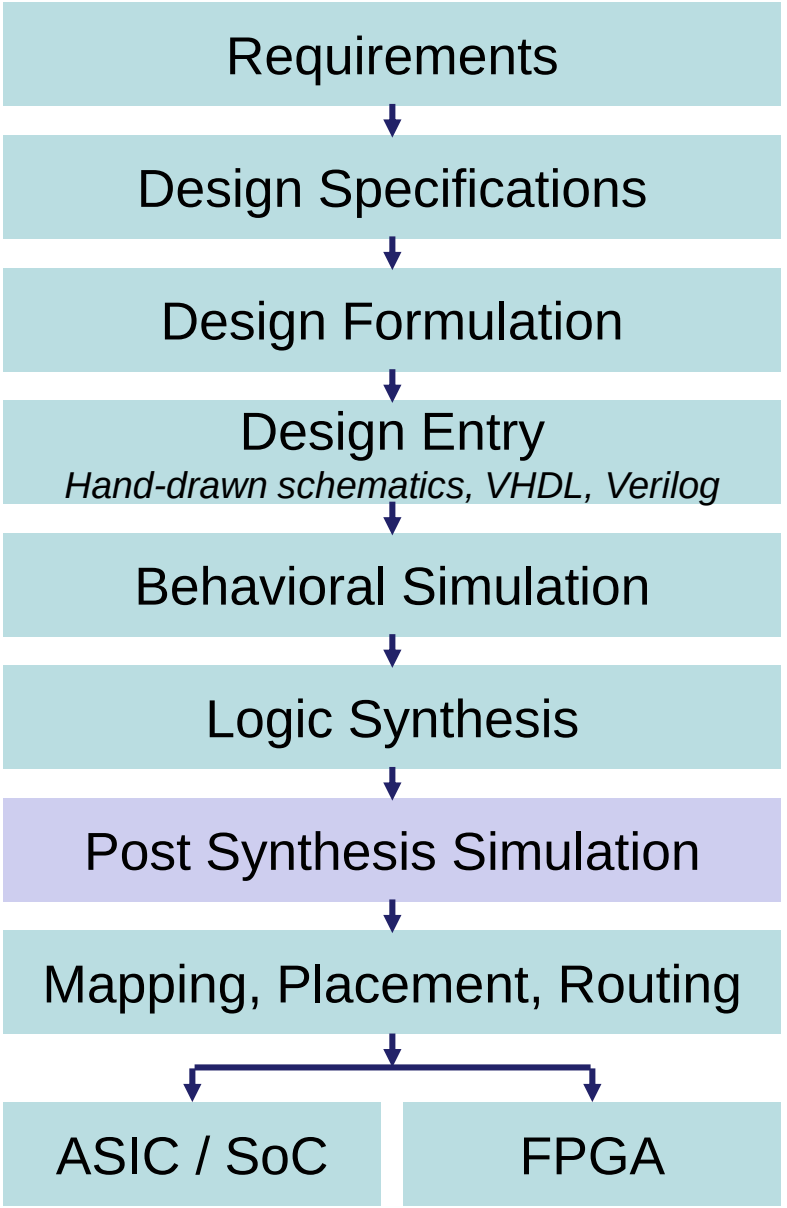
  - What is the gate-level representation (or “netlist”)?
    - A net refers to a connection of physical wires.
      - E.g., **`AND2X1(OUT, A, B)`**
    - A netlist is a file containing a list of nets.



Design Flow of Digital Systems (5/7)

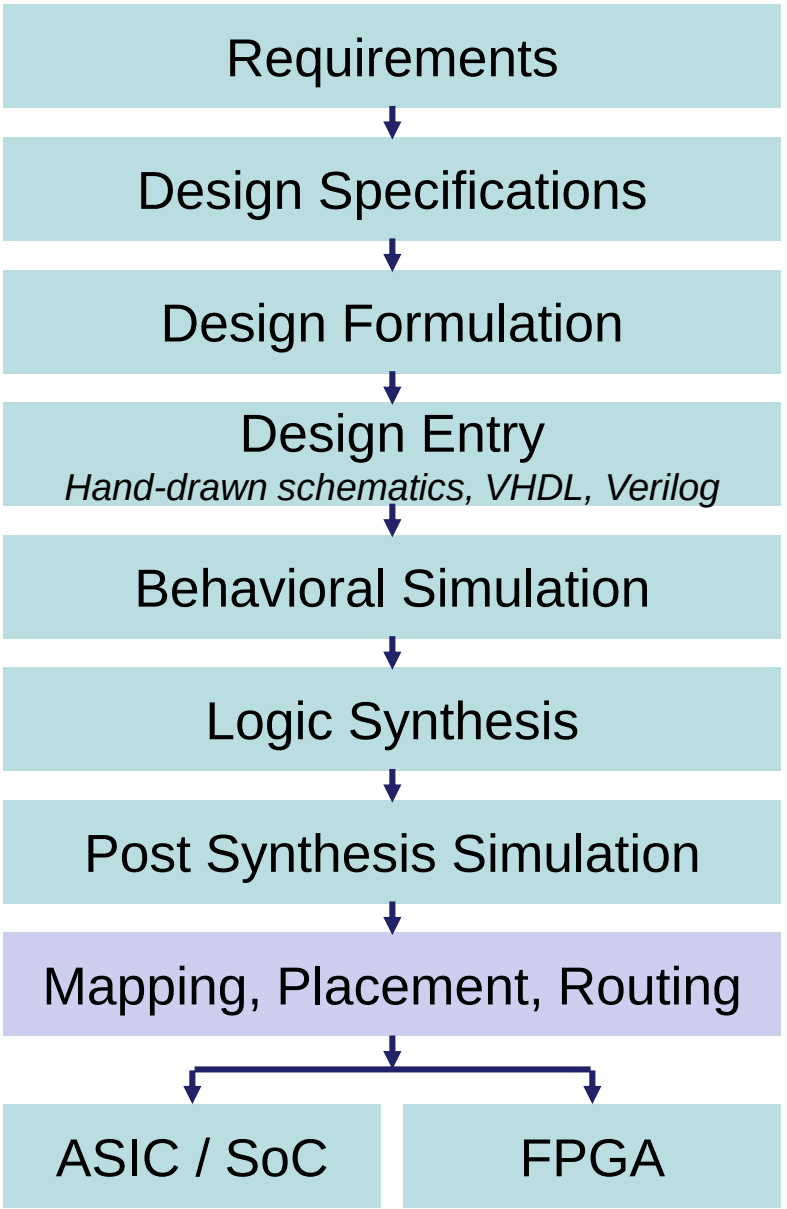
- Post-Synthesis Simulation
  - Performs simulation of the gate-level netlist to identify:
    - Timing errors;
    - Logical errors;
    - Power consumption issues.

- It is not necessary (and thus omitted in our lab exercises), but it is recommended in practice.



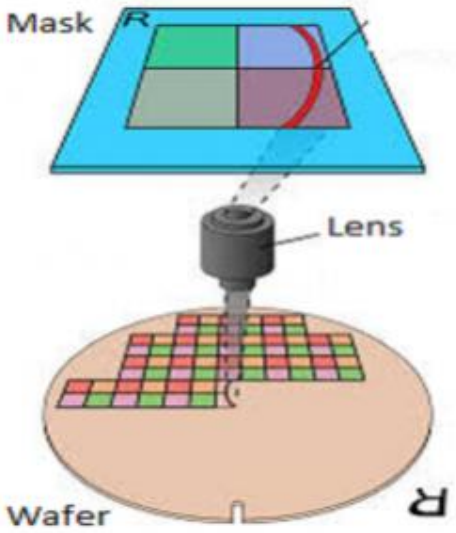
Design Flow of Digital Systems (6/7)

- Mapping, Placement, Routing
  - Mapping
    - Converts the gate-level netlist into the “onboard resources” of the **target**.
  - Placement and Routing
    - Placement: Determines the position of the resources.
    - Route: Connects the resources.
    - Both need to meet the timing and power constraints.
  - These are automated (and optimized) by CAD tool.

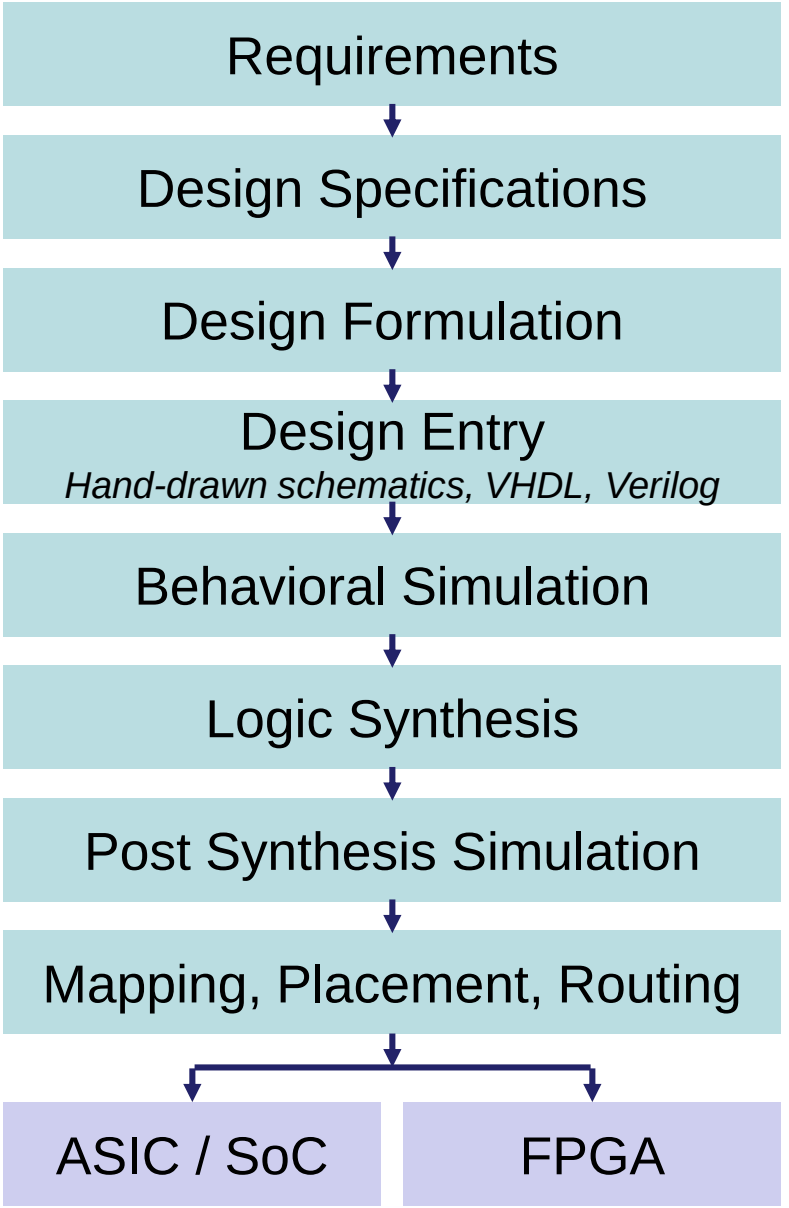


Design Flow of Digital Systems (7/7)

- Two most common **targets**:
  - AISC or SoC
    - The routed design is used to generate a photomask for producing integrated circuits (ICs).

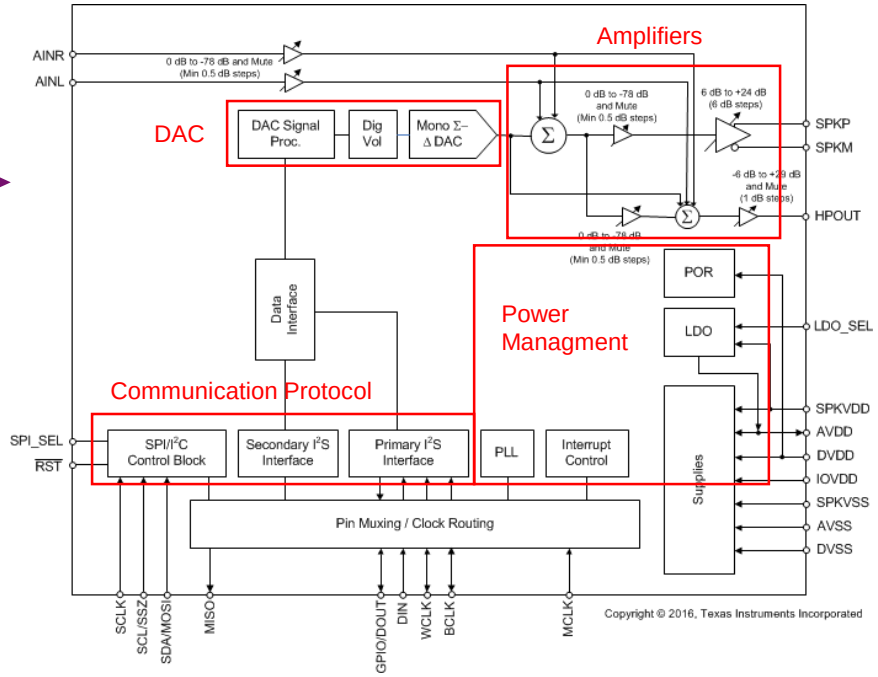


  - Field Programmable Gate Array (FPGA)
    - **Programming** simply involves writing a sequence of 0's and 1's into the programmable cells of FPGA.



What is ASIC?

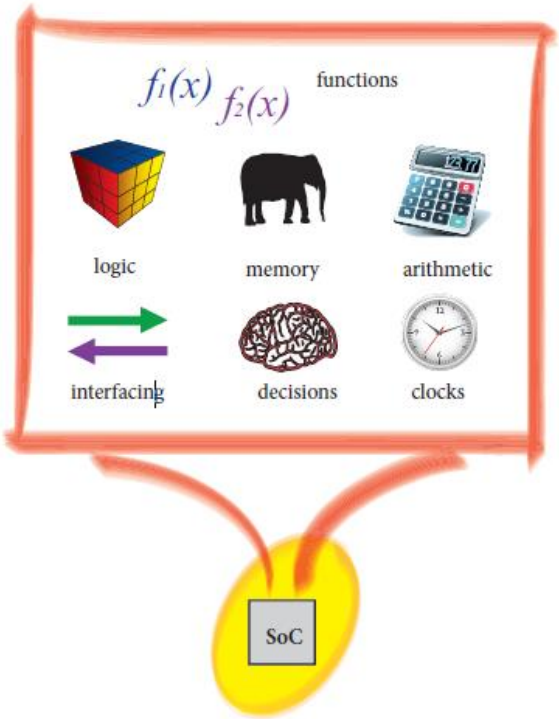
- **Application Specific Integrated Circuit (ASIC)**
  - A specialized chip aims at optimizing the performance and power consumption for certain applications.
  - Examples:
    - Imaging processors
    - Audio processors
    - Networking processors
    - Cryptographic processors
    - Networking processors
  - Features:
    - **High performance**
    - **Low power consumption**
    - **More expensive (smaller quantities)**
    - **Not reconfigurable**



TAS2505 Audio Processing Chip by Texas Instruments

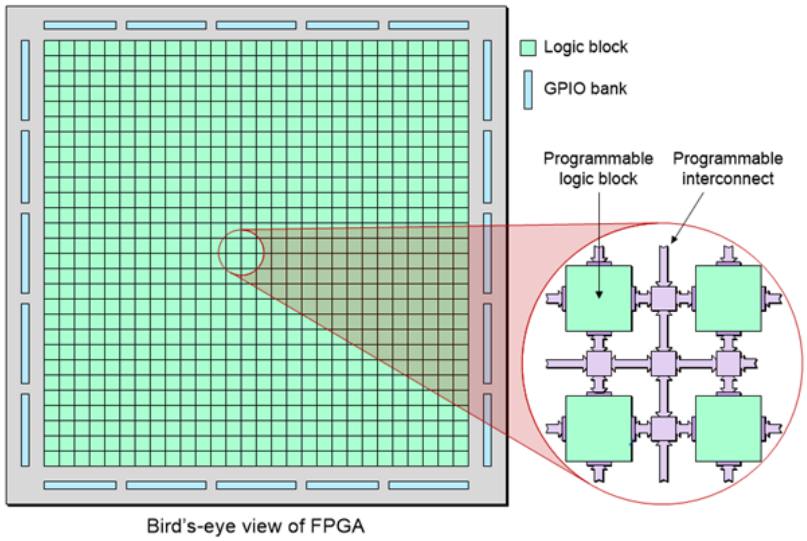
What is SoC?

- **System-on-Chip**
  - The implication is a single silicon chip can be used to implement the functionality of an entire system.
    - An SoC can combine all aspects of a digital system.
      - E.g., processing, high-speed logic, interfacing, memory, and etc.
  - Driven by the need for smaller, more portable devices
  - Features:
    - General purpose
    - Fairly high performance
    - Fairly low power consumption
    - Less expensive (Larger quantities)
    - Not reconfigurable in hardware



What is FPGA? (1/2)

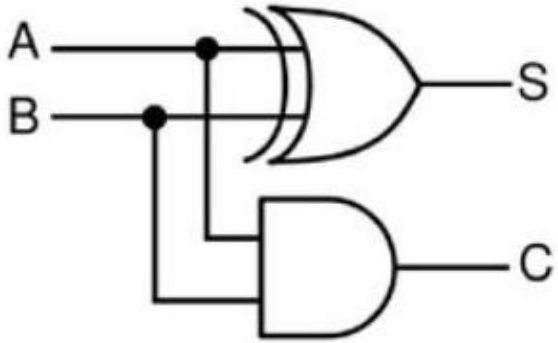
- How can FPGA be **reconfigurable**?
  - **Configurable Logic Block (CLB)** are the fundamental building blocks of FPGAs, which contain a combination of:
    - Look-Up Tables (**LUT**s) can be programmed to perform any logic function of its inputs for implementing **combinational logic**.
    - Flip-Flops (**FF**s) are used for storing state information, allowing for **sequential logic**.
    - Multiplexers (**MUX**s) help in routing signals within the CLB.
  - **Programmable Interconnects Point (PIP)** are the programmable connections between the different components of the FPGA.
    - Including CLBs, I/O blocks, and other resources.



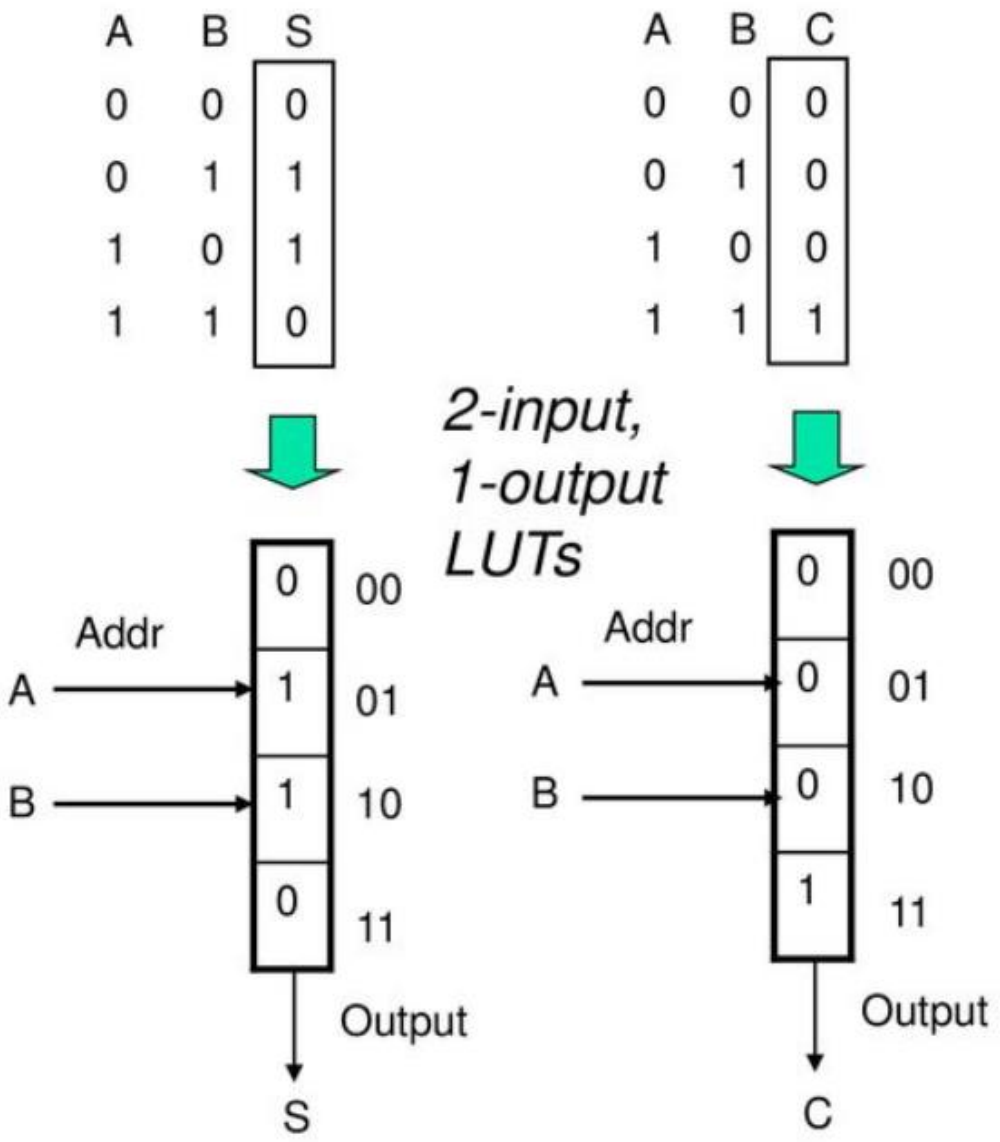
Bird's-eye view of FPGA

What is FPGA? (2/2)

- How can FPGA be **reconfigurable**?
  - Example: the half adder (combinational logic)

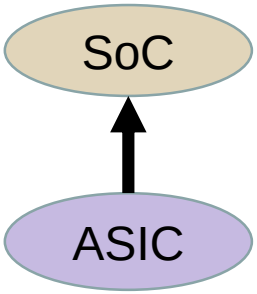


  - Truth tables can be implemented via small memories (LUTs).
    - Logic Input(s): Address,
    - Logic Output(s): Content.

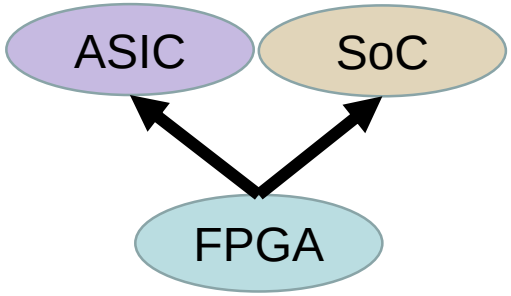


ASIC vs. SoC vs. FPGA

	ASIC	SoC	FPGA
Performance	Very high (Optimized)	High	Fine (Not optimized)
Energy Efficiency	Very high (Optimized)	High	Fine (Not optimized)
Price	Most expensive	Expensive	Least expensive
Reconfigurability	No	No	Yes
Purpose	**Specific Application**	**General Purpose**	**Rapid Prototyping**

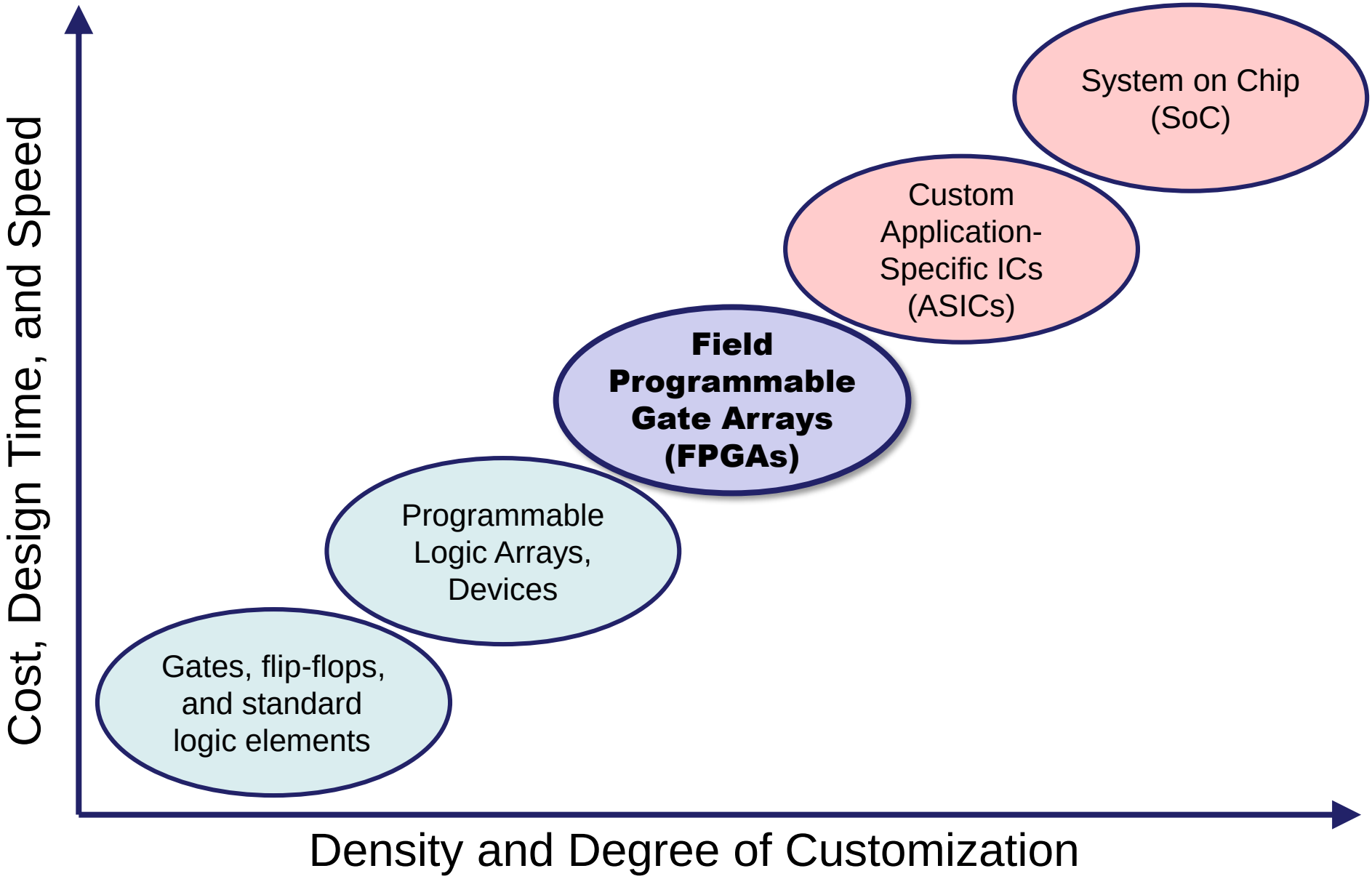


- Reuse in design

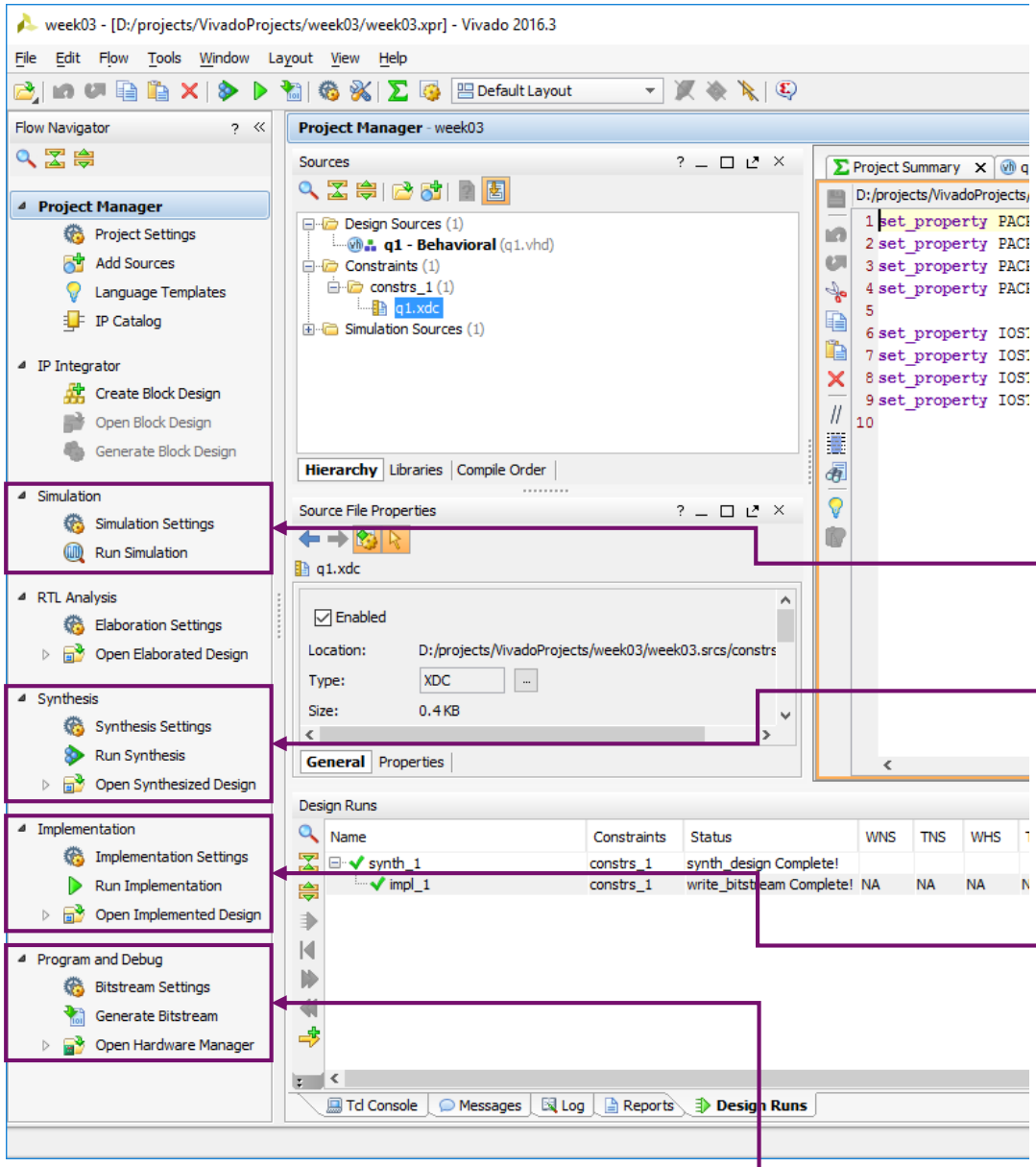


- Prototype
- Test and debug
- A proof of concept

Spectrum of Design Technologies



Design Flow on Vivado



Requirements

↓

Design Specifications

↓

Design Formulation

↓

Design Entry

Hand-drawn schematics, VHDL, Verilog

↓

Behavioral Simulation

↓

Logic Synthesis

↓

Post Synthesis Simulation

↓

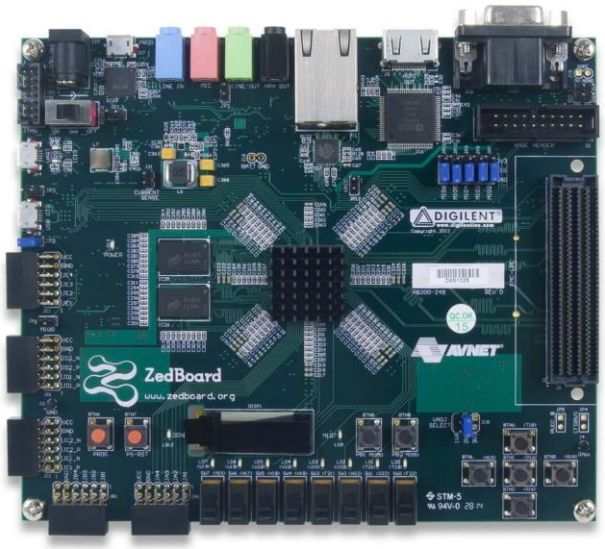
Mapping, Placement, Routing

↓

ASIC / SoC | FPGA

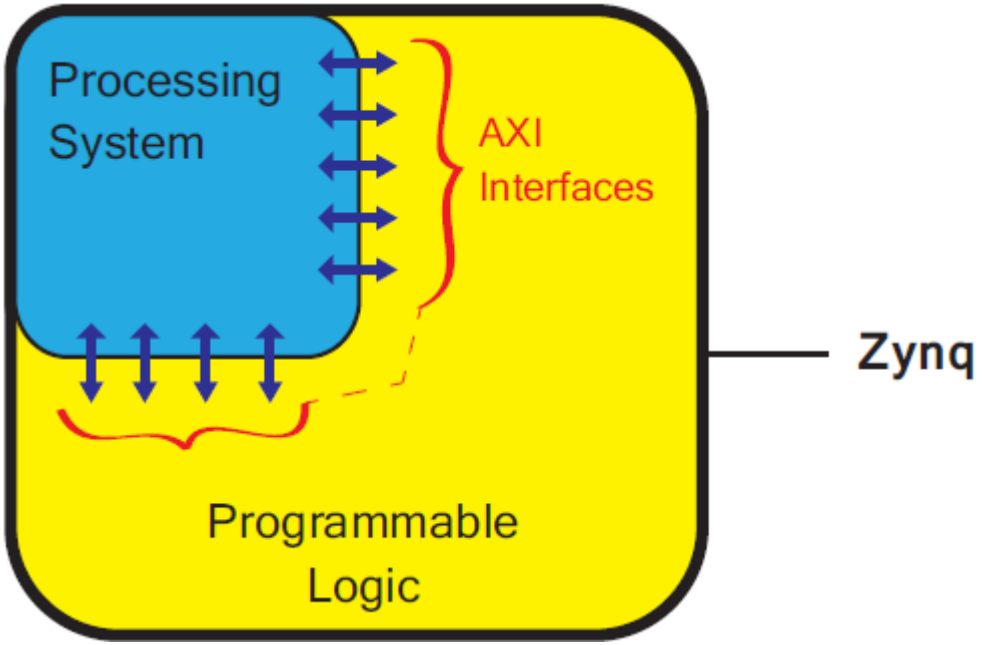
Outline

- Design Flow of Digital Systems
  - ① Requirements, Design Spec., and Design Formulation
  - ② Design Entry
  - ③ Behavioral Simulation
  - ④ Logic Synthesis
  - ⑤ Post-Synthesis Simulation
  - ⑥ Mapping, Placement, Routing
  - ⑦ Design Target
- Our Design Target: **Zynq ZedBoard**
  - ZedBoard Layout and Interfaces
  - Specifying ZedBoard in Vivado
  - Hardware Setup for ZedBoard
  - Programming the ZedBoard
  - Xilinx Design Constraints (XDC) File



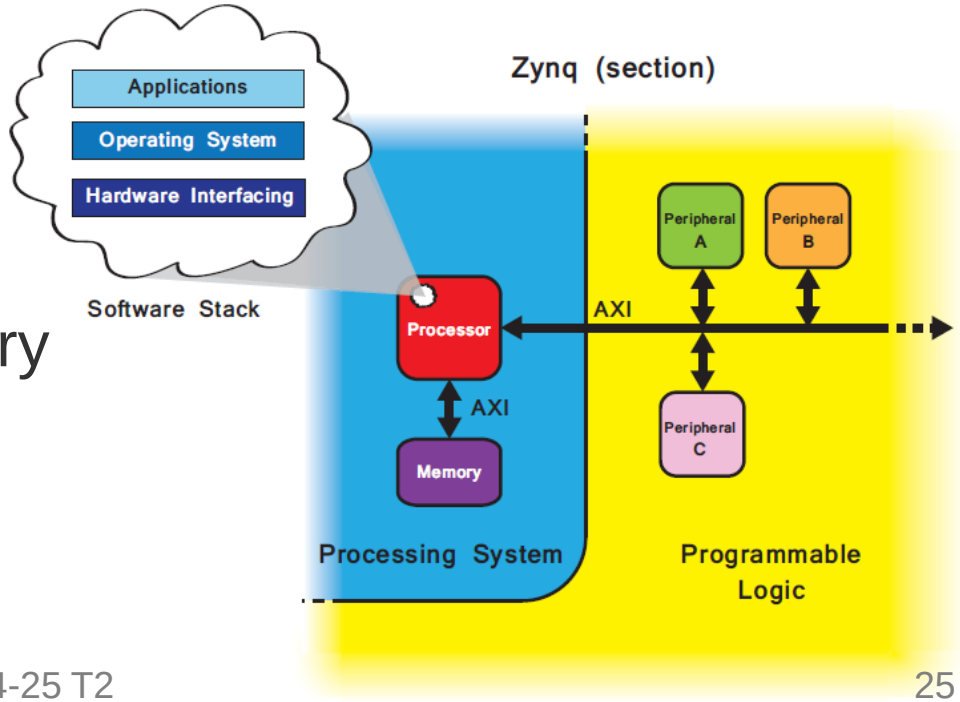
Zynq: All-Programmable SoC (1/2)

- All-Programmable SoC (APSoC): Zynq provides an more ideal platform for implementing flexible SoCs.
- Zynq comprises two main parts:
  - Programmable Logic (PL): equivalent to that of an FPGA
  - Processing System (PS): formed around a dual-core ARM Cortex-A9 processor



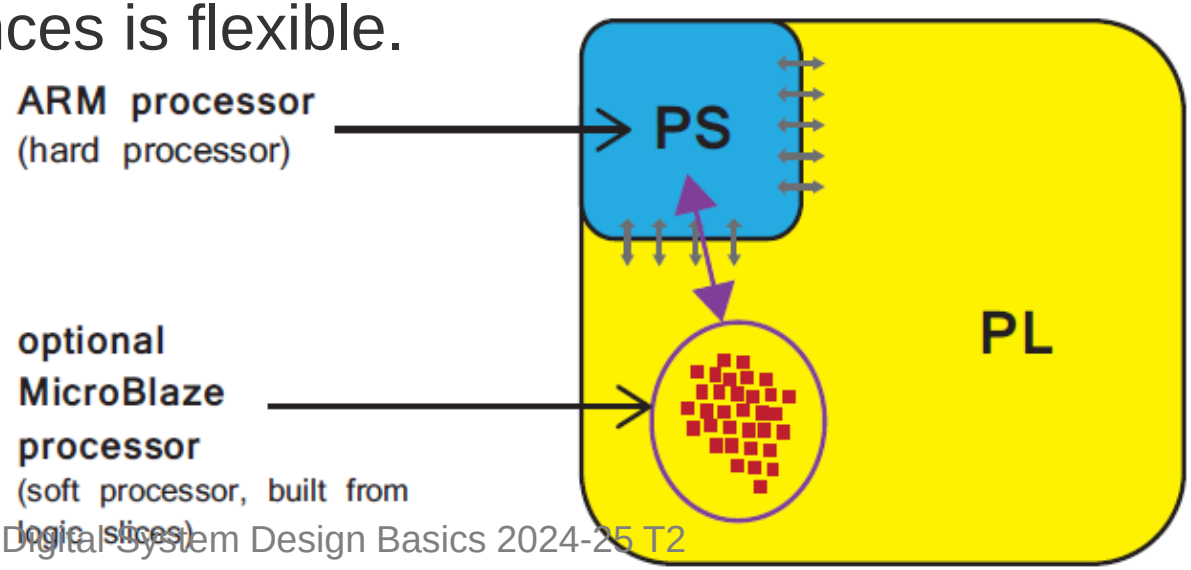
Zynq: All-Programmable SoC (2/2)

- **Programmable Logic (PL)**: Implements high-speed logic, arithmetic and data flow subsystems.
- **Processing System (PS)**: Supports software routines and/or operating systems.
  - The overall functionality of any designed system can be appropriately partitioned between hardware and software.
- **Advanced eXtensible Interface (AXI)**:
  - Links between the PL and PS are made using industry standard Advanced eXtensible Interface (AXI) connections.



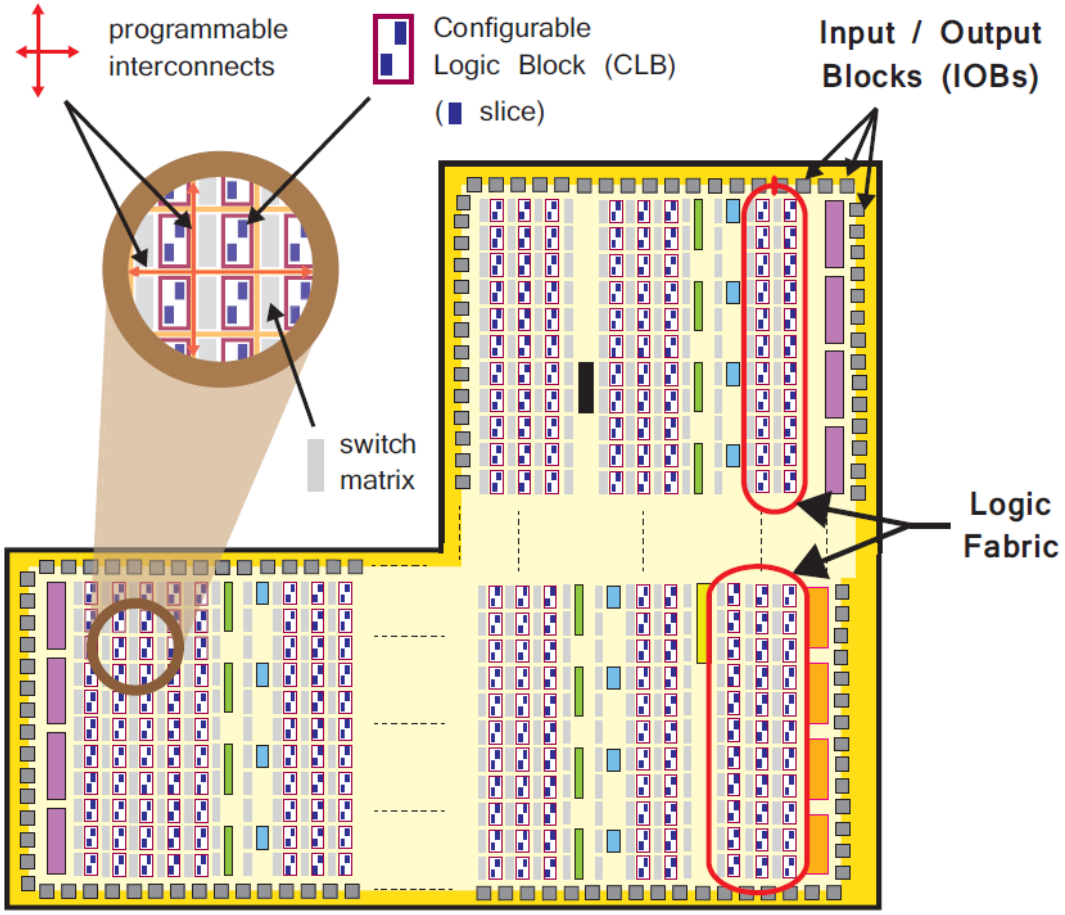
Processing System (PS)

- PS supports software routines and operating systems.
  - The overall functionality of any system can be partitioned.
- PS has a "hard" dual-core ARM Cortex-A9 processor.
  - Hard processors can achieve higher performance.
- By contrast, "soft" processor (e.g., Xilinx MicroBlaze) can be made by the programmable logic elements.
  - The number and precise implementation of soft processor instances is flexible.



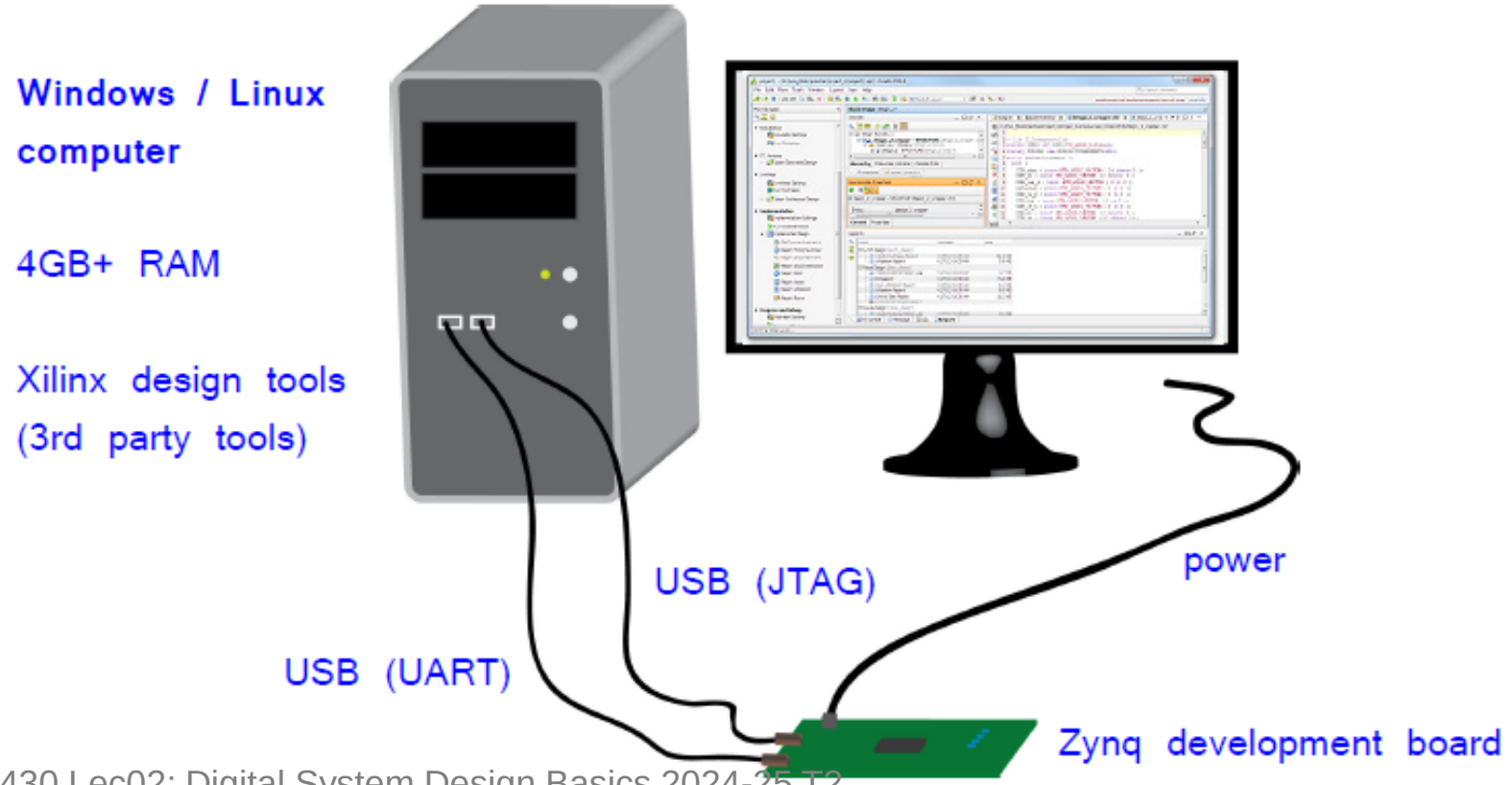
Programmable Logic (PL)

- PL section is ideal for implementing high-speed logic, arithmetic and data flow subsystems.
- PL is composed of general purpose FPGA logic fabric.



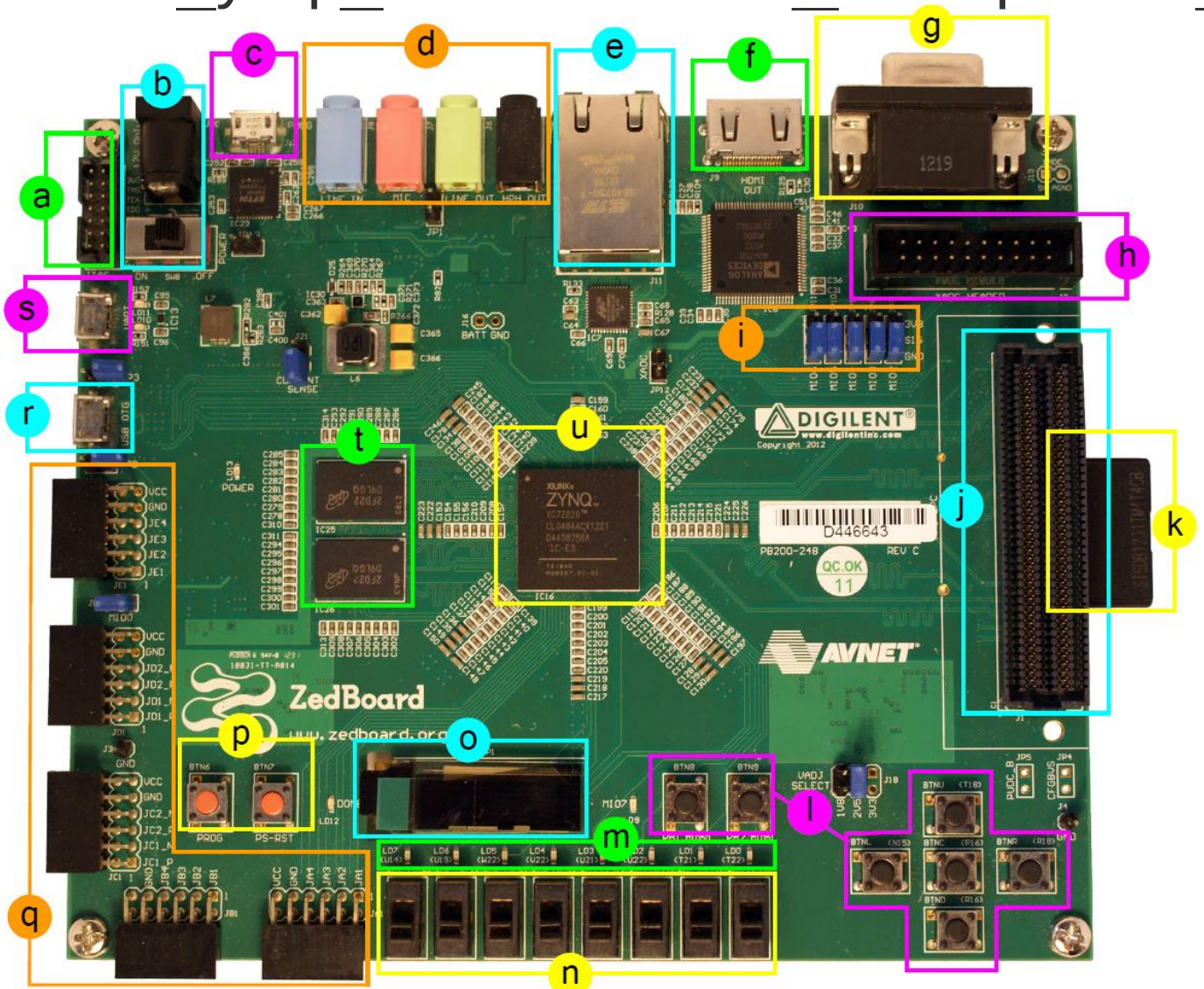
Zynq Development Setup

- Joint Test Action Group (JTAG): Downloading designs onto the development board over JTAG
- Universal Asynchronous Receiver/Transmitter (UART) and Terminal Applications: Interfacing and debugging



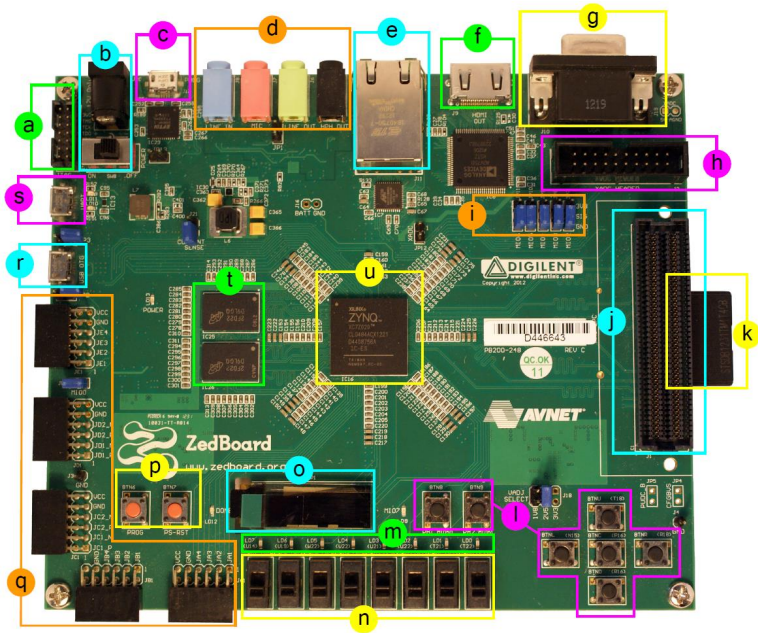
Our Board: Zynq ZedBoard

- ZedBoard: **Z**ynq **E**valuation and **D**evelopment **Board**



ZedBoard Layout and Interfaces

- ZedBoard features a ZC7Z020 Zynq device.
  - Artix-7 logic fabric, with a capacity of 13,300 logic slices, 220 DSP48E1s, and 140 BlockRAMs
  - DDR3 Memory, and Flash
  - Several peripheral interfaces



a Xilinx JTAG connector

b Power input and switch

c USB-JTAG (programming)

d Audio ports

e Ethernet port

f HDMI port (output)

g VGA port

h XADC header port

i Configuration jumpers

j FMC connector

k SD card (underside)

l User push buttons

m LEDs

n Switches

o OLED display

p Prog & reset push buttons

q 5 x Pmod connector ports

r USB-OTG peripheral port

s USB-UART port

t DDR3 memory

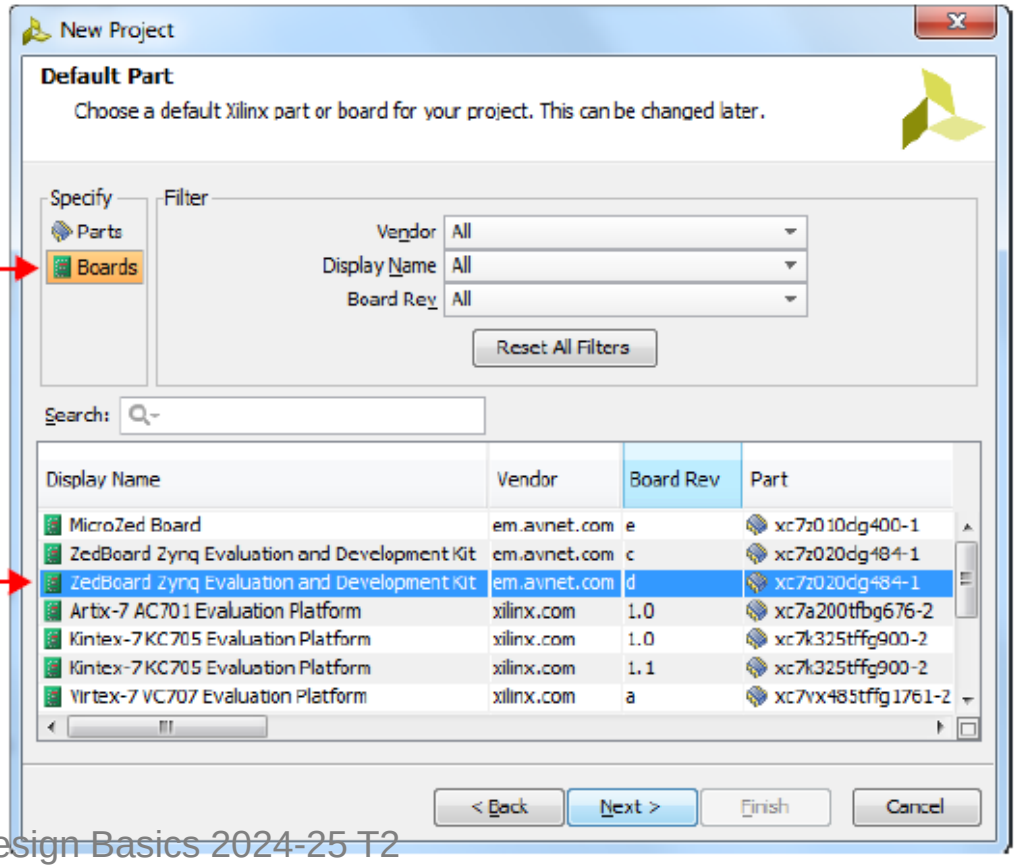
u Zynq device (+ heatsink)

Specifying ZedBoard in Vivado

- **ZedBoard Zynq Evaluation and Development Kit**:
  - The design tools have knowledge of the specific facilities and peripheral connections of ZedBoard.
- Target part: **xc7z020clg484-1**, Rev: **d**

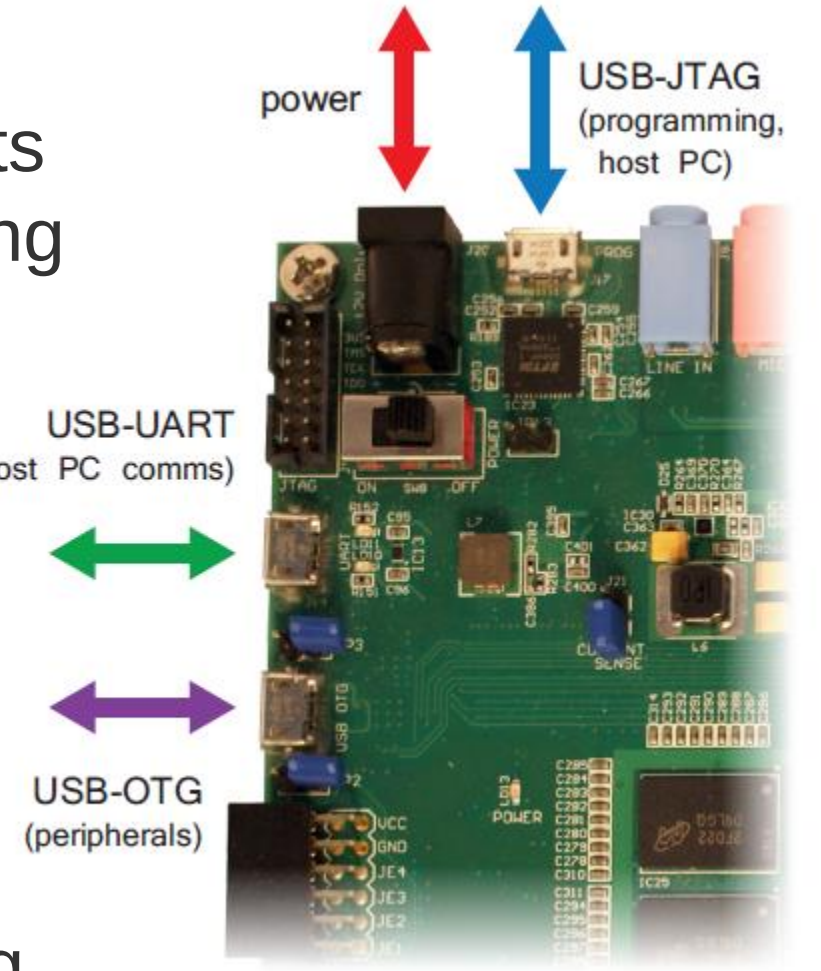
choose ***Boards*** here to see the selection...

select the ***ZedBoard*** (choose the 'Board Rev' for you board)



Hardware Setup for ZedBoard

- The ZedBoard must be connected to a power supply.
- By default, ZedBoard connects to the host PC for programming over USB-JTAG.
- An additional connection can be made over USB-UART.
  - If intending to facilitate simple board-PC communication using the Terminal application.
- Note that there is also a third micro-USB port for connecting USB peripherals (USB-OTG)



Programming the ZedBoard (1/2)

- ZedBoard can be programmed in four different ways:
  - **USB-JTAG:** This is the default and most straightforward method of programming the ZedBoard, given that it can be done directly over the USB-micro-USB cable supplied in the ZedBoard kit.
  - **Traditional JTAG:** A Xilinx JTAG connector is available on the board and may be used in place of the USB-JTAG connection, if desired. This will require a different type of cable or a *Digilent USB-JTAG programming cable*.
  - **Quad-SPI flash memory:** The non-volatile flash memory on the board can be used to store configuration data which persists when the board is powered off. Using this method removes the requirement for a wired connection to program the Zynq device.
  - **SD card:** There is an SD slot on the underside of the ZedBoard. This facility can be used to program the Zynq with files stored on the SD card, thus requiring no wired connections.

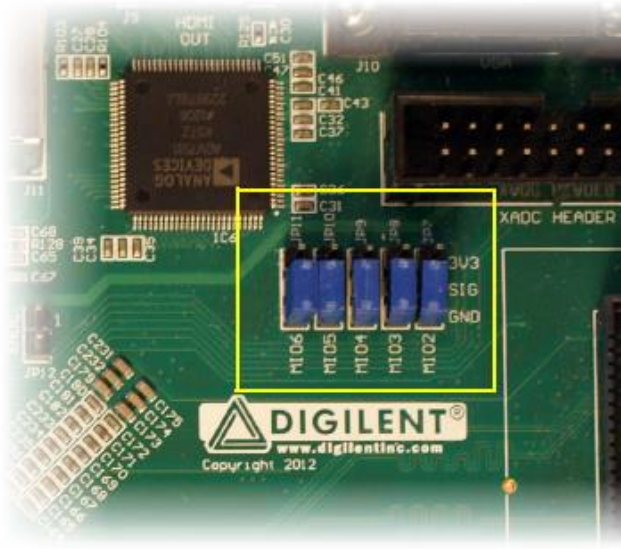
Zedboard Programming Guide in SDK: https://reference.digilentinc.com/learn/programmable-logic/tutorials/zedboard-programming-guide/start

Programming the ZedBoard (2/2)

- The ZedBoard user specifies the method of booting / programming via a set of jumper pins.
  - The middle three are for specifying programming source.

	MIO[6]	MIO[5]	MIO[4]	MIO[3]	MIO[2]
In Xilinx Technical Reference Manual...	Boot_ Mode[4]	Boot_ Mode[0]	Boot_ Mode[2]	Boot_ Mode[1]	Boot_ Mode[3]
JTAG Mode					
Cascaded JTAG[a]	-	-	-	-	0
Independent JTAG	-	-	-	-	1
Boot Device					
JTAG	-	0	0	0	-
Quad-SPI (flash)	-	1	0	0	-
SD Card[a]	-	1	1	0	-
PLL Mode					
PLL Used[a]	0	-	-	-	-
PLL Bypassed	1	-	-	-	-

Cascaded: A single JTAG connection is used to interface to the debug access ports in both the PS and PL.

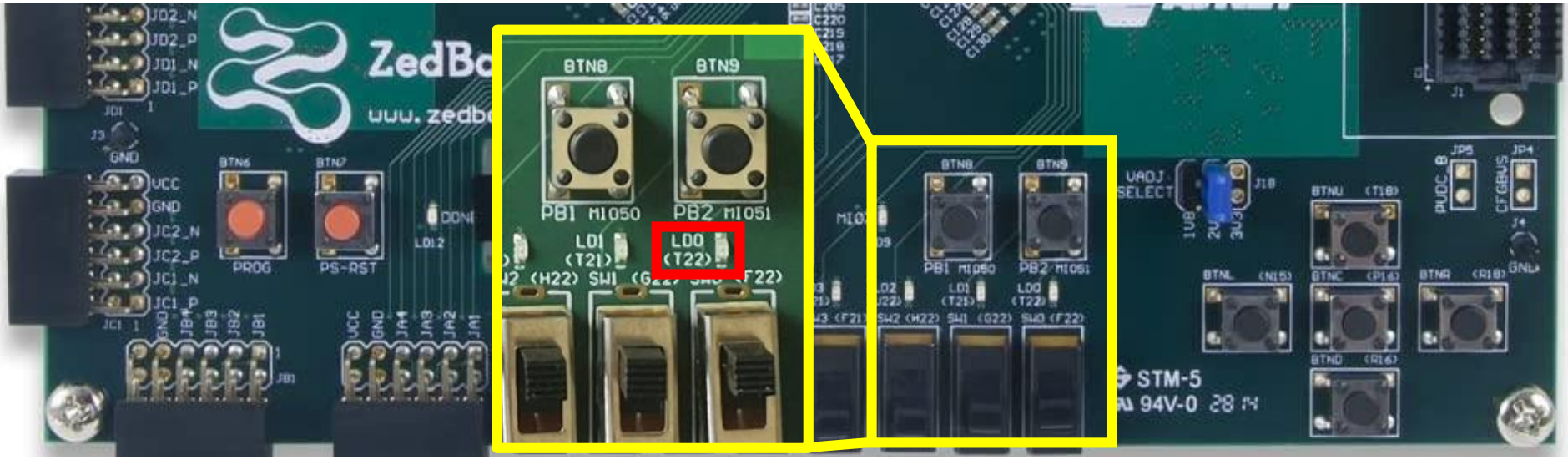


The *PLL* mode determines whether the process of configuring the device includes a phase of waiting for the PLL to lock.

Xilinx Design Constraints (XDC) (1/3)

- XDC: A file that plays the key to map external I/Os of the design to physical pins on the ZedBoard.
  - External interfaces examples: switches, LEDs
  - For example: **`C <= A and B;`**

```
set_property PACKAGE_PIN T22 [get_ports {C}]; # "LD0"
set_property PACKAGE_PIN F22 [get_ports {A}]; # "SW0"
set_property PACKAGE_PIN G22 [get_ports {B}]; # "SW1"
```



Xilinx Design Constraints (XDC) (2/3)

- Voltage levels also need to be specified in XDC:
  - `set_property IOSTANDARD LVCMOS33 [get_ports C]; # "LD0"`
  - `set_property IOSTANDARD LVCMOS25 [get_ports A]; # "SW0"`
  - `set_property IOSTANDARD LVCMOS25 [get_ports B]; # "SW1"`

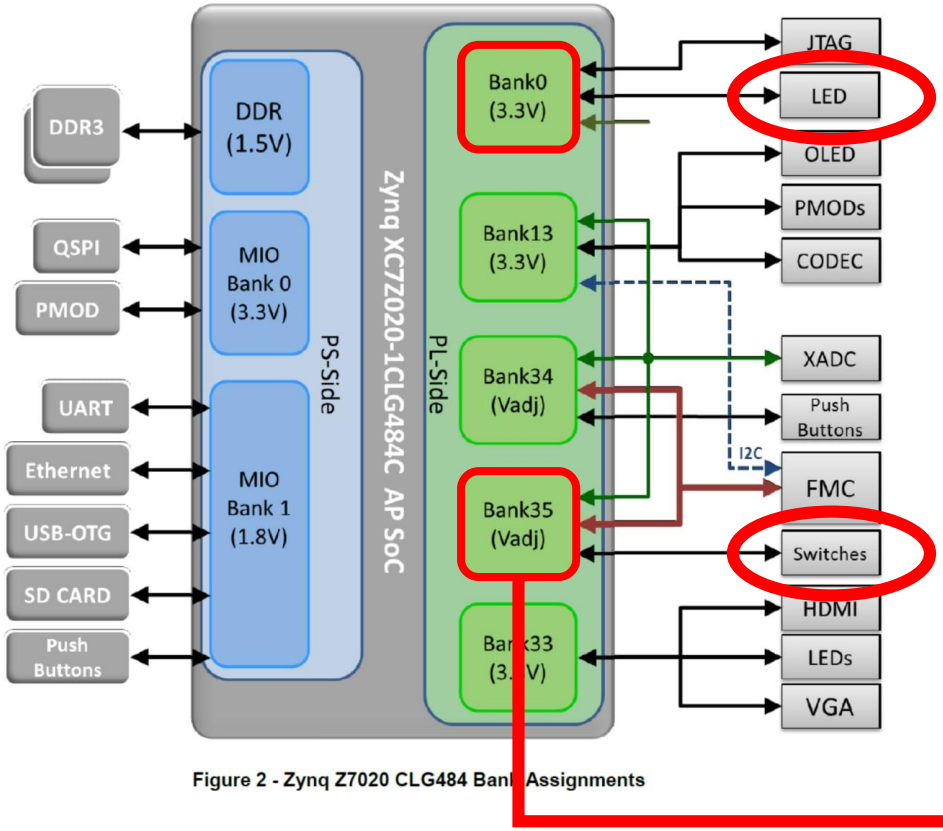
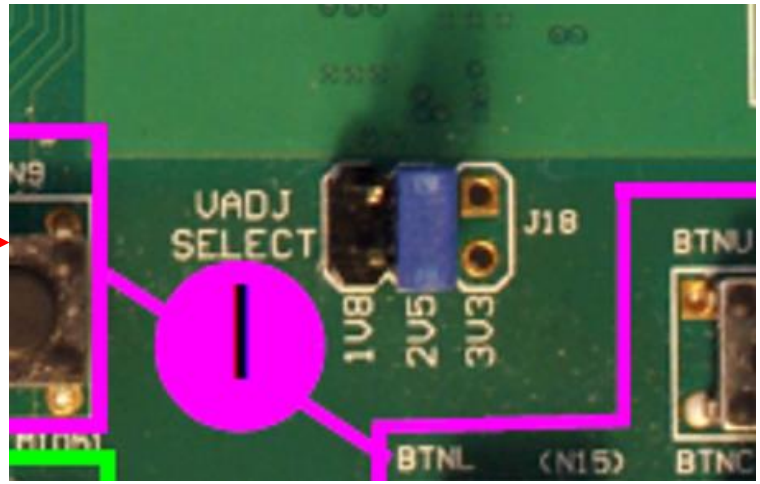


Figure 2 - Zynq Z7020 CLG484 Bank Assignments

The onboard jumper controls the **Vadj voltage** (see P.27 for the position).



Xilinx Design Constraints (XDC) (3/3)

- Together the .xdc file would be:

```
set_property PACKAGE_PIN T22 [get_ports {C}]; # "LD0"
set_property PACKAGE_PIN F22 [get_ports {A}]; # "SW0"
set_property PACKAGE_PIN G22 [get_ports {B}]; # "SW1"

set_property IOSTANDARD LVCMOS33 [get_ports C]; # "LD0"
set_property IOSTANDARD LVCMOS25 [get_ports A]; # "SW0"
set_property IOSTANDARD LVCMOS25 [get_ports B]; # "SW1"
```

Ready to Program

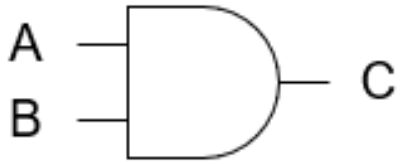
- You are ready to program the Zedboard with both the **.vhd** file and the **.xdc** file.

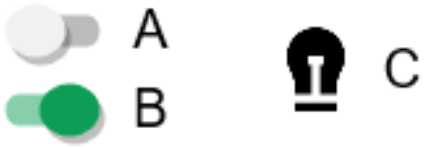
```
entity AND2x1 is
    port(A, B: in STD_LOGIC;
         C: out STD_LOGIC);
end AND2x1;

architecture Behavioral of
AND2x1 is
begin
    C <= A and B;
end Behavioral;
```

.vhd

```
set_property PACKAGE_PIN T22 [get_ports {C}];
# "LD0"
set_property PACKAGE_PIN F22 [get_ports {A}];
# "SW0"
set_property PACKAGE_PIN G22 [get_ports {B}];
# "SW1"

set_property IOSTANDARD LVCMOS33 [get_ports C];
# "LD0"
set_property IOSTANDARD LVCMOS25 [get_ports A];
# "SW0"
set_property IOSTANDARD LVCMOS25 [get_ports B];
# "SW1"
```

.xdc



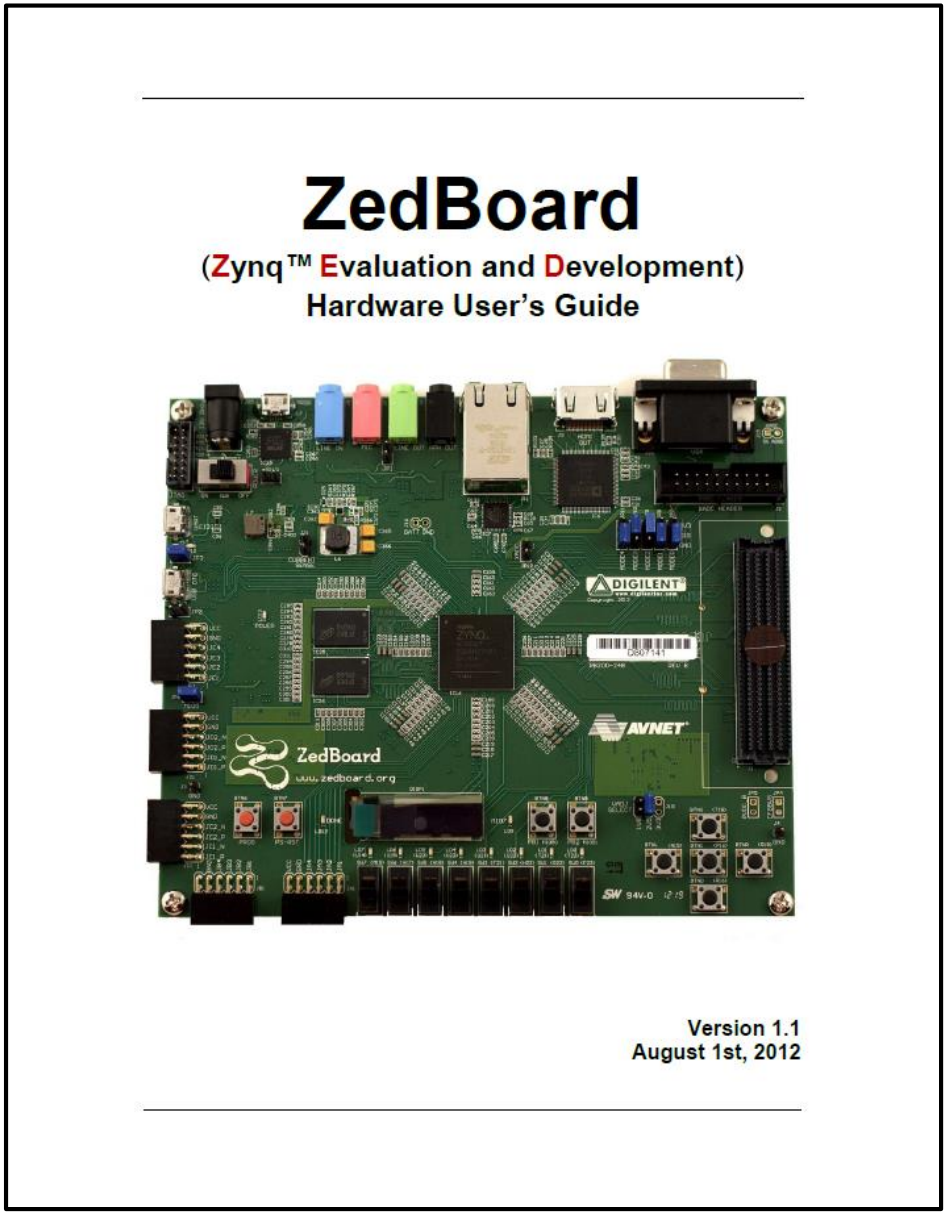


Class Exercise 2.2

- Complete the XDC file for the design **`C <= A and B`** with the following pin assignments on ZedBoard:
  - **`A`**: Switch #6 (SW6)
  - **`B`**: Switch #7 (SW7)
  - **`C`**: LED #7 (LD7)

Reference

- All the hardware configuration instructions could be found in ZedBoard Hardware User's Guide.



Summary

- Design Flow of Digital Systems
  - ① Requirements, Design Spec., and Design Formulation
  - ② Design Entry
  - ③ Behavioral Simulation
  - ④ Logic Synthesis
  - ⑤ Post-Synthesis Simulation
  - ⑥ Mapping, Placement, Routing
  - ⑦ Design Target
- Our Design Target: **Zynq ZedBoard**
  - ZedBoard Layout and Interfaces
  - Specifying ZedBoard in Vivado
  - Hardware Setup for ZedBoard
  - Programming the ZedBoard
  - Xilinx Design Constraints (XDC) File