Experiment No. 9

FPGA System Design using Vivado IP Integrator

OBJECTIVE: To construct the block-level digital system design using the Vivado IP integrator (IPI) and realize them on the Basys3 FPGA kit.

THEORY: This experiment is designed to introduce students to the digital design flow using Vivado IPI for Artix-7 FPGAs.

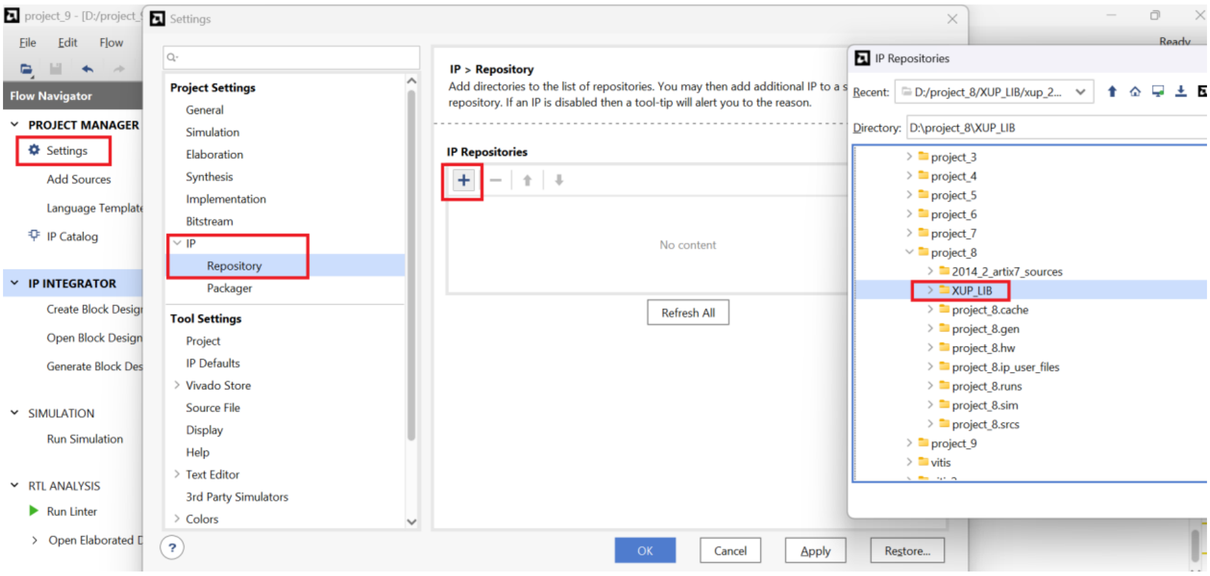
The Vivado IPI allows the creation of digital designs in schematic view through the basic functional IP blocks. The basic functional IP blocks are available on your desktop under "XUP_LIB" directory. Extract it into your working directory.

The design flow to create a digital circuit using Vivado IPI is detailed here, and summarized below for quick reference:

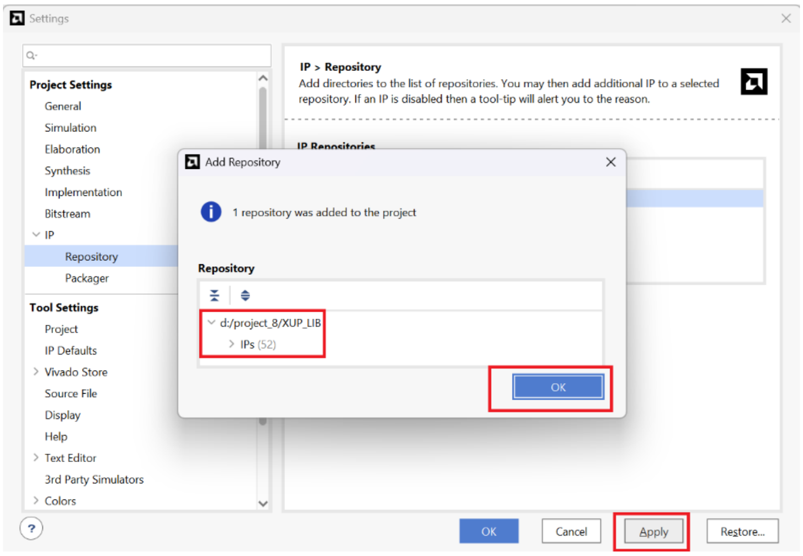
Step 1: Create a new project following the steps detailed in Exp.1. However, do not add any source files while creating the project.

Step 2: In the flow navigator, click "create block design," assign a meaningful name to your design, and click "OK."

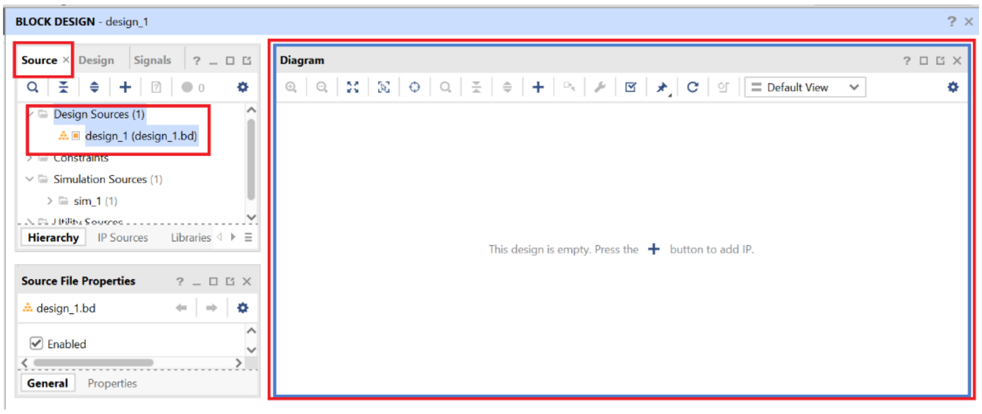
Step 3: Click "Settings" in the flow navigator, click "IP" and then "Repository", click "+", browse to the "XUP_LIB" directory, and then select it.



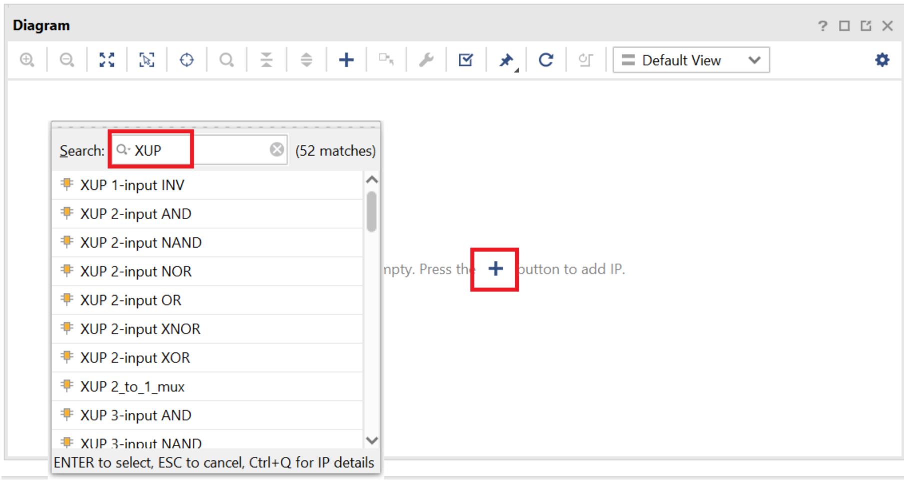
The added repository will be shown on the pop-up; check it once and then click "OK" followed by a click on "Apply" and "OK."



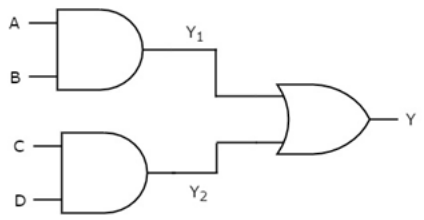
Step 4: Under the "Source" tab, expand the "Design sources" tab to see the created block design file (with .bd extension). Double-click it to open its design canvas.



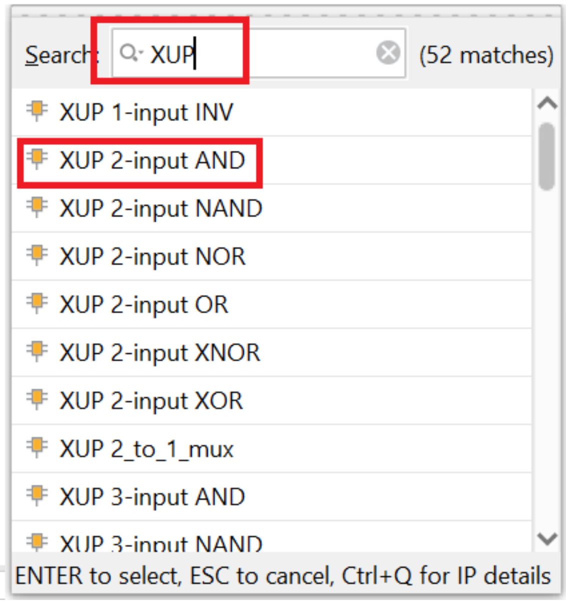
Step 5: Click "+" and type "XUP" in the search tab. You can see all the IP blocks available in the XUP directory that you can utilize for your design.

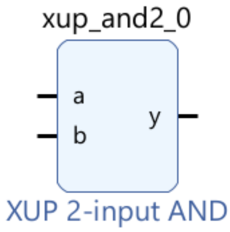


Step 6: Design a simple example shown below using the IPI:

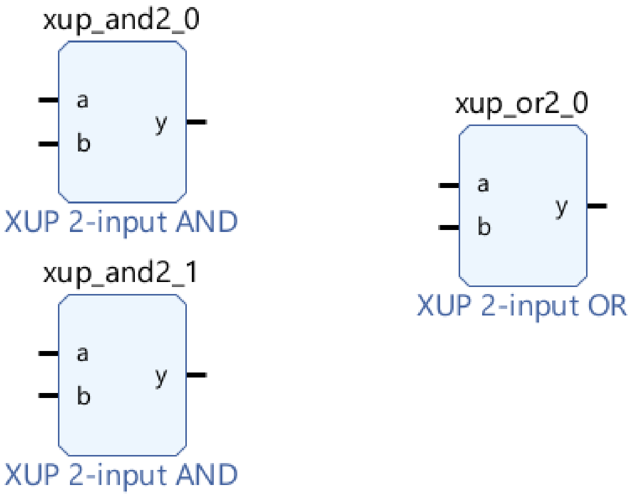


Double-click on "XUP 2-input AND" to add it to the design canvas:

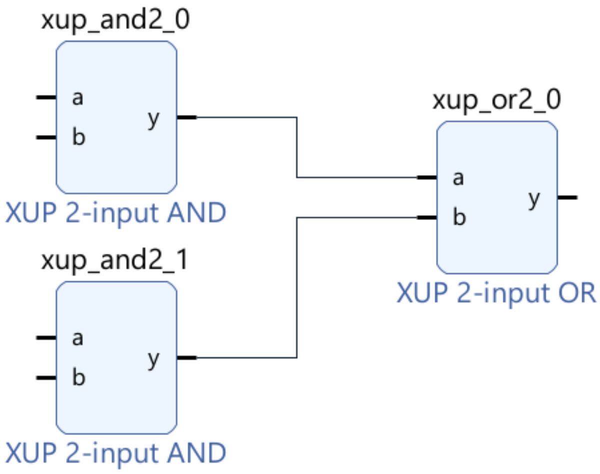




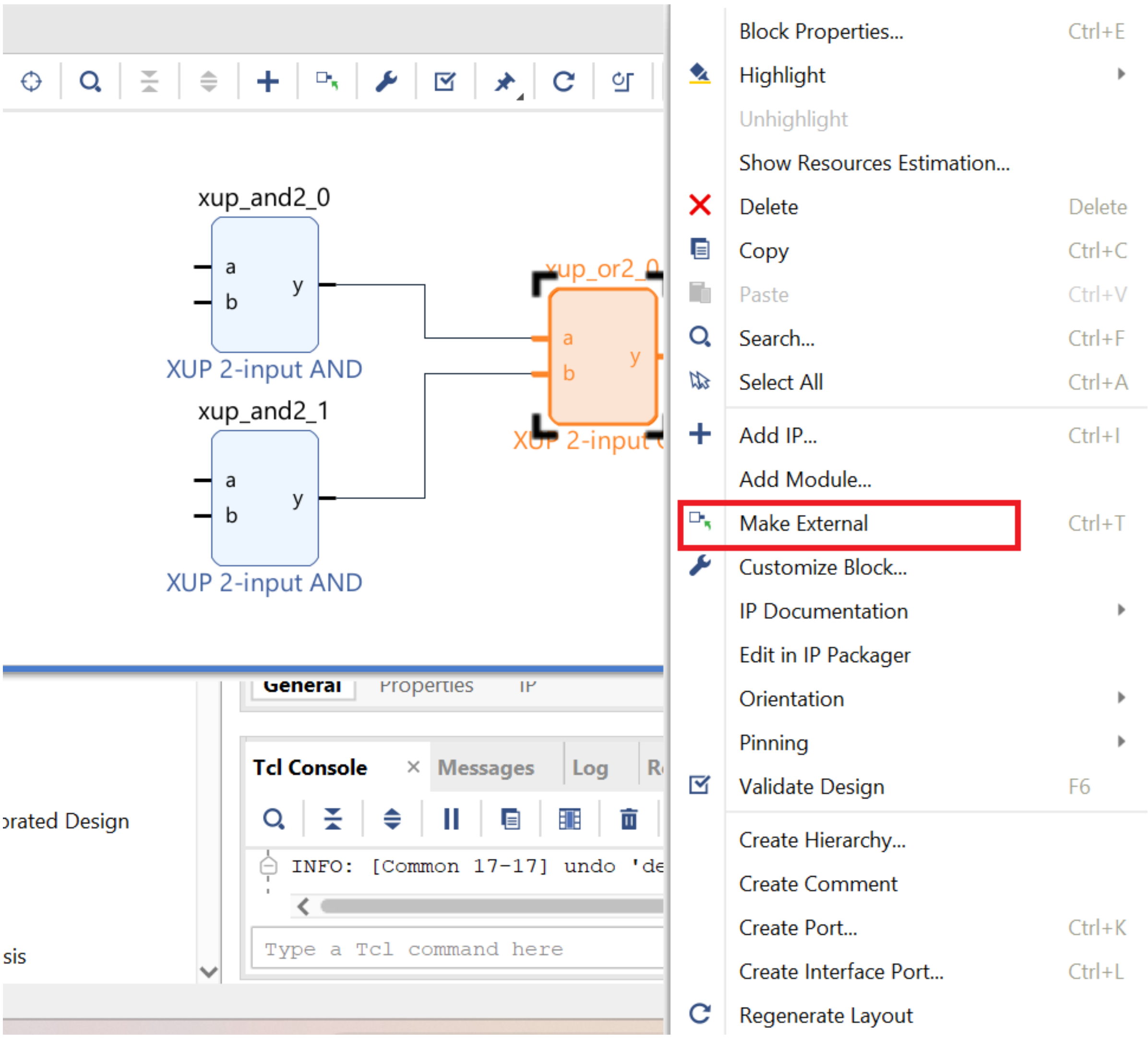
Similarly, add another instance of "XUP 2-input AND", and an instance of "XUP 2-input OR":



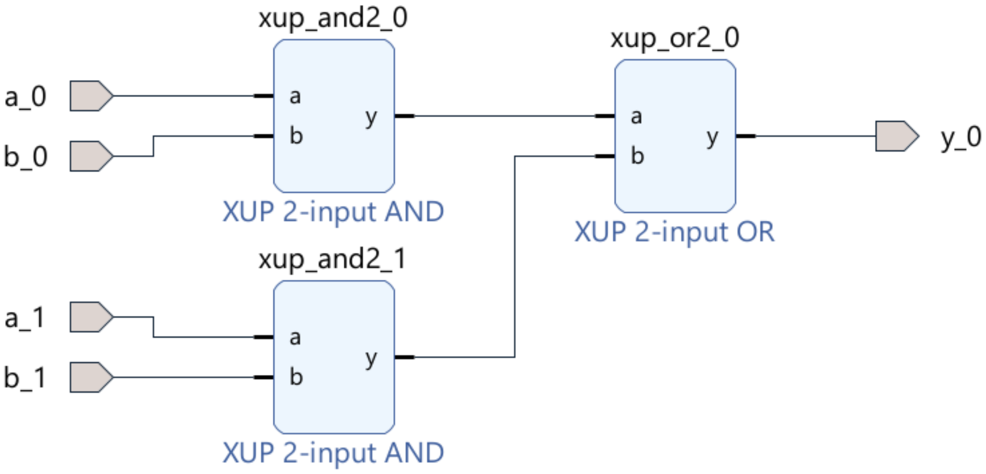
Interconnect the outputs of 2-input AND gates and the inputs of 2-input:



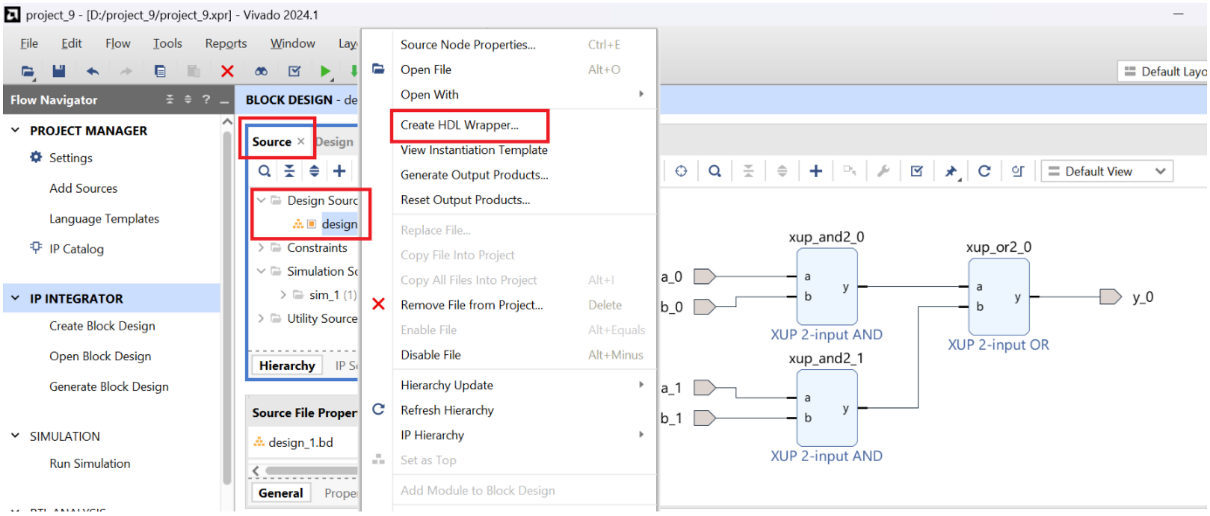
Right-click at the inputs of AND gates as well as at the output of OR gate one by one, and click "make external" to add external ports to them:



The completed design should look like this:



Step 7: Right-click on the block design(.bd) file under the "Source" tab, then click "Create HDL wrapper" and then click OK.



This creates the equivalent HDL code to the created design, and the associated files will be listed under the "Source" tab.

Step 8: Run the simulation, synthesis, and implementation following the steps detailed in Exp.7 and Exp.8.

Step 9: Generate the bitstream and download it into Basys3.

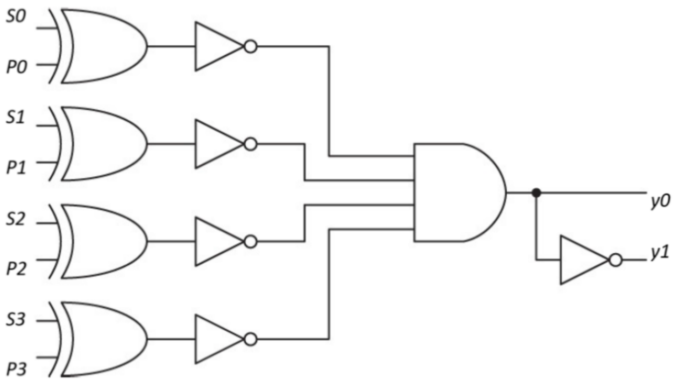
EXAMPLE 9.1:

Design a digital safe system using combinational circuits on Basys 3. Assume that the system has a four-bit predefined password. If the user input to the system matches the predefined password, turn an LED ON. Otherwise, turn another LED ON.

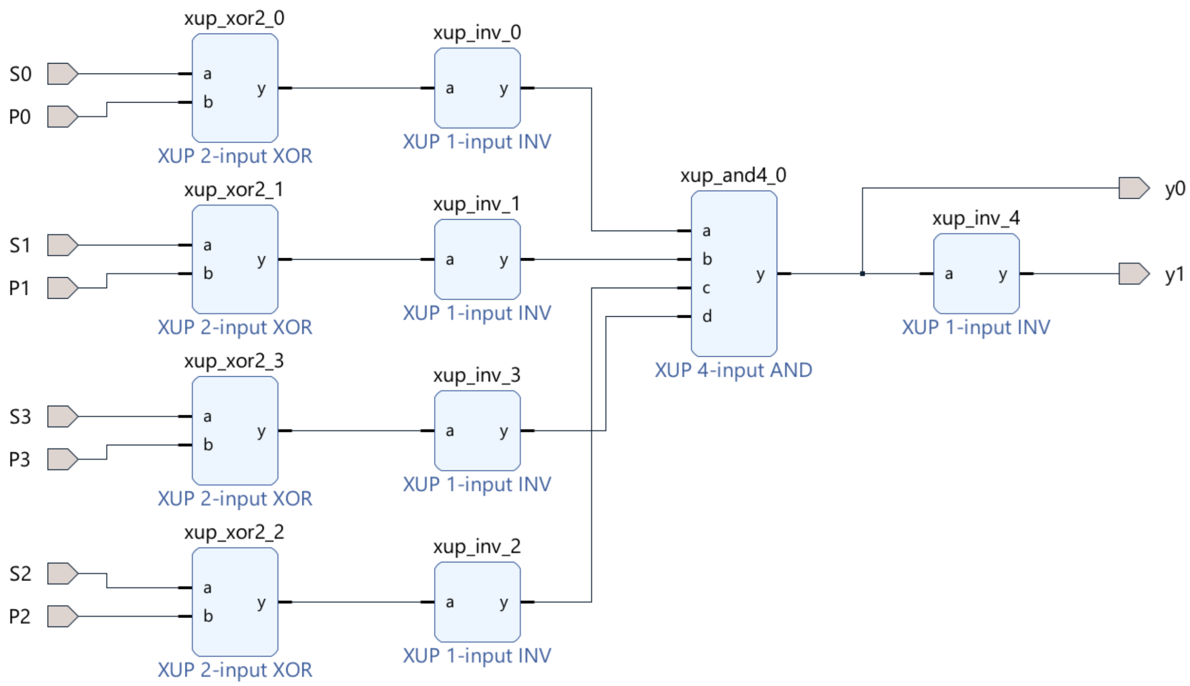
Solution:

The digital safe can be implemented using an XOR gate followed by a NOT gate for each bit to be tested. Therefore, if the input bit matches the corresponding password bit, then the XOR gate followed by NOT will give logic level 1. If all input bits match corresponding predefined password bits this way, the output will have logic level 1. Defining a second output by simply inverting the actual output can indicate the incorrect password.

Let's define S0, S1, S2, and S3 as the user inputs and P0, P1, P2, and P3 are the predefined password bits. y0 as the correct password indicator and y1 as the incorrect password indicator.



Block Design:



Simulation Results:

Input: Set P0=1, P1=0, P2=1, P3=1

Feed S0=1, S1= 100ns clock, S2=1, S3=1

Output: y0 and y1 varies based on correct and incorrect password

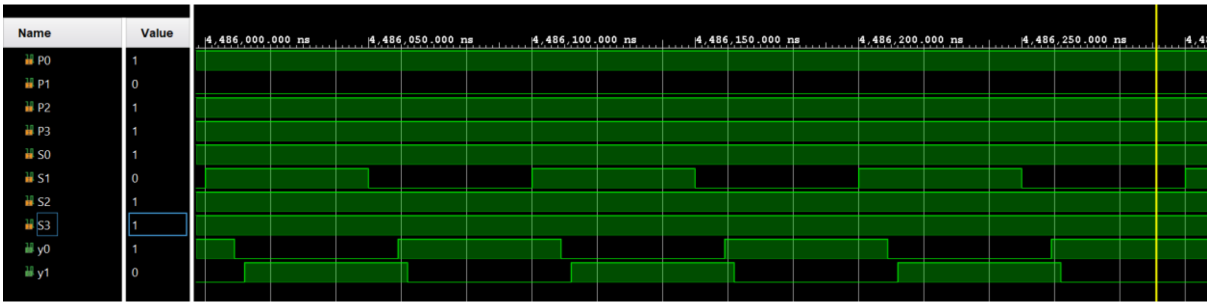


Fig. 9.1: Simulation results of example 9.1

IO pin assignment:

P0:R2	P2:U1	S0:W17	S2:V16	y0:V13
P1:T1	P3:W2	S1:W16	S3:V17	y1:V14

Hardware results:

Set password:

SW14	SW13	SW12	SW11
ON	OFF	ON	ON

Enter password:

SW3	SW2	SW1	SW0	LD8	LD7
ON	OFF	ON	ON	ON	OFF
For other combinations				OFF	ON

EXAMPLE 9.2:

Design a car park-occupied slot counting system, which counts how many slots are occupied at any given time and displays its count on a seven-segment display of Basys 3.

Solution:

Assume there is a car park with three slots, and we would like to know how many slots are occupied at a given time. Within the design, occupied slot locations are not necessary. Assume that we placed a sensor over each slot, which provides output logic level 1 when the slot is occupied. If the slot is empty, the sensor provides output logic level 0.

Let label the output of sensors as binary variables S_0, S_1, and S_2. The designed digital circuit will provide the output as a two-bit binary number C_1 (MSB) and C_0 (LSB). Therefore, we should cover all input combinations in terms of a truth table as:

Inputs			Outputs	
S0	**S1**	**S2**	**C1**	**C0**
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0

1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The Boolean equation for the above table is:

$$C_0 = \overline{S_o}.\overline{S_1}.S_2 + \overline{S_o}.S_1.\overline{S_2} + S_0.\overline{S_1}.\overline{S_2} + S_0.S_1.S_2$$

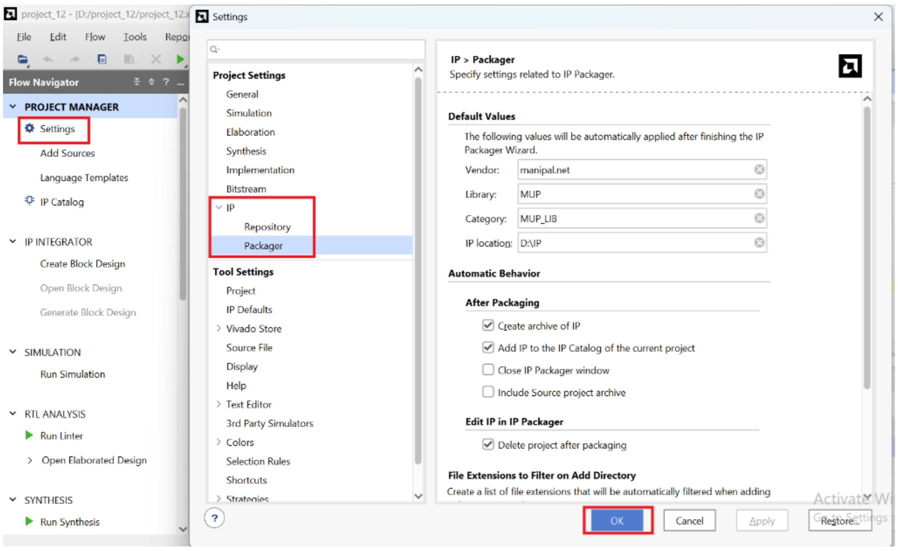
$$C_1 = \overline{S_o}.S_1.S_2 + S_0.\overline{S_1}.S_2 + S_0.S_1.\overline{S_2} + S_0.S_1.S_2$$

Now, the 2-bit binary output must be displayed on the seven-segment display. However, the Vivado in-built repositories don't have the IP for the segment-segment display HDL description. Therefore, a custom IP needs to be created using the steps below(Ref):

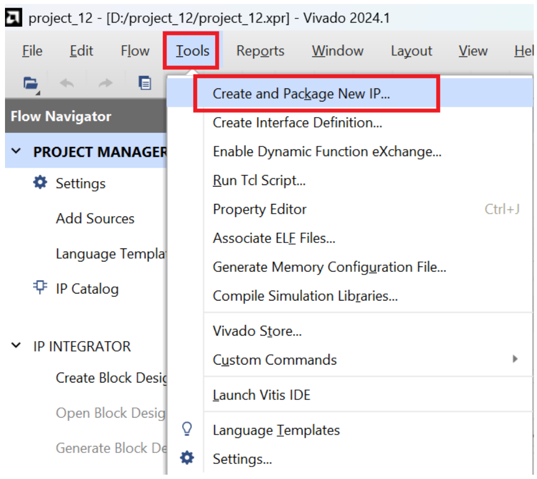
Step 1: Create a Vivado project following the steps described in Exp.1.

Step 2: Create an empty Verilog source file and name it "Seven_Seg.v". Copy the seven-segment HDL code from Example.7.1. Although it is optional, recommended to simulate it to verify the logical correctness of your HDL description.

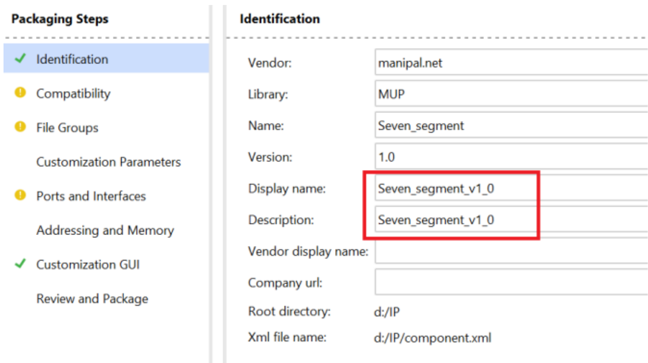
Step 3: In the flow navigator, open "settings" and then click "IP" → "packager," set the fields as below, and click "OK."



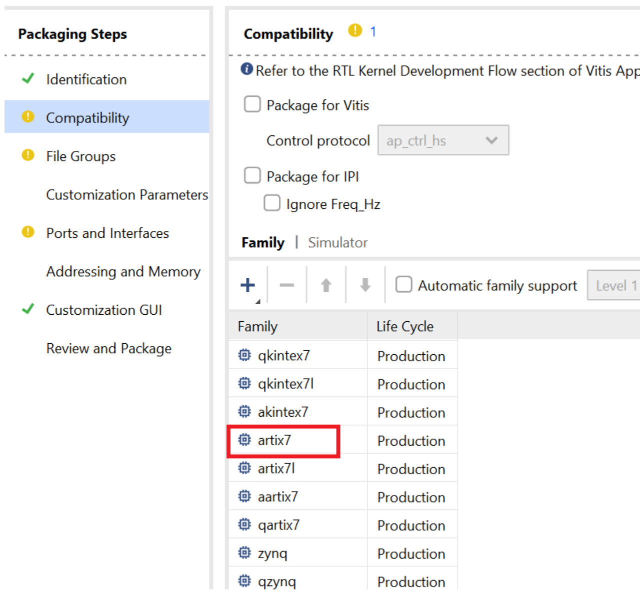
Step 4: Select "tools" and click "Create and Package New IP", click "Next", select "Package your current project", click "Next", give the IP location, click "Next", and "Finish".



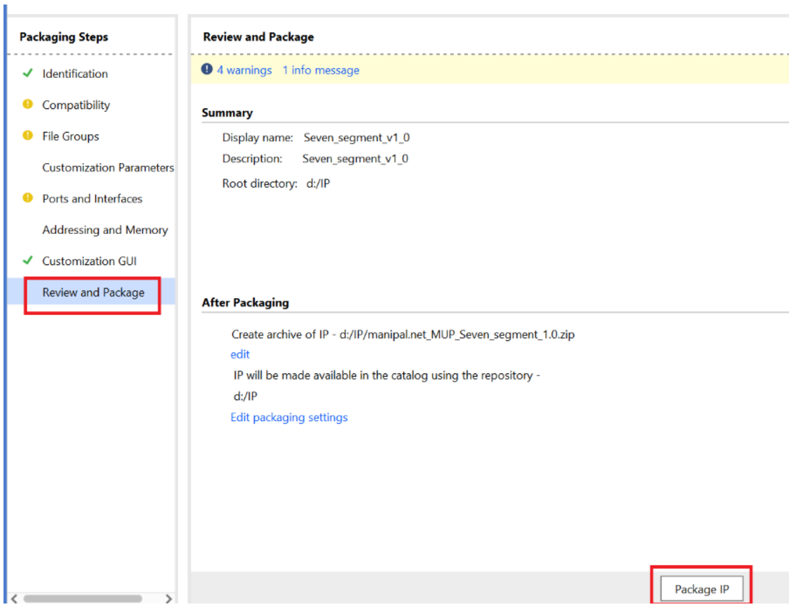
Step 5: Under “packaging steps”, select “Identification” and give a meaningful name and description to your IP:



In “compatibility,” check if the Artix-7 family is added:



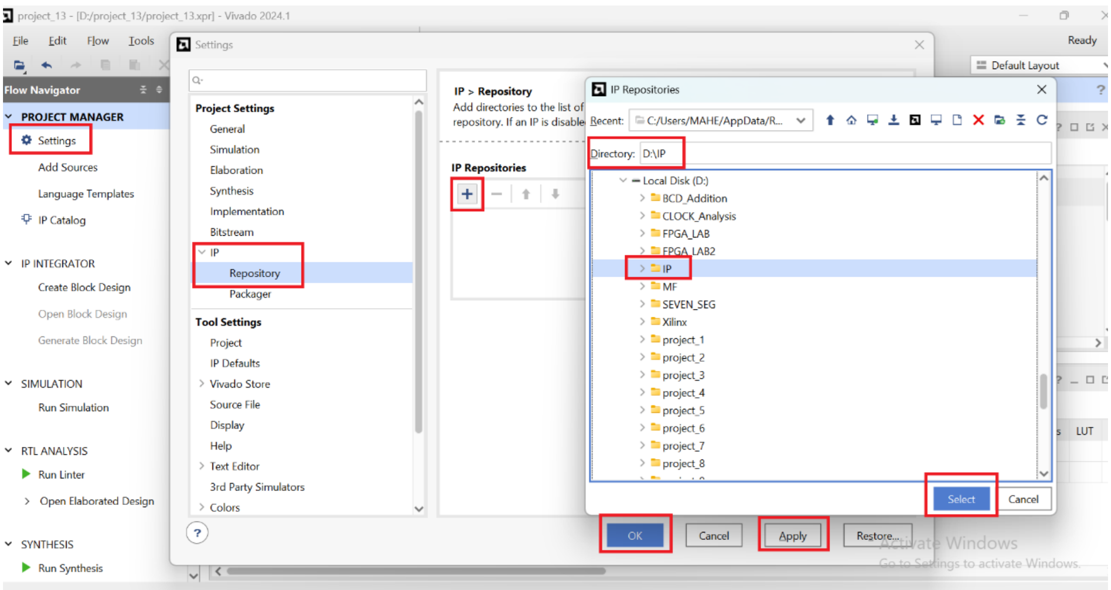
Go to “Review and Package” and click “Package IP”:



Your IP is created with the name “Seven_segment_v1_0” and saved at the given IP location.

Block design:

Create a new Vivado project, go to “settings”, click “IP” and then click “Repository”, click “+” and then browse to your “IP location” and “select” it. Click “OK”, “Apply” and “OK”.

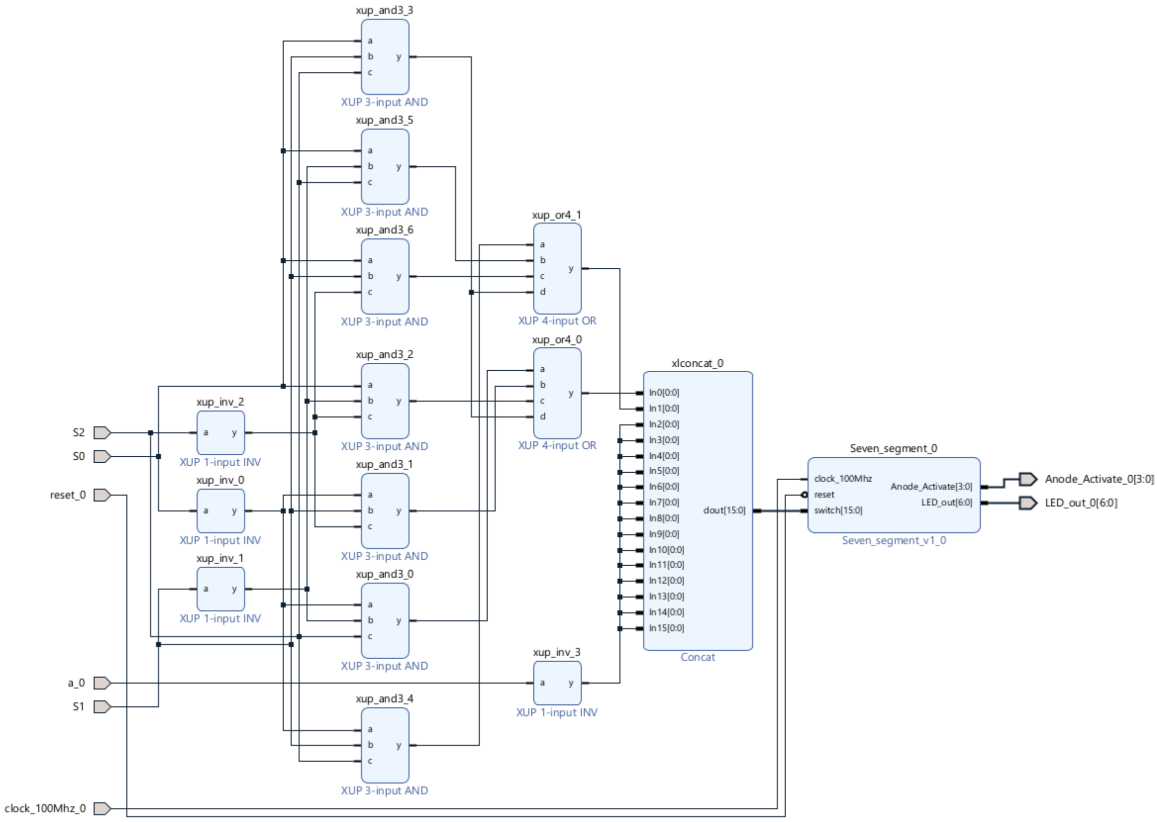


Similarly, add the “XUP_LIB” repository.

In the flow navigator, click “Create Block design” and select the “Seven_segment_v1_0” component in the search space of the design canvas. Add it to your design along with other necessary components listed in below table for the implementation of our Boolean equation:

Component	Library	Operation	Description	Quantity
xup_inv_2	XUP_LIB	NOT	For compliment operation of Boolean Equation	4
xup_and_3	XUP_LIB	AND	For product operation of Boolean Equation	7
xup_or_3	XUP_LIB	OR	For sum operation of Boolean Equation	2
xlconcat_0	in-built	Concatenate	To mask the 14 higher input bits to seven segment display IP	1
Seven_seg_v1_0	Custom	Binary to hex conversion	To receive 16-bit binary input and display the equivalent 4-digit Hexadecimal on SSD	1

Interconnect the components as shown in the schematic below, create the HDL wrapper, synthesize and implement it, and generate the bitstream to download it onto Basys 3.



IO pin assignment:

Inputs		Anode_Activate [3:0]		LED_out [6:0]			
S0	V17	Bit 3: Display 1	W4	Bit 6: Seg a	W7	Bit 2: Seg e	U5
S1	V16	Bit 2: Display 2	V4	Bit 5: Seg b	W6	Bit 1: Seg f	V5

S2	W16	Bit 1: Display 3	U4	Bit 4: Seg c	U8	Bit 0: Seg g	U7
a_0	R2	Bit 0: Display 4	U2	Bit 3: Seg d	V8		
Reset_0: T17				**Clock:** W5			

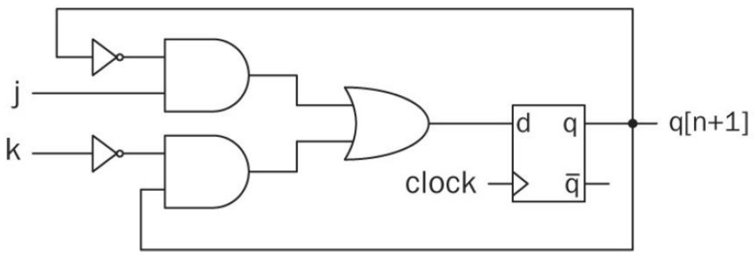
Hardware results:

SW15	SW2	SW1	SW0	Display on SSD
ON	OFF	OFF	OFF	**0**
ON	OFF	OFF	ON	**1**
ON	OFF	ON	OFF	**1**
ON	OFF	ON	ON	**2**
ON	ON	OFF	OFF	**1**
ON	ON	OFF	ON	**2**
ON	ON	ON	OFF	**2**
ON	ON	ON	ON	**3**

EXAMPLE 8.3: Design JK flipflop using D flipflop and implement it on Basys3 using Vivado IPI

Solution:

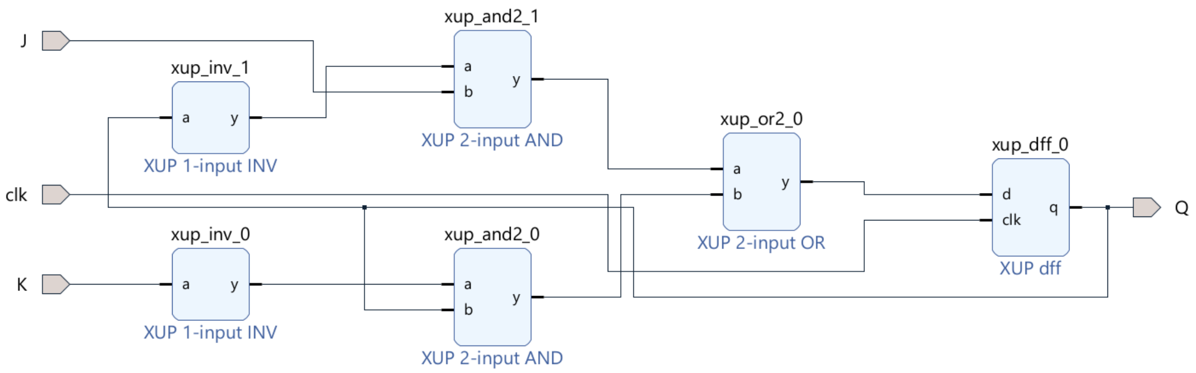
Logic diagram:



Truth table:

Inputs		Output
J	K	Q[n+1]
0	0	Q[n]
0	1	0
1	0	1
1	1	$\bar{Q}[n]$

Block design:



IO Pin assignment:

Input	**Pin No**	**Output**	**Pin No**
J	V16	Q	L1
K	V17		
clk	W5		

Hardware Results:

SW1	SW2	LD15
OFF	ON	OFF
OFF	OFF	OFF
ON	OFF	ON
OFF	OFF	ON

EXERCISE PROBLEMS:

1. Construct a block design for 2 to 4 decoder using Vivado IPI and realize it on Basys3.
2. Construct a block design for 4 to 2 priority encoder using Vivado IPI and realize it on Basys3.
3. Construct a block design for 3-bit parity generator and 3-bit parity checker using Vivado IPI and realize it on Basys3.
4. Construct a block design for 2 asynchronous counter using Vivado IPI and realize it on Basys3.
5. Construct a block design for the below state diagram using Vivado IPI and realize it on Basys3.