Lab 2: Adding a Hardware IP

National Chiao Tung University
Chun-Jen Tsai
3/23/2018

Lab Description

- In this lab, you will have to do two things:
  - Port `find_face.c` to a SW platform with FreeRTOS kernel
  - Add a HW IP module into Zynq 7020 to "accelerate" the performance of the find_face application
- For HW IP design, you must modify the template RTL code generated by Vivado to implement the inner-loop of the `compute_sad()` function
- Give your TA a demo on 4/11 (Wednesday night)

The Target of HW “Acceleration”

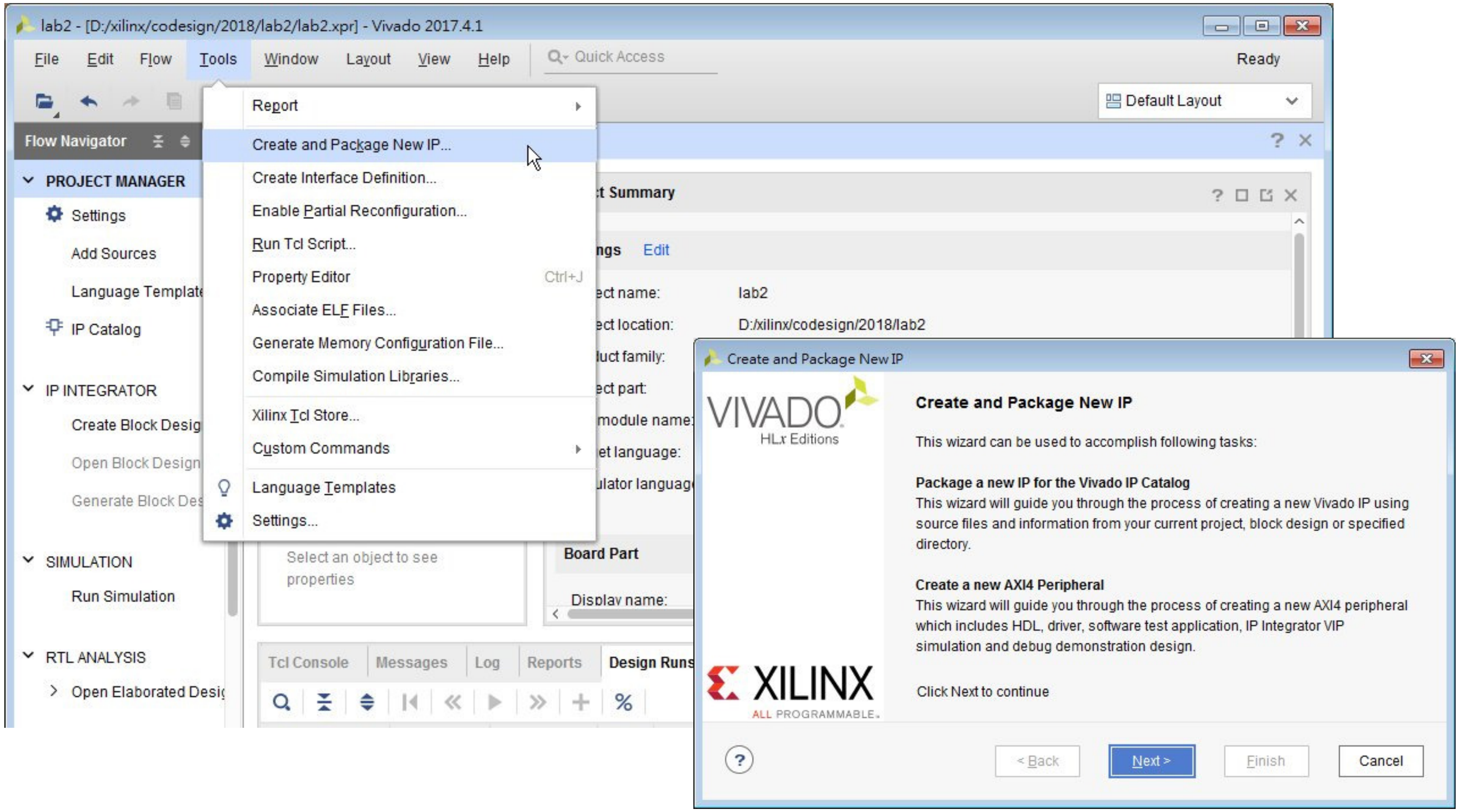
- We will try to accelerate the `compute_sad()` function:

```
int32 compute_sad(uint8 *image1, int w1, uint8 *image2, int w2, int h2,
                  int row, int col)
{
    int   x, y;
    int32 sad = 0;

    for (y = 0; y < h2; y++)
    {
        for (x = 0; x < w2; x++)
        {
            /* compute the sum of absolute difference */
            sad += abs(image2[y*w2+x] - image1[(row+y)*w1+(col+x)]);
        }
    }
    return sad;
}
```

Convert this section of code to an HW IP!

How to Create an IP on ZedBoard?

- Create an empty Zynq platform using Vivado, then select “Create and Package New IP” menu:



Create and Package New IP

This wizard can be used to accomplish following tasks:

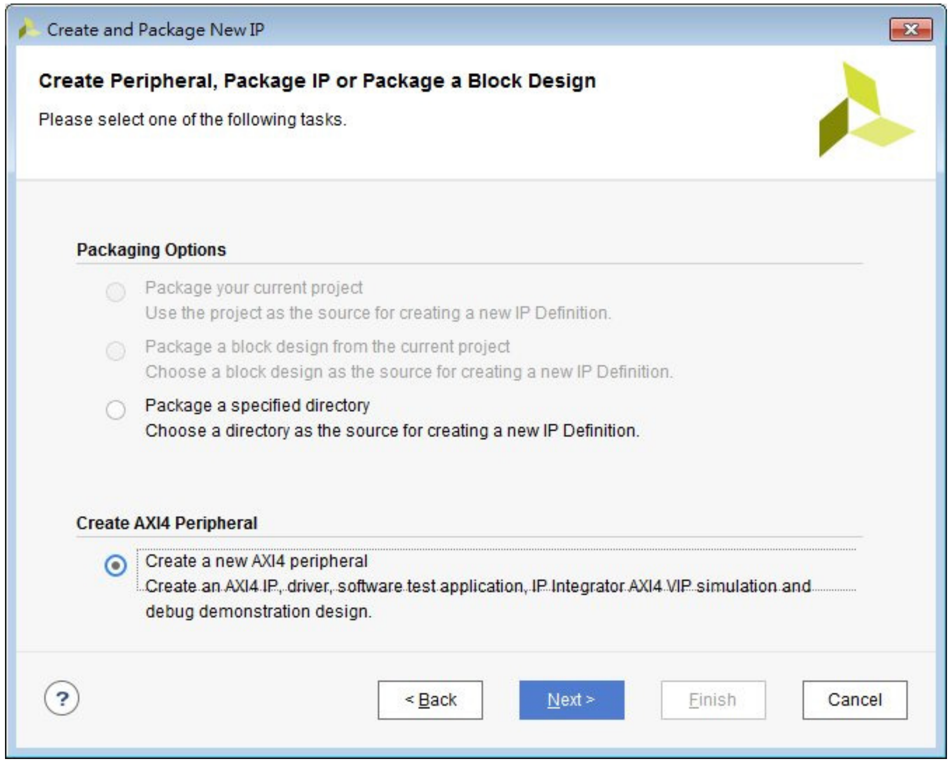
Package a new IP for the Vivado IP Catalog
This wizard will guide you through the process of creating a new Vivado IP using source files and information from your current project, block design or specified directory.

Create a new AXI4 Peripheral
This wizard will guide you through the process of creating a new AXI4 peripheral which includes HDL, driver, software test application, IP Integrator VIP simulation and debug demonstration design.

Click Next to continue

Select IP Type and Name Your IP

- Enter the IP Name, version number, and description:


Create and Package New IP

Create Peripheral, Package IP or Package a Block Design

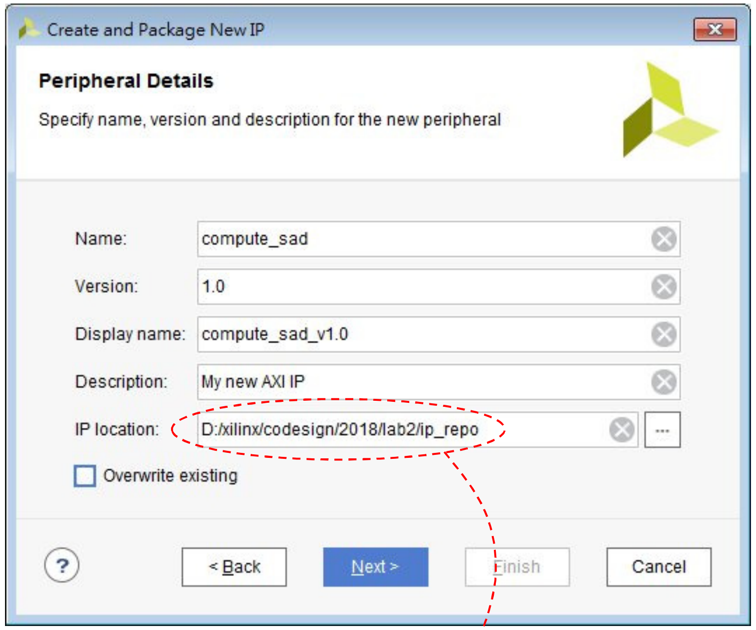
Please select one of the following tasks.

Packaging Options

- ○ Package your current project
  Use the project as the source for creating a new IP Definition.
- ○ Package a block design from the current project
  Choose a block design as the source for creating a new IP Definition.
- ○ Package a specified directory
  Choose a directory as the source for creating a new IP Definition.

Create AXI4 Peripheral

- [✓ Create a new AXI4 peripheral]
  Create an AXI4 IP, driver, software test application, IP Integrator AXI4 VIP simulation and debug demonstration design.


Create and Package New IP

Peripheral Details

Specify name, version and description for the new peripheral

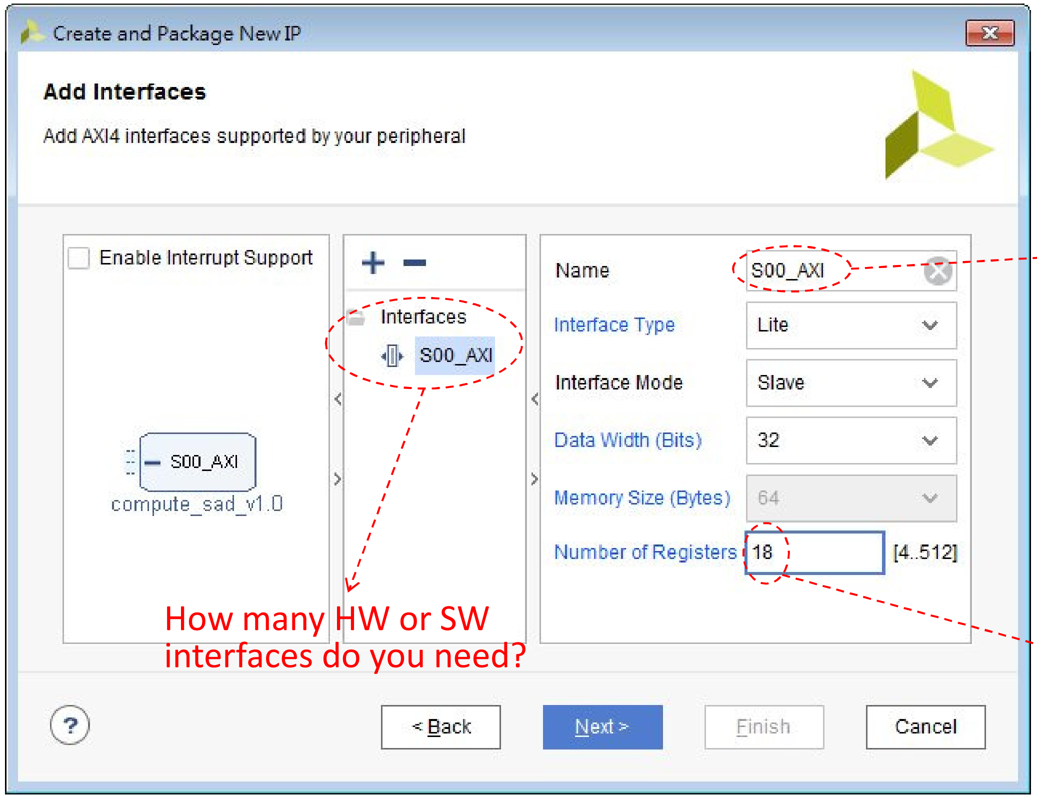
Name:	compute_sad
Version:	1.0
Display name:	compute_sad_v1.0
Description:	My new AXI IP
IP location:	D:/xilinx/codesign/2018/lab2/ip_repo

☐ Overwrite existing

Change the location of the "IP repository" so that it will locate inside your project directory. Note that the default location is in a shared location!

Configure Your IP

- You must determine the bus type, the slave/master mode, and the number of I/O registers of your IP



Create and Package New IP

Add Interfaces

Add AXI4 interfaces supported by your peripheral

Enable Interrupt Support

Interfaces
S00_AXI

Name	S00_AXI
Interface Type	Lite
Interface Mode	Slave
Data Width (Bits)	32
Memory Size (Bytes)	64
Number of Registers	18 [4..512]

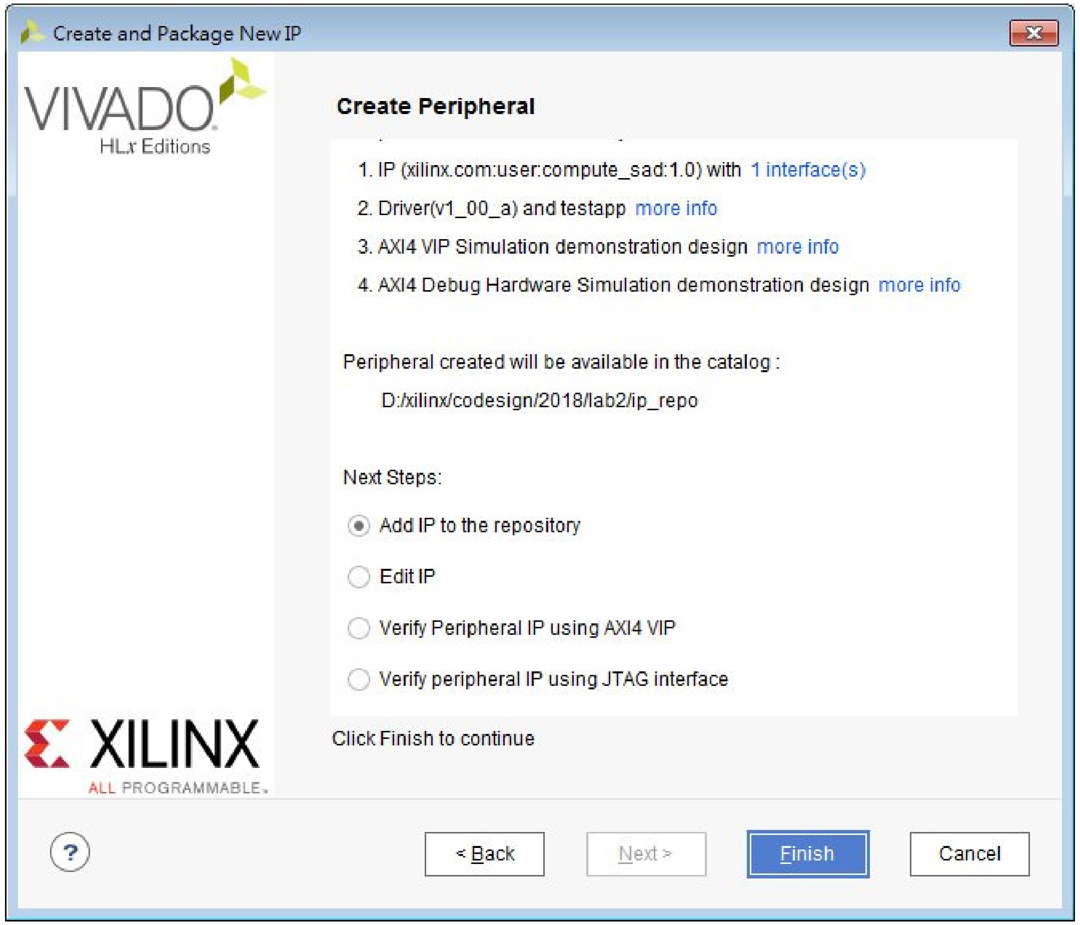
AXI4 bus type supported: Lite, Full, and Stream

How many HW or SW interfaces do you need?

How many registers do you need for the HW-SW cooperation?

Add the new IP to the IP Catalog

- Now you can add an template IP called `compute_sad` to the IP catalog so that it can be reused later



Create Peripheral

1. IP (xilinx.com:user:compute_sad:1.0) with 1 interface(s)
2. Driver(v1_00_a) and testapp more info
3. AXI4 VIP Simulation demonstration design more info
4. AXI4 Debug Hardware Simulation demonstration design more info

Peripheral created will be available in the catalog :

D:/xilinx/codesign/2018/lab2/ip_repo

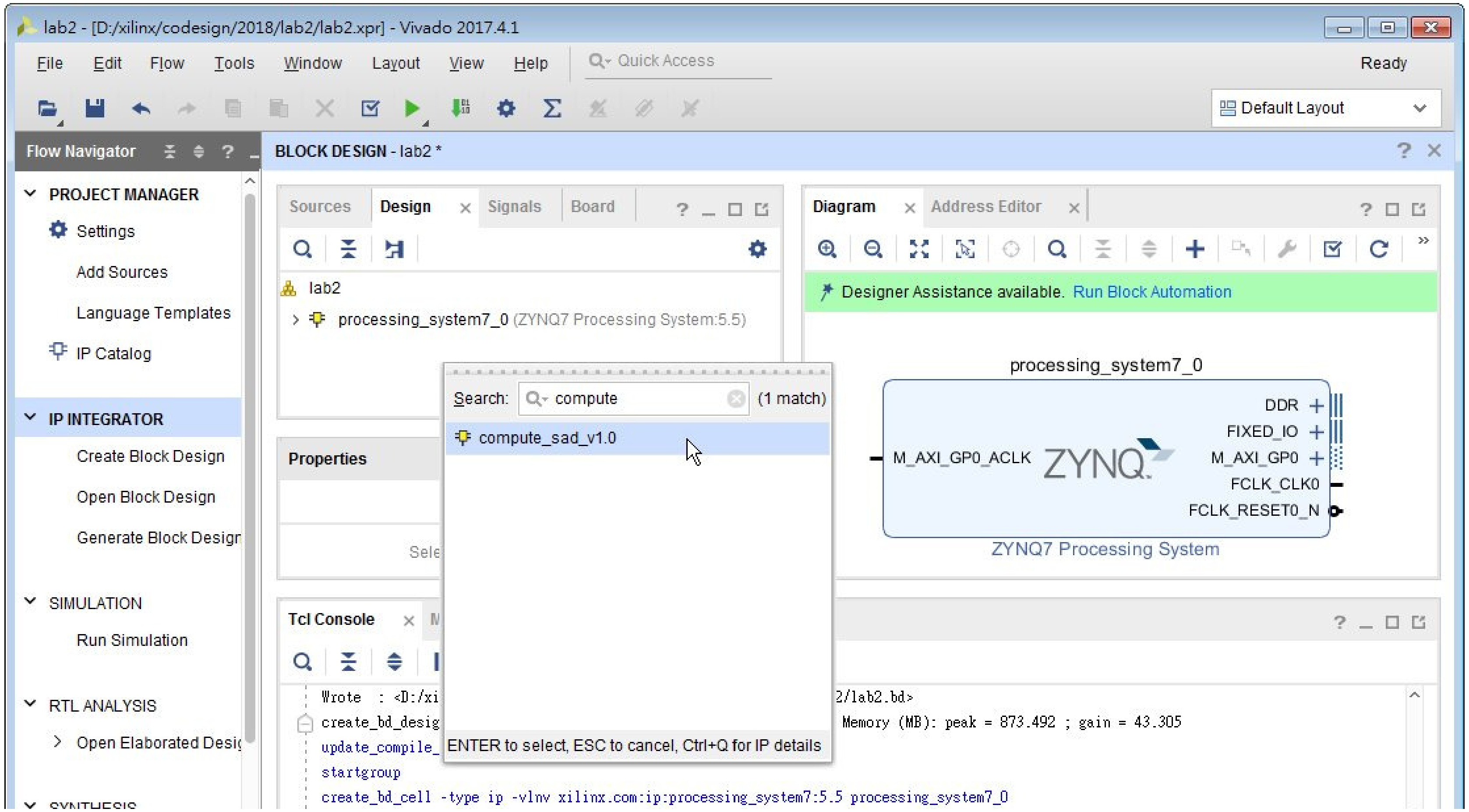
Next Steps:

[✓ Add IP to the repository]
Edit IP
Verify Peripheral IP using AXI4 VIP
Verify peripheral IP using JTAG interface

Click Finish to continue

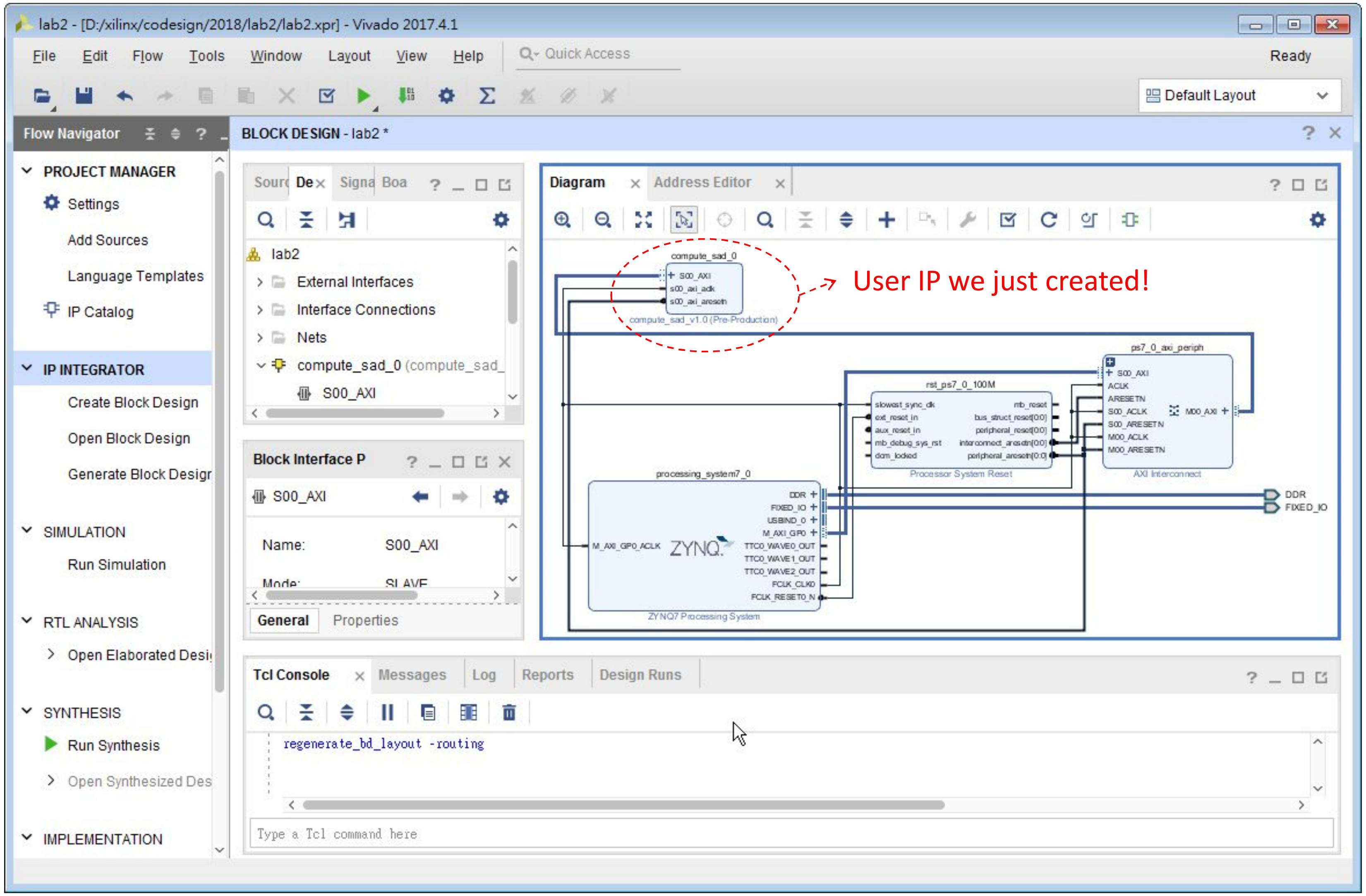
Add the IP to Your Block Diagram

- Add an instance of the new IP to the Zynq platform:



```
Wrote  : <D:/xi...2/lab2.bd>
create_bd_desig... Memory (MB): peak = 873.492 ; gain = 43.305
update_compile_...
startgroup
create_bd_cell -type ip -vlnv xilinx.com:ip:processing_system7:5.5 processing_system7_0
```

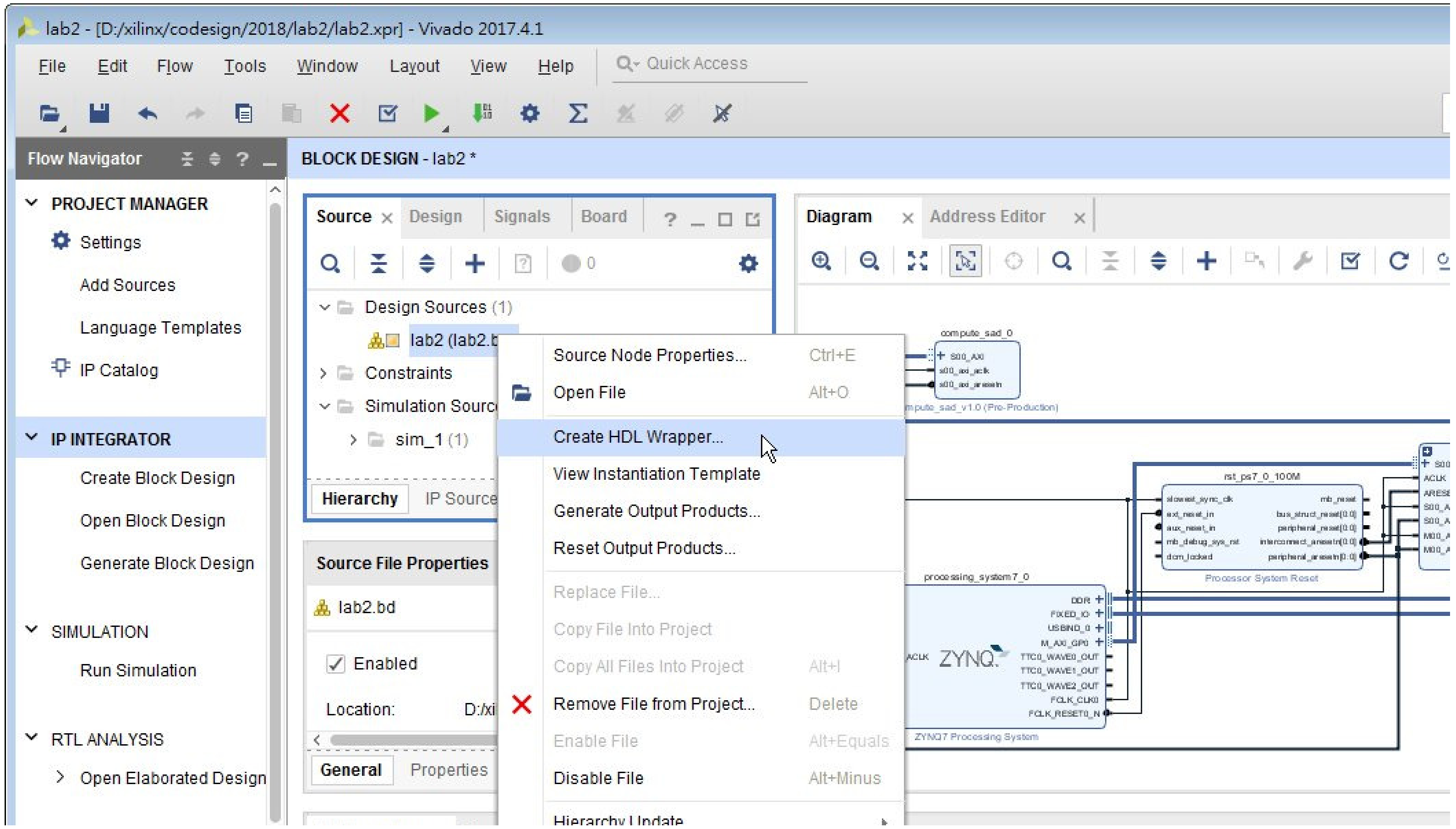
Run Block & Connection Automations



```
regenerate_bd_layout -routing
```

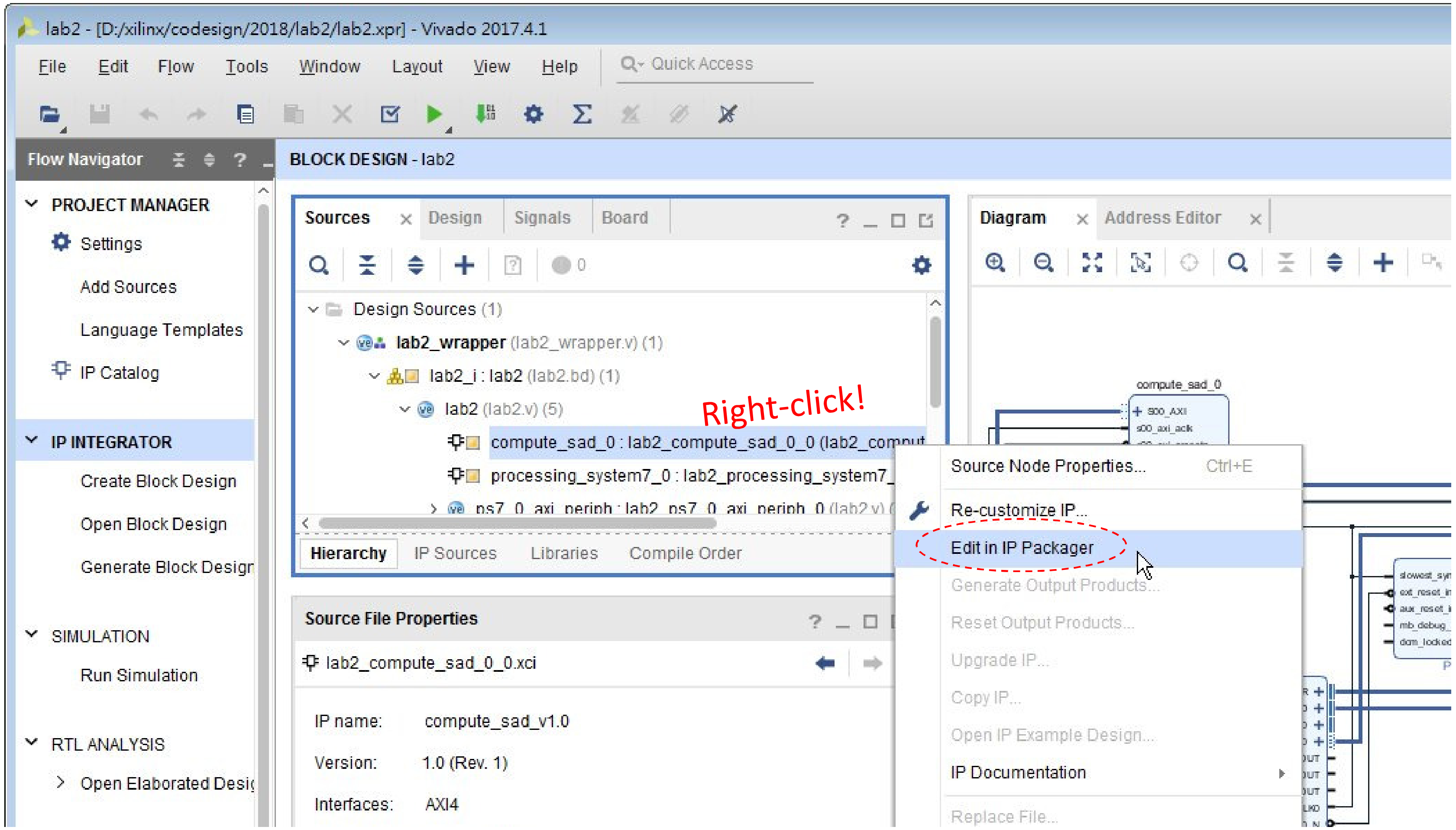
Create the HDL Wrapper

- Create HDL wrapper, including a template IP source code for the `compute_sad` module

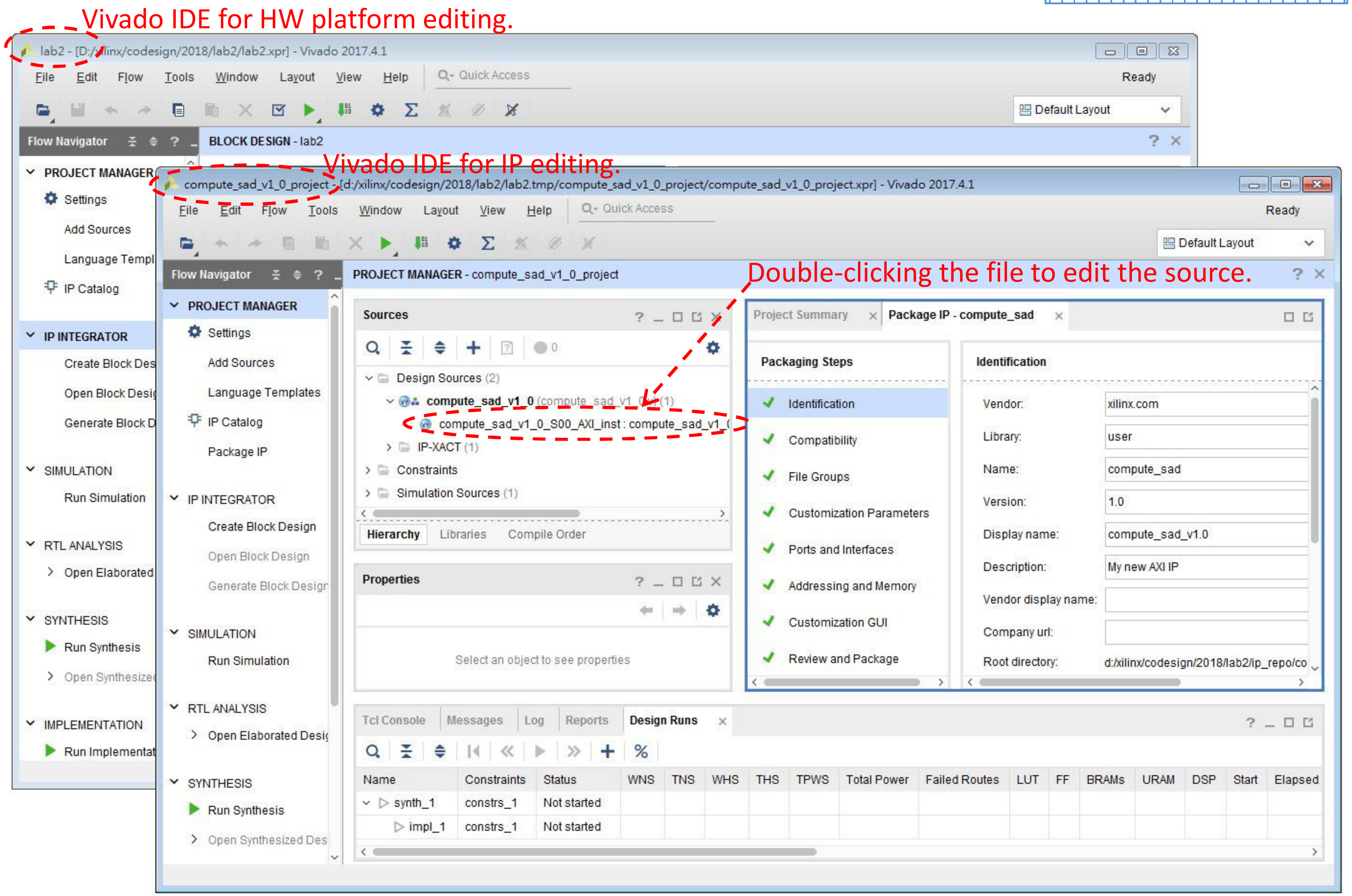


Now, Edit the IP Verilog Source Code

- ❑ Invoke another Vivado instance for IP editing:



IP Packager Runs in a 2nd Vivado IDE



Vivado IDE for HW platform editing.

Vivado IDE for IP editing.

Double-clicking the file to edit the source.

lab2 - [D:/xilinx/codesign/2018/lab2/lab2.xpr] - Vivado 2017.4.1

compute_sad_v1_0_project - [d:/xilinx/codesign/2018/lab2/lab2.tmp/compute_sad_v1_0_project/compute_sad_v1_0_project.xpr] - Vivado 2017.4.1

PROJECT MANAGER - compute_sad_v1_0_project

Sources

- Design Sources (2)
  - compute_sad_v1_0 (compute_sad_v1_0.v) (1)
    - compute_sad_v1_0_S00_AXI_inst : compute_sad_v1_(
  - IP-XACT (1)
- Constraints
- Simulation Sources (1)

Package IP - compute_sad

Packaging Steps

- ✓ Identification
- ✓ Compatibility
- ✓ File Groups
- ✓ Customization Parameters
- ✓ Ports and Interfaces
- ✓ Addressing and Memory
- ✓ Customization GUI
- ✓ Review and Package

Identification

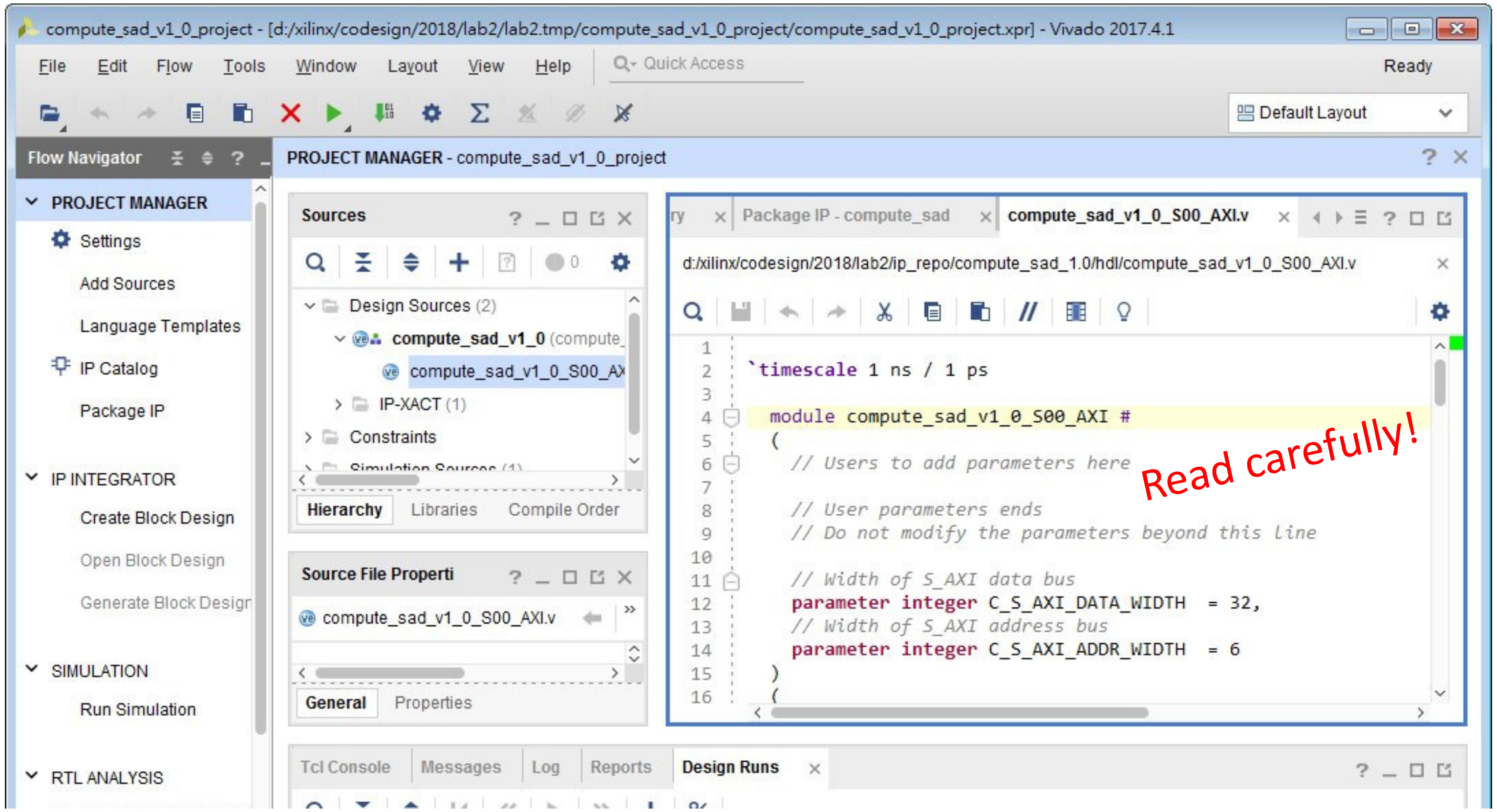
Vendor:	xilinx.com
Library:	user
Name:	compute_sad
Version:	1.0
Display name:	compute_sad_v1.0
Description:	My new AXI IP
Vendor display name:	
Company url:	
Root directory:	d:/xilinx/codesign/2018/lab2/ip_repo/co

Design Runs

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAMs	URAM	DSP	Start	Elapsed
synth_1	constrs_1	Not started														
impl_1	constrs_1	Not started														

The Template RTL Model

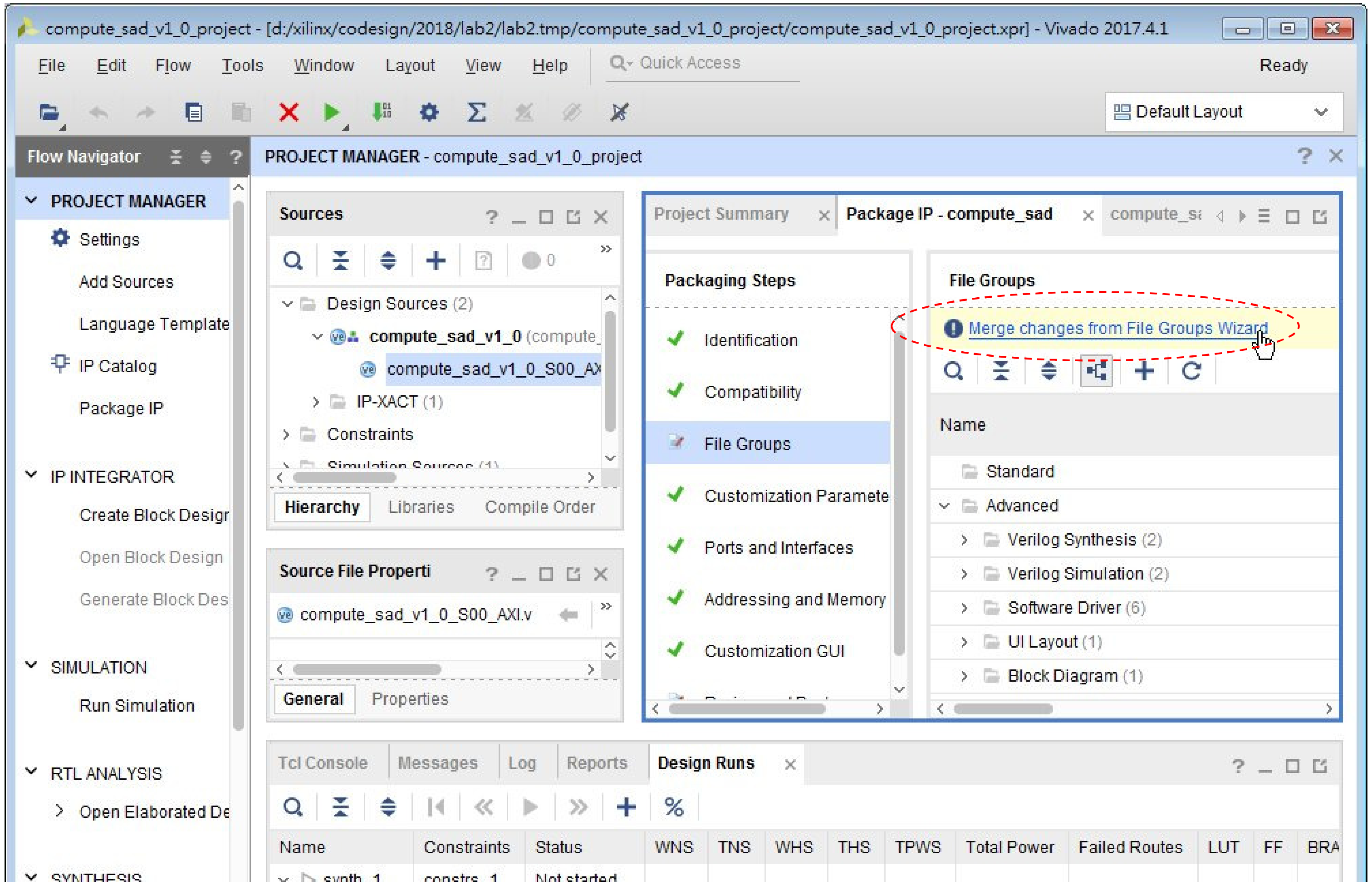
- The IP template contains the Verilog code to handle the read/write requests to the I/O registers from another IPs (including the processor)



```
`timescale 1 ns / 1 ps

	module compute_sad_v1_0_S00_AXI #
	(
		// Users to add parameters here

		// User parameters ends
		// Do not modify the parameters beyond this line

		// Width of S_AXI data bus
		parameter integer C_S_AXI_DATA_WIDTH	= 32,
		// Width of S_AXI address bus
		parameter integer C_S_AXI_ADDR_WIDTH	= 6
	)
	(
```

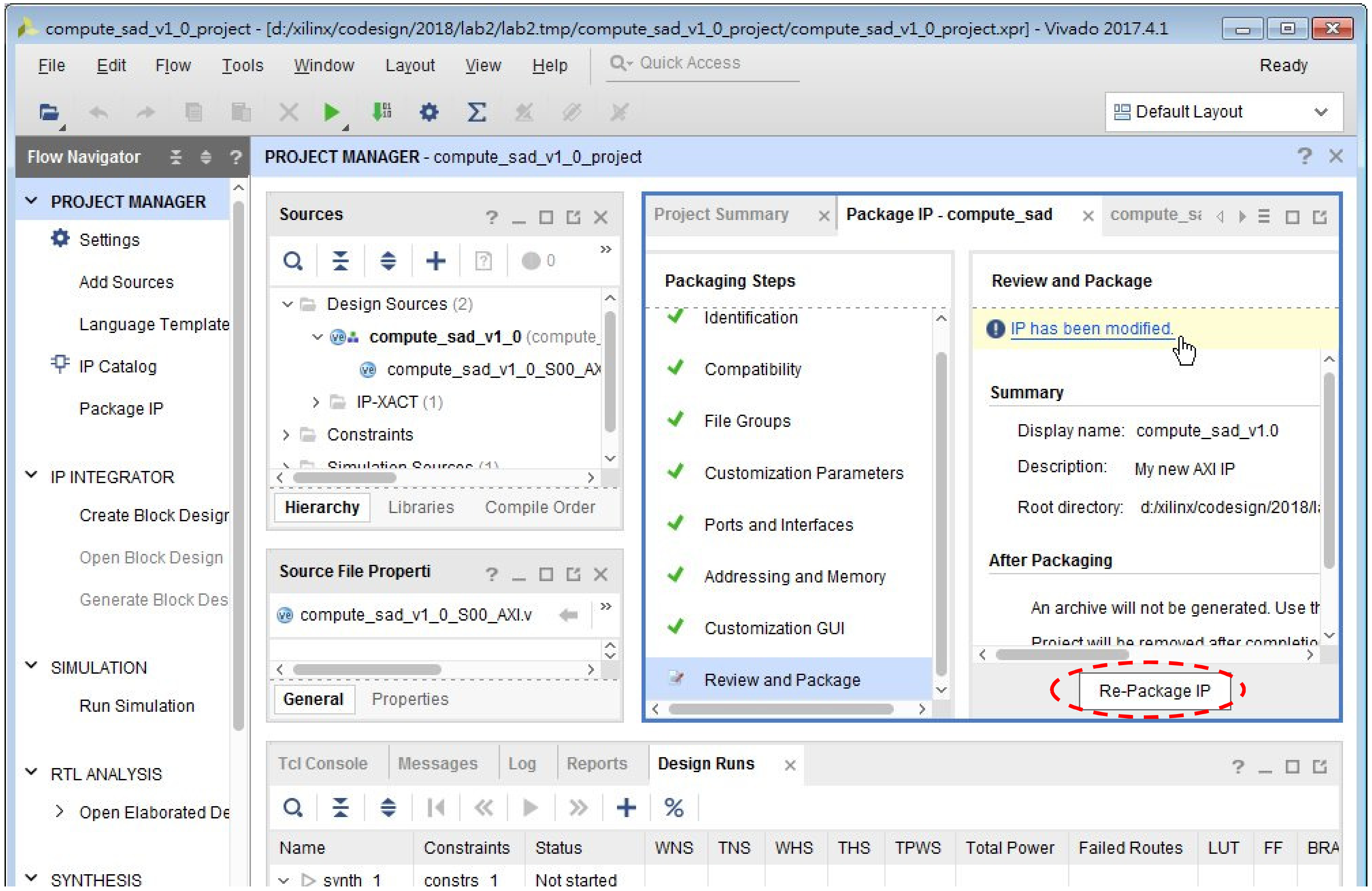
Update File Groups

- After IP editing, you must update the modified files



Repackage IP

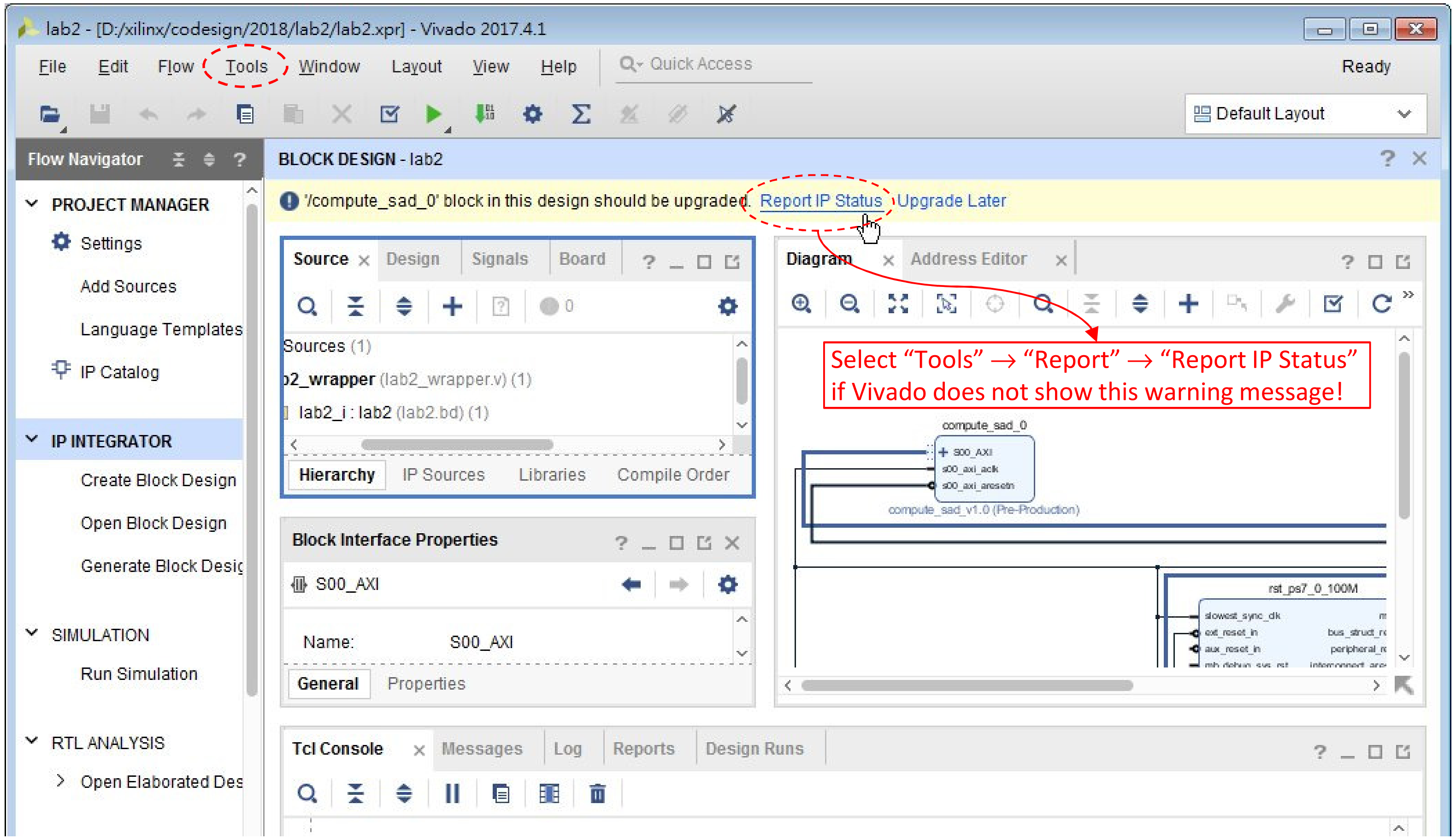
- You must repackage your IP when the editing is done



Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRA
synth_1	constrs_1	Not started										

Check IP Status after IP Editing

- If you changed an IP, you must upgrade the platform:



lab2 - [D:/xilinx/codesign/2018/lab2/lab2.xpr] - Vivado 2017.4.1

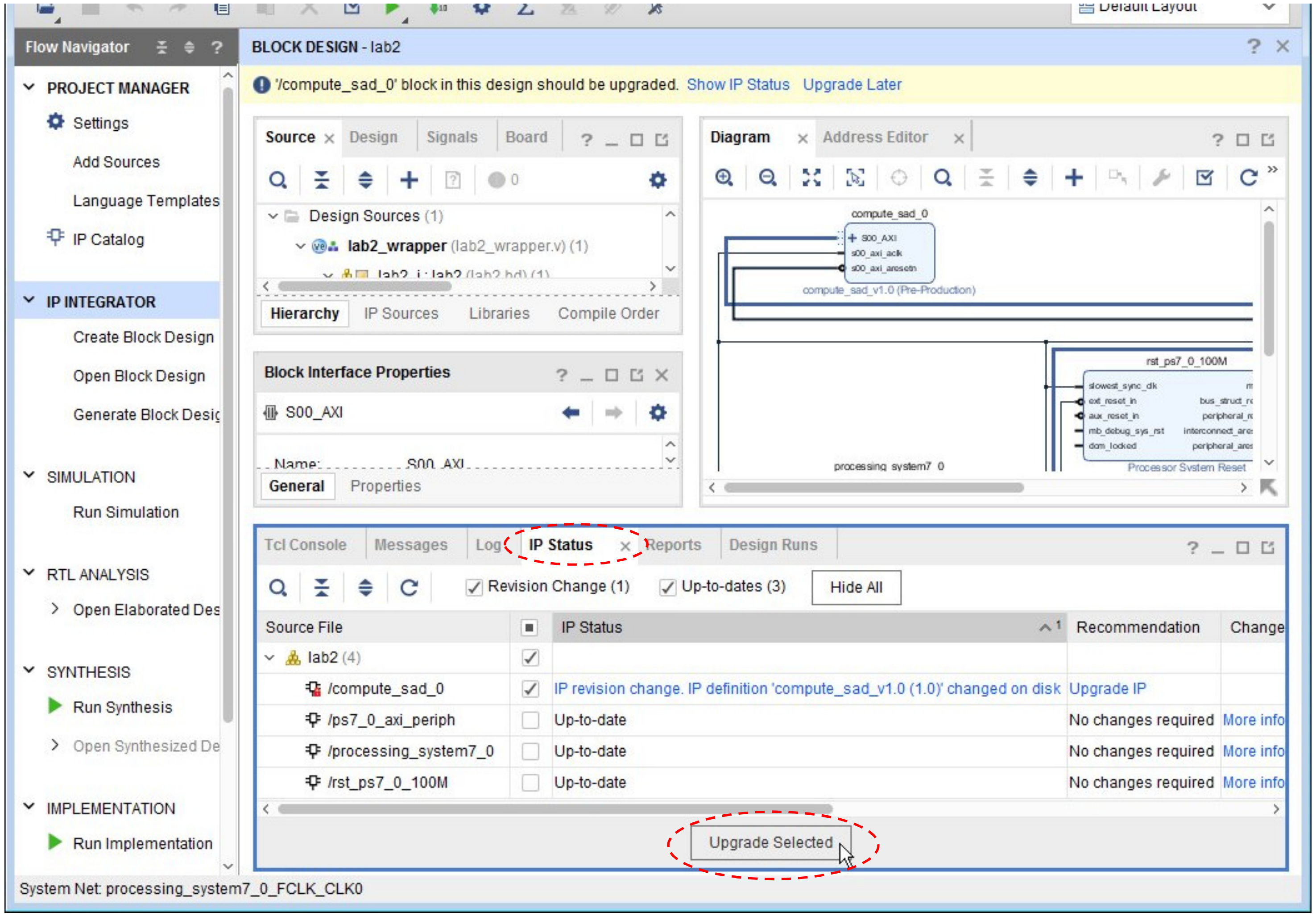
BLOCK DESIGN - lab2

'/compute_sad_0' block in this design should be upgraded. Report IP Status Upgrade Later

Select "Tools" → "Report" → "Report IP Status" if Vivado does not show this warning message!

Update IP and the Platform

- ❑ You must update the IP and refresh the platform:



'/compute_sad_0' block in this design should be upgraded. Show IP Status Upgrade Later

Source File	■	IP Status	Recommendation	Change
lab2 (4)	✓			
/compute_sad_0	✓	IP revision change. IP definition 'compute_sad_v1.0 (1.0)' changed on disk	Upgrade IP	
/ps7_0_axi_periph	☐	Up-to-date	No changes required	More info
/processing_system7_0	☐	Up-to-date	No changes required	More info
/rst_ps7_0_100M	☐	Up-to-date	No changes required	More info

System Net: processing_system7_0_FCLK_CLK0

How Can CPU Talk to the IP

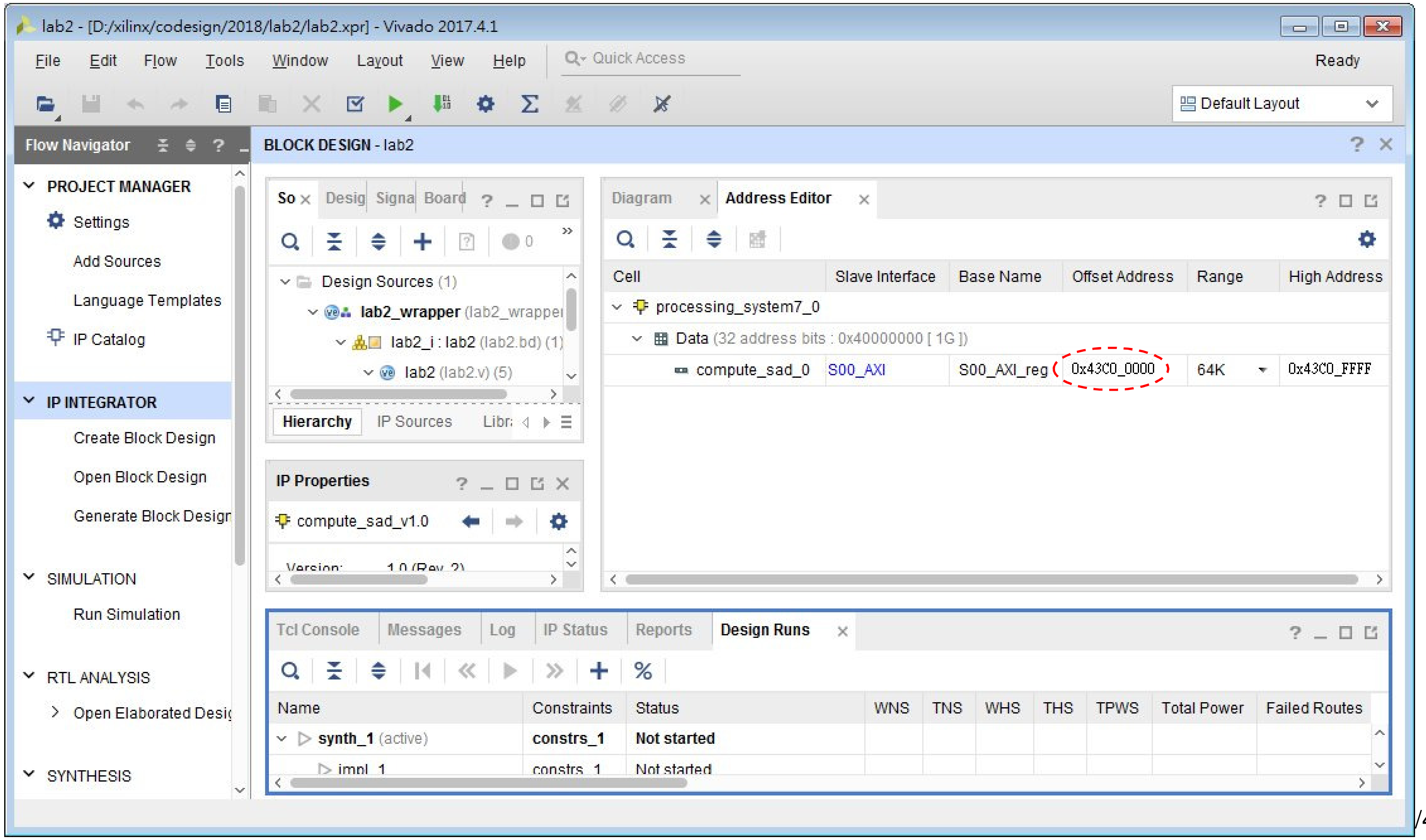
- Today, modern CPU communicates with an IP using memory-mapped I/O operations:
  - Some memory addresses (called I/O addresses) will be mapped to the registers inside a HW IP
  - CPU work with the IP by reading/writing these addresses

- Each HW IP in a Vivado HW platform has a base address assigned by the IP Integrator, all its registers are addressed w.r.t. the base address

SW addresses	LSB of Address offsets	Register to access
BASEADDR + 0	8'b00000000	slv_reg0
BASEADDR + 4	8'b00000100	slv_reg1
BASEADDR + 8	8'b00001000	slv_reg2
BASEADDR + 12	8'b00001100	slv_reg3
	. . .	

Checking the Base Address of the IP

❑ Use Window → Address Editor to see the I/O addresses



lab2 - [D:/xilinx/codesign/2018/lab2/lab2.xpr] - Vivado 2017.4.1

BLOCK DESIGN - lab2

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
compute_sad_0	S00_AXI	S00_AXI_reg	0x43C0_0000	64K	0x43C0_FFFF

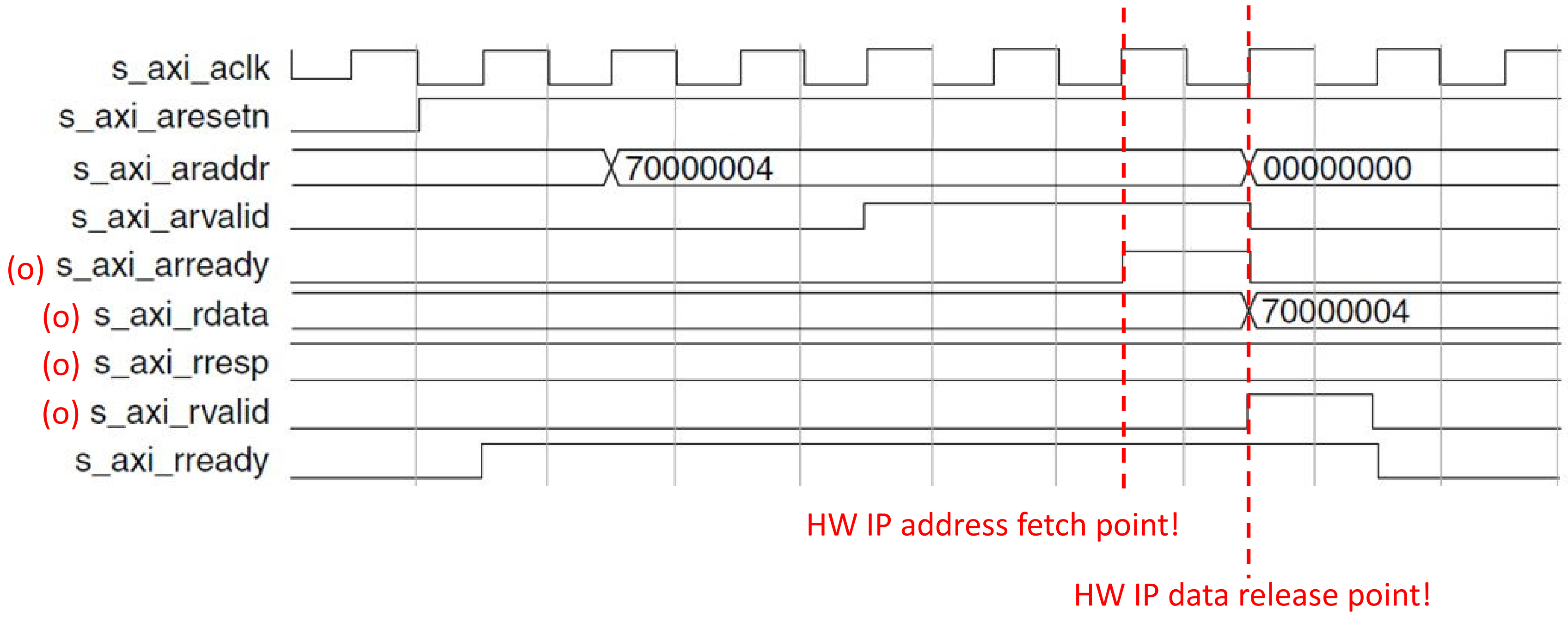
Design Sources (1)
lab2_wrapper (lab2_wrapper
lab2_i : lab2 (lab2.bd) (1)
lab2 (lab2.v) (5)

IP Properties
compute_sad_v1.0
Version: 1.0 (Rev. 2)

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes
synth_1 (active)	constrs_1	Not started							
impl_1	constrs_1	Not started							

AXI4-Lite[†] Single Read Operation

- If a register is read by the processor, the AXI bus signal "`axi_arready`" inside the IP will be triggered
- The timing diagram of a AXI4-Lite single-word read transaction within the HW IP:



† For AXI4 bus protocol details, please see *AMBA® AXI™ and ACE™ Protocol Specification*, ARM IHI 0022D, 2011.

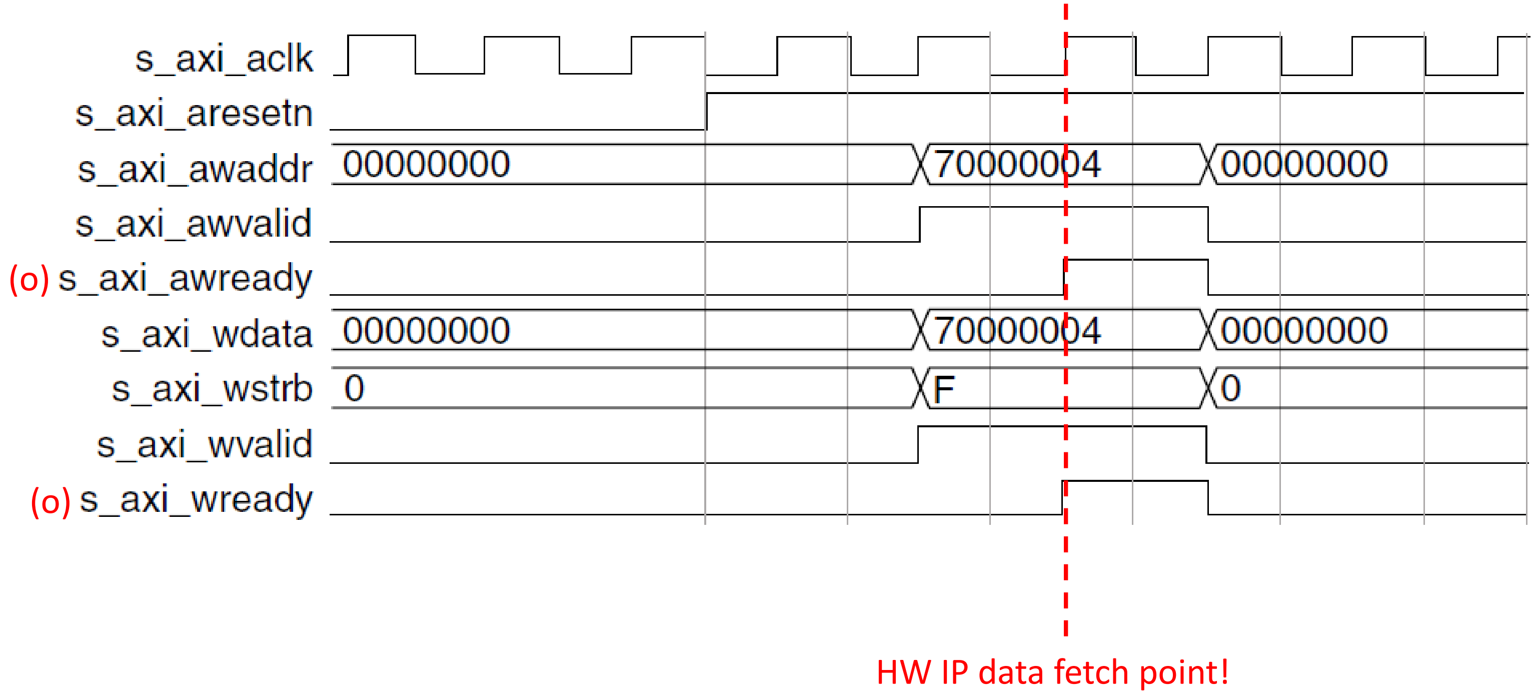
HDL Code for CPU Reading Registers

- The RTL code to handle register read:

```
assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;

always @(*)
  case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
    4'h0    : reg_data_out <= slv_reg0;
                          . . .
    4'h9    : reg_data_out <= slv_reg9;
    default : reg_data_out <= 0;
  endcase

always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    axi_rdata  <= 0;
  else if (slv_reg_rden)
    axi_rdata <= reg_data_out;  // register read data
end
```

AXI4-Lite† Single-Write Operation

- The timing diagram of a AXI4-Lite single-word write transaction within the HW IP:



HDL Code for CPU Writing Registers

- The RTL code to handle register write:

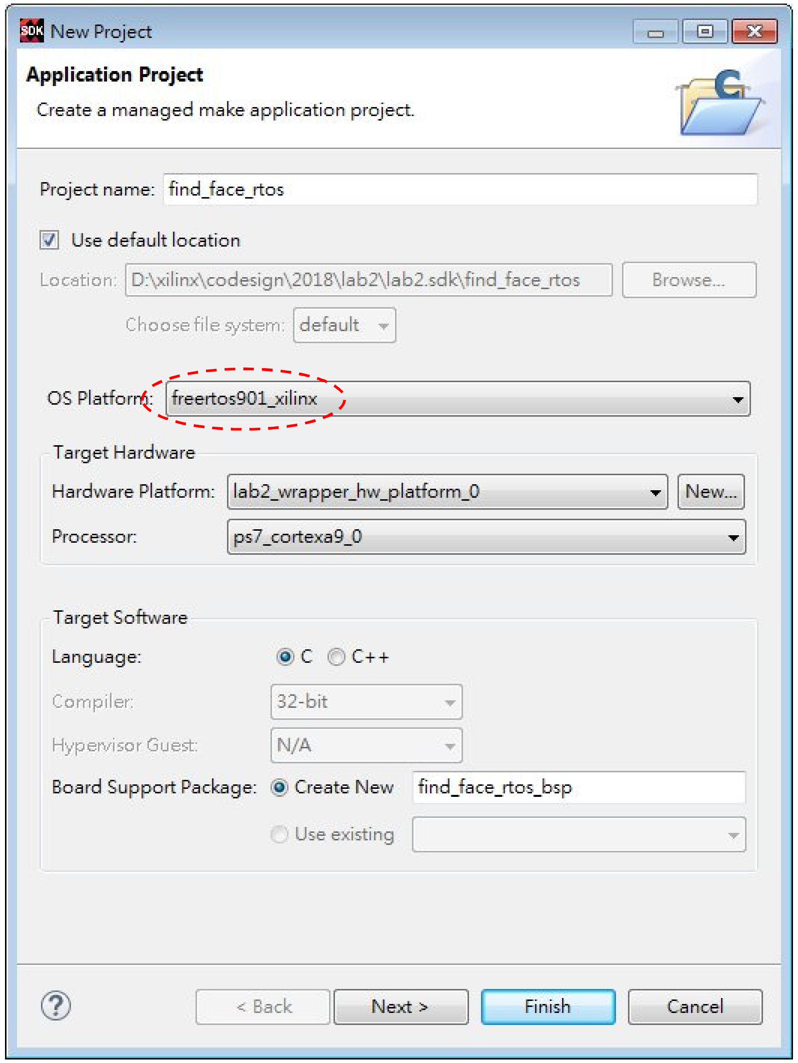
```
assign slv_reg_wren = axi_wready && S_AXI_WVALID &&
                                     axi_awready && S_AXI_AWVALID;

always @( posedge S_AXI_ACLK )
begin
  if ( S_AXI_ARESETN == 1'b0 )
    slv_reg0 <= 0; slv_reg1 <= 0; . . . slv_reg9 <= 0;
  else if ( slv_reg_wren )
    case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
                                   . . .
      4'h2:
        for ( byte = 0; byte <= (DATA_WIDTH/8)-1; byte = byte+1 )
        begin
          if ( S_AXI_WSTRB[byte] == 1 )
            slv_reg2[(byte*8) +: 8] <= S_AXI_WDATA[(byte*8) +: 8];
        end
                                   . . .

      default :
          begin
            slv_reg0 <= slv_reg0; . . . slv_reg9 <= 0;
          end
    endcase
end
```

How Many Registers Do You Need?

- For Lab2, the IP computes the sum-of-absolute-differences of 32 pixel pairs, where each pixel has 8 bits. In short, the HW IP needs 18 registers
  - Inputs:
    - Eight 32-bit registers, `face[0:7][31:0]`, for the face pixels.
    - Eight 32-bit registers, `group[0:7][31:0]`, for the group pixels
  - Input/output:
    - A command/status register, `hw_active`, to trigger the HW IP to begin the computation, and to notify CPU when it is done
  - Output:
    - A 32-bit register, `sad_result`, to return the result of SAD

Export HW and Create a SW Project

- ❑ Export the HW to the SDK
- ❑ Create a FreeRTOS application project, name it `find_face_rtos`.
- ❑ Select the template: "FreeRTOS Hello World", as your starting point, then port `find_face.c` from Lab 1 into this application



New Project

Application Project

Create a managed make application project.

Project name: find_face_rtos

[✓] Use default location

Location: D:\xilinx\codesign\2018\lab2\lab2.sdk\find_face_rtos

Choose file system: default

OS Platform: freertos901_xilinx

Target Hardware

Hardware Platform: lab2_wrapper_hw_platform_0

Processor: ps7_cortexa9_0

Target Software

Language: [● C] ○ C++

Compiler: 32-bit

Hypervisor Guest: N/A

Board Support Package: [● Create New] find_face_rtos_bsp

○ Use existing

FreeRTOS

- FreeRTOS^{TM} is a real time operating system (RTOS) from Real Time Engineers Ltd.
  - FreeRTOS supports almost all popular CPU architecture
  - Free to use in commercial products, no need to release code
  - Less than 20,000 lines of code; less than 9KB runtime image
  - Great for IoT's

- For general information, check FreeRTOS website:
  http://www.freertos.org

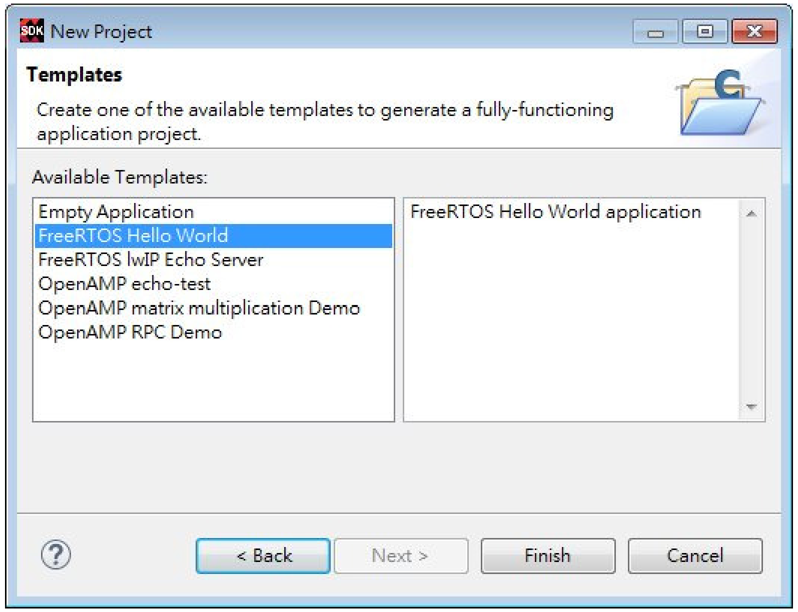
- For coding style and naming conventions, check:
  http://www.freertos.org/FreeRTOS-Coding-Standard-and-Style-Guide.html

FreeRTOS Kernel Model

- FreeRTOS is different from Linux:
  - Linux is a standalone executable code that spawns many processes, one for each application
  - FreeRTOS is a kernel library that is linked with the application program, it can only execute one single program, but the program can spawn many threads.
- Note that, a FreeRTOS kernel refers to its “threads” by the name “tasks.”
- A tutorial on FreeRTOS programming is available at: http://www.freertos.org/tutorial/

Create the Initial Template Application

- ❑ Select the Hello World template:
- ❑ Rename the main file to `find_face.c`



New Project

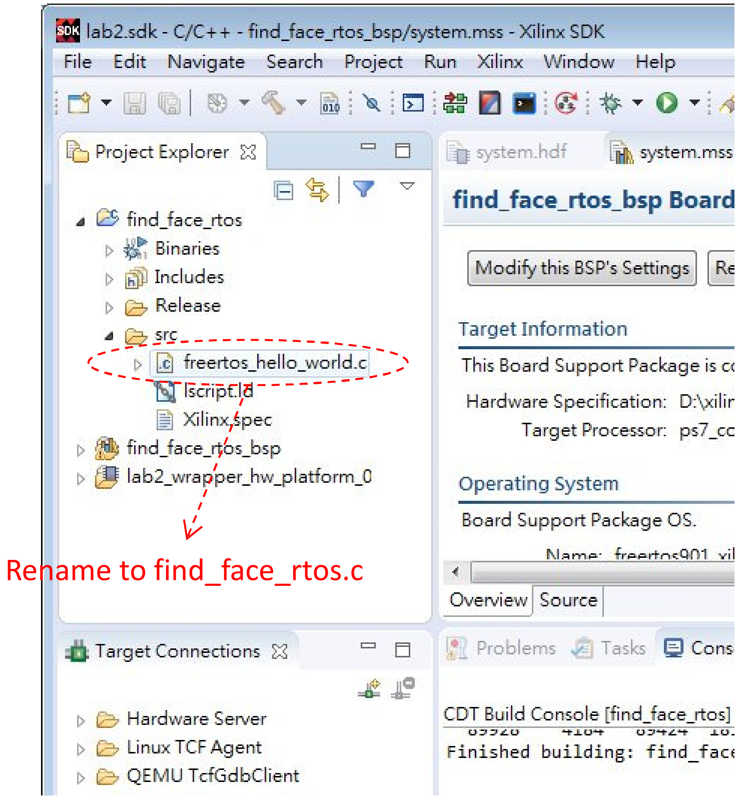
Templates

Create one of the available templates to generate a fully-functioning application project.

Available Templates:

- Empty Application
- FreeRTOS Hello World
- FreeRTOS lwIP Echo Server
- OpenAMP echo-test
- OpenAMP matrix multiplication Demo
- OpenAMP RPC Demo

FreeRTOS Hello World application

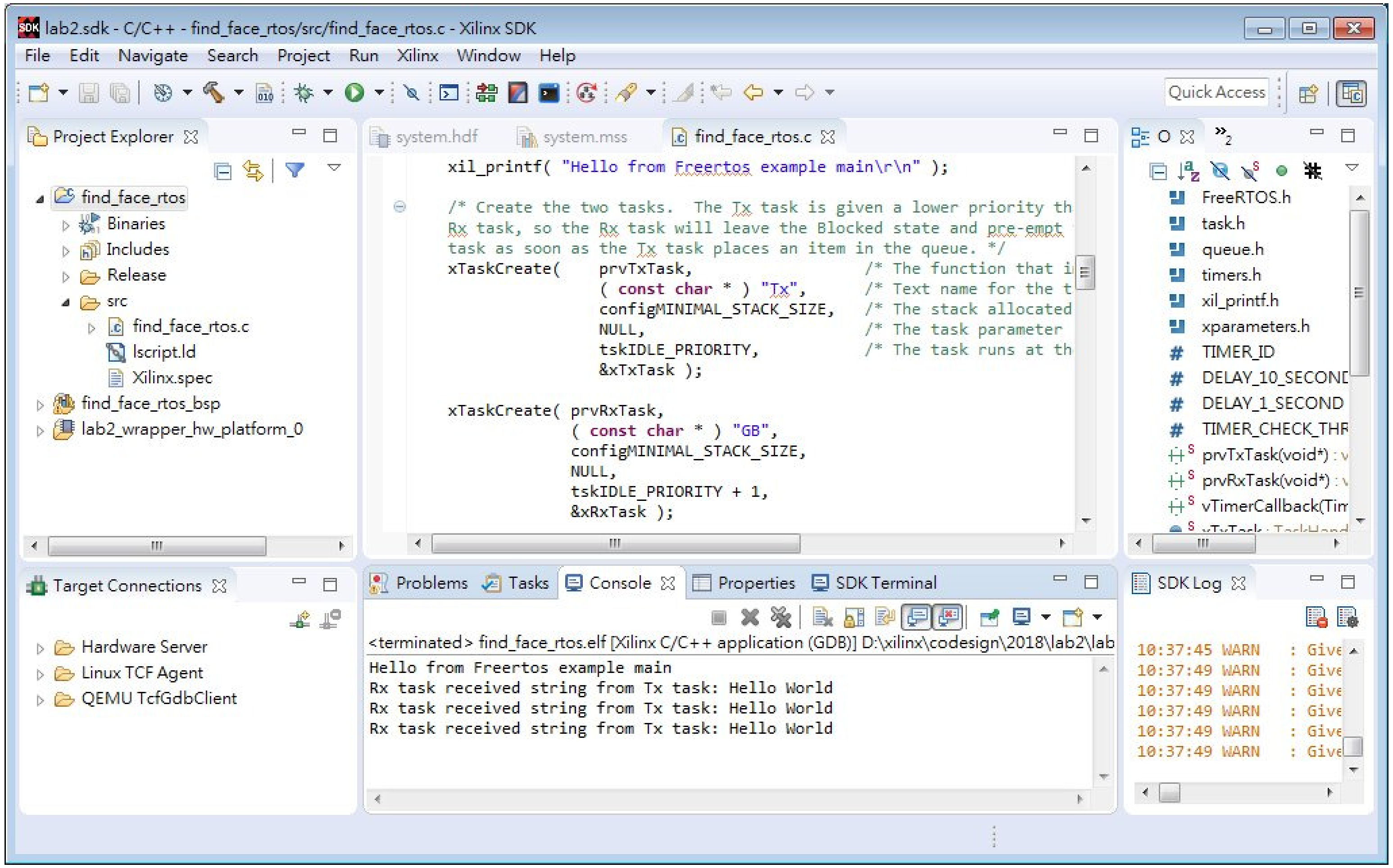


lab2.sdk - C/C++ - find_face_rtos_bsp/system.mss - Xilinx SDK

```
Finished building: find_face
```

FreeRTOS “Hello World” Template

- The “Hello World” template is a two-tread application:



```
    xil_printf( "Hello from Freertos example main\r\n" );

    /* Create the two tasks.  The Tx task is given a lower priority th
    Rx task, so the Rx task will leave the Blocked state and pre-empt
    task as soon as the Tx task places an item in the queue. */
    xTaskCreate(    prvTxTask,                  /* The function that i
                    ( const char * ) "Tx",      /* Text name for the t
                    configMINIMAL_STACK_SIZE,   /* The stack allocated
                    NULL,                       /* The task parameter
                    tskIDLE_PRIORITY,           /* The task runs at th
                    &xTxTask );

    xTaskCreate( prvRxTask,
                 ( const char * ) "GB",
                 configMINIMAL_STACK_SIZE,
                 NULL,
                 tskIDLE_PRIORITY + 1,
                 &xRxTask );
```

```
<terminated> find_face_rtos.elf [Xilinx C/C++ application (GDB)] D:\xilinx\codesign\2018\lab2\lab
Hello from Freertos example main
Rx task received string from Tx task: Hello World
Rx task received string from Tx task: Hello World
Rx task received string from Tx task: Hello World
```

```
10:37:45 WARN  : Give
10:37:49 WARN  : Give
10:37:49 WARN  : Give
10:37:49 WARN  : Give
10:37:49 WARN  : Give
10:37:49 WARN  : Give
```

Tasks in the “Hello World” Template

```
static void prvTxTask(void *pvParameters) {
  send_buf[] = "Hello World";
  const TickType_t x1second = pdMS_TO_TICKS(DELAY_1_SECOND);
  for( ;; ) {
    vTaskDelay(x1second);
    xQueueSend(xQueue, send_buf, 0UL);
  }
}

static void prvRxTask(void *pvParameters) {
  char recv_buf[15];
  for( ;; ) {
    xQueueReceive(xQueue, recv_buf, portMAX_DELAY);
    xil_printf("Rx task received string from Tx task: %s\r\n", recv_buf);
    RxtaskCntr++;
  }
}

int main() {
                                    . . .
  /* Create the two tasks */
  xTaskCreate(prvTxTask, "Tx", STACK_SIZE, NULL, TASK_PRIORITY, &xTxTask);
  xTaskCreate(prvRxTask, "GB", STACK_SIZE, NULL, NULL, TASK_PRIORITY + 1, &xRxTask);
  xQueue = xQueueCreate(1, sizeof(send_buf));
                                    . . .
}
```

Thread Control in “Hello World”

- The two tasks in “Hello World” only runs for about 10 seconds. There is a timer set by the main program that runs for 10 seconds.
- When the time is up, a timer call-back routine will be invoked and kills the two application tasks in the main program.
- The two tasks in “Hello World” communicates to each other by a message queue.

Limitations of FreeRTOS on Zynq

- ❑ FreeRTOS on Zynq do not support two CPU cores running in Symmetric Multi-Processor (SMP) mode, it only supports running two CPUs in AMP mode

- ❑ If you create multiple threads using FreeRTOS API, these threads will all be running on one processor

  → you cannot speed up `find_face` by simply rewriting the program using multiple threads

Porting Guide for `find_face.c` (1/2)

- Make sure you change the linker script so that you have enough heap space
- The memory management API in FreeRTOS are pvPortMalloc() and vPortFree(). The default stack size and heap size are defined in FreeRTOSConfig.h. Make sure they are big enough, e.g.,

```
#define configMINIMAL_STACK_SIZE ( ( unsigned short ) 0x2000 )

#define configTOTAL_HEAP_SIZE ( ( size_t ) ( 0x1000000 ) )
```

Porting Guide for `find_face.c` (2/2)

- The FreeRTOS kernel handles thread and memory management, it does not do I/O management → you need Xilinx standalone BSP APIs for low-level I/Os
- Be careful with printf() or xil_printf() if you have more than one threads
- Make sure you enable the BSP option to support the FAT I/O library so you can access the SD card

Measuring Time Using FreeRTOS

- You can use the system call `xTaskGetTickCount()` to get the time counter and convert it to milliseconds:

```
static void find_face(void *pvParameters)
{
    TickType_t time_tick;
    unsigned long msec;

    time_tick = xTaskGetTickCount();

    /* main computations */
    . . .

    time_tick = xTaskGetTickCount() - time_tick;
    msec = time_tick * (1000 / configTICK_RATE_HZ);
}
```

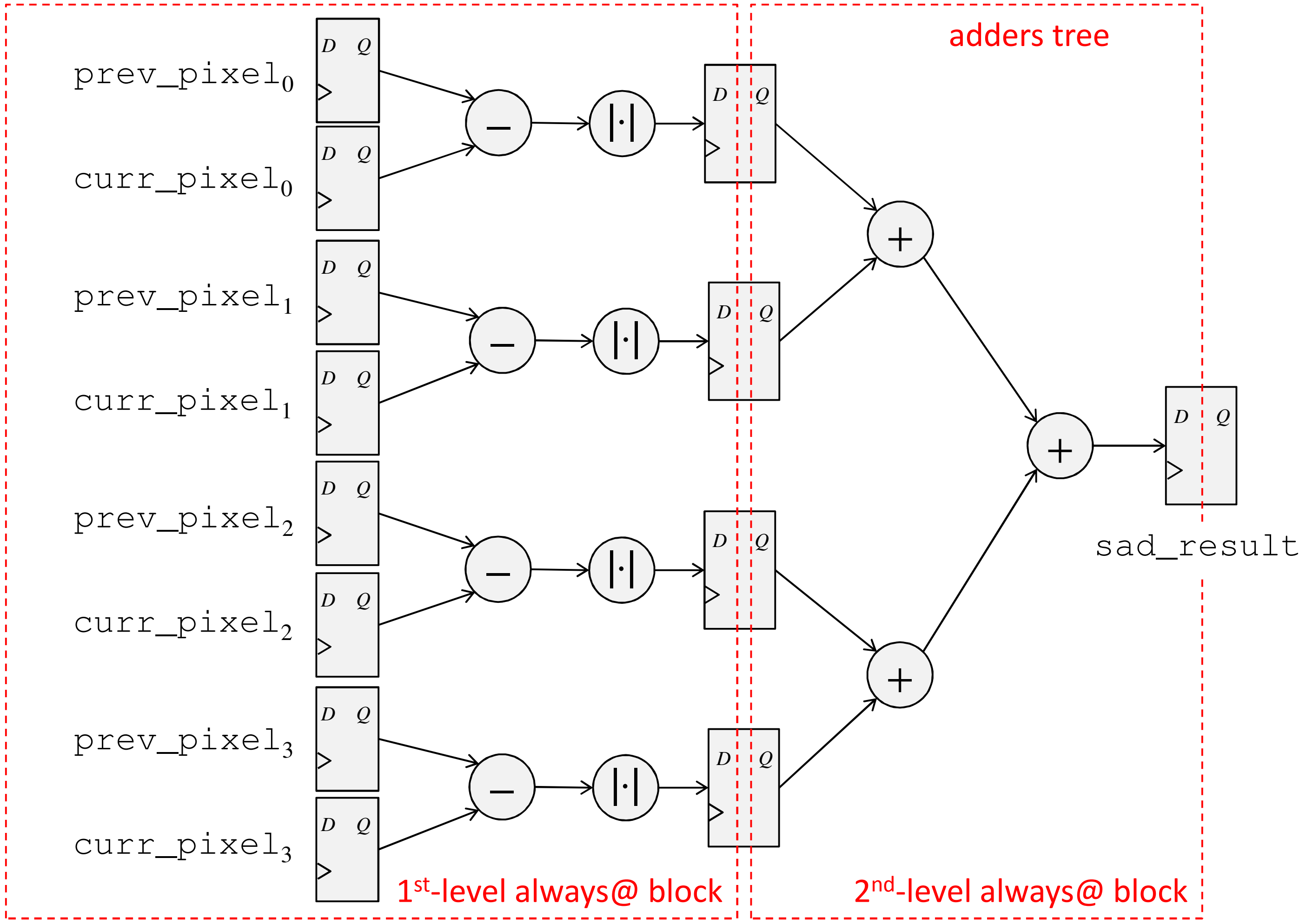
What You Have to Do for Lab 2

- Modify `compute_sad_v1_0_S00_AXI.v` to
  - Control the registers `hw_active` so that it can be used for synchronization between the HW and the SW
    - If SW writes `1` to `hw_active`, the HW logic starts computing
    - When output is ready, HW sets `hw_active` to `0` to notify SW
  - Write a circuit block to compute the SAD of 16 pixels and save the result in `sad_result`

- Port `find_face.c` to `find_face_rtos.c`

Example of SAD Logic Diagram

- Assume that we want to compute the SAD of 4 pixels:



Example of 1st-level always@ Blocks

- An always@ block for computing the absolute differences is as follows:

```
reg  [8:0]   abs_diff[0:3];
wire [8:0]   f_pixel[0:3];   // face pixels
wire [8:0]   g_pixel[0:3];   // group pixels
wire [8:0]   diff[0:3];

assign f_pixel[0] = {1'b0, slv_reg0[31:24]};
assign g_pixel[0] = {1'b0, slv_reg8[31:24]};
assign diff[0] = f_pixel[0] - g_pixel[0];
            . . .

always@(posedge S_AXI_ACLK)
begin
  if (S_AXI_ARESETN == 1'b0)
    abs_diff[0] <= 0;
             . . .
  else
    abs_diff[0] <= (diff[0][8] == 1’b1)? -diff[0] : diff[0];
             . . .
end
```

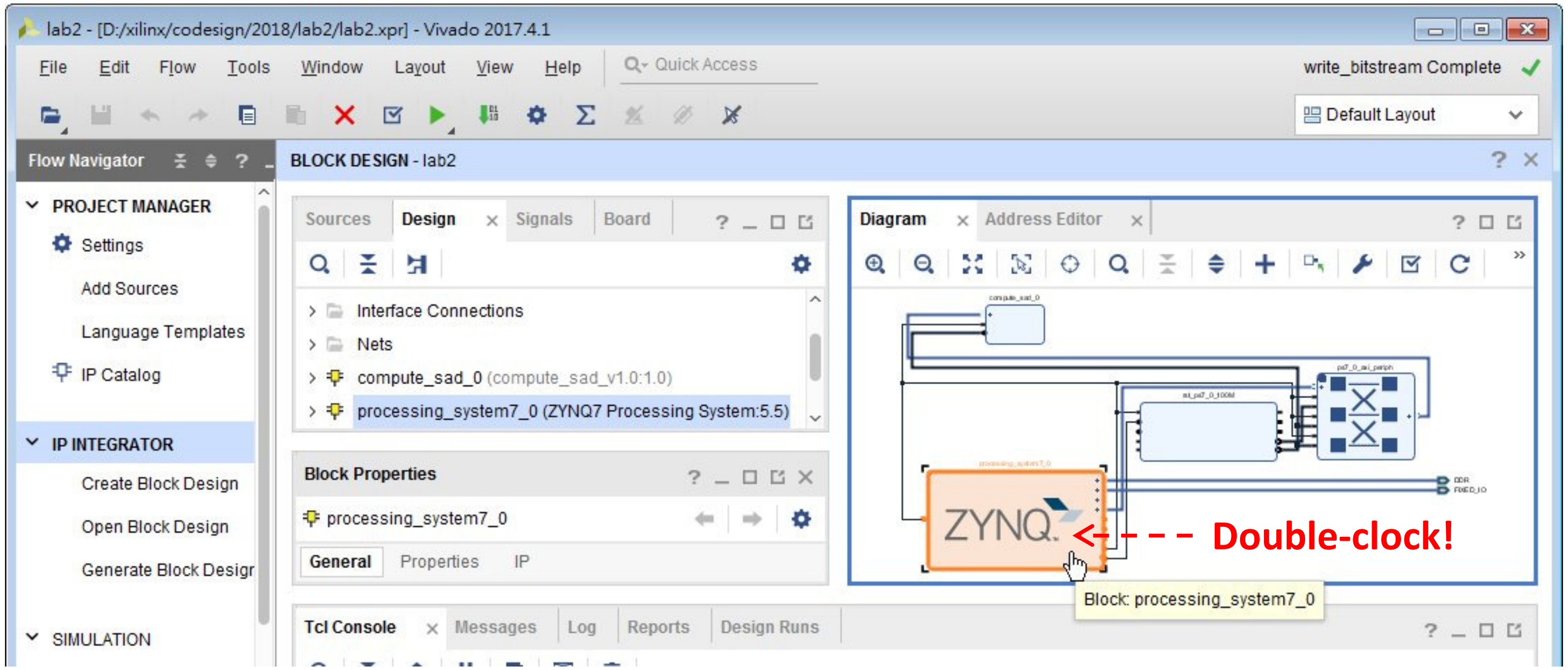
Example of 2nd-level always@ Blocks

- An always@ block for computing the summation of outputs from level-1 logic is as follows:

```
reg  [10:0]  sad_result;
wire [9:0]   partial_sum[0:1];

assign partial_sum[0] = abs_diff[0] + abs_diff[1];
assign partial_sum[1] = abs_diff[2] + abs_diff[3];

always @(posedge S_AXI_ACLK)
  begin
    if (S_AXI_ARESETN == 1'b0)
      sad_result <= 0;
    else
      sad_result <= partial_sum[0] + partial_sum[1];
end
```

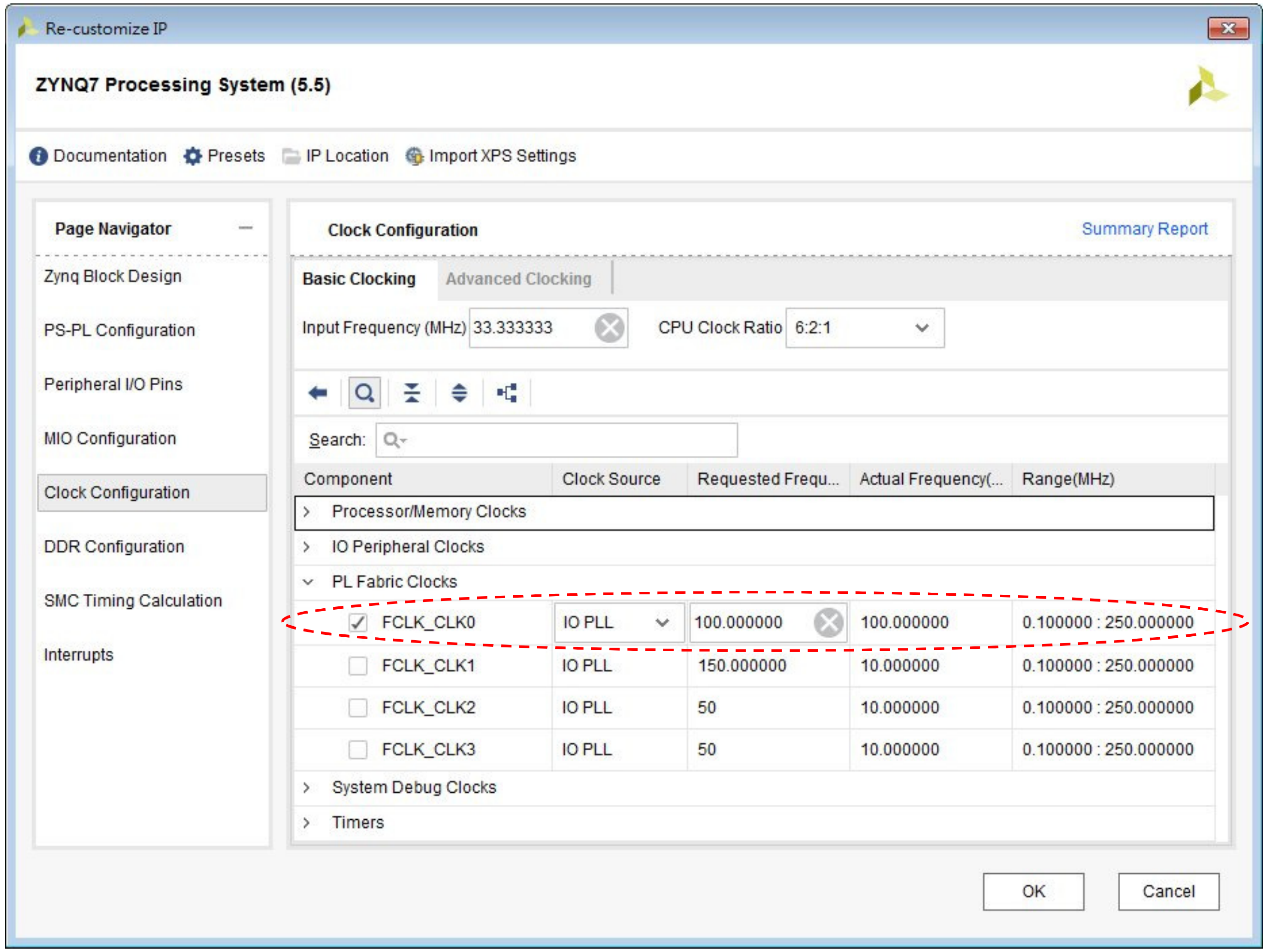
Timing Requirement

- By default, the user IP is running at 100 MHz; thus, your IP must have a critical path shorter than 10ns!
- If the IP does not meet the timing, there are two options:
  - Further divide the logic path into more stages
  - Change the clock setting in the Zynq IP to lower the PL clock



Changing the Clock Rate of the HW IP

- Double-clicking the Zynq IP Block, you can set clocks



Re-customize IP

ZYNQ7 Processing System (5.5)

Documentation | Presets | IP Location | Import XPS Settings

Page Navigator: Zynq Block Design, PS-PL Configuration, Peripheral I/O Pins, MIO Configuration, Clock Configuration, DDR Configuration, SMC Timing Calculation, Interrupts

Clock Configuration — Summary Report

Basic Clocking | Advanced Clocking

Input Frequency (MHz) 33.333333 — CPU Clock Ratio 6:2:1

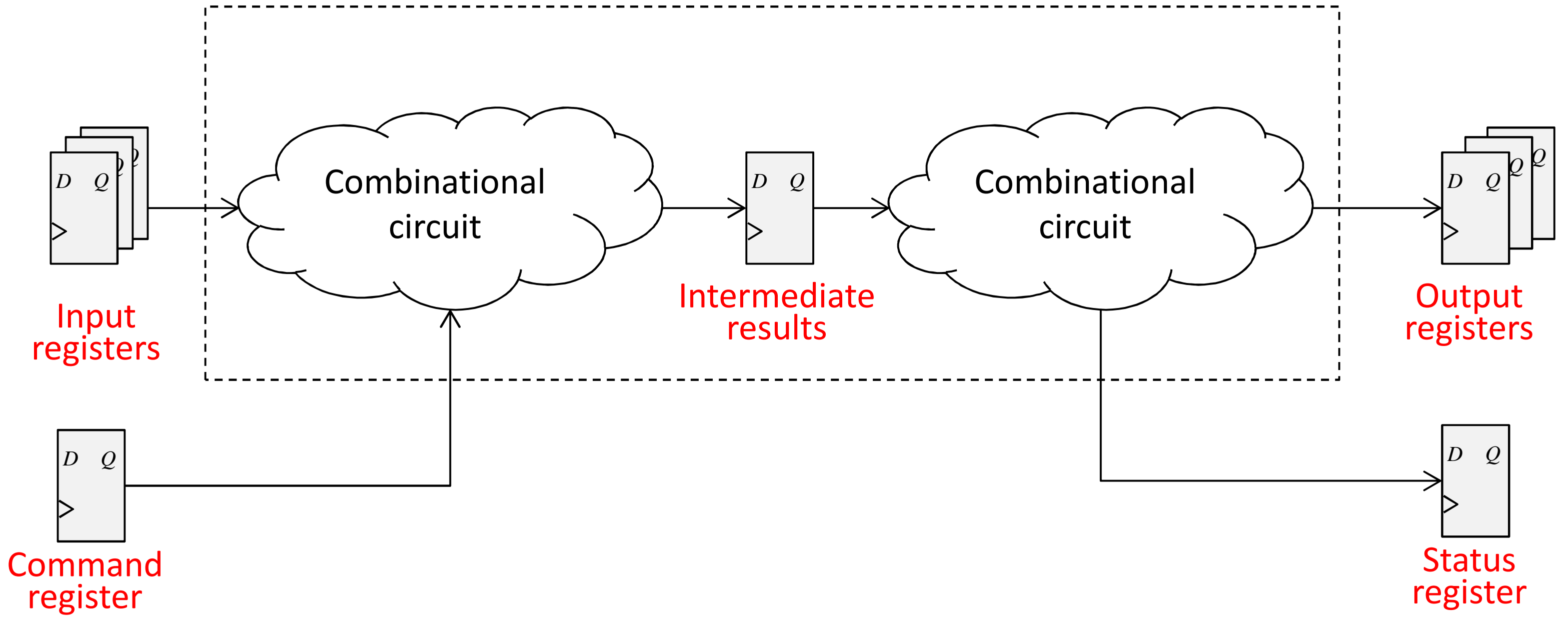
Search:

Component	Clock Source	Requested Frequ...	Actual Frequency(...	Range(MHz)
› Processor/Memory Clocks				
› IO Peripheral Clocks				
⌄ PL Fabric Clocks				
✓ FCLK_CLK0	IO PLL	100.000000	100.000000	0.100000 : 250.000000
☐ FCLK_CLK1	IO PLL	150.000000	10.000000	0.100000 : 250.000000
☐ FCLK_CLK2	IO PLL	50	10.000000	0.100000 : 250.000000
☐ FCLK_CLK3	IO PLL	50	10.000000	0.100000 : 250.000000
› System Debug Clocks				
› Timers				

OK | Cancel

HW-SW Interface of `compute_sad()`

- From the SW side, we would like to invoke the HW IP using three-step protocols as follows:
  - Send data into IP by writing to its input registers
  - SW wait for the IP to finish calculation
  - Read the output registers of the IP for the results



Example `compute_sad()` Invocation

- Sample code for calling the HW IP (naive, but working):

```c
#include <xparameters.h>

volatile int *face_regs  = (int *) (XPAR_COMPUTE_SAD_0_S00_AXI_BASEADDR +  0);
volatile int *group_regs = (int *) (XPAR_COMPUTE_SAD_0_S00_AXI_BASEADDR + 32);
volatile int *hw_active  = (int *) (XPAR_COMPUTE_SAD_0_S00_AXI_BASEADDR + 64);
volatile int *sad_result = (int *) (XPAR_COMPUTE_SAD_0_S00_AXI_BASEADDR + 68);

int32 compute_sad(uint8 *img1, int w1, uint8 *img2, int w2, int h2, int row, int col)
{
    int x, y;
    int sad = 0;

    for (y = 0; y < h2; y++)
    {
        /* send one row of 32 pixels into HW IP registers */
        memmove((void *) group_regs, img1+(row+y)*w1+col, sizeof(int)*8);
        memmove((void *) face_regs, img2+y*w2, sizeof(int)*8);
        *hw_active = 1;

        while (*hw_active)  ; /* busy waiting, not good but faster */
        sad += *sad_result;
    }
    return sad;
}
```

Note: 1. You can also try using memcpy() and see what happens.
2. Endian issues can be handled in your HW IP, if necessary.

Final Warning

- Although we try to accelerate the execution of the program by HW acceleration, you will fail to speed up the application if you do things correctly.
- Communication overhead can seriously degrade the performance of a complex HW-SW system.