

# Port-Level I/O Protocols

2020.1

## Abstract

This lab demonstrates how to implement I/O ports and protocols using high-level synthesis.

## Objectives

After completing this lab, you will be able to:

- Identify the default I/O protocol for ports
- Select the appropriate I/O protocol for ports
- Add directives to select the interface type

## Introduction

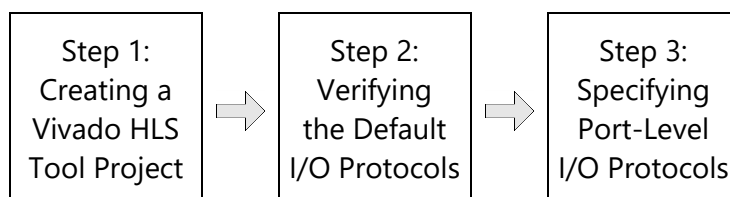
This lab illustrates how RTL ports are added to a C design. Interface synthesis assigns a protocol to each of these ports. I/O protocol synchronizes data transfer with the synthesized logic.

### Port-Level Interface Protocol

The I/O protocol that is created depends on the type of C argument and on the default.

After the block-level protocol has been used to start the operation of the block, the port-level I/O protocols are used to sequence data into and out of the block.

## General Flow



## Before starting the lab

## Step 0

Browse to the `C:\Xilinx_trn\HLS_2020\lab4_port_level_protocol\source` directory using Windows Explorer. Open, with your preferred text editor, and explore the C code to understand algorithm and data dependencies.

### 0-1. Open lab4.h file and review it.

```
1  #ifndef LAB4_IO_H_
2  #define LAB4_IO_H_
3      typedef short data_sc;
4      #define N 4
5      void lab4(data_sc in_s, data_sc in_ar[N], data_sc *in_p, data_sc *out_p);
6
7  #endif
```

Figure 4-1: lab4 definition file

New data type `data_sc` is defined in the file. The data type will be used in source code and testbench. A value for `N` is defined as 4.

### 0-2. Open lab4.c file and review it.

```
1  #include "lab4.h"
2
3  void lab4(data_sc in_s, data_sc in_ar[N], data_sc *in_p, data_sc *out_p)
4  {
5      data_sc temp = 0;
6      for (short i=0; i<N; i++)
7          temp += in_s * in_ar[i] + *in_p;
8      *out_p = temp;
9  }
```

Figure 4-2: lab4 source file

This example uses a simple design to focus on the Port I/O interface implementation. The arguments for the function `lab4` are:

- `in_s` – input scalar,
- `in_ar[N]` – input array,
- `*in_p` – input pointer,
- `*out_p` – output pointer.

The Port I/O protocols created for the function arguments are discussed in this lab exercise.

**0-3. Open lab4\_test.c file and review it.**

```
1  #include <stdio.h>
2  #include "lab4.h"
3  int main()
4  {
5      data_sc inA = 1;
6      data_sc inB = 1;
7      data_sc inAR[N] = {10, 11, 12, 13};
8      data_sc outF;
9      data_sc ref_ar[3] = {50, 100, 150};
10     data_sc temp_t;
11     int pass=0;
12     // Call the function for 3 transactions
13     for (int i=0; i<3; i++) {
14         lab4 (inA, inAR, &inB, &outF );
15         temp_t = outF;
16         fprintf(stdout, "  got=%d expected=%d \n", outF, ref_ar[i]);
17         // Test the output against expected results
18         if (outF == ref_ar[i])
19             pass = 1;
20         else
21             pass = 0;
22         inA=inA+1;
23         inB=inB+1;
24     }
25     if (pass == 1) {
26         fprintf(stdout, "-----Pass!-----\n");
27         return 0;}
28     else {
29         fprintf(stderr, "-----Fail!-----\n");
30         return 1;
31     }
32 }
```

**Figure 4-3: lab4 testbench**

There are three invocations of the function lab4 and three sets of expected results are provided.

## Creating a Vivado HLS Tool Project

## Step 1

---

In this step, you will create a new Vivado® HLS tool project, add source files, and provide solution settings for the default solution using the Vivado HLS tool command prompt.

### 1-1. Complete the **lab4.tcl** file.

- 1-1-1. Browse to the `C:\Xilinx_trn\HLS_2020\lab4_port_level_protocol` directory using Windows Explorer and open **lab4.tcl** with your preferred text editor.
- 1-1-2. Complete the following sections in the file opened:
  - Project name (*lab4\_prj*)
  - Top-level function for synthesis (*lab4*)
  - Source files (*./source/lab4.c*)
  - Test bench (*./source/lab4\_test.c*)
  - A solution name (*sol1*)
  - Xilinx device (*xa7a12tcsq325-1Q*)
  - Clock period (*6*)
  - Clock uncertainty (*0.1*)
  - Run the C simulation
  - Run the C synthesis
  - Run the C/RTL Cosimulation using the following options: *-trace\_level all -tool xsim*

An example of the **lab4.tcl** file completed is on the figure.

```
1 # Insert the command to create new project
2 open_project -reset lab4_prj
3
4 # Insert the command to add design file
5 add_files ./source/lab4.c
6
7 # Insert the command to specify the top-level function
8 set_top lab4
9
10 # Insert the command to add testbench file
11 add_files -tb ./source/lab4_test.c
12
13 #solution sol1 ( default)
14 # Insert the command to create the solution sol1
15 open_solution -reset "sol1"
16
17 # Insert the command to associate the device to the solution
18 set_part {xa7a12tcsg325-1Q}
19
20 # Insert the command to associate clock period to the solution
21 create_clock -period 6 -name clk
22 set_clock_uncertainty 0.1
23
24 # Insert the command to run C simulation
25 csim_design -clean
26
27 # Insert the command to Synthesize the design
28 csynth_design
29
30 # Insert the command to perform C/RTL Cosimulation
31 cosim_design -trace_level all -tool xsim
32
33 exit
```

Figure 4:lab4.tcl file with the base solution

1-1-3. Save and close the **lab4.tcl** file.

**1-2. Launch the Vivado HLS tool command prompt and the lab3.tcl file.**

1-2-1. For Windows 10: **Start > Xilinx Design Tools > Vivado HLS 2020.1 Command Prompt.**

1-2-2. Enter the following command to change the directory to the current working directory:

```
cd C:\Xilinx_trn\HLS_2020\lab4_port_level_protocol
```

1-2-3. Enter the following command in the Vivado HLS tool command prompt to run the lab4.tcl file:

```
vivado_hls -f lab4.tcl
```

**Note:** Observe that the C and C/RTL co-simulation simulations passed.

1-2-4. Enter the following command to open the Vivado HLS tool project:

```
vivado_hls -p lab4_prj
```

The Vivado HLS tool GUI opens.

## Verifying the Default I/O Protocols

## Step 2

In this step, you will verify the default I/O protocols generated by the tool.

The types of I/O protocol that you can add to C function arguments by interface synthesis depends on the argument type.

The possible options for each function argument are described below.

	Argument Type	Scalar		Array			Pointer or Reference			HLS:: Stream
		Input	Return	I	I/O	O	I	I/O	O	
Block-Level Protocol	ap_ctrl_none									
	ap_ctrl_hs		D							
	ap_ctrl_chain									
AXI Interface Protocol	axis									
	s_axilite									
	m_axi									
No I/O Protocol	ap_none	D					D			
	ap_stable									
Wire Handshake Protocol	ap_ack									
	ap_vld								D	
	ap_ovld							D		
	ap_hs									
Memory Interface Protocol: RAM : FIFO	ap_memory			D	D	D				
	bram									
	ap_fifo									D
Bus Protocol	ap_bus									

Supported D = Default Interface
  Not Supported

Figure 4-5: I/O Protocol Types

Function Argument	I/O Protocol Options
<i>in_s</i>	<p>This is pass-by-value argument that can be implemented with the following I/O protocols:</p> <ul style="list-style-type: none"><li>• <i>ap_none</i>: No I/O protocol. <b>This is the default for inputs.</b></li><li>• <i>ap_stable</i>: No I/O protocol.</li><li>• <i>ap_ack</i>: Implemented with an associated output acknowledge port.</li><li>• <i>ap_vld</i>: Implemented with an associated input valid port.</li><li>• <i>ap_hs</i>: Implemented with both input valid and output acknowledge ports.</li></ul>
<i>*in_p</i>	<p>This is a pass-by-reference input that can be implemented with the following I/O protocols:</p> <ul style="list-style-type: none"><li>• <i>ap_none</i>: No I/O protocol. <b>This is the default for inputs.</b></li><li>• <i>ap_stable</i>: No I/O protocol.</li><li>• <i>ap_ack</i>: Implemented with an associated output acknowledge port.</li><li>• <i>ap_vld</i>: Implemented with an associated input valid port.</li><li>• <i>ap_hs</i>: Implemented with both input valid port and output acknowledge ports</li><li>• <i>ap_fifo</i>: A FIFO interface with associated output write and input FIFO full ports.</li><li>• <i>ap_bus</i>: A Vivado HLS tool bus interface protocol.</li></ul>

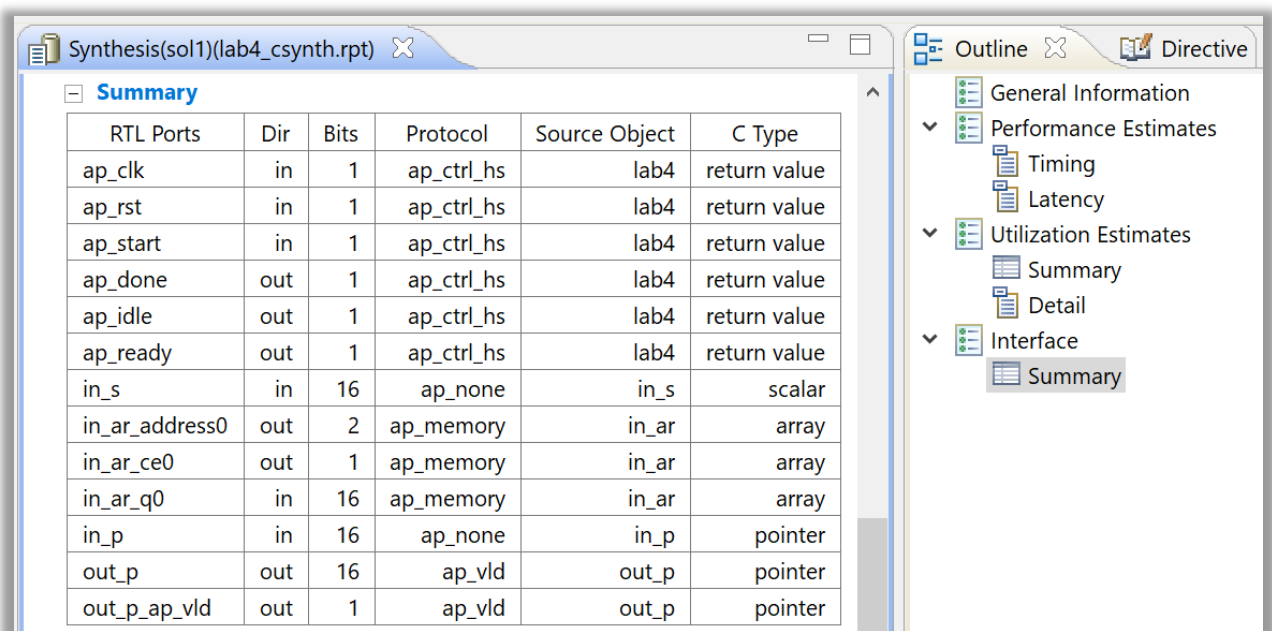
<i>in_arr[N]</i>	<p>This is an input array argument that can be implemented with the following I/O protocols:</p> <ul style="list-style-type: none"><li>• <i>ap_memory</i>: This is a standard block RAM interface with data, address, chip-enable, and write-enable ports. The interface can be implemented as a single-port or dual-port interface. If the Vivado® HLS tool can determine that using a dual-port interface will reduce the initial interval, it will automatically implement a dual-port interface. The RESOURCE directive is used to specify the memory resource and if this directive is specified on the array with a single-port block RAM, a single-port interface will be implemented. Conversely, if a dual-port interface is specified using the RESOURCE directive and the Vivado HLS tool determines that this interface provides no benefit, it will automatically implement a single-port interface. <b>This is the default for inputs.</b></li><li>• <i>bram</i>: The bram interface mode is functionally identical to the <i>ap_memory</i> interface. The only difference is how the ports are implemented when the design is used in the Vivado IP integrator:<ul style="list-style-type: none"><li>• An <i>ap_memory</i> interface is displayed as multiple and separate ports.</li><li>• A <i>bram</i> interface is displayed as a single grouped port that can be connected to a Xilinx block RAM using a single point-to-point connection.</li></ul></li><li>• <i>ap_fifo</i>: If the array is accessed in a sequential manner an <i>ap_fifo</i> interface can be used. The Vivado HLS tool will reject it if the tool determines that the data access is not sequential, report a warning if it cannot determine if the access is sequential, or issue no message if it determines the access is sequential. The <i>ap_fifo</i> interface can only be used for reading or writing, not both.</li></ul>
------------------	---



<i>*out_p</i>	<p>This is a pass-by-reference output that can be implemented with the following I/O protocols:</p> <ul style="list-style-type: none"> <li>• <i>ap_none</i>: No I/O protocol.</li> <li>• <i>ap_ack</i>: Implemented with an associated input acknowledge port.</li> <li>• <i>ap_vld</i>: Implemented with an associated output valid port. This is the default for outputs. <b>This is the default for outputs.</b></li> <li>• <i>ap_ovld</i>: Implemented with an associated output valid port.</li> <li>• <i>ap_hs</i>: Implemented with both output valid port and input acknowledge ports</li> <li>• <i>ap_fifo</i>: A FIFO interface with associated output write and input FIFO full ports.</li> <li>• <i>ap_bus</i>: A Vivado HLS tool bus interface protocol.</li> </ul>
---------------	--

## 2-1. View the Synthesis report.

2-1-1. Select **Interface** in the Outline pane to review the Interface Summary.

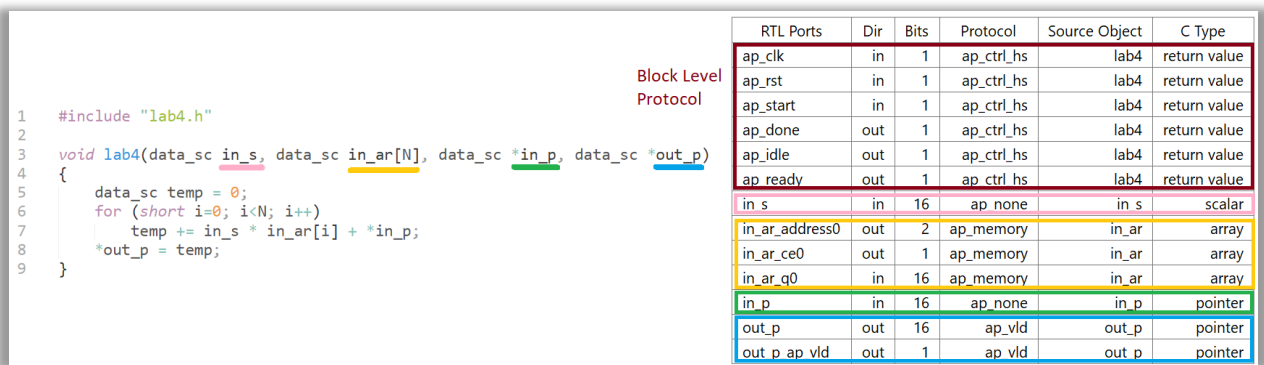


**Figure 4-6: Interface Summary Report**

The Interface Summary shows how the arguments in the C source are using the default protocols:

- The design has a *clock*, *reset*, and the default block-level I/O protocol *ap\_ctrl\_hs*.
- Input data (*in\_s*), which is an input scalar, has the port-level I/O protocol *ap\_none*.

- Input array (*in\_ar*) argument has been synthesized to a RAM port (I/O protocol *ap\_memory*) with the following ports:
  - A data port (*in\_ar\_q0*).
  - An address port (*in\_ar\_address0*).
  - Control port of a chip-enable (*in\_ar\_ce0*).
- Input data (*in\_p*), which is input pointer, has the port-level I/O protocol *ap\_none*.
- Output data (*out\_p*), which is output pointer, has been synthesized to a Port-Level I/O protocol *ap\_vld* which is a subset of the *ap\_hs* interface type. The *ap\_vld* Port-level I/O protocol provides the following signals:
  - Data port (*out\_p*)
  - Valid signal to indicate when data is ready (*out\_p\_ap\_vld*).



**Figure 4-7: Interface Summary Report - Block-Level and Port-Level I/O Protocols for the solution sol1**

2-1-2. Select **Analysis** perspective.

2-1-3. Open Performance Profile.

Performance Profile		Resource Profile			
	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
lab4	-	21	-	22	-
Loop 1	no	20	5	-	4

**Figure 4-8: Performance Profile Report**

### Question 1

How many cycles are for the following delays?

Trip Count -

Iteration Latency -

Loop Latency -

Function Latency -

Initiation interval -

- 2-1-4. Open Scheduler=>Scheduler Viewer and check how the function scheduled in comparison with the source code and with the Performance Profile figures.

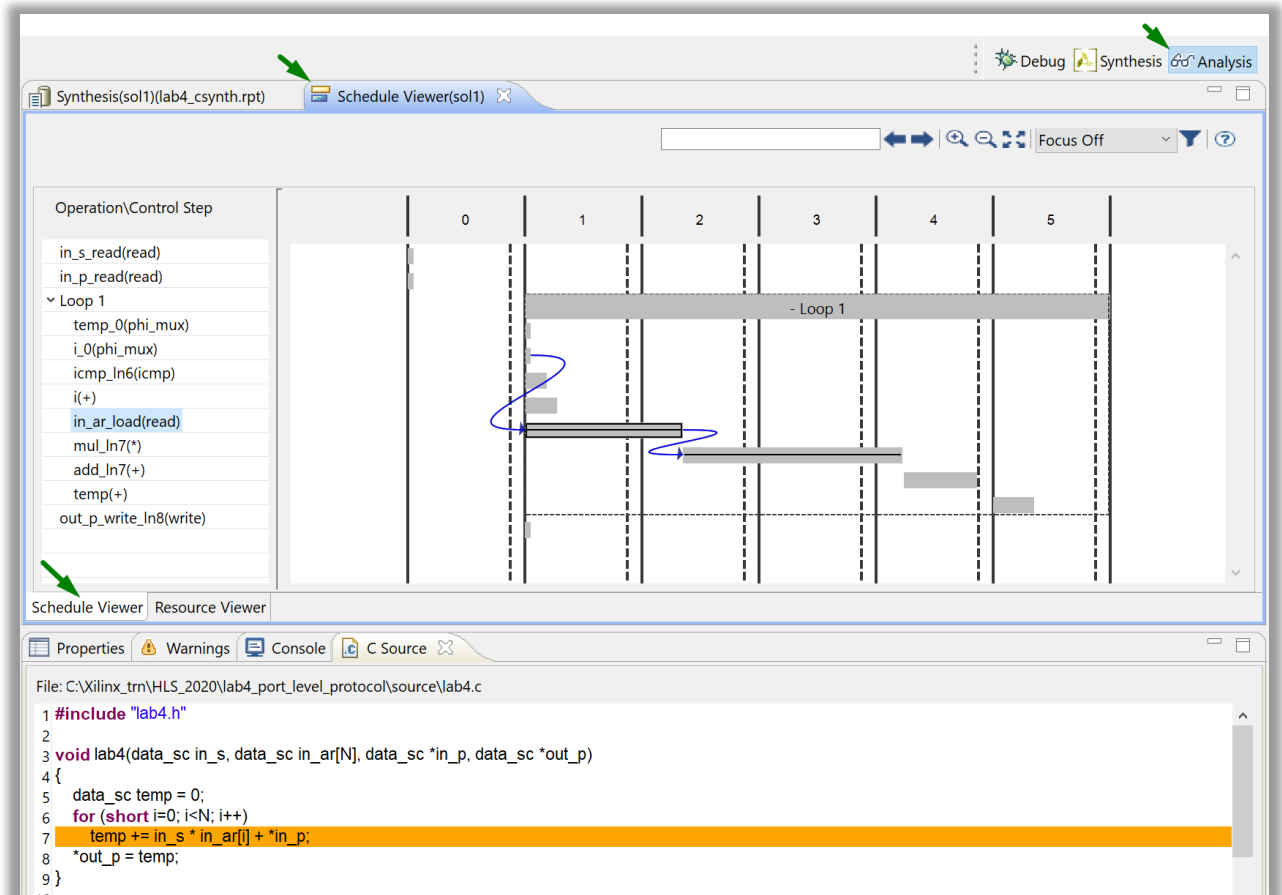


Figure 4-9: Scheduler Viewer Report

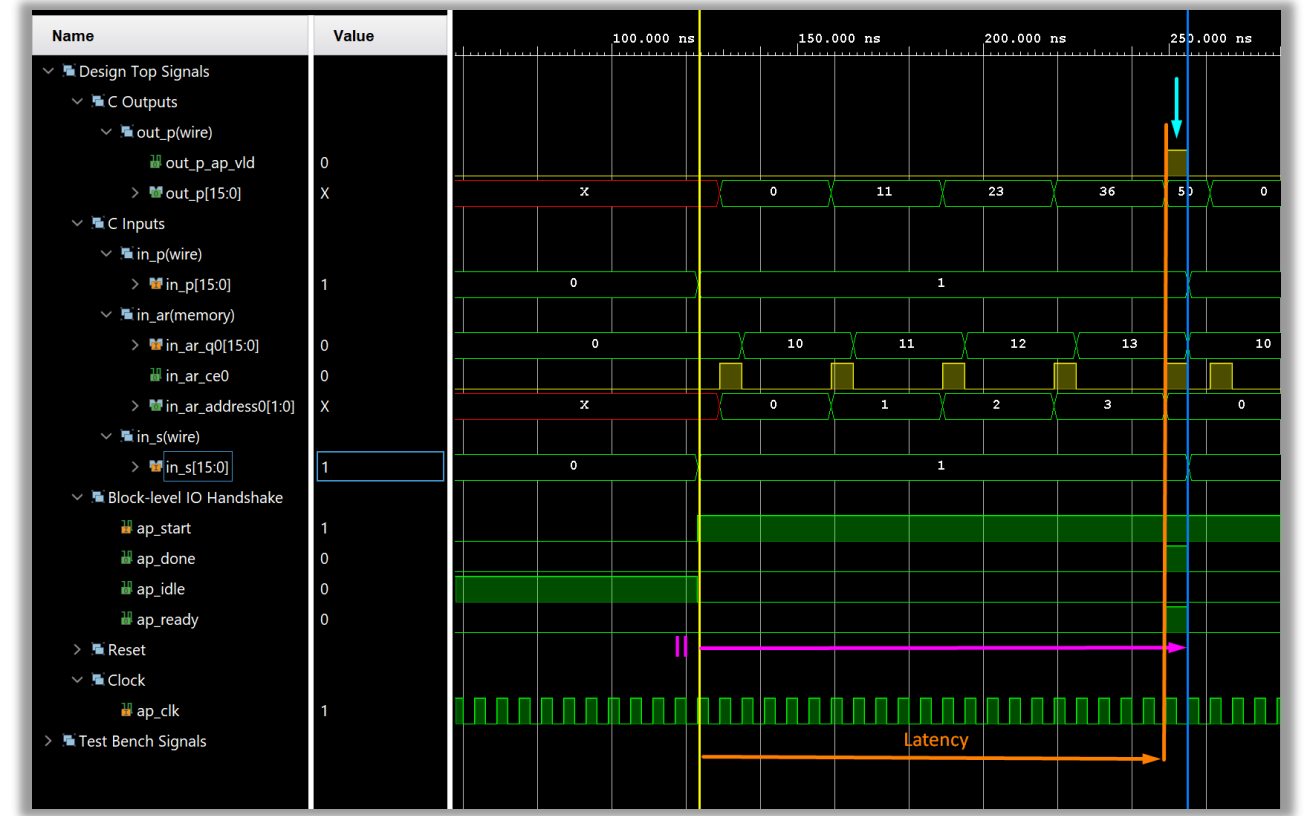
- 2-2. View the trace files that has been generated for the solution sol1.

- 2-2-5. Click the Open Wave Viewer (  ) button from the toolbar.

Alternatively, select **Solution > Open Wave Viewer**.

2-2-6. Expand the Design Top Signals as shown on the figure below.

2-2-7. Change Radix for the *out\_p*, *in\_p*, *in\_ar\_q0*, *in\_s* ports to UNSIGNED DECIMAL as shown below.



**Figure 4-10: Wave Viewer for the solution sol1**

2-2-8. Check and compare Latency and Iteration Interval with Performance profile figures.

## Questions 2

Are there any control signals for the ports in\_s and in\_p? Explain why it is so.


Are there any control signals for the ports in\_ar? Explain how the protocol works.

Is there a control signal for the ports out\_p? Explain how the protocol works.

## Specifying the Port-Level I/O Protocols

## Step 3

3-1. Create new solution and specify the I/O protocols for the ports *in\_ar[N]*.

3-1-1. Select **Project** > **New Solution** or click the **New Solution** icon (  ) from the toolbar.

3-1-2. Enter the Name of the new solution as **sol2**.

3-1-3. Leave the other options at their default settings.

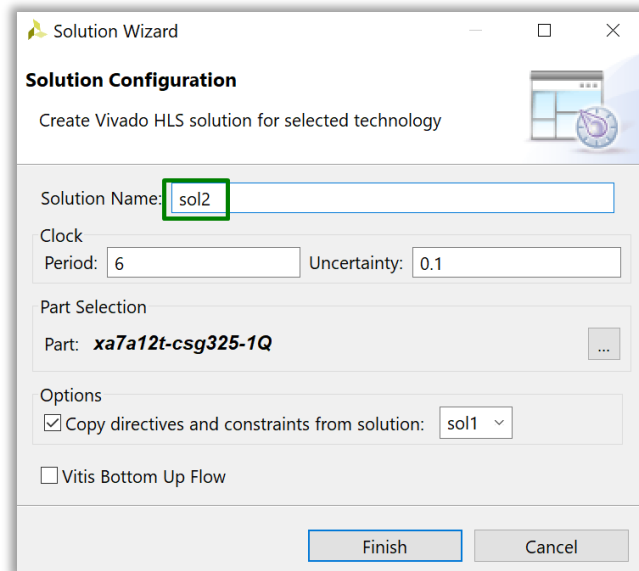


Figure 4-11: Setting a new solution sol2

3-1-4. Click **Finish**.

3-2. Specify the I/O protocol for the port *in\_ar*.

3-2-1. Ensure that the **lab4.c** file is opened and Directive tab is selected.

3-2-2. Right-click **in\_ar** in the Directive tab and select **Insert Directive**.

3-2-3. Select **INTERFACE** from the Directive drop-down list.

3-2-4. Select **ap\_fifo** from the mode (optional) drop-down list.

3-2-5. Select **Directive File** in the Destination field.

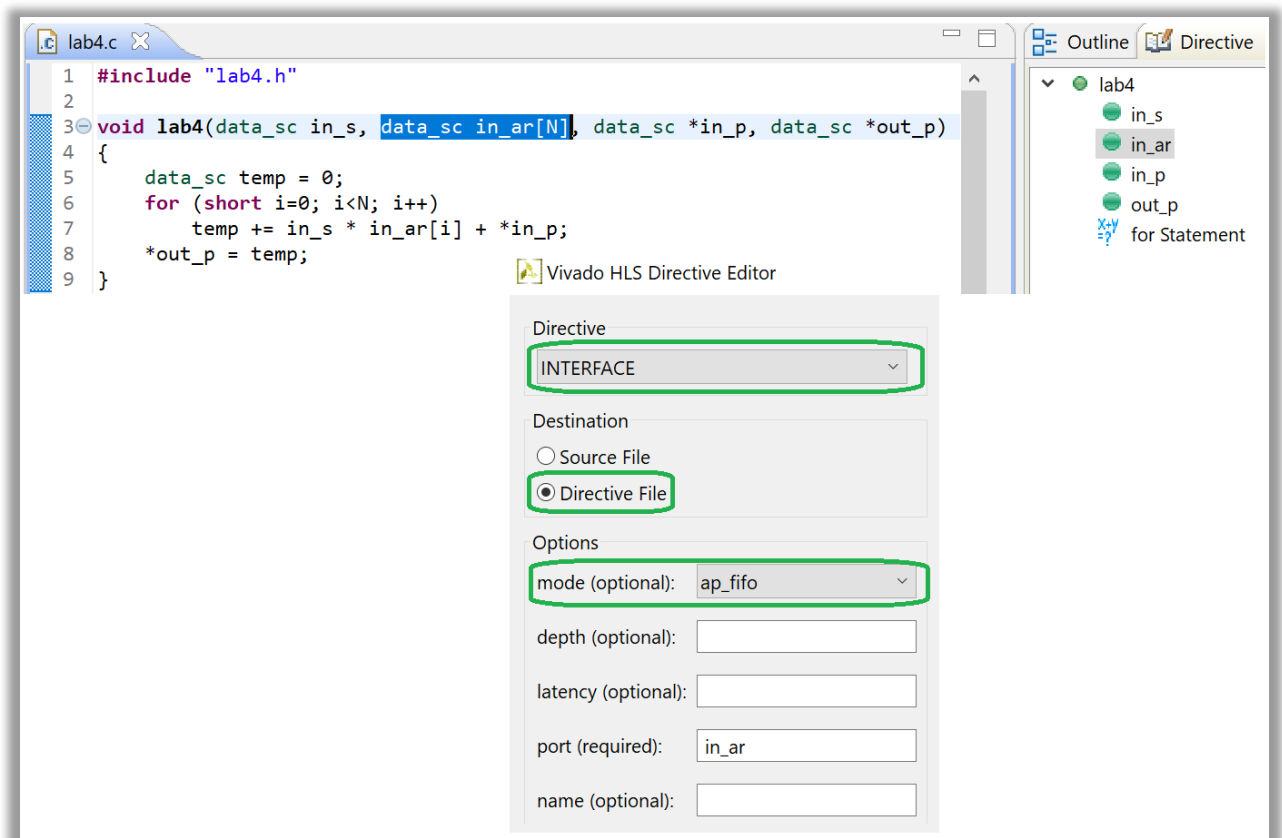


Figure 4-12: Selecting the Interface Type `ap_fifo` for the `in_ar` port

3-2-6. Click **OK**.

3-2-7. Make sure that the final view of the INTERFACE directives of the ports in the Directive tab is similar to the figure below.

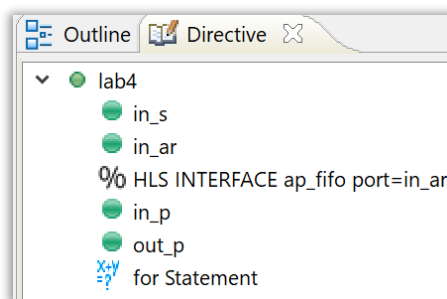


Figure 4-13: The final view of the Directive Tab

3-2-8. Expand the **lab4\_prj** > **sol2** > **Constraints** folder in the Explorer pane.

3-2-9. Double-click the **directives.tcl** file to open it.

You should see that the directive added through the GUI is now in the *directives.tcl* file.

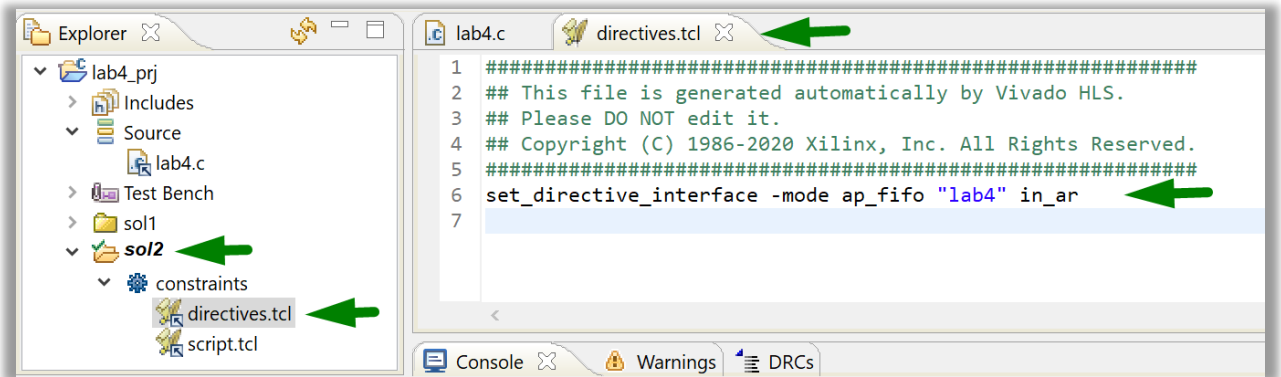


Figure 4-14: Opening the directives.tcl file

### 3-3. Synthesize the Vivado HLS design.

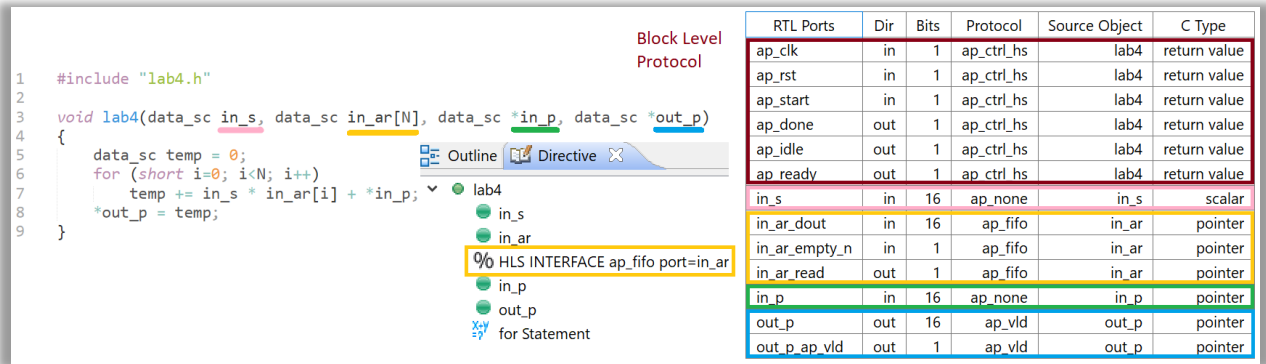
3-3-1. Select **Solution** > **Run C Synthesis** > **Active Solution** or click the **Run Synthesis** icon in the menu bar.

### 3-4. View the Synthesis report.

*You should see that the Synthesis report in the Information window opens automatically when synthesis is completed.*

3-4-1. Select **Interface** in the Outline pane to review the Interface Summary:

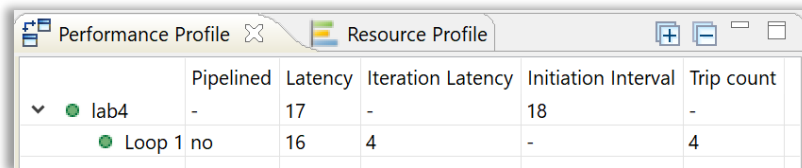
- The design has a *clock*, *reset*, and the default block-level I/O protocol *ap\_ctrl\_hs*.
- Input data (*in\_s*), which is an input scalar, has the port-level I/O protocol *ap\_none*.
- Input array (*in\_ar*) argument has been synthesized to FIFO port (I/O protocol *ap\_fifo*) with the following ports:
  - A data port (*in\_ar\_dout*).
  - An *fifo\_empty* signal with active level zero (*in\_ar\_empty\_n*).
  - A read *fifo* signal (*in\_ar\_read*).
- Input data (*in\_p*), which is input pointer, has the port-level I/O protocol *ap\_none*.
- Output data (*out\_p*), which is output pointer, has been synthesized to a Port-Level I/O protocol *ap\_vld* which is a subset of the *ap\_hs* interface type. The *ap\_vld* Port-level I/O protocol provides the following signals:
  - Data port (*out\_p*)
  - Valid signal to indicate when data is ready (*out\_p\_ap\_vld*).



**Figure 4-15: Interface Summary Report - Block-Level and Port-Level I/O Protocols for the solution sol2**

3-4-2. Select **Analysis** perspective.

3-4-3. Open Performance Profile to review the figures.



**Figure 4-16: Performance Profile Report for the solution sol2**

3-4-4. Open Scheduler=>Scheduler Viewer and check how the function scheduled in comparison with the source code and with the Performance Profile figures.



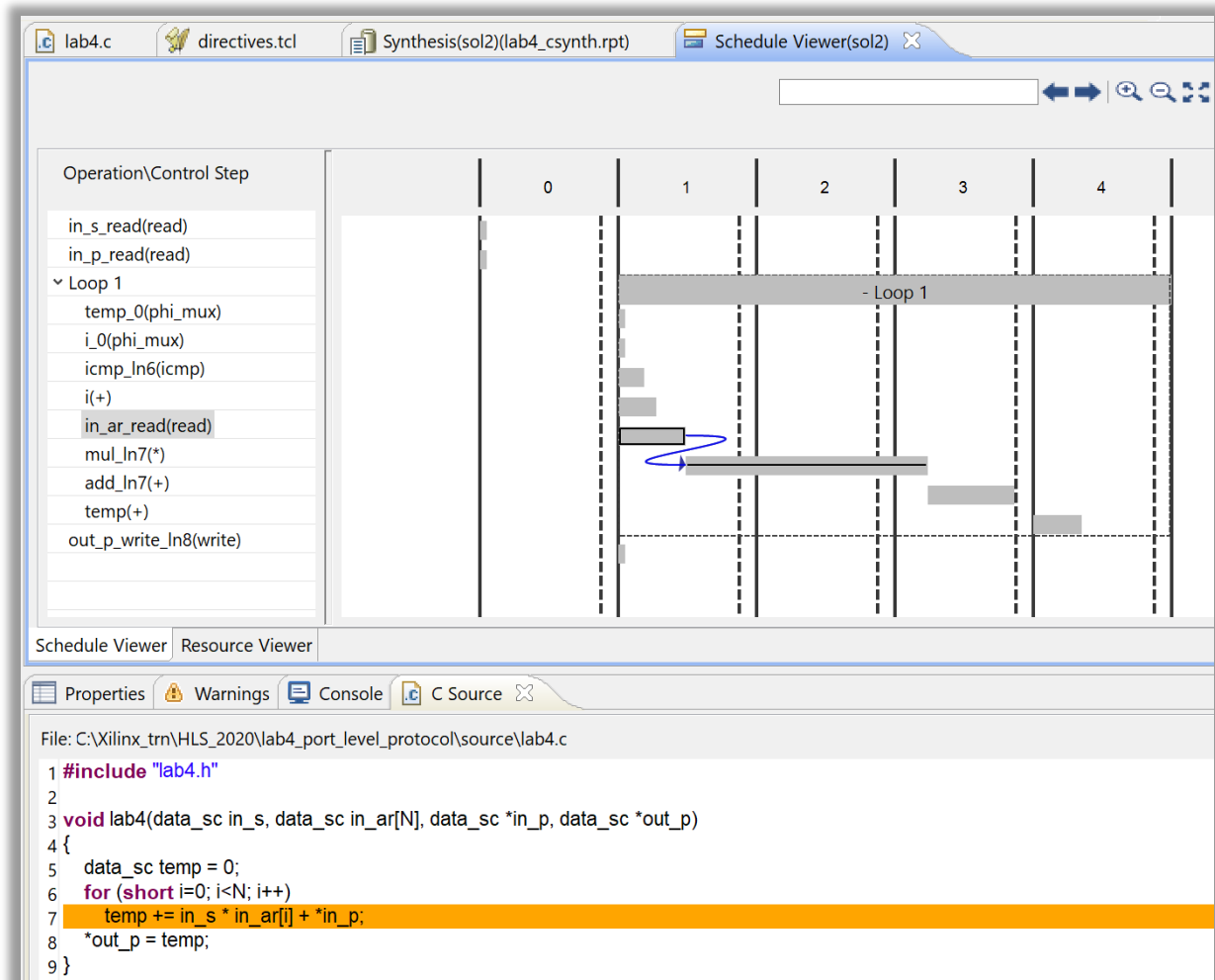


Figure 4-17: Scheduler Viewer Report for the solution sol2

### 3-5. Generate and view Waves for the solution sol2 in the Vivado simulator.

3-5-5. Change perspective to Synthesis: click the Synthesis Tap.

3-5-6. Launch Co-Simulation procedure: select **Solution** > **Run C/RTL Cosimulation**.

3-5-7. In a window appeared click OK button.

Wait until the Co-Simulation procedure is finished.

3-5-8. Click the Open Wave Viewer (  ) button from the toolbar.

Alternatively, select **Solution** > **Open Wave Viewer**.

This will open the Vivado IDE with the RTL waveforms traces.

- 3-5-9. Expand the Design Top Signals as shown below and change Radix for the *out\_p*, *in\_p*, *in\_ar\_dout*, *in\_s* ports to UNSIGNED DECIMAL as shown below.

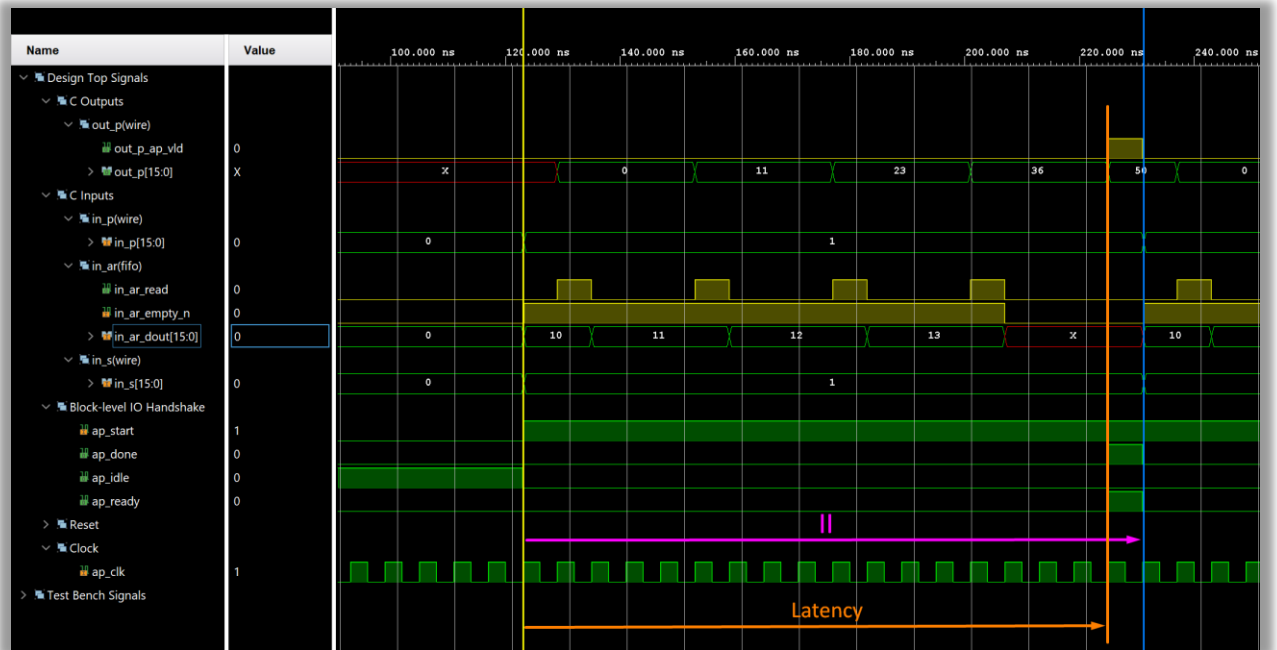


Figure 4-18: Wave Viewer for the solution sol2

- 3-5-10. Check and compare Latency and Iteration Interval with Performance profile figures.

### Question 3


What are the differences in the control signals for the ports *in\_ar* in the solutions sol1 and sol2? How the protocol works in sol2?

---



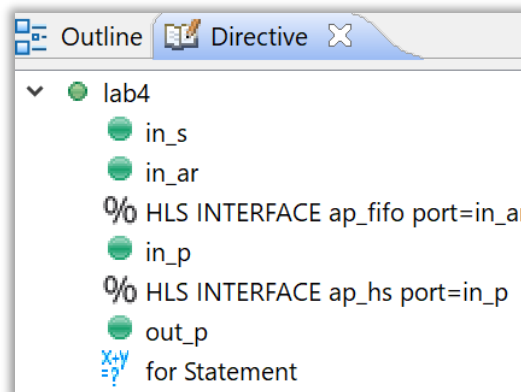
---

**3-6. Create a new solution and specify the I/O protocols for the port *in\_ar[N]*.**

- 3-6-1. In Vivado HLS tool GUI select **Project > New Solution** or click the **New Solution** icon (  ) from the toolbar.
- 3-6-2. Enter the Name of a new solution – sol3.
- 3-6-3. Leave the other options at their default settings.
- 3-6-4. Click **Finish**.

**3-7. Specify the I/O protocols for the port *in\_p*.**

- 3-7-1. Ensure that the **lab4.c** file is opened and Directive tab is selected.
- 3-7-2. Right-click **in\_p** in the Directive tab and select **Insert Directive**.
- 3-7-3. Select **INTERFACE** from the Directive drop-down list.
- 3-7-4. Select **ap\_hs** from the mode (optional) drop-down list.
- 3-7-5. Select **Directive File** from the Destination field.
- 3-7-6. Click **OK**.
- 3-7-7. Make sure that the final view of the INTERFACE directives of the ports in the Directive tab is similar to the figure below.



**Figure 4-19: The final View of the Directive Tab for the solution sol3**

- 3-7-8. Expand the **lab4\_prj > sol3 > Constraints** folder in the Explorer pane.
- 3-7-9. Double-click the **directives.tcl** file to open it.

You should see that the directives added through the GUI are now in the *directives.tcl* file.

```
1 #####
2 ## This file is generated automatically by Vivado HLS.
3 ## Please DO NOT edit it.
4 ## Copyright (C) 1986-2020 Xilinx, Inc. All Rights Reserved.
5 #####
6 set_directive_interface -mode ap_fifo "lab4" in_ar
7 set_directive_interface -mode ap_hs "lab4" in_p
```

**Figure 4-20: Opening the directives.tcl file for the solution sol3**

### 3-8. Synthesize the Vivado HLS design.

3-8-1. Select **Solution** > **Run C Synthesis** > **Active Solution** or click the **Run Synthesis** icon in the menu bar.

### 3-9. View the Synthesis report.

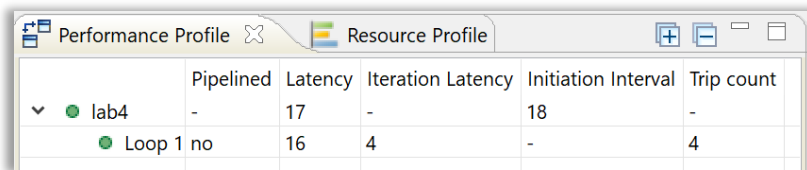
*You should see that the Synthesis report in the Information window opens automatically when synthesis is completed.*

3-9-1. Select **Interface** in the Outline pane to review the Interface Summary:

- Pay attention on in\_p port I/O Protocol Interface. It should be ap\_hs.

3-9-2. Select **Analysis** perspective.

3-9-3. Open Performance Profile review the figures.



	Pipelined	Latency	Iteration Latency	Initiation Interval	Trip count
lab4	-	17	-	18	-
Loop 1	no	16	4	-	4

**Figure 4-21: Performance Profile Report for the solution sol3**

### 3-10. Generate and view Waves for the solution sol2 in the Vivado simulator.

3-10-4. Change perspective to Synthesis: click the Synthesis Tap.

3-10-5. Launch Co-Simulation procedure: select **Solution** > **Run C/RTL Cosimulation**.

Wait until Co-Simulation procedure is finished.

3-10-6. Click the Open Wave Viewer (  ) button from the toolbar.

Alternatively, select **Solution** > **Open Wave Viewer**.

3-10-7. In a window appeared click OK button.

This will open the Vivado IDE with the RTL waveforms traces.

3-10-8. Expand the Design Top Signals as shown below and change Radix for the *out\_p*, *in\_p*, *in\_ar\_dout*, *in\_s* ports to UNSIGNED DECIMAL as shown below.

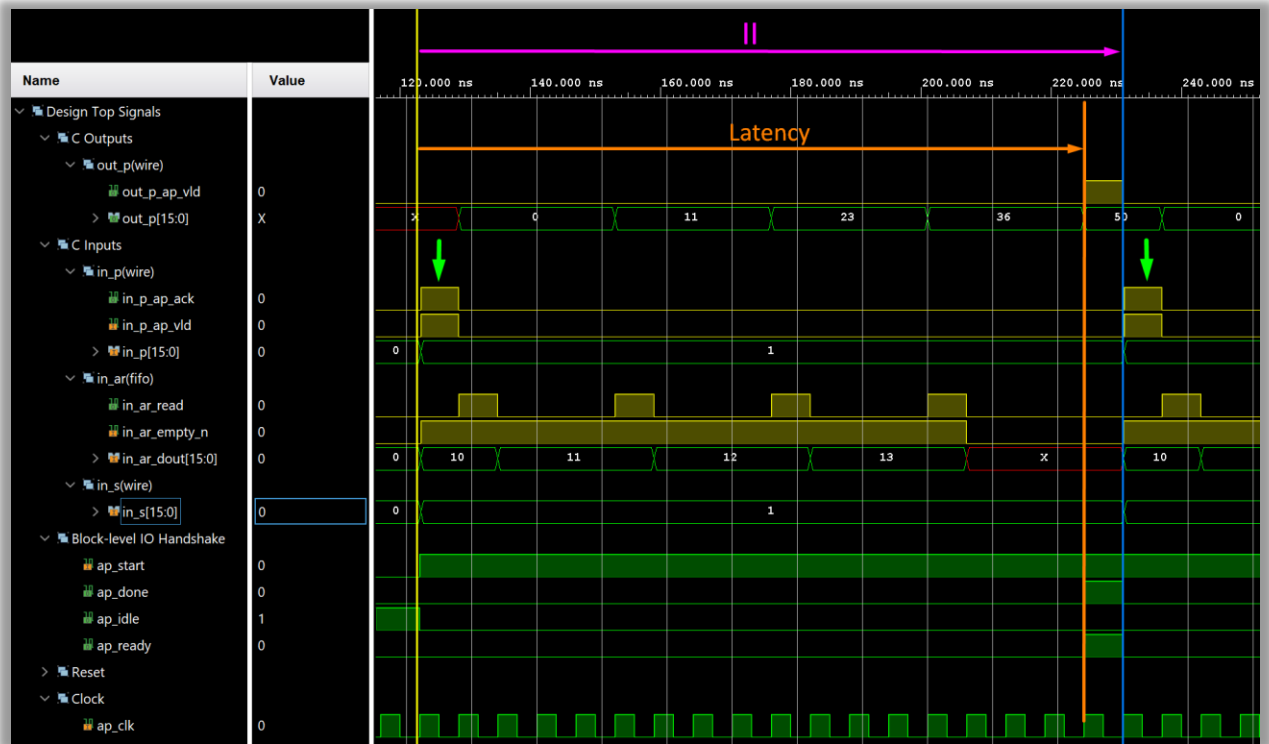


Figure 4-22: Interface Summary Report - Block-Level and Port-Level I/O Protocols

3-10-9. Check and compare Latency and Iteration Interval with Performance profile figures.

### Questions 4

Why there are the differences in the control signals for the port *in\_p* and port *in\_s*? What are the differences?

---



---

What are there the differences in the control signals for the ports *in\_p* in the solutions sol2 and sol3?

---



---

## Compare the solutions

## Step 4

---

### 4-1. Select all solutions for compare.

4-1-1. In Vivado HLS tool GUI select **Project > Compare Reports**.

4-1-2. In the window appeared select all solutions for comparing and click OK.

### 4-2. Complete the spreadsheet and select the optimal solution.

4-2-1. Use data from the *compare\_report* tab to complete the spreadsheet (A template of the spreadsheet is in file *./source/lab4\_ex.xlsx*).

Pay attention that Latency (ns) is calculated by multiplying Clock estimated (ns) on Latency (cycles).

4-2-2. Construct the diagram by using Latency (ns) and FF, LUT resources (DSP, BRAM18, URAM have constant values and can be omitted).

### Question 5

Explain which solution, in your opinion, is the best (criteria is max performance and min resources) and why do you think so.

---

---

4-2-3. Select **File > Exit** to close the Vivado HLS tool.

4-2-4. Close the Vivado HLS Command Prompt.

## Summary

In this lab, you learned how to apply directives to implement different types of port-Level I/O protocols. You also reviewed the Interface Summary to see the types of port-level I/O protocols that were implemented.