XILINX

Introduction to I/O Interfaces

2021.1

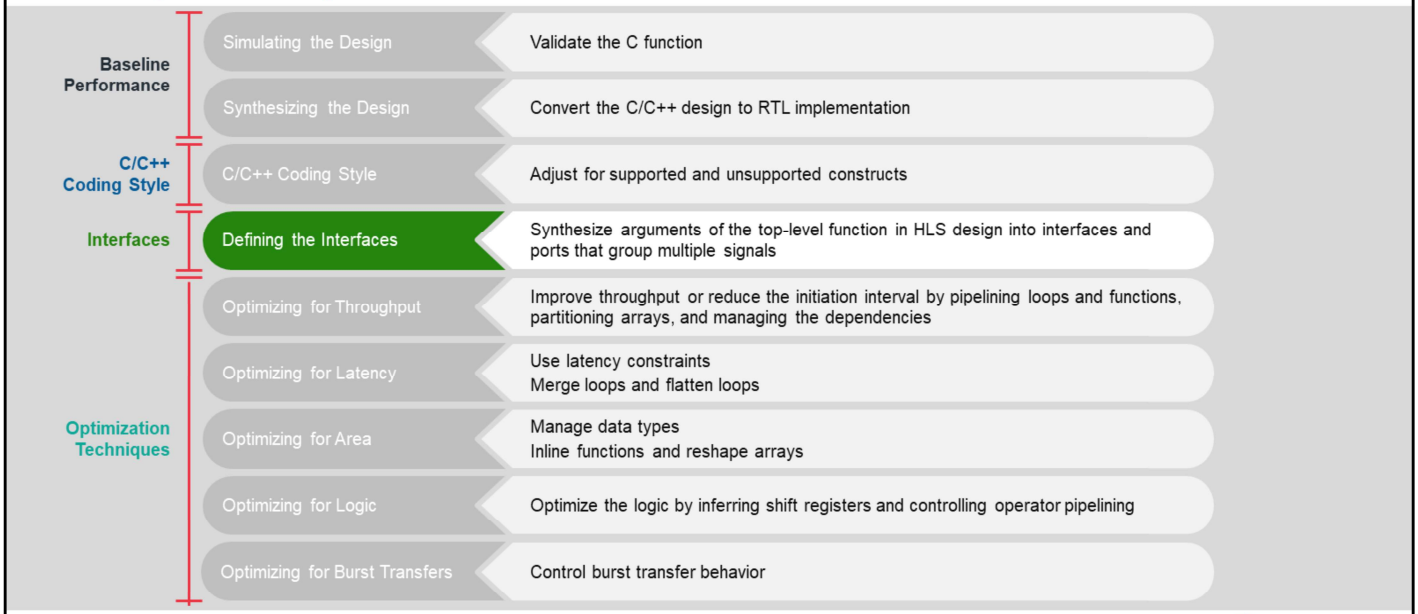
Objectives

After completing this module, you will be able to:

- List the types of I/O abstracted in the Vitis™ HLS tool
- Distinguish between block-level and port-level I/O protocols

HLS Design Methodology

First optimization step is to define the interfaces



XILINX.

Once the functionality of the design is verified and the design is synthesized, the very first optimization step to do is to define the interfaces.

HLS Design Methodology

What is the interface?

The interface is defining the I/O protocol for each argument in the top-level function

The HLS tool will choose the type of I/O protocol based on the data type of the function arguments

User can override the default I/O protocol using the INTERFACE directive

```
void fir ( data_t*y,
coef_t c[4],
data_t x)
```

XILINX.

Let's first understand what is the interface.

The interface is defining the I/O protocol for each argument in the top-level function. By default, the HLS tool will choose the type of I/O protocol based on the data type of the function arguments.

However, the user can override the default I/O protocol using the INTERFACE directive.

Key Attributes of C Code

Functions:
Entire code is made up of functions that represent the design hierarchy; the same in the hardware

Inputs/Outputs:
Arguments of a top-level function are transformed into input/output interfaces of the hardware

Types:
All the variables are of a defined type and this type can influence the area and the performance of a design

```
void fir (
  data_t *y,
  coef_t c[4],
  data_t x
  ) {
  static data_t shift_reg[4];
  acc_t acc;
  int i;
acc=0;
loop: for (i=3;i>=0;i--) {
  if (i==0) {
    acc+=x*c[0];
    shift_reg[0]=x;
  } else {
    shift_reg[i]=shift_reg[i-1];
    acc+=shift_reg[i] *  c[i];
}
  }
  *y=acc>>2;
}
```


There are various key attributes of a C code. Let's define these one by one.

- Functions: The entire code is made up of functions that represent the design hierarchy; the same in the hardware.
- Input/outputs: The arguments of a top-level function are transformed into input/output interfaces of the hardware.
- Types: All the variables are of a defined type and this type can influence the area and the performance of a design.

Key Attributes of C Code

Loops:
Functions typically contain loops
Handling loops can have an impact on area and performance

Arrays:
Arrays can impact the device area and become performance bottlenecks

Operators:
Operators can be shared to control the area and can be assigned to the specific hardware implementations to meet performance

```
void fir (
  data_t *y,
  coef_t c[4],
  data_t x
  ) {
  static data_t shift_reg[4];
  acc_t acc;
  int i;
acc=0;
loop: for (i=3;i>=0;i--) {
  if (i==0) {
    acc+=x*c[0];
    shift_reg[0]=x;
  } else {
    shift_reg[i]=shift_reg[i-1];
    acc+=shift_reg[i] * c[i];
}
  }
  *y=acc>>2;
}
```


XILINX.

- Loops: Functions typically contain loops. How these loops are handled can have an impact on area and performance.
- Arrays: These are used often in C code. These can impact the device area and become performance bottlenecks.
- Operators: These in the C code may require sharing to control the area. These operators can be assigned to the specific hardware implementations to meet performance.

Example: Combinatorial Design

Adder Example

```
#include "adders.h"
int adders(int in1, int in2, int in3) {
    int sum;
    sum = in1 + in2 + in3;
    return sum;
}
```

Design synthesized with 100 ns clock

- Clock period of a clock is large
- Allows all the adders to fit in one clock cycle
- Making the design a combinatorial design

```
==========================================
Performance Estimates
==========================================
+ Summary of timing analysis:
* Estimated clock period (ns): 3.45
+ Summary of overall latency (clock cycles):
* Best-case latency:          0
* Average-case latency:       0
* Worst-case latency:         0
```

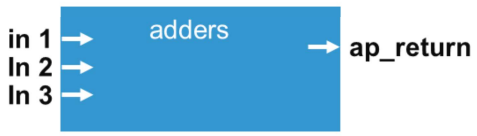

XILINX.

Let's understand the combinatorial design with a simple adder example in which the output is a sum of three input and is synthesized with a 100 ns clock.

Since the clock period of a clock is large, it allows all the adders to fit in one clock cycle, making the design a combinatorial design.

Example: Combinatorial Design

Vitis HLS tool thus creates a combinatorial design with the default port types



All the ports added by the Vitis HLS tool start with a prefix ap_

sum variable → ap_return port

Note: No handshaking signals are required or created for this design by default by the HLS tool

XILINX.

The Vitis HLS tool thus creates a combinatorial design with the default port types. All the ports added by the Vitis HLS tool start with a prefix ap_.

In this C code, the function ‘return’, which is a sum variable, gets transformed into ap_return RTL port.

Note that no handshaking signals are required or created for this design by default by the HLS tool.

Example: Sequential Design

Adder Example

```
#include “adders.h”
int adders(int in1, int in2, int in3) {
    int sum;
    sum = in1 + in2 + in3;
    return sum;
}
```

Design synthesized with 3 ns clock

XILINX.

Let’s understand sequential design through the same adder example.

But this same adder example is now synthesized with 3 ns clock.

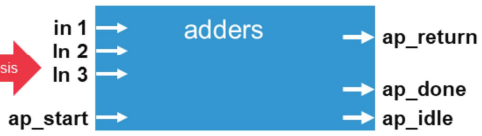
Example: Sequential Design

- Clock period of a clock is small
- Design now takes more than one cycle to complete this addition operation
- Vitis HLS tool creates a sequential design with the default port types

Adder Example

```
#include "adders.h"
int adders(int in1, int in2, int in3) {
      int sum;
      sum = in1 + in2 + in3;
      return sum;
}
```

Synthesis



Block-level handshaking signals are added to this sequential design by default

These block-level handshaking signals created by the HLS tool tell the design:

When to start using: ap_start signal

When the design has completed the operation using: ap_done signal

When the design is idle using: ap_idle signal

XILINX.

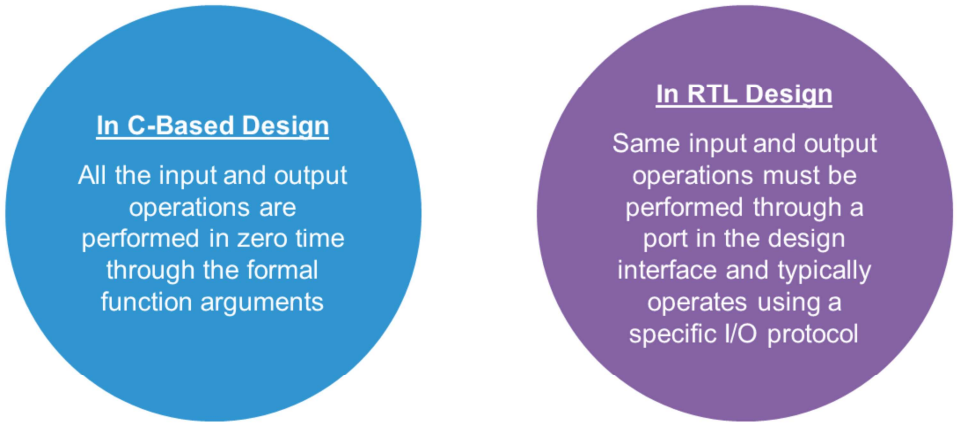
Since, the clock period of a clock is small, thus, the design now takes more than one cycle to complete this addition operation and the Vitis HLS tool creates a sequential design with the default port types.

Similar to the combinatorial design example, the function return gets transformed into ap_return RTL port.

In addition to this signal, block-level handshaking signals are added to this sequential design by default.

These block-level handshaking signals created by the HLS tool tell the design when to start using ap_start signal, when the design has completed the operation using ap_done signal and when the design is idle using ap_idle.

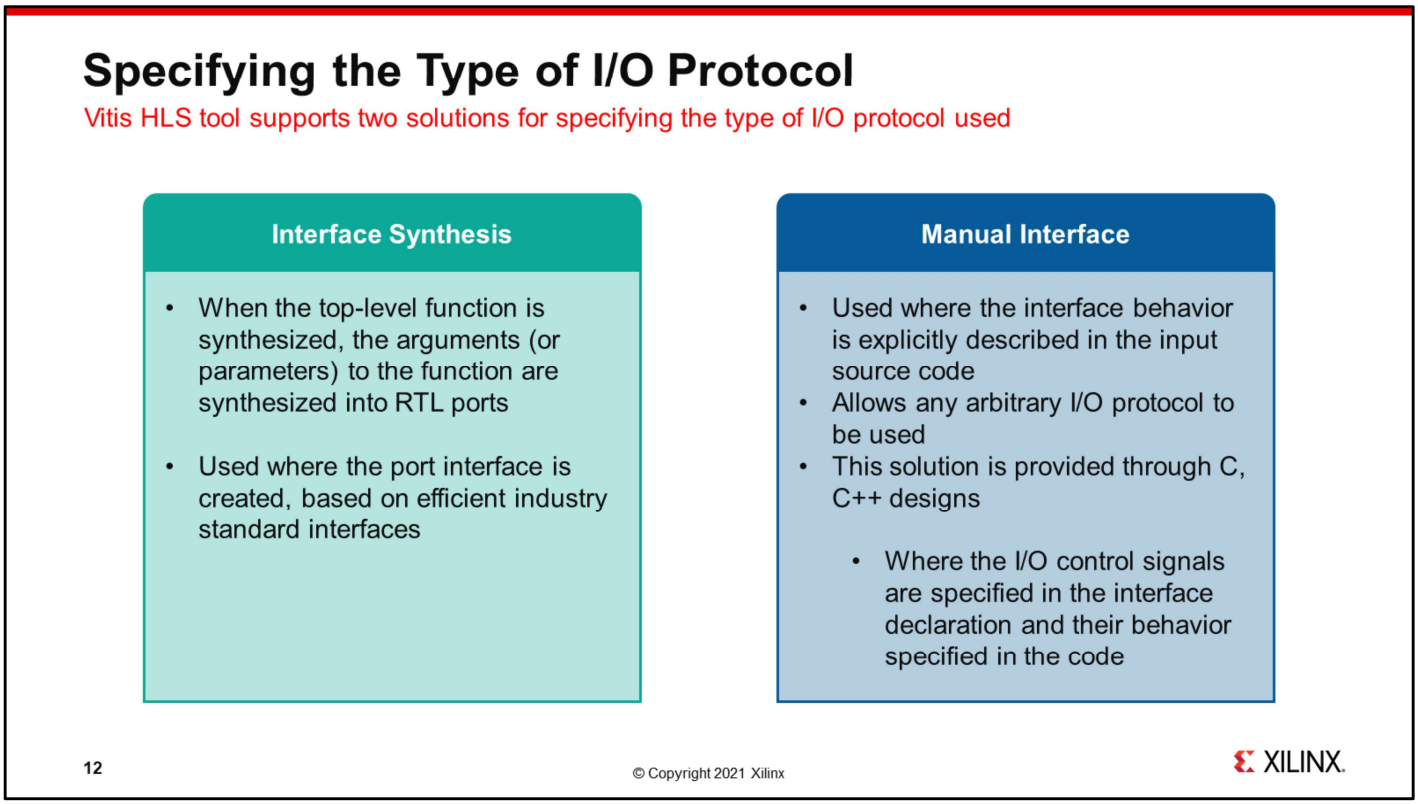
Specifying the Type of I/O Protocol



XILINX.

In C based design, all the input and output operations are performed in zero time through the formal function arguments.

In an RTL design, these same input and output operations must be performed through a port in the design interface and typically operates using a specific input-output protocol.



The Vitis HLS tool supports two solutions for specifying the type of I/O protocol used: interface synthesis and manual interface.

Interface Synthesis
When the top-level function is synthesized, the arguments (or parameters) to the function are synthesized into RTL ports. This process is called interface synthesis. Where the port interface is created, based on efficient industry standard interfaces.

Manual Interface
Manual interface is where the interface behavior is explicitly described in the input source code. It allows any arbitrary I/O protocol to be used. This solution is provided through C and C++ and SystemC designs, using the Vitis HLS tool, where the I/O control signals are specified in the interface declaration and their behavior is specified in the code.

Interface Synthesis

Vitis HLS tool defines interfaces automatically, using industry standards to specify the protocol used

Type of interface depends on:

- Data type
- Direction of the parameters of the top-level function
- Target flow for the active solution
- Default interface configuration settings
- Any specified INTERFACE pragmas or directives

Interface defines three elements of the kernel:

- Channels for data to flow into or out of the HLS design
- Port protocol that is used to control the flow of data through the data channel, defining when the data is valid and when it can be read or written (port-level protocols)
- Execution control scheme for the HLS design, specifying the operation of the kernel or IP as pipelined or sequential (block-level protocols)

13

XILINX.

The Vitis HLS tool defines interfaces automatically, using industry standards to specify the protocol used.

The type of interfaces that the Vitis HLS tool creates depends on the data type and direction of the parameters of the top-level function, the target flow for the active solution, the default interface configuration settings as specified by config_interface, and any specified INTERFACE pragmas or directives.

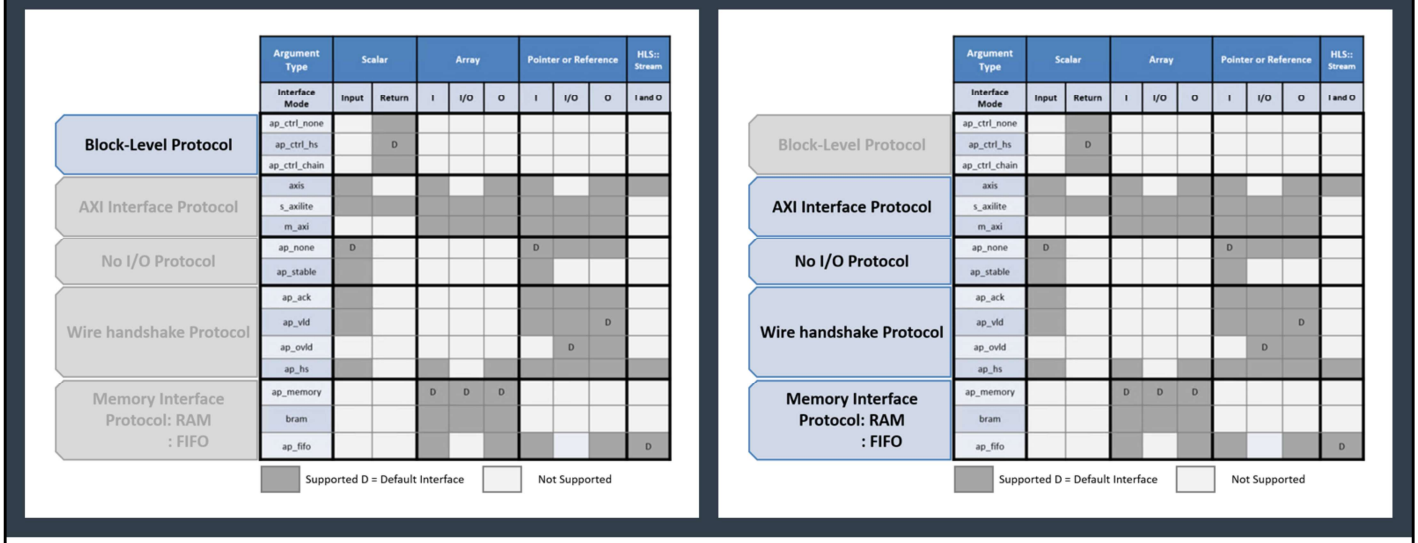
The interface defines three elements of the kernel:

- The interface defines channels for data to flow into or out of the HLS design. Data can flow from a variety of sources external to the kernel or IP, such as a host application, an external camera or sensor, or from another kernel or IP implemented on the Xilinx device. The default channels for Vitis environment kernels are AXI adapters.
- The interface defines the port protocol that is used to control the flow of data through the data channel, defining when the data is valid and when it can be read or written.
- The interface also defines the execution control scheme for the HLS design, specifying the operation of the kernel or IP as pipelined or sequential.

The choice and configuration of interfaces is a key to the success of your design. The Vitis HLS tool tries to simplify the process by selecting default interfaces for the target flows.

Interface Synthesis Protocols

Block-Level Protocol | **Port-Level Protocol**



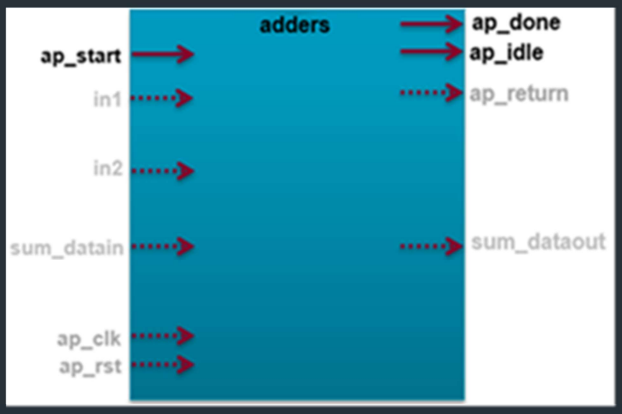
XILINX.

There are two categories of interface protocols supported by the Vitis HLS tool: block-level protocols and port-level protocols.

Interface Synthesis Protocols

Block-Level Protocol

- These signals control the RTL block, independently of any port-level I/O protocols
- By default, a block-level interface protocol is added to the design
- Block-level interface protocols like:
  - ap_ctrl_none
  - ap_ctrl_hs
  - ap_ctrl_chain



XILINX.

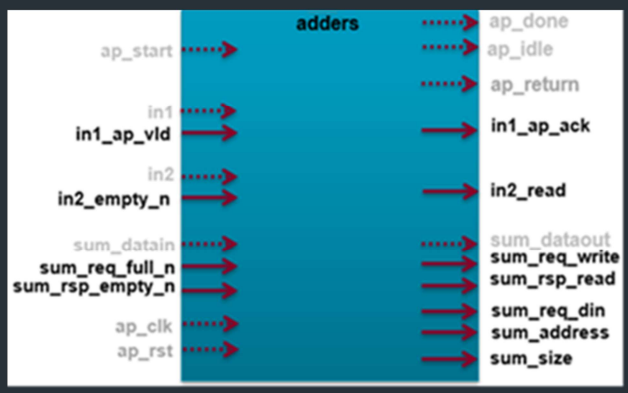
Let's first look at block-level protocols.

These signals control the RTL block, independently of any port-level I/O protocols. By default, a block-level interface protocol is added to the design. There are block level interface protocols like ap_ctrl_none, ap_ctrl_hs, and ap_ctrl_chain which can only be specified on the function or the function return.

Interface Synthesis Protocols

Port-Level Protocol

- The port-level I/O protocol interfaces are basically the data ports
- They are created for each argument in the top-level function and the function return
- I/O protocol created depends on the type of C argument and on the default, type selected by the tool



XILINX.

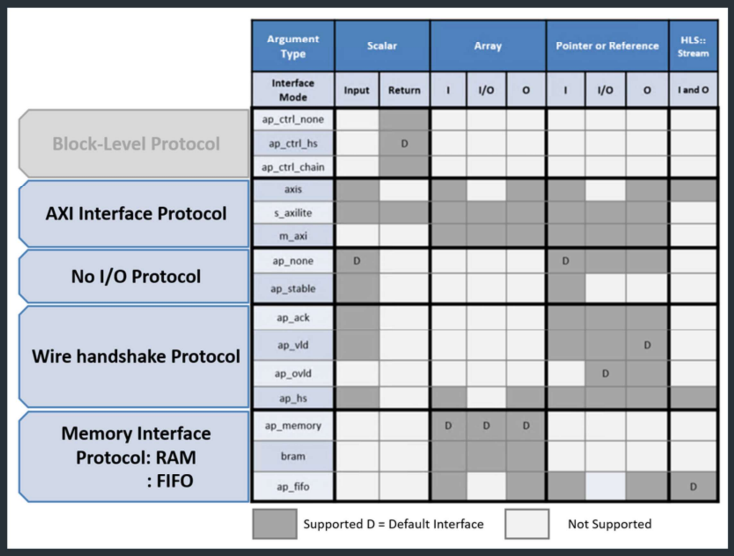
Let's now look at port-level protocols.

The port-level I/O protocol interfaces are basically the data ports. They are created for each argument in the top-level function and the function return. The I/O protocol created depends on the type of C argument and on the default type selected by the tool.

Interface Synthesis Protocols

Port-Level Protocol

- A complete list of all possible port-level I/O protocols is shown
- The port-level protocols are:
  - AXI4 interface protocols
  - No I/O protocol
  - Wire handshake protocols
  - Memory interface protocols



XILINX.

A complete list of all possible port-level I/O protocols is shown.

The port-level protocols are:
- AXI4 interface protocols
- No I/O protocol
- Wire handshake protocols
- Memory interface protocols

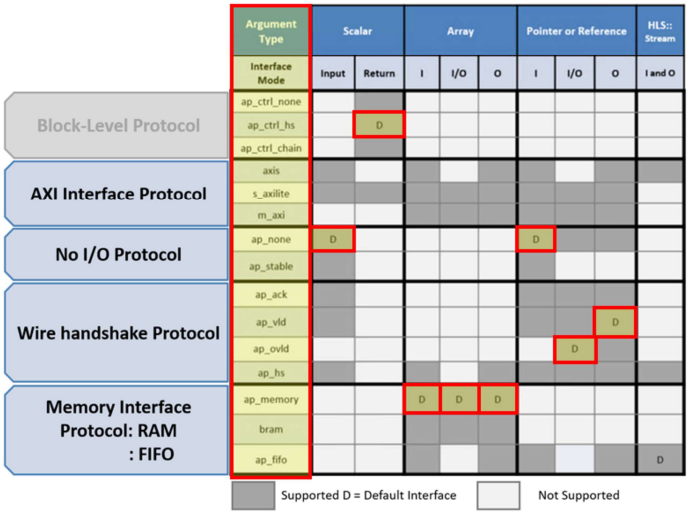
Interface Synthesis Protocols

After the block-level protocol has been used to control the operation of the block

Port-level I/O protocols are used to sequence data into and out of the block

Type of interfaces that are created by the interface synthesis depends on:

- Type of C argument
- Default interface mode
- INTERFACE optimization directive

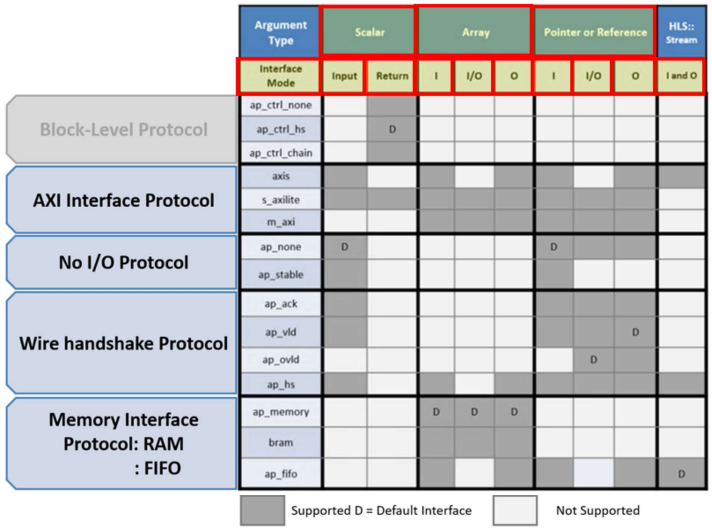


XILINX.

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

The type of interfaces that are created by the interface synthesis depends on:
- Type of C argument
- Default interface mode
- INTERFACE optimization directive

Interface Synthesis Protocols



- D: Default interface mode for each type If an illegal interface is specified, Vitis HLS tool issues a message and implements the default interface mode
- I: Input arguments, which are only read
- O: Output arguments, which are only written to
- I/O: Input/output arguments, which are both read and written
- Supported
- Not supported

XILINX.

The diagram shows the interface protocol mode that you can specify on each type of C argument.

The first column shows the interface mode, and the next columns show the type of a C function argument such scalar, array or a pointer or reference. The type column is then sub-divided into either input, output or return or IO columns, wherever applicable, specifying which interface mode can be applied to what type of argument.

Vitis HLS Tool Optional I/O: Function Arguments

Function arguments of a C code get synthesized into input and/or output data ports

- in1 argument from code → in1 port
- in2 argument from code → in2 port

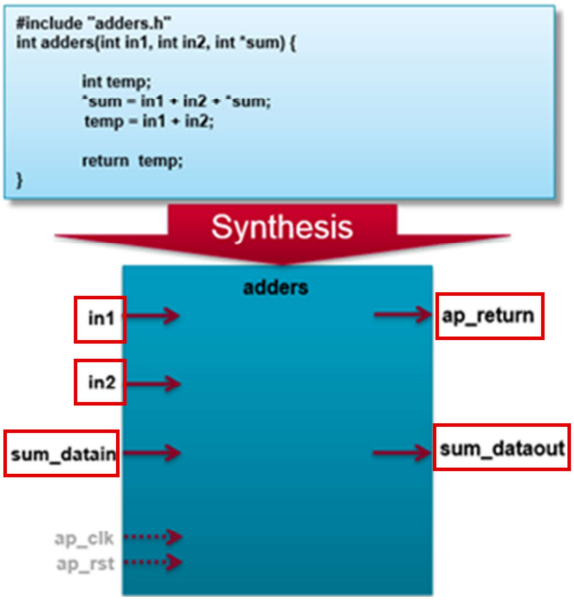
Function return of a C code gets synthesized into an output port called ap_return

- return temp variable → ap_return port

Pointers used in the C code can be read from and written to the separate input/output ports

- sum pointer reads → sum_datain port
- sum pointer writes → sum_dataout port

The arrays used in the C code can be synthesized into read and/or write ports similar to pointers



XILINX.

The graphic here shows the function code and the synthesized hardware for that code.

The function arguments of a C code get synthesized into data ports. There is an equivalent data ports ‘in1’ and ‘in2’ for the function arguments.

The function return of a C code gets synthesized into an output port called ap_return. There is an equivalent ‘ap_return’ port for the return ‘temp’ variable in the code.

The pointers used in the C code can be read from and written to the separate input/output ports. There is an input port (sum_datain) and the output port (sum_dataout) for the ‘sum’ pointer, which does the pointer read and writes.

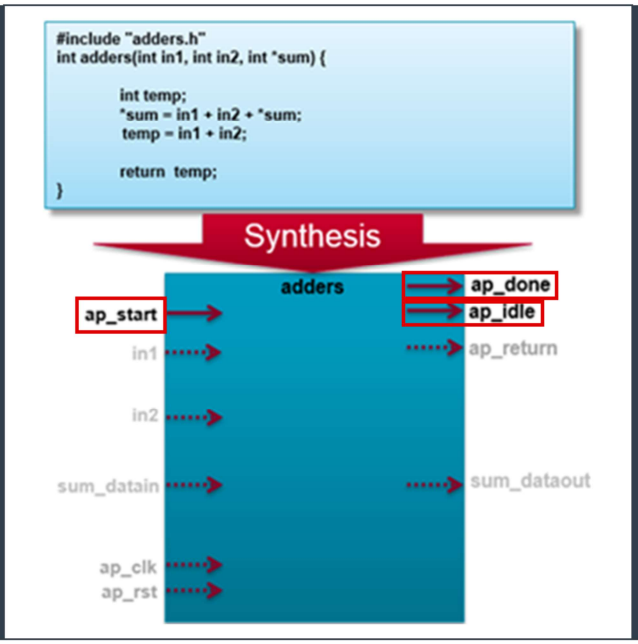
The arrays used in the C code can be synthesized into read and/or write ports similar to pointers.

Block-Level Protocols

Block-level protocols control and indicate the operational status of the RTL block

The interface signals included in these block-level protocols are:

- ap_start: Indicates when the block can start processing data
- ap_idle: Indicates if the design is idle
- ap_done: Indicates the design has completed its operation and also indicates when any function return is valid
- ap_ready: Indicates when the block is ready to accept new inputs



XILINX.

Let's understand the block-level protocols in depth.

Block-level protocols are the control protocols added at the RTL block level, which control and indicate the operational status of the RTL block.

The interface signals included in these block level protocols are:
- ap_start: Indicates when the block can start processing data
- ap_idle: Indicates if the design is idle
- ap_done: Indicates the design has completed its operation and also indicates when any function return is valid
- ap_ready: Indicates when the block is ready to accept new inputs. The additional ap_ready signal is added if the function is pipelined.

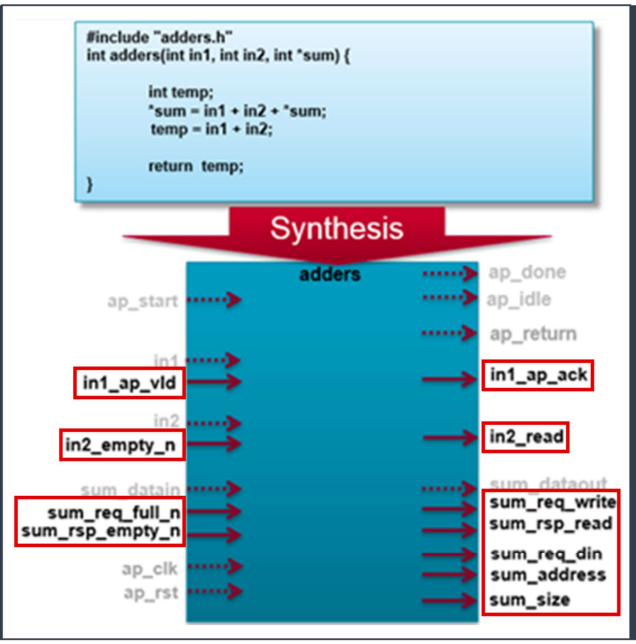
Port-Level I/O Protocols

Port level protocols sequence the data to and from the data port

Design is automatically synthesized to account for I/O signals

- in1 → in1_ap_vld and in1_ap_ack
- In2 → in2_empty_n and in2_read

List of port-level protocols → user can select an I/O protocol for each port



XILINX.

Let's understand the port-level protocols in depth.

The port-level protocols are added at the port level and they sequence the data to and from the data port.

The interface synthesis is applied to the design and the design is automatically synthesized to account for I/O signals. The ‘in1’ and ‘in2’ function arguments are converted to ‘in1_ap_vld’, ‘in1_ap_ack’ and ‘in2_empty_n’, ‘in2_read’, respectively.

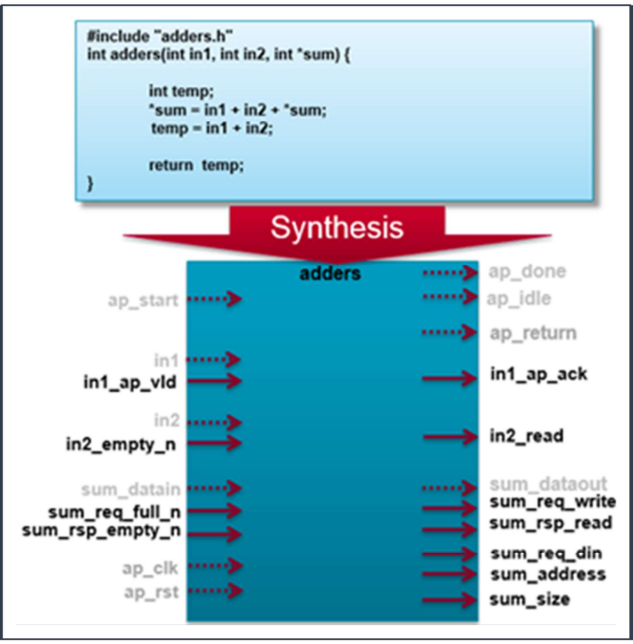
There is a list of port level protocols that are available in the Vitis HLS tool, from which a user can select an I/O protocol for each port.

Port-Level I/O Protocols

This allows users to easily connect to the surrounding blocks

- 'sum' pointer from the code gets converted to equivalent ports based on the I/O protocol selected

Non-standard interfaces are also supported in C/C++ using an arbitrary protocol definition



XILINX.

This allows users to easily connect to the surrounding blocks. The 'sum' pointer from the code gets converted to equivalent ports, based on the I/O protocol selected.

Non-standard interfaces are also supported in C/C++ using an arbitrary protocol definition.

INTERFACE Directive

How do we define the block-level and port-level I/O protocols in the source code?

INTERFACE directive

XILINX.

The question here is that how do we define the block-level and port-level I/O protocols in the source code?

By using the INTERFACE directive, the block-level and port-level I/O protocols are defined in the source code.

INTERFACE Directive

Block-level protocol is specified by using the INTERFACE directive with port=return

Port-level I/O protocols are specified by using the INTERFACE directive with port=<port name>

These interfaces protocols can be defined using the GUI as well

```
4   out_T dot(in_T a[N], in_T b[N], in_T  init, out_T *total)   {
5
6   #pragma HLS  INTERFACE  ap_ctrl_h port=return
7
8
9   #pragma HLS INTERFACE  ap_fifo depth=16 port=b
10  #pragma HLS INTERFACE  ap_fifo depth=16 port=a
11
12
13  int sum = init;
14  int tmp = 0;
15
16  for(int i=0; i < N; i++){
17                tmp = a[i] * b[i];
18                sum += tmp;
19  }
20
21  *total = *total + sum;
22  return sum;
23
24  }
25
```

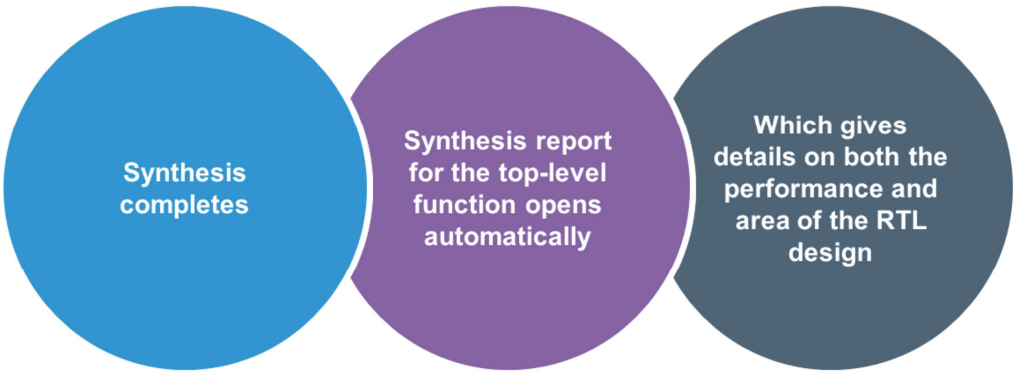

XILINX.

The block-level protocol highlighted here in yellow is specified by using the INTERFACE directive with port=return.

The port-level I/O protocols highlighted here in green are specified by using the INTERFACE directive with port=<port name>.

These interfaces protocols can be defined using the GUI as well.

Vitis HLS Tool Interfaces



XILINX.

When the synthesis completes, the synthesis report for the top-level function opens automatically, which gives details on both the performance and area of the RTL design.

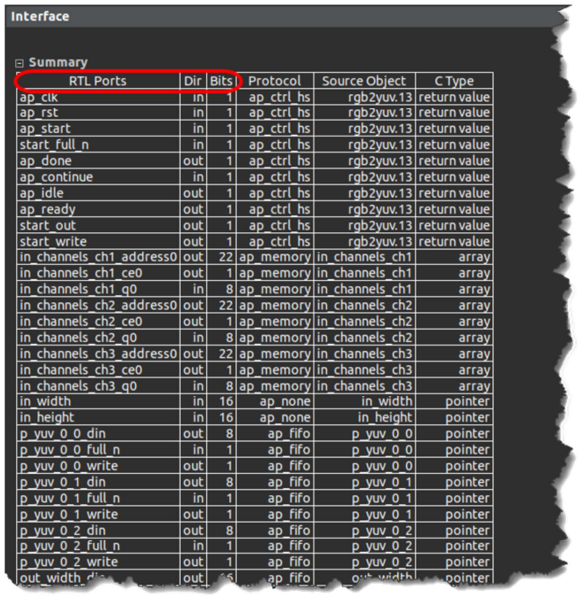
Vitis HLS Tool Interfaces

Interface summary section shows how the function arguments have been synthesized into RTL ports

RTL ports are created when that source object is synthesized with the stated I/O protocol

RTL port names grouped with their protocol and the source object

Size and the direction of the ports



Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	rgb2yuv.13	return value
ap_rst	in	1	ap_ctrl_hs	rgb2yuv.13	return value
ap_start	in	1	ap_ctrl_hs	rgb2yuv.13	return value
start_full_n	in	1	ap_ctrl_hs	rgb2yuv.13	return value
ap_done	out	1	ap_ctrl_hs	rgb2yuv.13	return value
ap_continue	in	1	ap_ctrl_hs	rgb2yuv.13	return value
ap_idle	out	1	ap_ctrl_hs	rgb2yuv.13	return value
ap_ready	out	1	ap_ctrl_hs	rgb2yuv.13	return value
start_out	out	1	ap_ctrl_hs	rgb2yuv.13	return value
start_write	out	1	ap_ctrl_hs	rgb2yuv.13	return value
in_channels_ch1_address0	out	22	ap_memory	in_channels_ch1	array
in_channels_ch1_ce0	out	1	ap_memory	in_channels_ch1	array
in_channels_ch1_q0	in	8	ap_memory	in_channels_ch1	array
in_channels_ch2_address0	out	22	ap_memory	in_channels_ch2	array
in_channels_ch2_ce0	out	1	ap_memory	in_channels_ch2	array
in_channels_ch2_q0	in	8	ap_memory	in_channels_ch2	array
in_channels_ch3_address0	out	22	ap_memory	in_channels_ch3	array
in_channels_ch3_ce0	out	1	ap_memory	in_channels_ch3	array
in_channels_ch3_q0	in	8	ap_memory	in_channels_ch3	array
in_width	in	16	ap_none	in_width	pointer
in_height	in	16	ap_none	in_height	pointer
p_yuv_0_0_din	out	8	ap_fifo	p_yuv_0_0	pointer
p_yuv_0_0_full_n	in	1	ap_fifo	p_yuv_0_0	pointer
p_yuv_0_0_write	out	1	ap_fifo	p_yuv_0_0	pointer
p_yuv_0_1_din	out	8	ap_fifo	p_yuv_0_1	pointer
p_yuv_0_1_full_n	in	1	ap_fifo	p_yuv_0_1	pointer
p_yuv_0_1_write	out	1	ap_fifo	p_yuv_0_1	pointer
p_yuv_0_2_din	out	8	ap_fifo	p_yuv_0_2	pointer
p_yuv_0_2_full_n	in	1	ap_fifo	p_yuv_0_2	pointer
p_yuv_0_2_write	out	1	ap_fifo	p_yuv_0_2	pointer
out_width_din	out	16	ap_fifo	out_width	pointer

XILINX.

The interface summary section at the end of the report shows how the function arguments have been synthesized into RTL ports.

The RTL ports are created when that source object is synthesized with the stated I/O protocol.

The left most column shows the RTL port names, grouped with their protocol and the source object.

The report also shows the size and the direction of the ports.

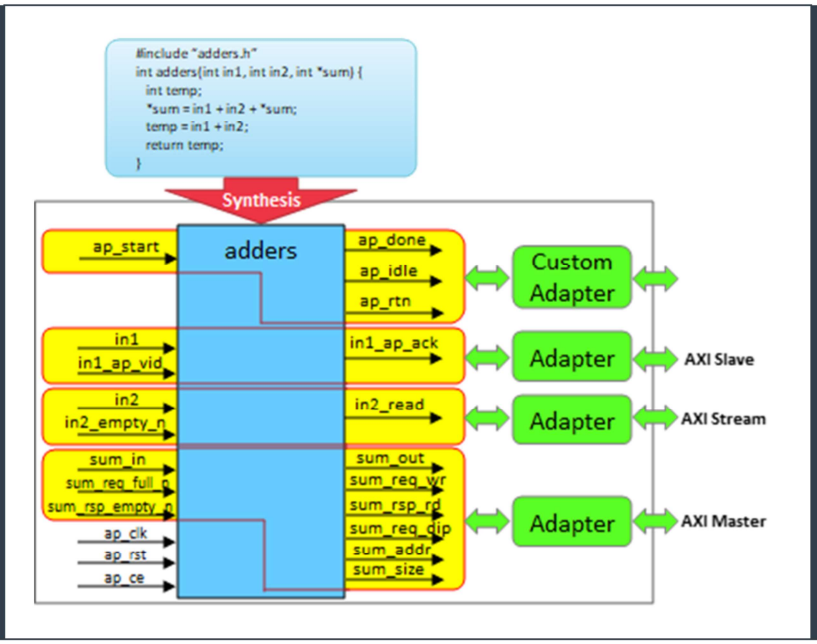
Vitis HLS Tool I/O Options: IP Adapters

Bus interfaces are used for the Vivado® IP integrator environment

Bus interfaces include:
- AXI4 Slave
- AXI4 Stream
- AXI4 Master

IP adapter cores are used to connect the I/O protocols to the bus interfaces

These interfaces are implemented in the RTL wrapper used in the IP generation flow



XILINX.

The bus interfaces are used for the Vivado IP integrator environment.

The bus interfaces include:
- AXI4 Slave
- AXI4 Stream
- AXI4 Master

IP adapter cores are used to connect the I/O protocols to the bus interfaces.

These interfaces are implemented in the RTL wrapper used in the IP generation flow.

Apply Your Knowledge

Multiple Choice Question

Interface synthesis is used _____.

- Only for the sequential design
- When the top-level function arguments are synthesized into RTL ports
- Where the behavior is explicitly described in the source code
- None of the above

Correct answer:
When the top-level function arguments are synthesized into RTL ports

Correct feedback:
You selected the correct answer.

Incorrect feedback:
The correct answer is “When the top-level function arguments are synthesized into RTL ports”.

Try again feedback:
Hint: Recall the definition of interface synthesis.

Apply Your Knowledge

Multiple Choice Question

Which are the block-level interfaces from the following? (Select all that apply)

- ☐ ap_start
- ☐ ap_none
- ☐ ap_ack
- ☐ ap_done

Correct answers:
- ap_start
- ap_done

Correct feedback:
You selected the correct answer.

Incorrect feedback:
The correct answers are “ap_start” & “ap_done”.

Try again feedback:
Hint: Block-level interfaces control the RTL block.

Summary

- Interface synthesis: When the top-level function is synthesized, the arguments (or parameters) to the function are synthesized into RTL ports
- Two categories of interfaces
  - Block-level protocols
    - Default handshakes at the block level
  - Port-level protocols
    - Wide support for all standard I/O protocols
    - Protocol is dependent on the C variable type (pointer, etc.)