XILINX.

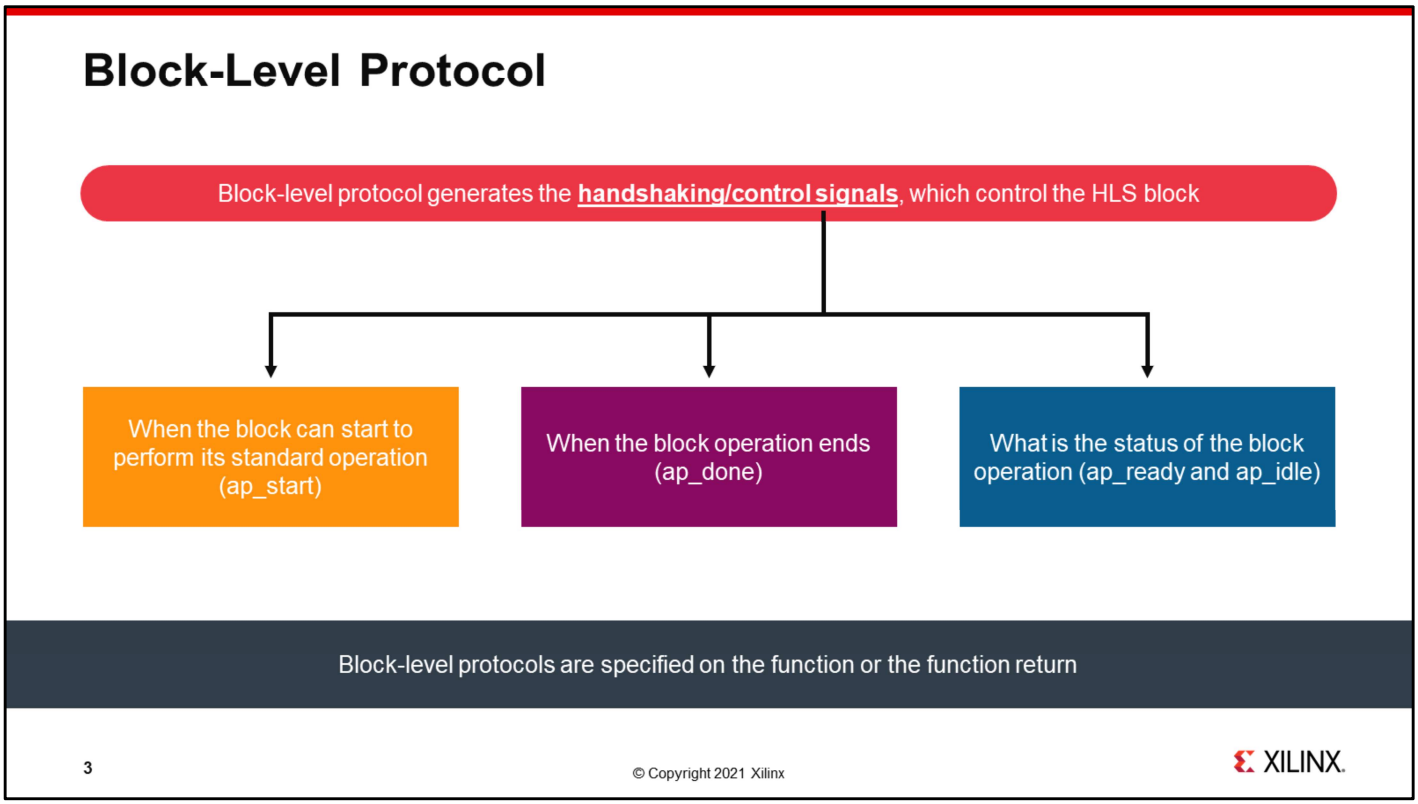
Block-Level Protocols

2021.1

Objectives

After completing this module, you will be able to:

- List the types of block-level protocols abstracted in the Vitis™ HLS tool



Let's first understand what a block-level protocol is.

The block-level protocol generates the handshaking or control signals, which control the HLS block.

These control signals indicate:

- When the block can start to perform its standard operation (ap_start)
- When the block operation ends (ap_done)
- What is the status of the block operation (ap_ready and ap_idle)

These block-level protocols are specified on the function or the function return.

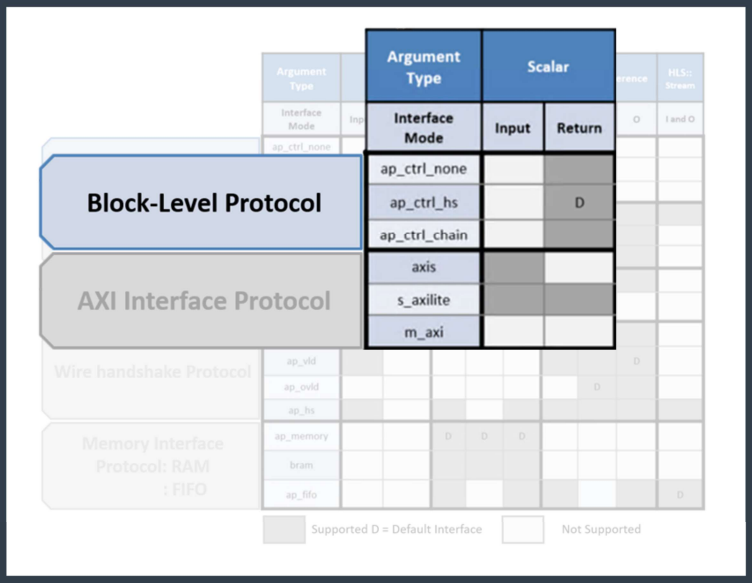
Block-Level Protocol

Three types of block-level protocols:

- **ap_ctrl_hs**
- **ap_ctrl_chain**
- **ap_ctrl_none**

Function return → specified as an AXI4-Lite protocol using **s_axilite**

- All the ports in the block-level protocol are grouped into the AXI4-Lite protocol
- Common practice when another device, such as a CPU or a processor, is used to configure and control when the block starts and stops operation



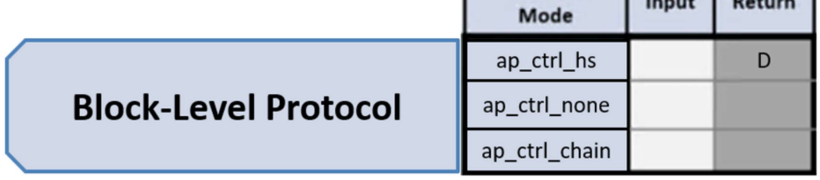
XILINX.

There are three types of block-level protocols:

- ap_ctrl_hs
- ap_ctrl_chain
- ap_ctrl_none

Apart from this, if the function return is specified as an AXI4-Lite protocol using s_axilite, all the ports in the block-level protocol are grouped into the AXI4-Lite protocol. This is a common practice when another device, such as a CPU or a processor, is used to configure and control when the block starts and stops operation.

Block-Level Protocol



Argument Type	Scalar	
Interface Mode	Input	Return
ap_ctrl_hs		D
ap_ctrl_none		
ap_ctrl_chain		

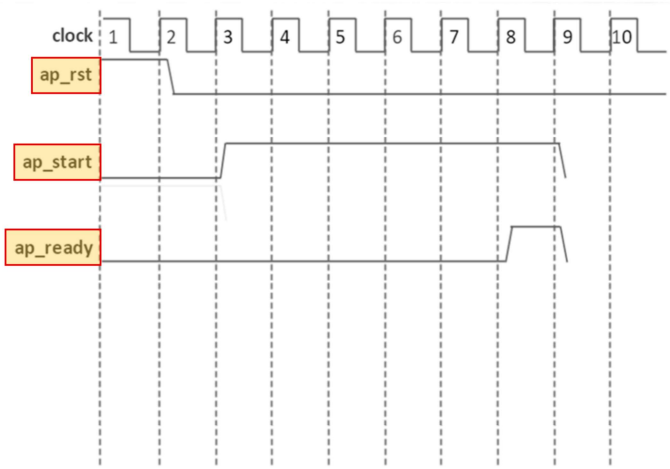
XILINX.

ap_ctrl_hs

Behavior of the control signals created by ap_ctrl_hs

After the reset signal is applied:

1. Design starts when “ap_start” control signal goes High
2. ap_start signal should remain High until ap_ready signal goes high



XILINX.

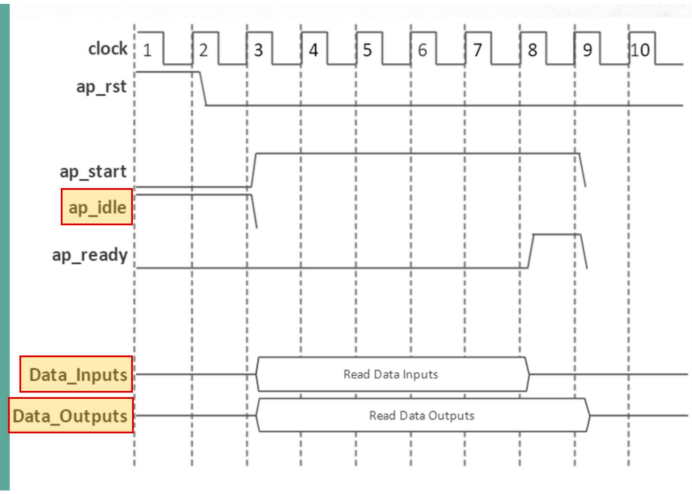
Let’s discuss what happens when the ap_ctrl_hs block-level protocol is used. The behavior of the control signals created by ap_ctrl_hs block-level protocol is shown here.

After the reset signal is applied:

1. Block waits for “ap_start” control signal to go High before it begins operation.
2. The ap_start signal should remain High until ap_ready signal indicates that the design is ready for new inputs.

ap_ctrl_hs

3. Output “ap_idle” goes Low when “ap_start” is sampled High
4. Data can now be read on the input ports and write on the output ports
   a. First input data can be sampled on the first clock edge after “ap_idle” goes Low

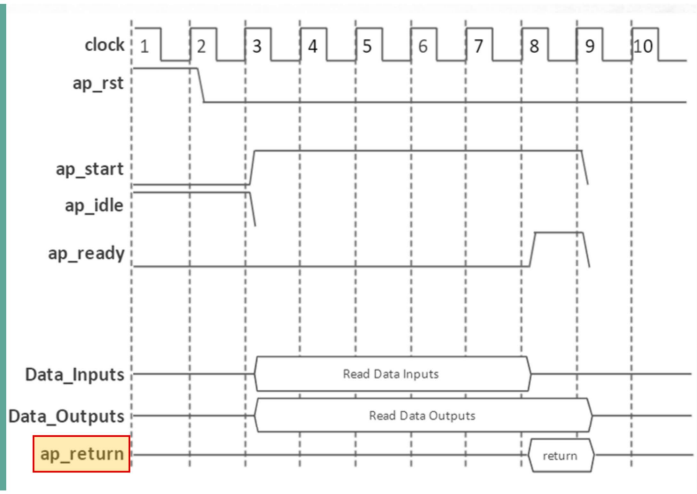


XILINX.

3. Output “ap_idle” goes Low when “ap_start” is sampled High.
4. Data can now be read on the input ports and write on the output ports.
   a. The first input data can be sampled on the first clock edge after “ap_idle” goes Low.

ap_ctrl_hs

5. When the block completes all operations, any return value is written to port ap_return
   a. If there was no function return, there is no ap_return port on the RTL
   b. Other outputs may be written to at any time until the block completes and are independent of this I/O protocol

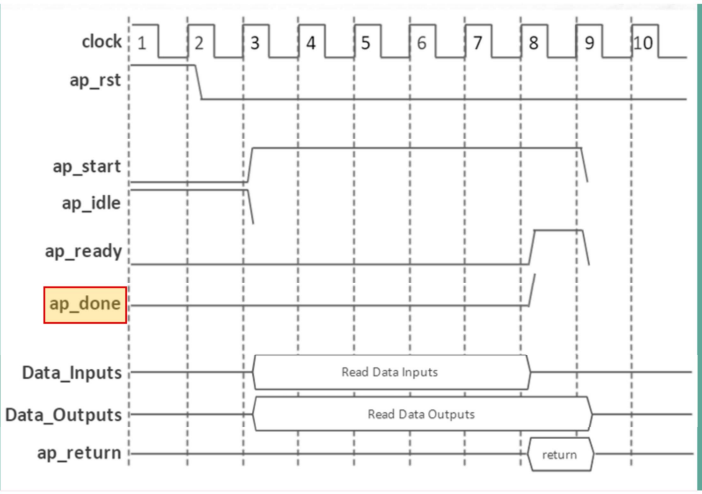


XILINX.

5. When the block completes all operations, any return value is written to port ap_return.
   a. If there was no function return, there is no ap_return port on the RTL block.
   b. Other outputs may be written to at any time until the block completes and are independent of this I/O protocol.

ap_ctrl_hs

6. Output “ap_done” goes High when the block completes operation
7. Idle signal goes High one cycle after “ap_done” and remains High until the next time “ap_start” is sampled High



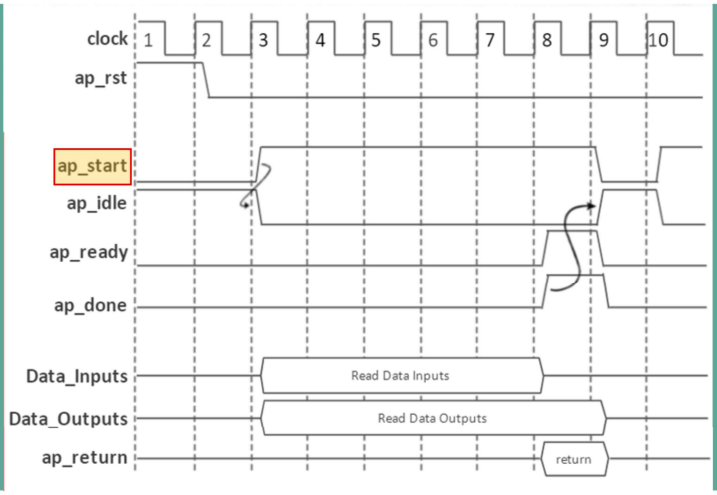
XILINX.

6. Output “ap_done” goes High when the block completes operation.
    a. When the design is ready to accept new inputs, the ap_ready signal goes High.
    b. If there is an ap_return port, the data on this port is valid when “ap_done” is High.

7. The idle signal goes High one cycle after “ap_done” and remains High until the next time “ap_start” is sampled High (indicating the block should begin operation).

ap_ctrl_hs

8. If the “ap_start” signal is High when “ap_done” goes High:
   a. “ap_idle” signal remains Low
   b. Block immediately starts its next execution (or next transaction)
   c. Next input can be read on the next clock edge



XILINX.

8. If the “ap_start” signal is High when “ap_done” goes High:
   a. The “ap_idle” signal remains Low.
   b. Block immediately starts its next execution (or next transaction).
   c. The next input can be read on the next clock edge.

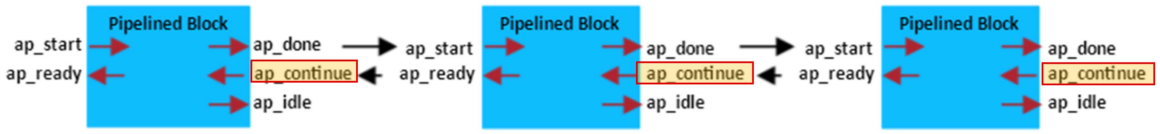
ap_ctrl_chain

ap_ctrl_chain protocol is used to support the pipeline chains

Similar to ap_ctrl_hs protocol but provides an additional input port ap_continue

Asserting the ap_continue signal Low:

- Informs the design that the downstream block that consumes the data is not ready to accept new data
- Design stalls when it reaches the final state of the current transaction
- Output data is presented on the protocol
- ap_done signal can be asserted High and the design remains in this state until ap_continue is asserted High



11

XILINX.

Let's discuss what happens when the ap_ctrl_chain block-level protocol is used. The ap_ctrl_chain protocol is used to support the pipeline chains. It is similar to the ap_ctrl_hs protocol but provides an additional input port ap_continue.

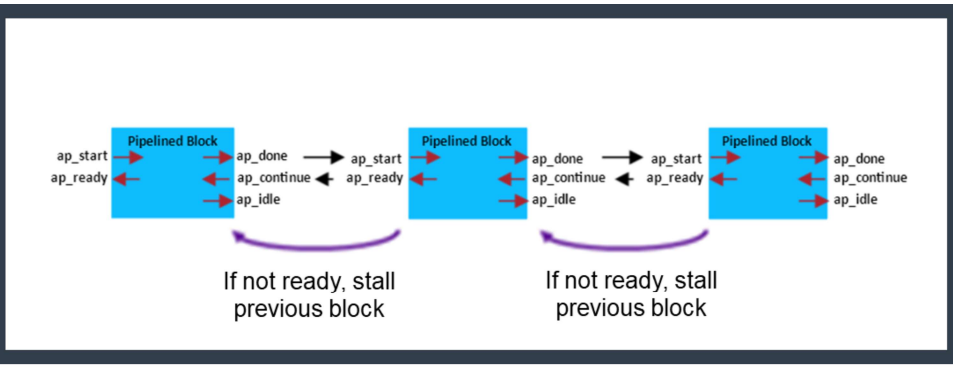
Asserting the ap_continue signal Low:

- Informs the design that the downstream block that consumes the data is not ready to accept new data.
- Design stalls when it reaches the final state of the current transaction.
- Output data is presented on the protocol.
- ap_done signal can be asserted High and the design remains in this state until ap_continue is asserted High.

ap_ctrl_chain

Provides "back pressure" signal which allows the downstream blocks to prevent any more data being generated

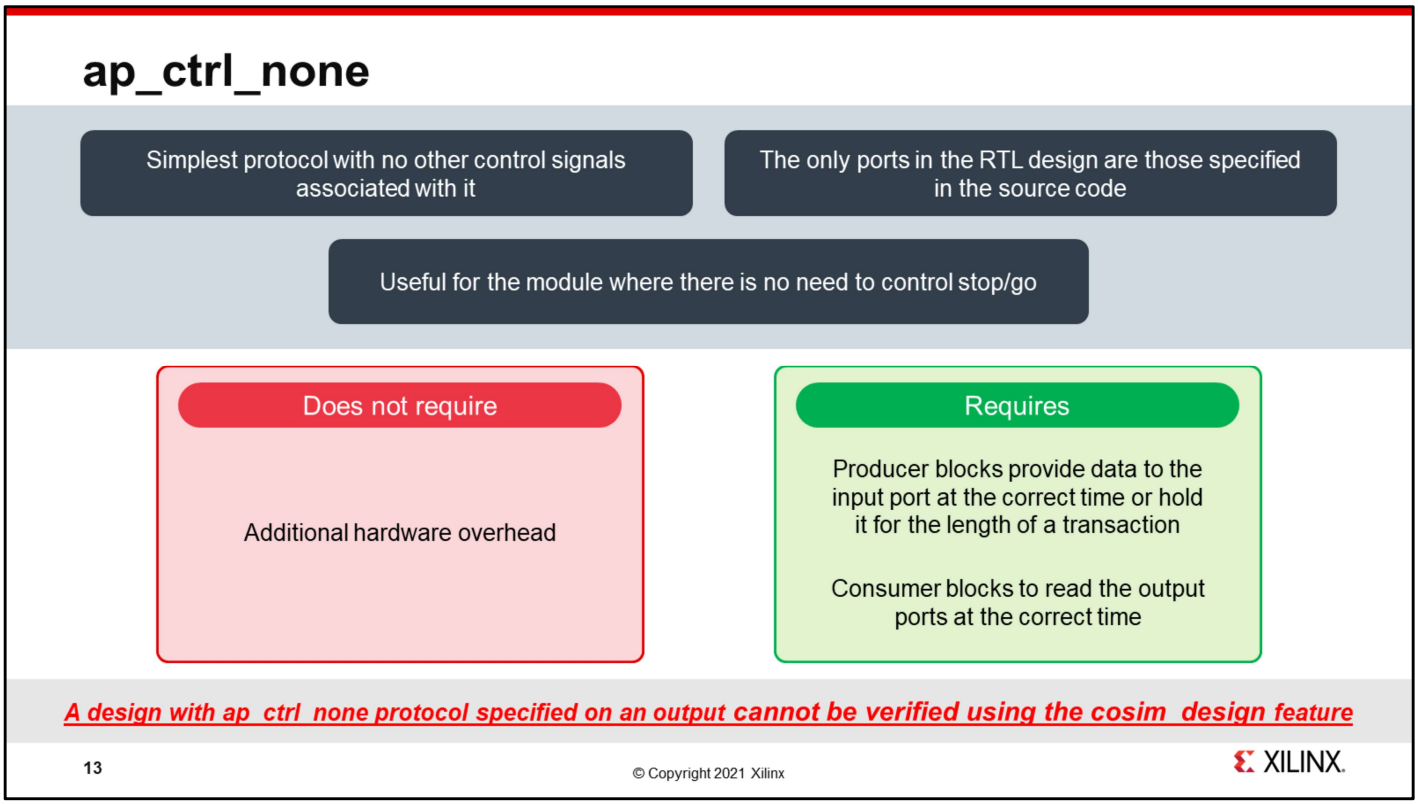
When integrating several HLS modules together in RTL domain



XILINX.

The ap_ctrl_chain protocol provides "back pressure" signal which allows the downstream blocks to prevent any more data being generated.

Xilinx recommends using the ap_ctrl_chain block-level I/O protocol when chaining the HLS tool blocks together.



The ap_ctrl_none block-level protocol is the simplest protocol and has no other control signals associated with it.

Neither the input nor the output signals have any associated control ports indicating when data is read or written. The only ports in the RTL design are those specified in the source code.

It is useful for the module where there is no need to control stop/go. For example: combinational logic example with Initiation Interval equals to 1.

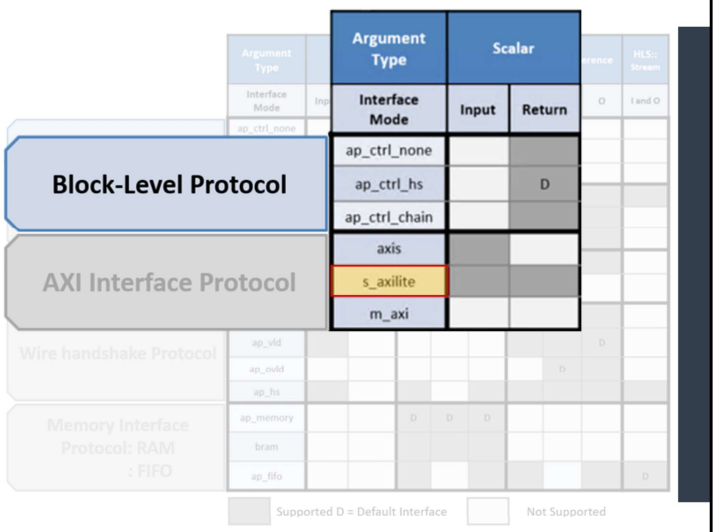
An ap_ctrl_none protocol requires no additional hardware overhead but does require that the producer blocks provide data to the input port at the correct time or hold it for the length of a transaction, until the design completes, and consumer blocks are required to read the output ports at the correct time.

A design with ap_ctrl_none protocol specified on an output cannot be verified using the cosim_design feature.

s_axilite

AXI Slave Lite I/O protocol used to control the HLS block: s_axilite

Common practice used when the processing system (PS) or processor is used to configure the HLS block



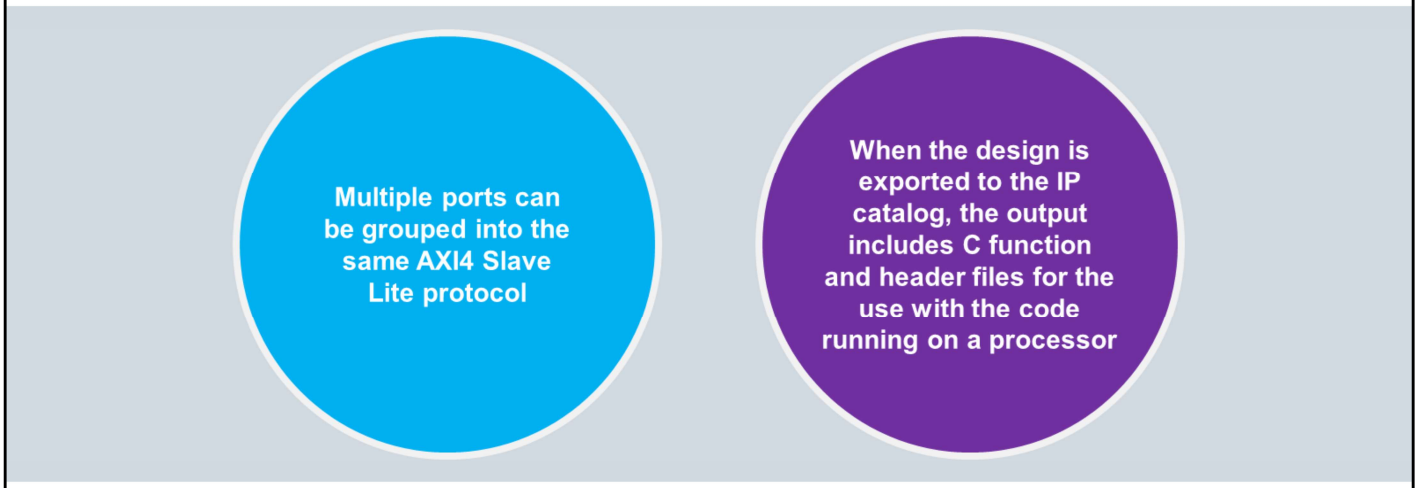
XILINX.

An AXI Slave Lite I/O protocol can be used to control the HLS block by using the memory-mapped block-level: s_axilite.

This is a common practice used when the processing system or processor is used to configure the HLS block.

s_axilite

Features of the AXI4 Slave Lite protocol



15

XILINX.

The features of the AXI4 Slave Lite protocol provided by Vitis HLS are:

- Multiple ports can be grouped into the same AXI4 Slave Lite protocol
- When the design is exported to the IP Catalog, the output includes C function and header files for the use with the code running on a processor

s_axilite

Hardware header file provides a complete list of the memory mapped locations for the ports grouped into the AXI4 Slave Lite protocol

Understand how the hardware ports operate

Device will read any inputs grouped into the AXI4 Slave Lite protocol from the register in the protocol

```
// 0x00 : Control signals
//        bit 0  - ap_start (Read/Write/SC)
//        bit 1  - ap_done (Read/COR)
//        bit 2  - ap_idle (Read)
//        bit 3  - ap_ready (Read)
//        bit 7  - auto_restart (Read/Write)
//        others - reserved
// 0x04 : Global Interrupt Enable Register
//        bit 0  - Global Interrupt Enable (Read/Write)
//        others - reserved
// 0x08 : IP Interrupt Enable Register (Read/Write)
//        bit 0  - Channel 0 (ap_done)
//        bit 1  - Channel 1 (ap_ready)
// 0x0c : IP Interrupt Status Register (Read/TOW)
//        bit 0  - Channel 0 (ap_done)
//        others - reserved
```

16

XILINX.

The hardware header file provides a complete list of the memory mapped locations for the ports grouped into the AXI4 Slave Lite protocol. To correctly program the registers in the AXI4 Slave Lite protocol, there is some requirement to understand how the hardware ports operate.

For example, to start the block operation the ap_start register must be set to 1.

The device will then proceed and read any inputs grouped into the AXI4 Slave Lite protocol from the register in the protocol.

s_axilite

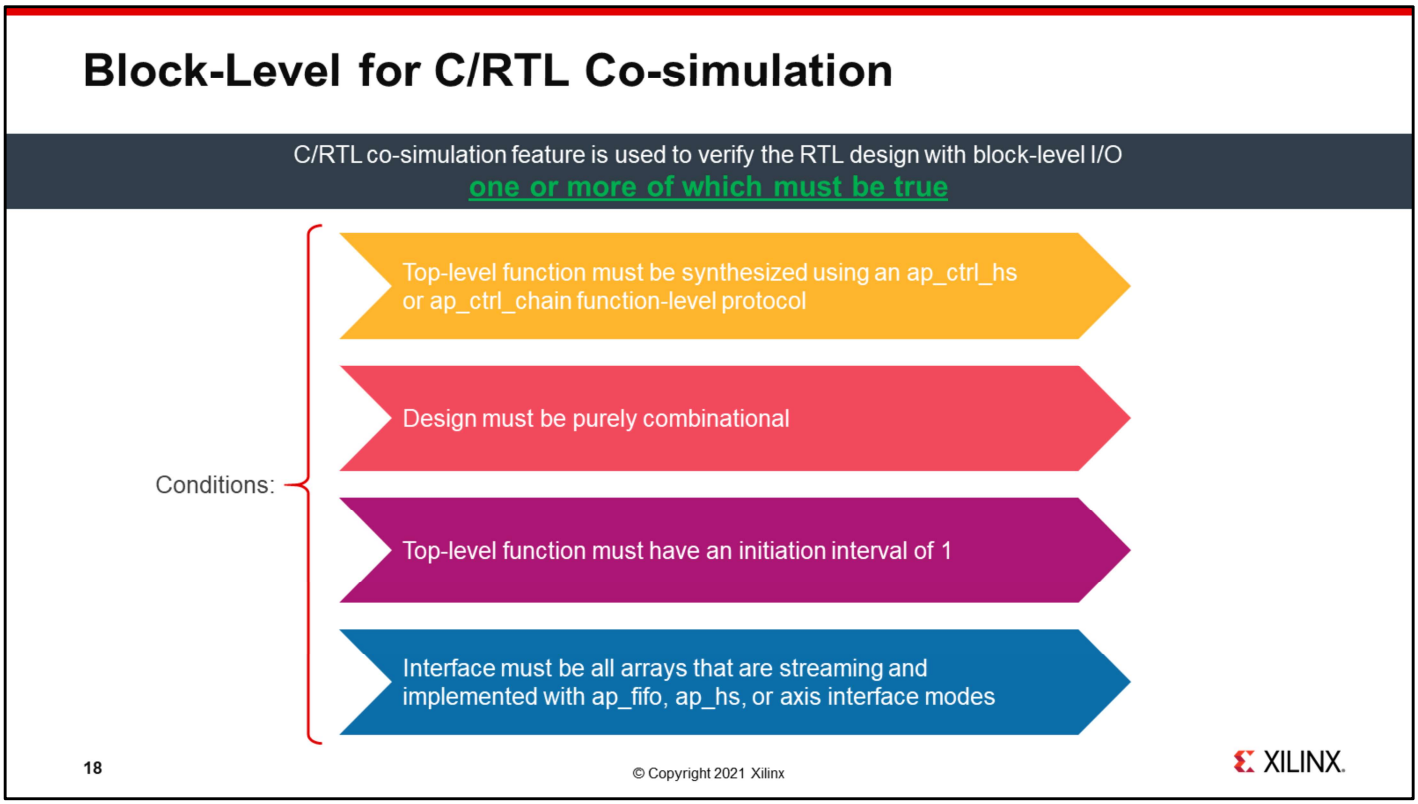
When the block completes operation:

- ap_done, ap_idle, and ap_ready registers will be set by the hardware output ports
- Results for any output ports grouped into the AXI4 Slave Lite protocol read from appropriate register

```
// 0x00 : Control signals
//        bit 0  - ap_start (Read/Write/SC)
//        bit 1  - ap_done (Read/COR)
//        bit 2  - ap_idle (Read)
//        bit 3  - ap_ready (Read)
//        bit 7  - auto_restart (Read/Write)
//        others - reserved
// 0x04 : Global Interrupt Enable Register
//        bit 0  - Global Interrupt Enable (Read/Write)
//        others - reserved
// 0x08 : IP Interrupt Enable Register (Read/Write)
//        bit 0  - Channel 0 (ap_done)
//        bit 1  - Channel 1 (ap_ready)
// 0x0c : IP Interrupt Status Register (Read/TOW)
//        bit 0  - Channel 0 (ap_done)
//        others - reserved
```


XILINX.

When the block completes operation, the ap_done, ap_idle, and ap_ready registers will be set by the hardware output ports and the results for any output ports grouped into the AXI4 Slave Lite protocol read from the appropriate register.



As we know that the C/RTL co-simulation feature is used to verify the RTL design with block-level, here are few conditions one or more of which must be true:

- Top-level function must be synthesized using an ap_ctrl_hs or ap_ctrl_chain function-level protocol
- Design must be purely combinational
- Top-level function must have an initiation interval of 1
- Interface must be all arrays that are streaming and implemented with ap_fifo, ap_hs, or axis interface modes

Block-Level for C/RTL Co-simulation

If one of these conditions is not met, C/RTL co-simulation halts with the message shown below

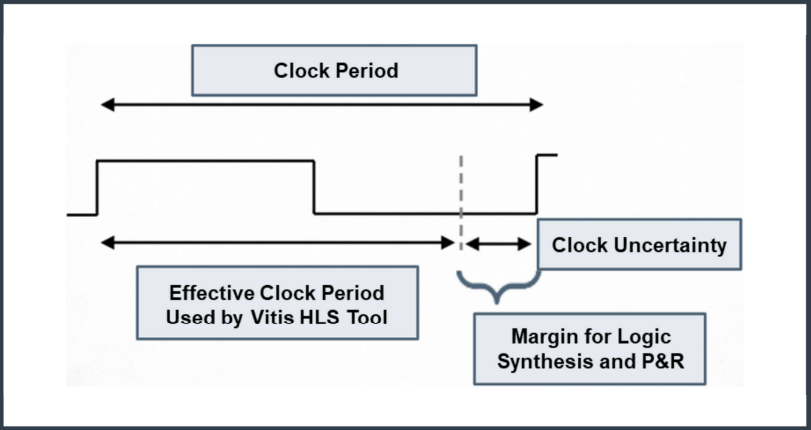
```
@E [SIM-345] Cosim only supports the following 'ap_ctrl_none' designs: (1)
combinational designs; (2) pipelined design with task interval of 1; (3) designs with
array streaming or hls_stream ports.
@E [SIM-4] *** C/RTL co-simulation finished: FAIL ***
```


If one of these conditions is not met, C/RTL co-simulation halts with the message shown below.

Clock, Reset, and RTL Output

For C and C++ Designs

- Only a single clock is supported
- Same clock is applied to all functions in the design



XILINX.

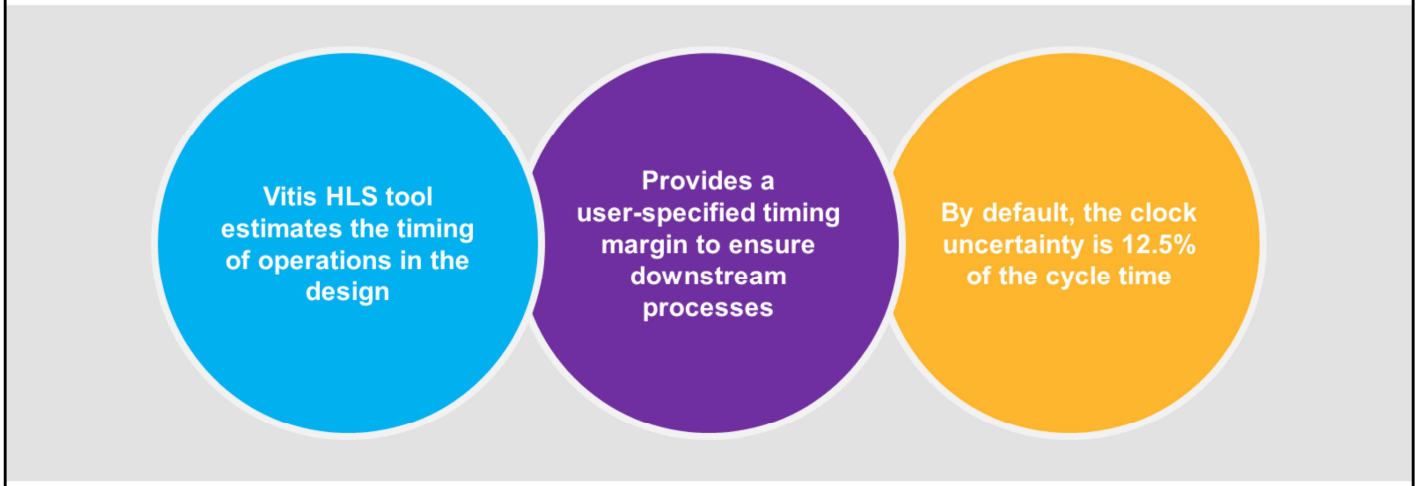
Let's look at how the clock, reset, and RTL output is managed using block-level protocols.

For C and C++ designs, only a single clock is supported. The same clock is applied to all functions in the design.

To calculate the effective clock period used for synthesis by the HLS tool, the Vitis HLS tool subtracts the clock uncertainty from the clock period as shown.

Using the clock frequency and device target information…

Clock, Reset, and RTL Output

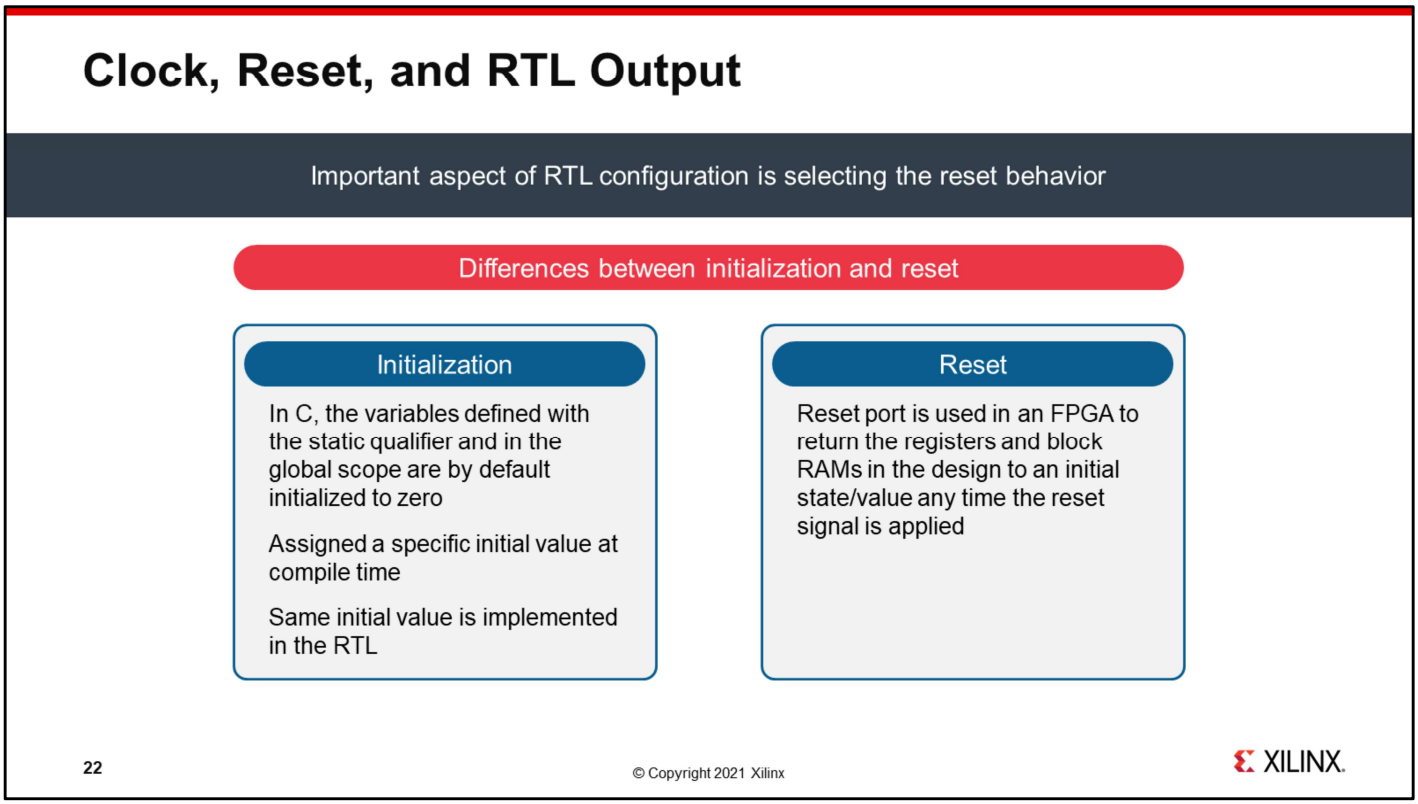


21

XILINX.

…the Vitis HLS tool estimates the timing of operations in the design and provides a user-specified margin to ensure downstream processes, such as logic synthesis and place & route, have enough timing margin to complete their operations.

By default, the clock uncertainty is 12.5% of the cycle time. The value can be explicitly specified beside the clock period.



Typically, the most important aspect of RTL configuration is selecting the reset behavior. When discussing reset behavior, it is important to understand the differences between initialization and reset.

Initialization
In C, the variables defined with the static qualifier and those defined in the global scope are by default initialized to zero.

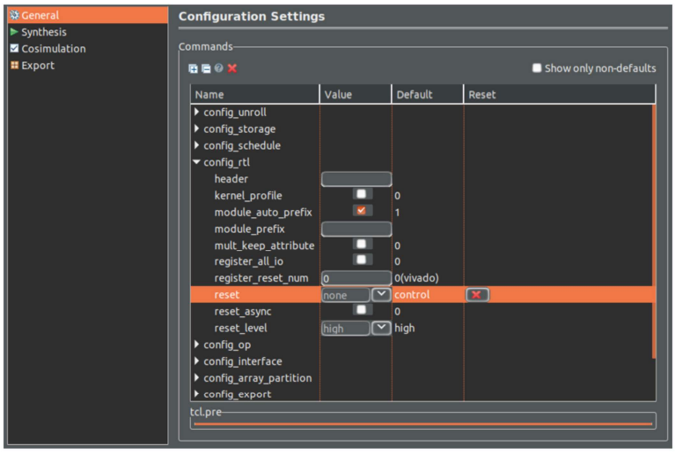
Optionally, these variables may be assigned a specific initial value at compile time (at time zero) and never again. In both cases, the same initial value is implemented in the RTL.

Reset
The reset port is used in an FPGA to return the registers and block-RAMs in the design to an initial state/value any time when the reset signal is applied.

Clock, Reset, and RTL Output

Presence and behavior of the RTL reset port is controlled using the **config_rtl** configuration

Accessed using menus Solution > Solution Settings > General > config_rtl



XILINX.

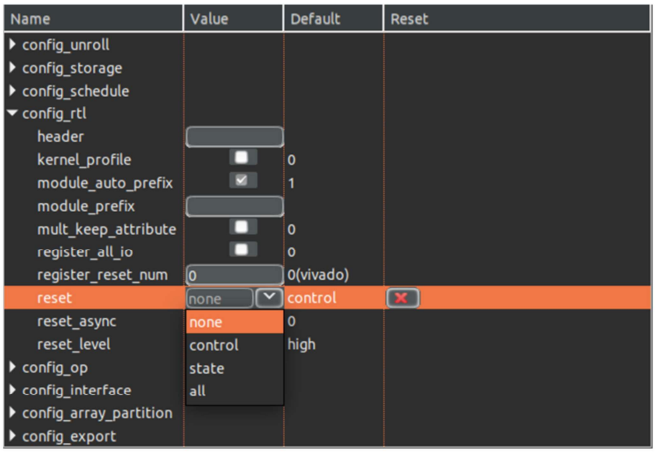
The presence and behavior of the RTL reset port is controlled using the config_rtl configuration.

This configuration is accessed using menus Solution > Solution Settings > General > config_rtl as shown.

Clock, Reset, and RTL Output

The reset settings include:

- Ability to set the polarity of the reset
- Whether the reset is synchronous or asynchronous
- Controls which registers are reset when the reset signal is applied



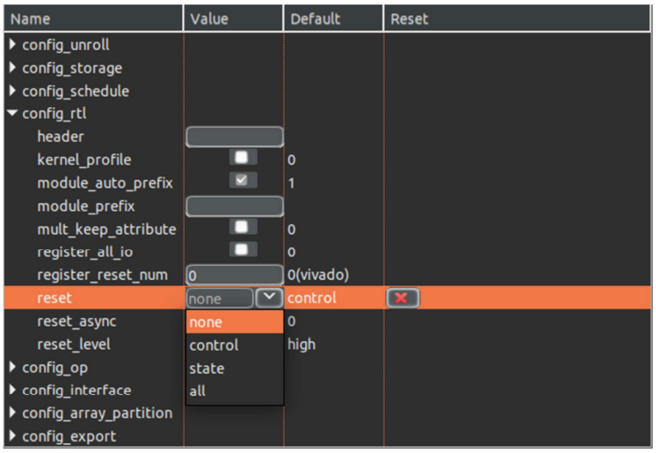
XILINX.

The reset settings include the ability to set the polarity of the reset and whether the reset is synchronous or asynchronous but more importantly it controls, through the reset option, which registers are reset when the reset signal is applied.

Clock, Reset, and RTL Output

The reset option has four settings:

- **none**: No reset is added to the design
- **control**: Default and ensures all control registers are reset
- **state**: Adds a reset to control registers plus any registers or memories derived from static and global variables in the C code
- **all**: This adds a reset to all registers and memories in the design



XILINX.

The reset option has four settings:

- none: No reset is added to the design
- control: This is the default and ensures all control registers are reset
- state: This option adds a reset to control registers plus any registers or memories derived from static and global variables in the C code
- all: This adds a reset to all registers and memories in the design

Apply Your Knowledge

Multiple Choice Question

What control signals are added by the ap_ctrl_hs protocol? (Select all that apply)

- ☐ ap_start
- ☐ ap_idle
- ☐ ap_continue
- ☐ ap_done

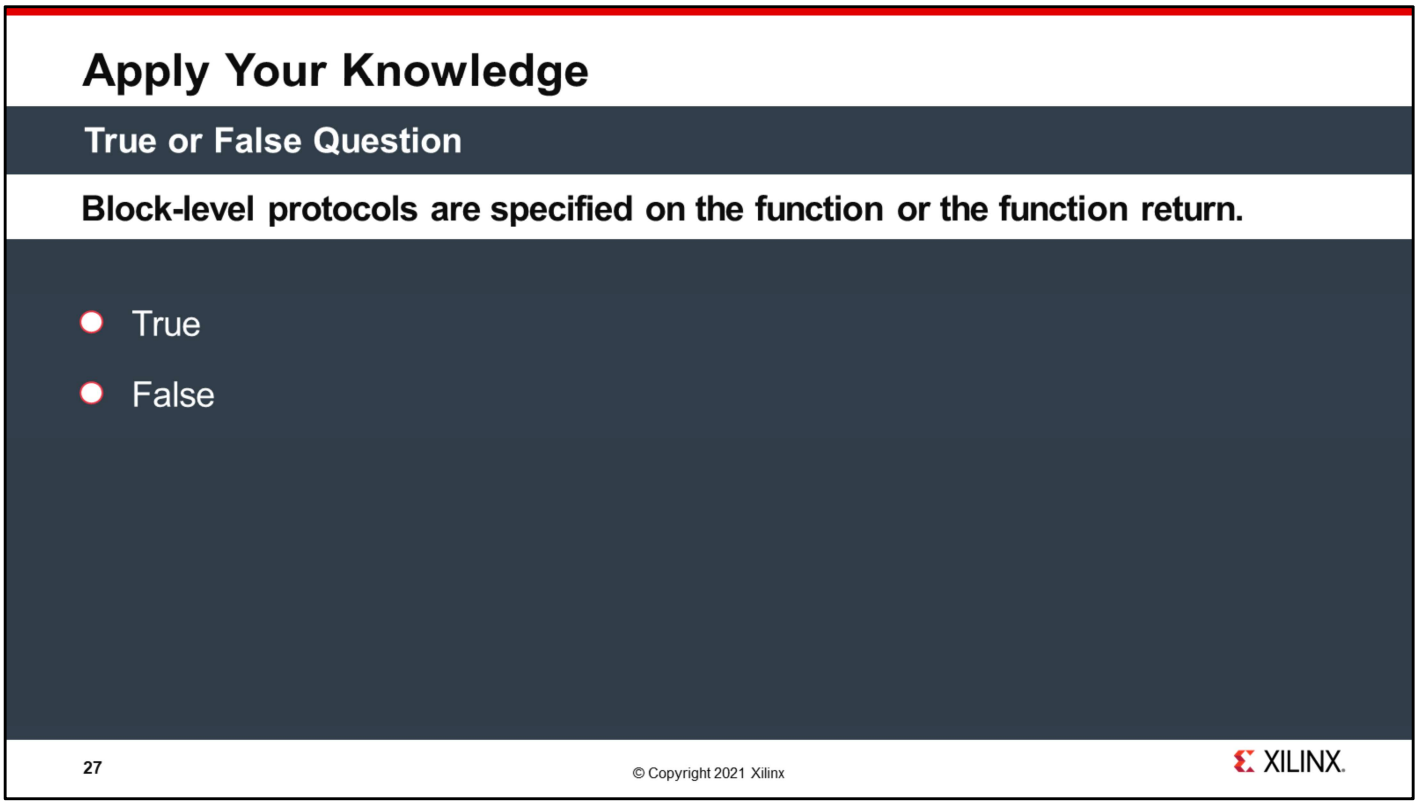
XILINX.

Correct answers:
"ap_start", "ap_idle" & "ap_done"

Correct feedback:
Your response is correct.

Incorrect feedback:
The correct answers are "ap_start", "ap_idle" & "ap_done".

Try again feedback:
Hint: Recall the main difference between the ap_ctrl_hs and ap_ctrl_chain protocols



Correct answer:
True

Correct feedback:
You selected the correct answer

Incorrect feedback:
The correct answer is “True”.

Summary

- Block-level interface protocols generate the handshaking or control signals that control the HLS block
- Three types of block-level protocols:
  - ap_ctrl_none: No block-level protocol
  - ap_ctrl_hs: Default block-level protocol
  - ap_ctrl_chain: Protocol to support chaining