Table 13: **Argument Types** *(cont'd)*

Argument Type	Scalar		Pointer to an Array			Hls::stream
Interface Mode	**Input**	**Return**	**I**	**I/O**	**O**	**I and O**
axis						D
m_axi			D	D	D	
s_axi_lite	D	D	D	D	D	

In the table you can see that two different defaults are specified for pointer arguments: `m_axi` and `s_axi_lite`. This is because the top-level pointers use the `m_axi` interface to access memory, and use the`s_axi_lite` interface to receive the offset into the memory address space.

The function `return` also has two different defaults: `s_axi_lite` for the returned value, and the block protocol (`ap_ctrl_chain`) used to control the function call in HW.

TIP: *The `s_axi_lite` interface is the default interface for scalar inputs, as well as the I/O port control interface for all ports except streaming.*

The following sections provides an overview of each interface mode, and full details on the interface protocols, including waveform diagrams.

Block-Level I/O Protocols

Vitis HLS uses the interface types `ap_ctrl_none`, `ap_ctrl_hs`, and `ap_ctrl_chain` to specify whether the RTL is implemented with block-level handshake signals. Block-level handshake signals specify the following:

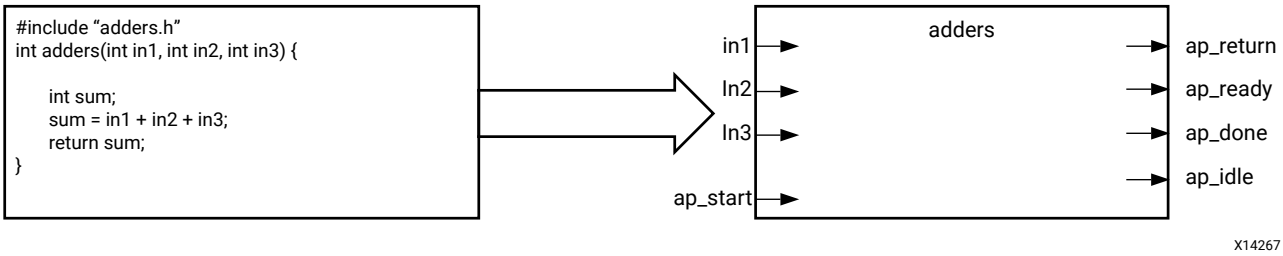
- When the design can start to perform the operation
- When the operation ends
- When the design is idle and ready for new inputs

You can specify these block-level I/O protocols on the function or the function return. If the C/C++ code does not return a value, you can still specify the block-level I/O protocol on the function return. If the C/C++ code uses a function return, Vitis HLS creates an output port `ap_return` for the return value.

TIP: *If the function return is specified as an AXI4-Lite interface (`s_axilite`) all the ports in the block-level interface are grouped into the AXI4-Lite interface, called `control` by default. This is a common practice when another device, such as a CPU, is used to configure and control when the block starts and stops operation.*

The `ap_ctrl_hs` block-level I/O protocol is the default. The following figure shows the resulting RTL ports and behavior when Vitis HLS implements `ap_ctrl_hs` on a function. In this example, the function returns a value using the `return` statement, and Vitis HLS creates the `ap_return` output port in the RTL design. If a function `return` statement is not included in the C/C++ code, this port is not created.

Figure 47: **Example ap_ctrl_hs Interface**



The `ap_ctrl_chain` interface mode is similar to `ap_ctrl_hs` but provides an additional input signal `ap_continue` to apply back pressure. Xilinx recommends using the `ap_ctrl_chain` block-level I/O protocol when chaining Vitis HLS blocks together.

ap_ctrl_none

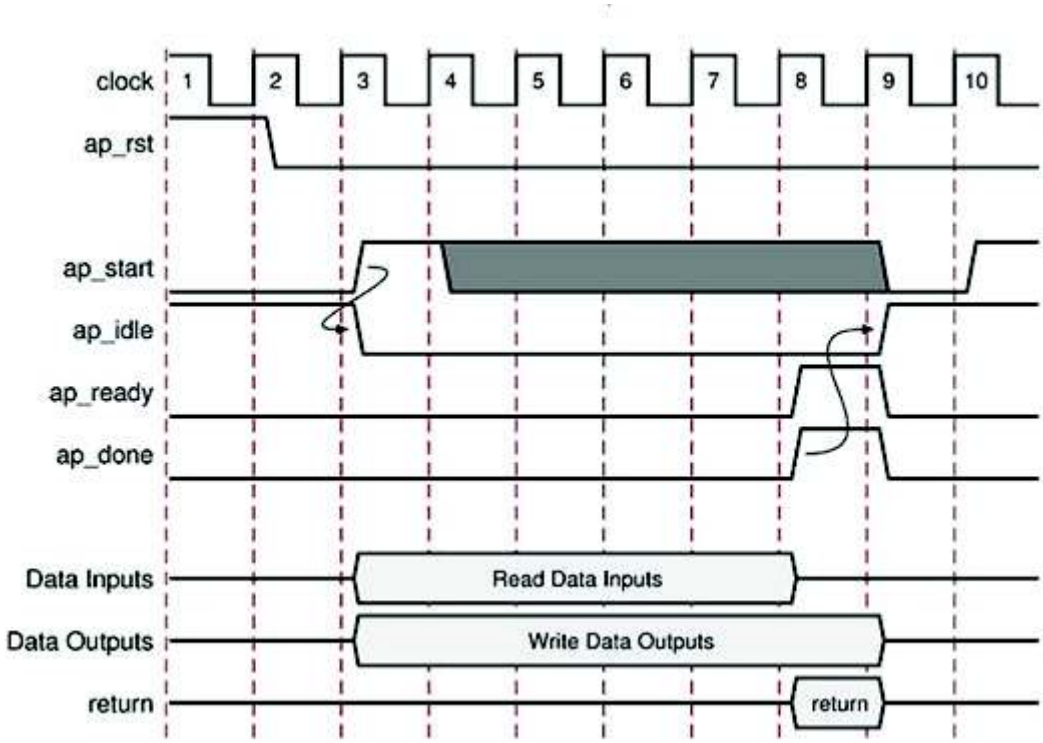
If you specify the `ap_ctrl_none` block-level I/O protocol, the handshake signal ports (`ap_start`, `ap_idle`, `ap_ready`, and `ap_done`) shown in the figure above are not created. You can use this protocol to create a block without control signals, as used in free-running kernels.

IMPORTANT! *If you use the `ap_none` block-level I/O protocol on your design, you must meet at least one of the conditions for C/RTL co-simulation as described in Interface Synthesis Requirements to verify the RTL design.*

ap_ctrl_hs

The following figure shows the behavior of the block-level handshake signals created by the `ap_ctrl_hs` I/O protocol for a non-pipelined design.

Figure 48: **Behavior of ap_ctrl_hs Interface**



After reset, the following occurs:

1. The block waits for `ap_start` to go High before it begins operation.
2. Output `ap_idle` goes Low immediately to indicate the design is no longer idle.
3. The `ap_start` signal must remain High until `ap_ready` goes High. Once `ap_ready` goes High:
   - If `ap_start` remains High the design will start the next transaction.
   - If `ap_start` is taken Low, the design will complete the current transaction and halt operation.
4. Data can be read on the input ports.
5. Data can be written to the output ports.

   ***Note*:** The input and output ports can also specify a port-level I/O protocol that is independent of this block-level I/O protocol. For details, see Port-Level I/O Protocols.
6. Output `ap_done` goes High when the block completes operation.

   ***Note*:** If there is an `ap_return` port, the data on this port is valid when `ap_done` is High. Therefore, the `ap_done` signal also indicates when the data on output `ap_return` is valid.
7. When the design is ready to accept new inputs, the `ap_ready` signal goes High. Following is additional information about the `ap_ready` signal:

- The ap_ready signal is inactive until the design starts operation.
- In non-pipelined designs, the ap_ready signal is asserted at the same time as ap_done.
- In pipelined designs, the ap_ready signal might go High at any cycle after ap_start is sampled High. This depends on how the design is pipelined.
- If the ap_start signal is Low when ap_ready is High, the design executes until ap_done is High and then stops operation.
- If the ap_start signal is High when ap_ready is High, the next transaction starts immediately, and the design continues to operate.

8. The ap_idle signal indicates when the design is idle and not operating. Following is additional information about the ap_idle signal:
   - If the ap_start signal is Low when ap_ready is High, the design stops operation, and the ap_idle signal goes High one cycle after ap_done.
   - If the ap_start signal is High when ap_ready is High, the design continues to operate, and the ap_idle signal remains Low.

ap_ctrl_chain

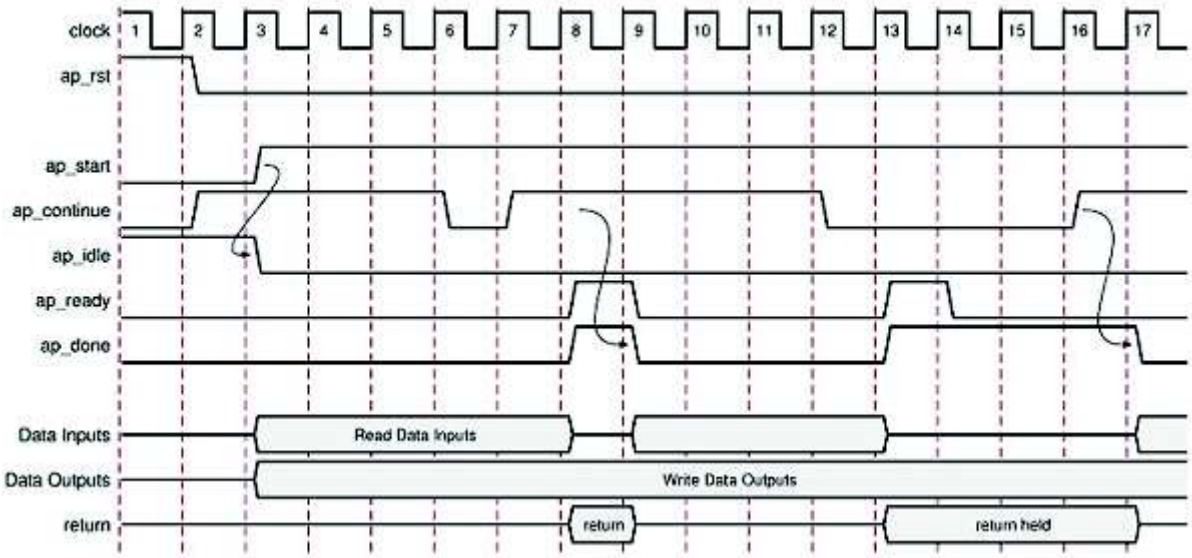
The ap_ctrl_chain block-level I/O protocol is similar to the ap_ctrl_hs protocol but provides an additional input port named ap_continue. An active-High ap_continue signal indicates that the downstream block that consumes the output data is ready for new data inputs. If the downstream block is not able to consume new data inputs, the ap_continue signal is Low, which prevents upstream blocks from generating additional data.

The ap_ready port of the downstream block can directly drive the ap_continue port. Following is additional information about the ap_continue port:

- If the ap_continue signal is High when ap_done is High, the design continues operating. The behavior of the other block-level I/O signals is identical to those described in the ap_ctrl_hs block-level I/O protocol.
- If the ap_continue signal is Low when ap_done is High, the design stops operating, the ap_done signal remains High, and data remains valid on the ap_return port if the ap_return port is present.

In the following figure, the first transaction completes, and the second transaction starts immediately because ap_continue is High when ap_done is High. However, the design halts at the end of the second transaction until ap_continue is asserted High.

Figure 49: **Behavior of ap_ctrl_chain Interface**



Port-Level I/O Protocols

By default input pointers and pass-by-value arguments are implemented as simple wire ports with no associated handshaking signal. For example, in the `sum_io` function discussed in Interface Synthesis Overview, the input ports are implemented without an I/O protocol, only a data port. If the port has no I/O protocol, (by default or by design) the input data must be held stable until it is read.

By default output pointers are implemented with an associated output valid signal to indicate when the output data is valid. In the `sum_io` function example, the output port is implemented with an associated output valid port (`sum_o_ap_vld`) which indicates when the data on the port is valid and can be read. If there is no I/O protocol associated with the output port, it is difficult to know when to read the data.

TIP: *It is always a good idea to use an I/O protocol on an output.*

Function arguments which are both read from and written to are split into separate input and output ports. In the `sum_io` function example, the `sum` argument is implemented as both an input port `sum_i`, and an output port `sum_o` with associated I/O protocol port `sum_o_ap_vld`.

If the function has a return value, an output port `ap_return` is implemented to provide the return value. When the RTL design completes one transaction, this is equivalent to one execution of the C/C++ function, the block-level protocols indicate the function is complete with the `ap_done` signal. This also indicates the data on port `ap_return` is valid and can be read.

***Note*:** The return value of the top-level function cannot be a pointer.