XILINX.

Port-Level I/O Protocols: Memory Interfaces

2021.1

Objectives

After completing this module, you will be able to:

- Explain how memory interface port-level interfaces protocols are implemented
- Describe how pointer interfaces are implemented

Interface Type: Memory Interface

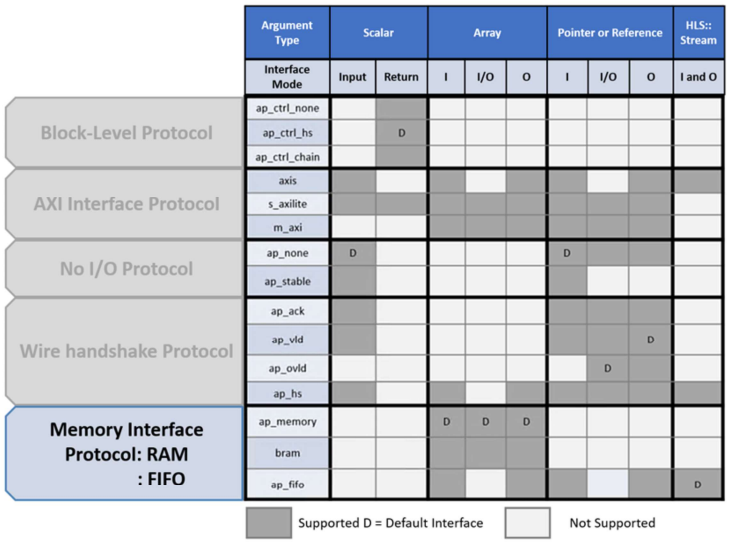
Memory interface protocols can be inferred from:

Array Arguments

- Array arguments can be synthesized to RAM or FIFO ports
  - When the FIFOs are specified on array ports, the ports must be read only or write only

Pointer Arguments

- Pointer (and references in C++) argument can be synthesized to FIFO ports



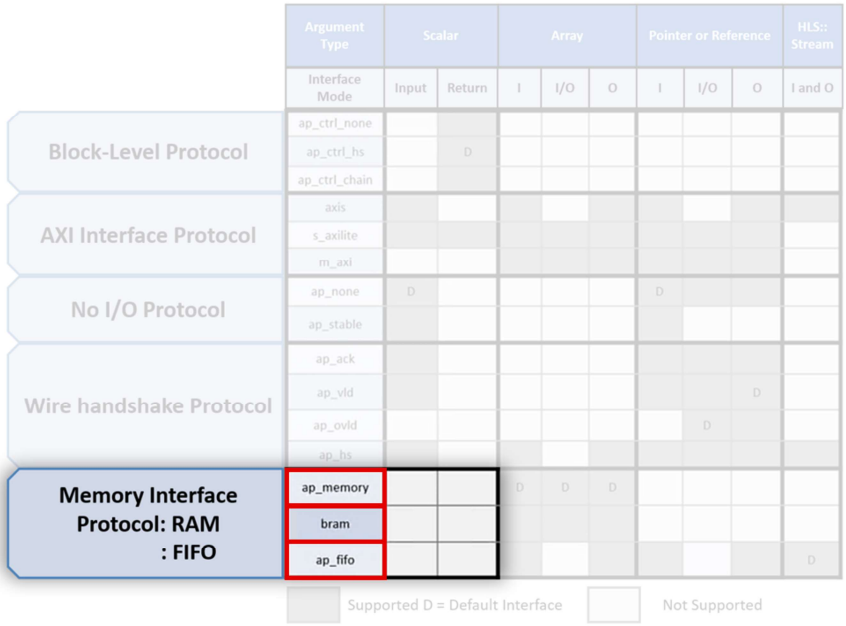
XILINX.

Let's learn how a port-level protocol is implemented using memory interface protocols.

The memory interface protocol can be inferred from array and pointer arguments.

- The array arguments can be synthesized to RAM or FIFO ports.
  - When the FIFOs are specified on array ports, the ports must be read only or write only.
- The pointer and references in C++ argument can be synthesized to FIFO ports.

Memory I/O Protocols



XILINX.

There are three types of memory interfaces that can be implemented:

- ap_memory
- bram
- ap_fifo

ap_memory

Array arguments are implemented by default as an ap_memory interface

ap_memory interface may be implemented as:

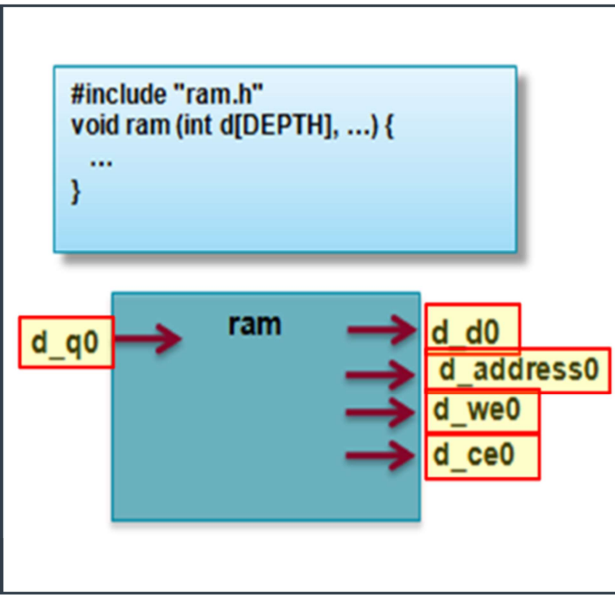
The array arguments are implemented by default as an ap_memory interface.

An ap_memory interface may be implemented as a single-port or a dual-port RAM.

ap_memory

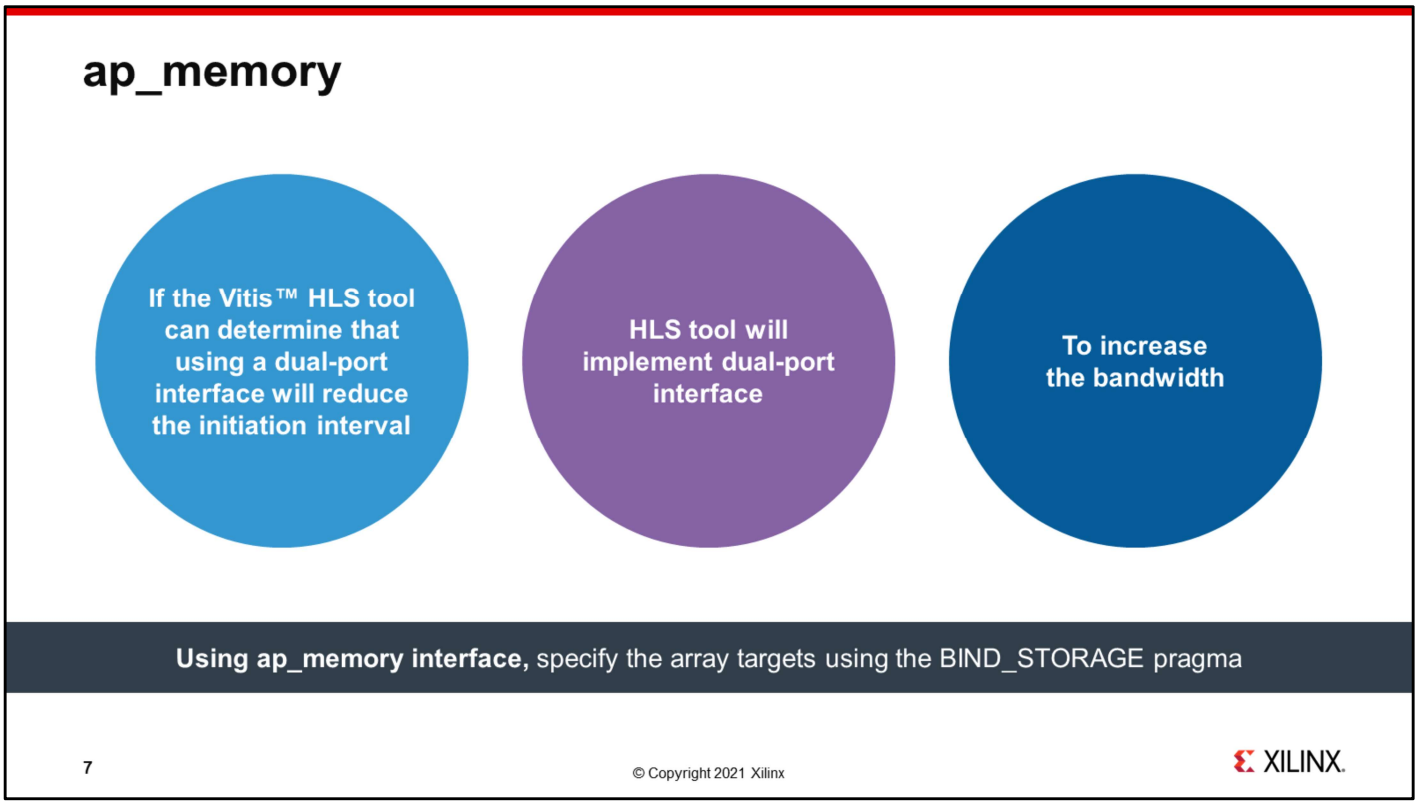
Example shown here is a standard single port block RAM interface with:

- Data
- Address
- Write-enable
- Chip-enable



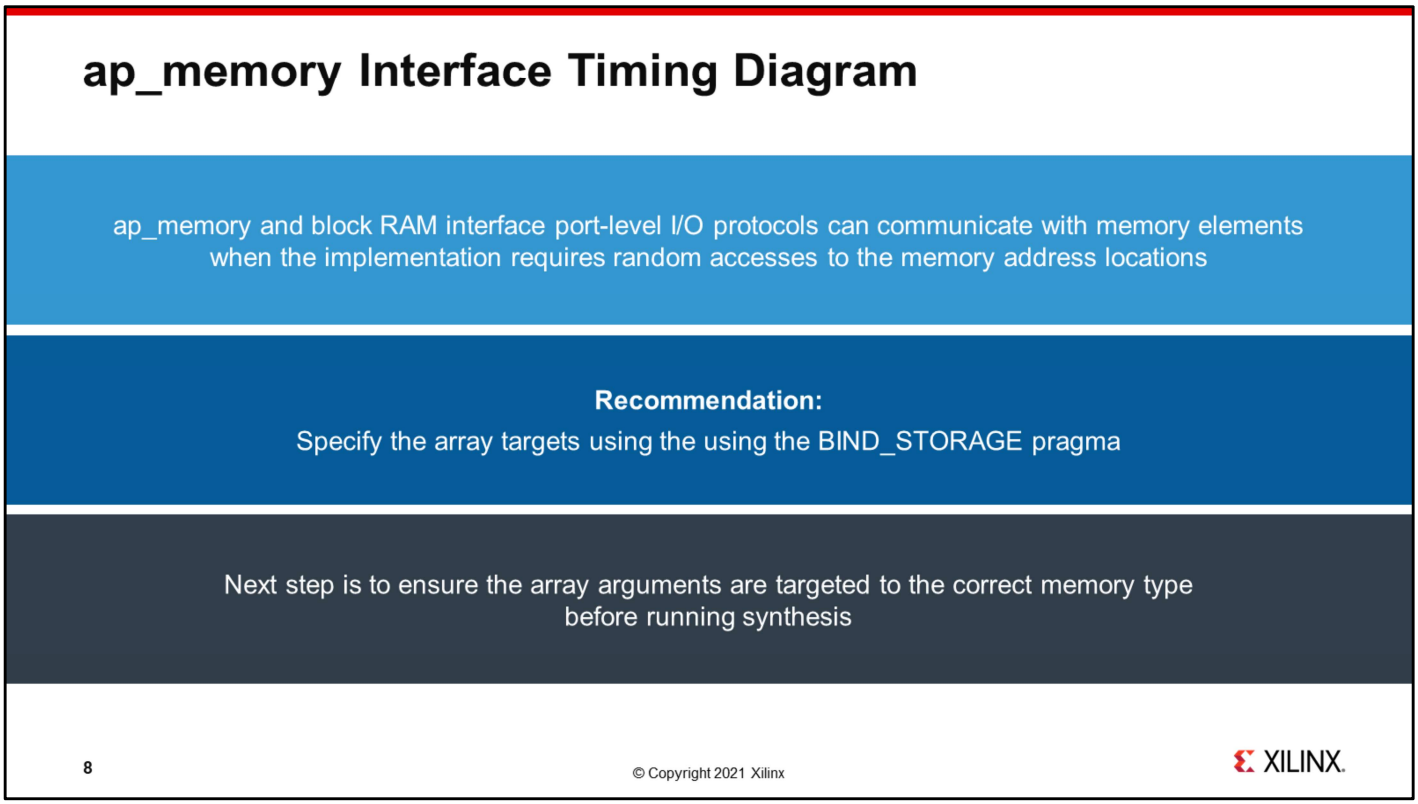
XILINX.

The example here shows the single port RAM. This is a standard block RAM interface with data, address, write-enable, and chip-enable ports generated.



If the Vitis HLS tool can determine that using a dual-port interface will reduce the initiation interval, it will automatically implement a dual-port interface to increase the bandwidth.

When using an ap_memory interface, specify the array targets using the BIND_STORAGE pragma. If no target is specified for the arrays, Vitis HLS determines whether to use a single or dual-port RAM interface.



Let's understand ap_memory interface through a timing diagram.

The ap_memory and block RAM interface port-level I/O protocols can communicate with memory elements (for example, RAMs and ROMs), when the implementation requires random accesses to the memory address locations.

When using an ap_memory interface, it is recommended to specify the array targets using the BIND_STORAGE pragma.

Once this is done, the next step is to ensure the array arguments are targeted to the correct memory type before running synthesis.

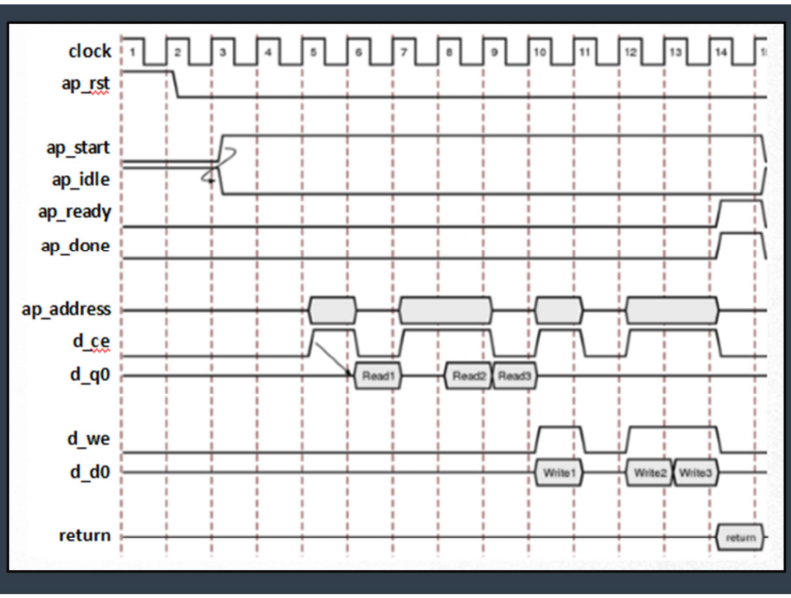
ap_memory Interface Timing Diagram

An array d is specified as a single-port block RAM

Port names are based on the C function arguments

Once the reset is applied:

- After start → block begins normal operation
- Reads are performed by applying an address on the output address ports while asserting the output signal d_ce
- Writes are performed by asserting the output ports d_ce and d_we while simultaneously applying the address and output data d_d0

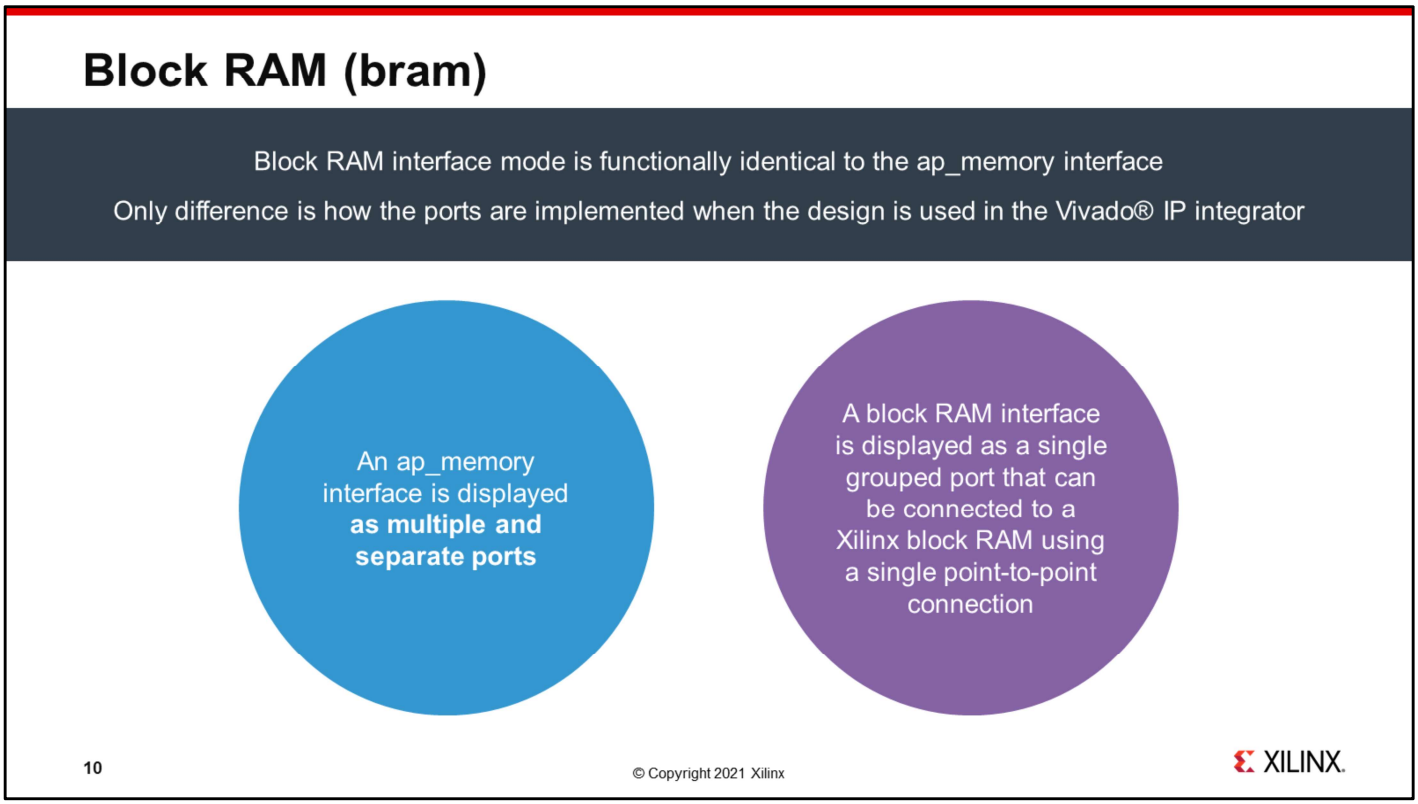


XILINX.

The graphic here shows the timing diagram of the ap_memory interface when an array named d is specified as a single-port block RAM. The port names are based on the C function argument. For example, if the C argument is d, the chip enable is d_ce, and the input data is d_q0 based on the output/q port of the block RAM etc.

Once the reset is applied:

- After start is applied, the block begins normal operation.
- Reads are performed by applying an address on the output address ports while asserting the output signal d_ce.
- For a default block RAM, the design expects the input data d_q0 to be available in the next clock cycle.
- Write operations are performed by asserting the output ports d_ce and d_we while simultaneously applying the address and output data d_d0.

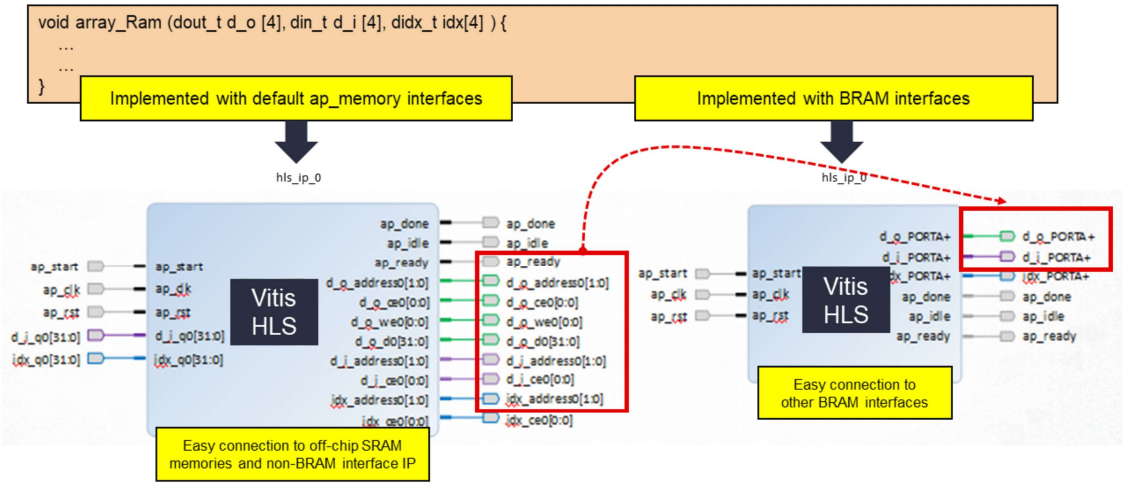


The block RAM interface mode is functionally identical to the ap_memory interface.

The only difference is how the ports are implemented when the design is used in the Vivado IP integrator:

- An ap_memory interface is displayed as multiple and separate ports.
- A block RAM interface is displayed as a single grouped port that can be connected to a Xilinx block RAM using a single point-to-point connection.

Block RAM (bram)



Default ap_memory interfaces

Connections to off-chip SRAM memories and non-block RAM interface IP are available

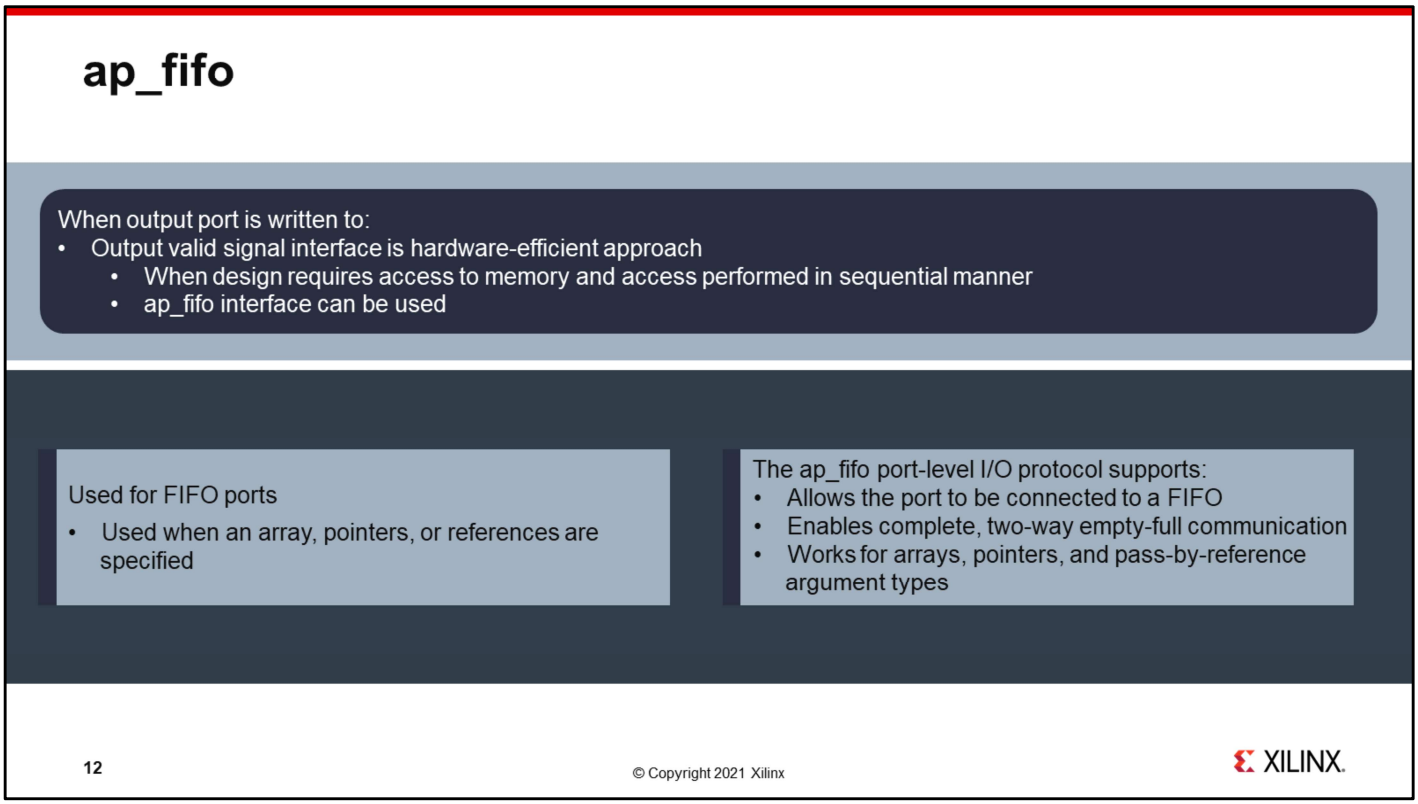
Connections to other block RAM interfaces are available

XILINX.

The block on the left-hand side shows the implementation of a code with a default ap_memory interfaces. The connections to off-chip SRAM memories and non-block RAM interface IP are available.

The block on the right-hand side shows the implementation of a code with block RAM interfaces. The connections to other block RAM interfaces are available.

The difference between the number of ports that are generated for these interfaces can be seen here.



When an output port is written to, its associated output valid signal interface is the most hardware-efficient approach when the design requires access to a memory element, and the access is always performed in a sequential manner. If the array is accessed in a sequential manner, an ap_fifo interface can be used. This interface is used for FIFO ports. It is used when an array, pointers, or references are specified.

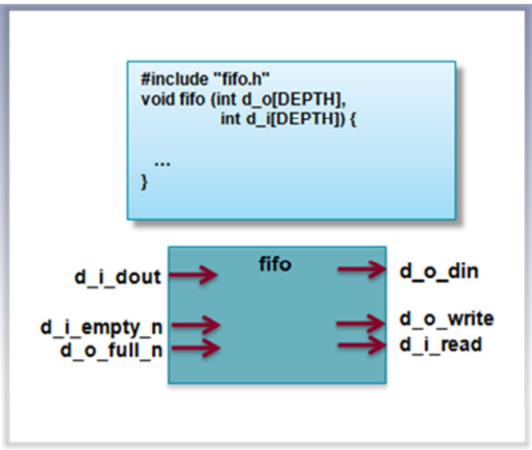
The ap_fifo port-level I/O protocol supports the following:

- Allows the port to be connected to a FIFO
- Enables complete, two-way empty-full communication
- Works for arrays, pointers, and pass-by-reference argument types

ap_fifo

Example shown is a FIFO example and the standard read/write and full/empty ports are generated

- There must be separate arrays for read and write
- For pointers/references, split it into in and out ports
- It uses the standard FIFO single port model
- It requires single cycle for both reads and writes



Note: Vitis HLS tool will halt if it determines the data access is not sequential and thus reports a warning

XILINX.

The example shown is a FIFO example and the standard read/write and full/empty ports are generated.

- There must be separate arrays for read and write.
- For pointers/references, split it into in and out ports.
- It uses the standard FIFO single port model. It requires single cycle for both reads and writes.
- It is to be noted that the Vitis HLS tool will halt if it determines the data access is not sequential and thus reports a warning.

ap_fifo Read Interface Timing Diagram

ap_fifo port-level I/O protocol

14

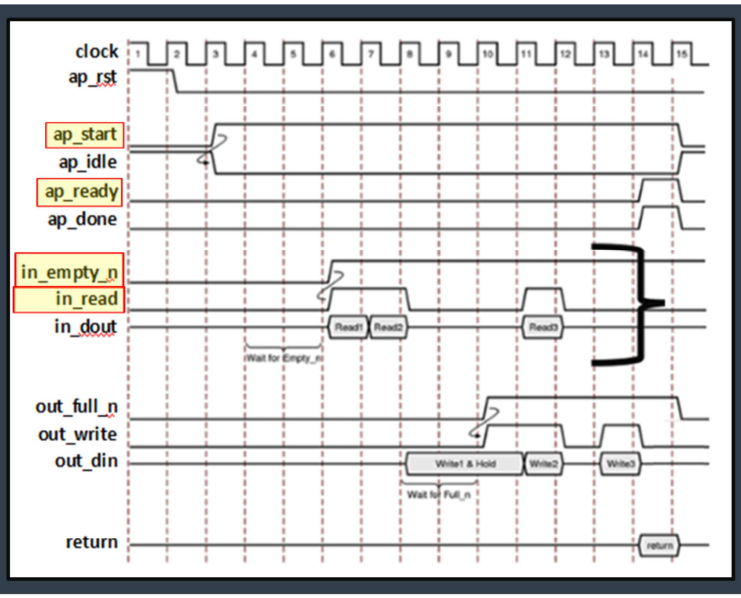
XILINX.

The ap_fifo port-level I/O protocol allows the port to be connected to a FIFO and enables complete, two-way empty/full communication. It also supports arrays, pointers, and pass-by-reference argument types.

ap_fifo Read Interface Timing Diagram

During the inputs or ap_fifo read operation:

- After start → block begins normal operation
- If the input port is ready to be read but the FIFO is empty → the design stalls and waits for data to become available
- When the FIFO contains data → an output acknowledges and the in_read is asserted High to indicate that the data was read



XILINX.

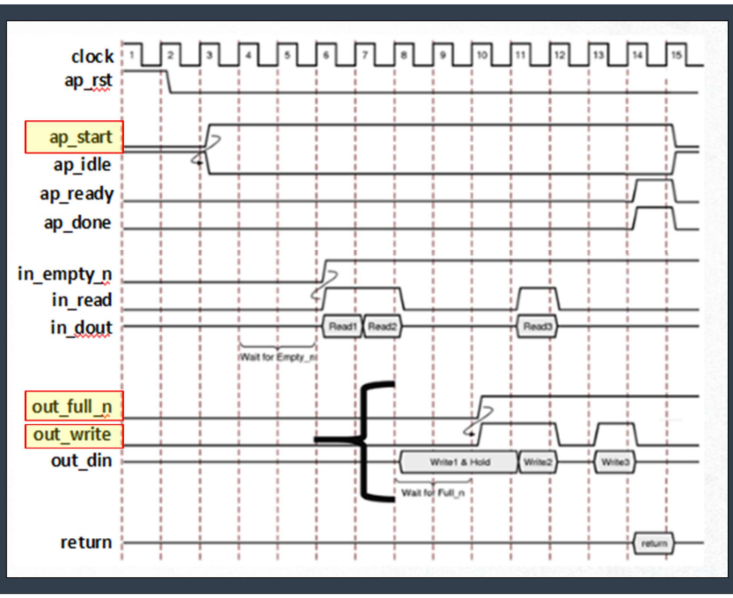
During the inputs or ap_fifo read operation, the following occurs:

- After start is applied, the block begins normal operation.
- If the input port is ready to be read but the FIFO is empty as indicated by input port in_empty_n Low, the design stalls and waits for data to become available.
- When the FIFO contains data as indicated by input port in_empty_n High, an output acknowledges and the in_read is asserted High to indicate that the data was read in this cycle.

ap_fifo Write Interface Timing Diagram

During the outputs or ap_fifo write operation, the following occurs:

- After start → block begins normal operation
- If an output port is ready to be written to but the FIFO is full → the data is placed on the output port but the design stalls and waits for the space to become available in FIFO
- When space becomes available in the FIFO → the output acknowledges and the signal out_write is asserted to indicate
- If the top-level function or the top-level loop is pipelined using the-rewind option → Vitis HLS tool creates an additional output port with the suffix _lwr

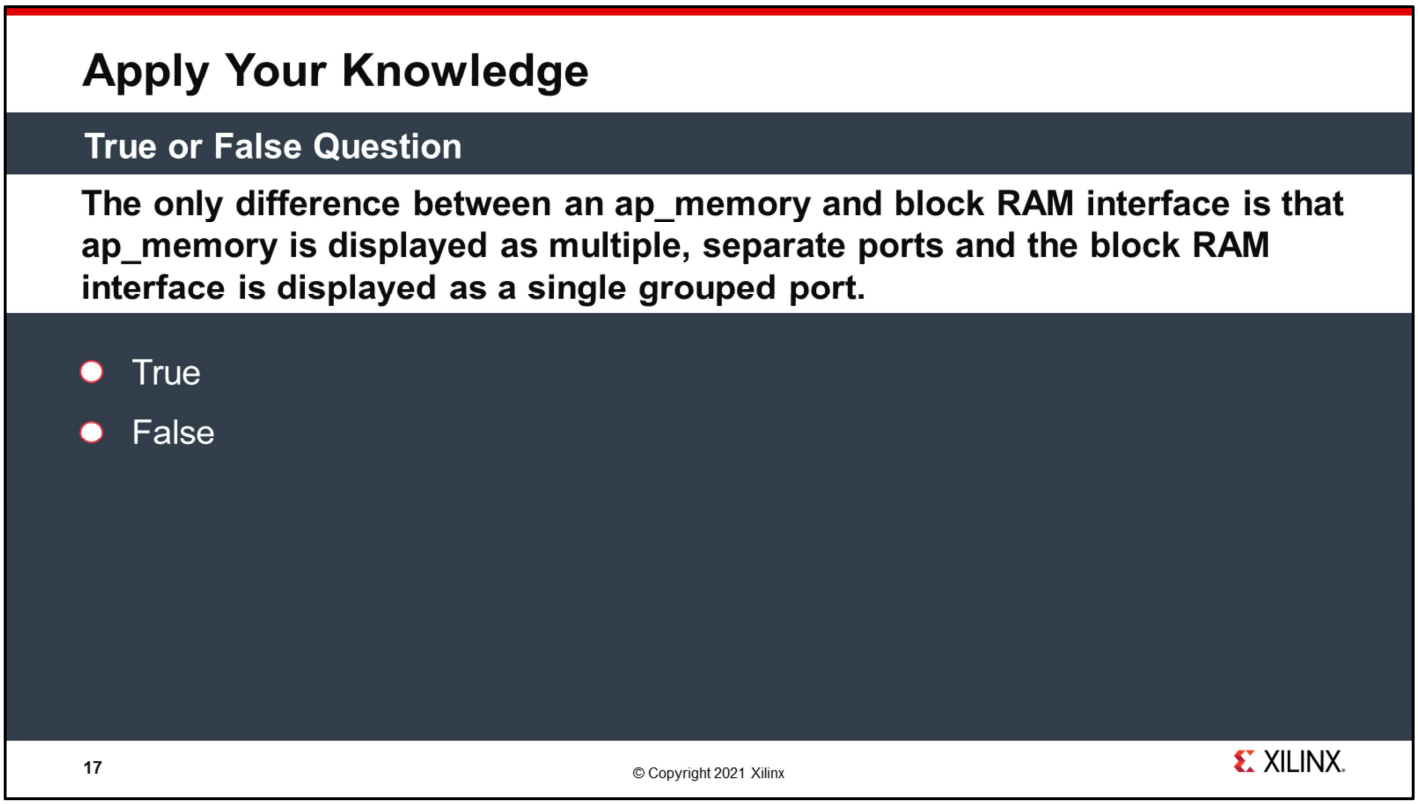


XILINX.

During the outputs or ap_fifo write operation, the following occurs:

- After start is applied, the block begins normal operation.
- If an output port is ready to be written to but the FIFO is full as indicated by out_full_n Low, the data is placed on the output port but the design stalls and waits for the space to become available in the FIFO.
- When space becomes available in the FIFO as indicated by out_full_n High, the output acknowledges and the signal out_write is asserted to indicate that the output data is valid.
- If the top-level function or the top-level loop is pipelined using the -rewind option, the Vitis HLS tool creates an additional output port with the suffix _lwr.

When the last write to the FIFO interface completes, the _lwr port goes active High.



Correct answer:
True

Correct feedback:
You selected the correct answer.

Incorrect feedback:
The correct answer is “True”.

Apply Your Knowledge

Multiple Choice Question

If there is an ap_memory interface used, the Vitis HLS tool will try to reduce the initiation interval by doing what?

- By default, the Vitis HLS tool implements a single-port interface only
- Automatically implement a dual-port interface
- User will need to specify the interface used by the Vitis HLS tool by using the BIND_STORAGE directive
- None of the above

Correct answer:
Automatically implements a dual-port interface

Correct feedback:
You selected the correct answer.

Incorrect feedback:
The correct answer is “Automatically implements a dual-port interface”.

Try again feedback:
Hint: The Vitis HLS tool always tries to reduce the initiation interval.

Summary

- Array arguments are implemented by default as an ap_memory interface
  - Array arguments can by synthesized to RAM or FIFO ports
  - Pointer (and references in C++) can be synthesized to FIFO ports
- An ap_memory interface can be implemented as a single-port or dual-port interface
- An ap_fifo interface is used for FIFO ports