



XILINX

ALL PROGRAMMABLE™

Creating Memory

2014.1

Objectives

After completing this module, you will be able to:

- Explain the difference between instantiation and inference
- Enumerate the different types of memory available in a Xilinx FPGA
- Describe the coding practice used to infer memory

Review



- **Review**
- Coding to Instantiate Memory
- Coding to Infer Memory
- Summary

Instantiation versus Inference

- **Instantiation** *explicitly* defines what silicon resources are to be used
 - Instantiation is the basis for the structural coding style
- **Inference** *suggests* what silicon resource might be used
 - Synthesis tools make decision
 - Synthesis tool options help guide the determination

```
myDFF: process (clk)
begin
    if rising_edge(clk) then
        if (reset = '1') then
            Q <= '0';
        else
            Q <= D;
        end if;
    end if;
end process myDFF;
```

Inferring a D type Flip-Flop
with synchronous reset

```
library UNISIM;
use UNISIM.vcomponents.all;
```

```
myDFF: FDR port map
(D=>D, C=>clk,
 R=>reset, Q=>Q);
```

Instantiating a D type Flip-Flop
with synchronous reset

When to Instantiate and When to Infer

➤ Inference is strongly recommended

- Allows the tools the flexibility to make performance and resource choices
- Tools optimize more efficiently
- Faster to code
- More portability among families
- Simulate faster

➤ Some primitives cannot be inferred

- Some operations of DSP blocks
- Some types of memories

➤ When it cannot be inferred, try

- Vivado® IP catalog

FPGA Memory Resources

➤ Distributed memory

- Many LUTs* can serve as small memories
 - Amount and size of LUT memories varies across the FPGA families
- Configurable as single, dual, or multi-port**

➤ Block

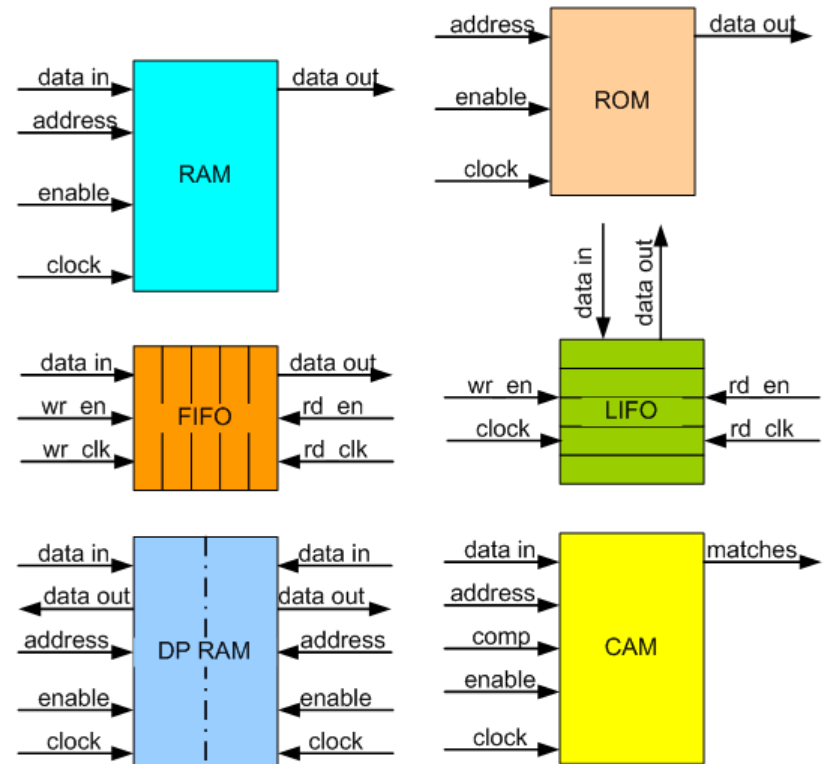
- Larger chunks of memory
 - Many block RAMs have special capabilities such as
 - True dual-port***
 - Built-in FIFO
 - ECC
 - Power management

➤ External

- Largest units of memory
- Xilinx provides interfaces for various types of memory (FLASH or DDR2, for example)

Commonly Used Types of Memory

- **RAM** – random access R & W
- **ROM** – random access R only
- **FIFO** – First-In, First-Out
 - Synch or Asynch
- **LIFO** – "stack"
- **Associative/CAM**
 - Reports matches between contents and data
- **Dual-Port**
 - Enables access to the same memory contents from two locations




Coding to Instantiate Memory



- Review
- **Coding to Instantiate Memory**
- Coding to Infer Memory
- Summary

Requirements and Suggestions

- **Designer must have intimate understanding of the specific FPGA architecture**
- **Designer must preplan how the resources are to be utilized before beginning**
 - Instantiation does not allow swapping of implementation types without recoding
- **Have a copy of the library reference guide handy**
 - The Language Templates  provide a convenient shortcut
- **Best design practice**
 - Infer where possible—many primitives are supported
 - Vivado IP catalog is the next best choice
 - Create a "wrapper" with a standard port signal set and implement the specifics within the wrapper
 - This allows the designer to have multiple wrappers for different implementations and architectures without having to modify the "calling" module

Memory Primitives

➤ **Varies by family**

- Size, speed, quantity, capabilities, total block RAMs/select (distributed) RAMs

➤ **7 series FPGA examples**

- RAM16x4S (created from MLUTs)
- RAM128x1D (Dual port created from MLUTs)
- RAM32M (Multi port created from block RAM)
- RAMB18SDP (block RAM)
- RAMB36SDP (block RAM)

➤ **Other families have similar members**

➤ **Library guide contains**

- Instantiation templates in both VHDL and Verilog (as does the Language Templates)
- Detailed information on all the options

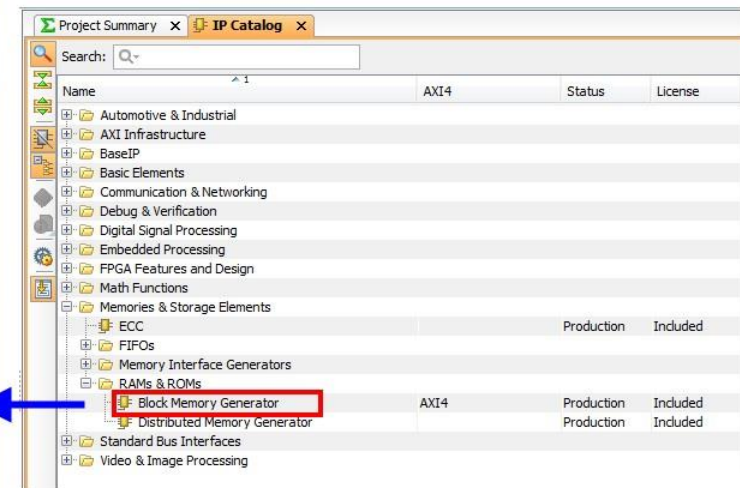
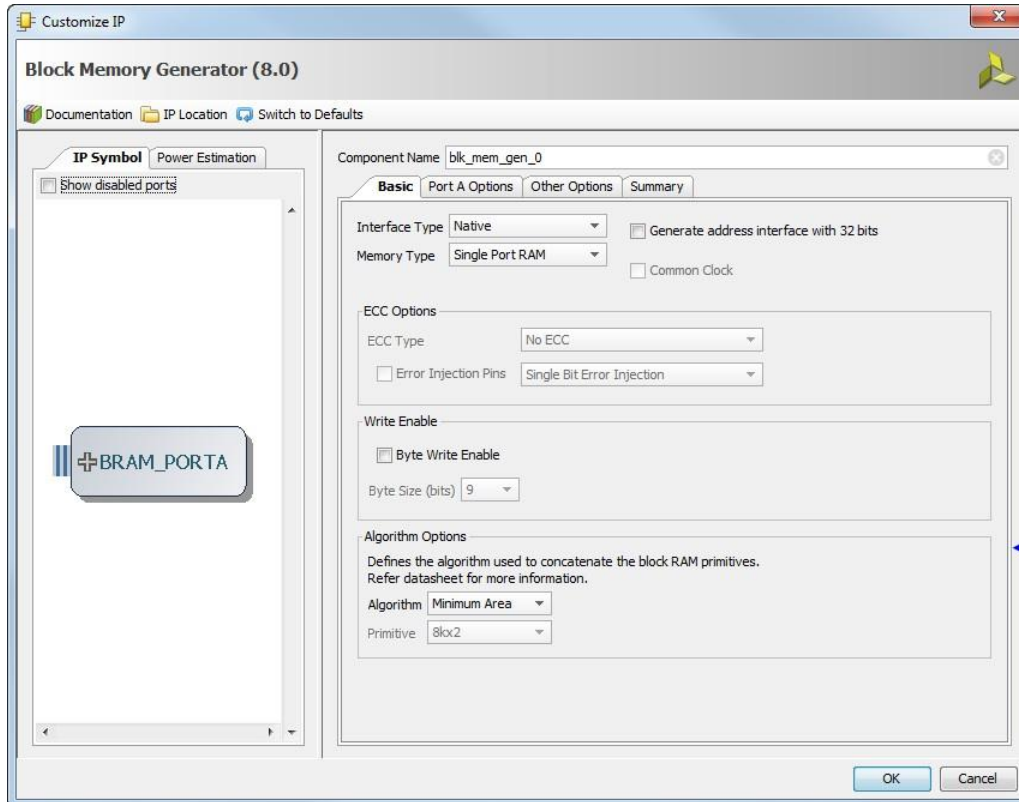
Instantiating the Memory Primitive

- Include the UNISIM library as it contains the primitive component's declaration
 - Otherwise designer is responsible for declaring the primitive
- Use the Language Templates to reduce typographical errors and insure all ports and generics are used

```
library UNISIM;  
use UNISIM.vcomponents.all;  
  
bram1: RAMB36_inst : RAMB36  
  generic map (DOA_REG => 0,  
    <snip>  
  )  
  port map (  
    CASCADEOUTLATA => CASCADEOUTLATA,  
    <snip>  
  );
```

IP Catalog

➤ Customize IP can quickly produce many types of memory configurations



Coding to Infer Memory



- Review
- Coding to Instantiate Memory
- **Coding to Infer Memory**
- Summary

What Kind of Memories Can Be Inferred?

➤ Single-port RAMs and ROMs

- Synchronous can utilize block RAM or distributed RAM
Asynchronous utilizes only distributed RAM

➤ Simple dual- and multi-port RAMs and ROMs

- Obviously, the ROM has no write access

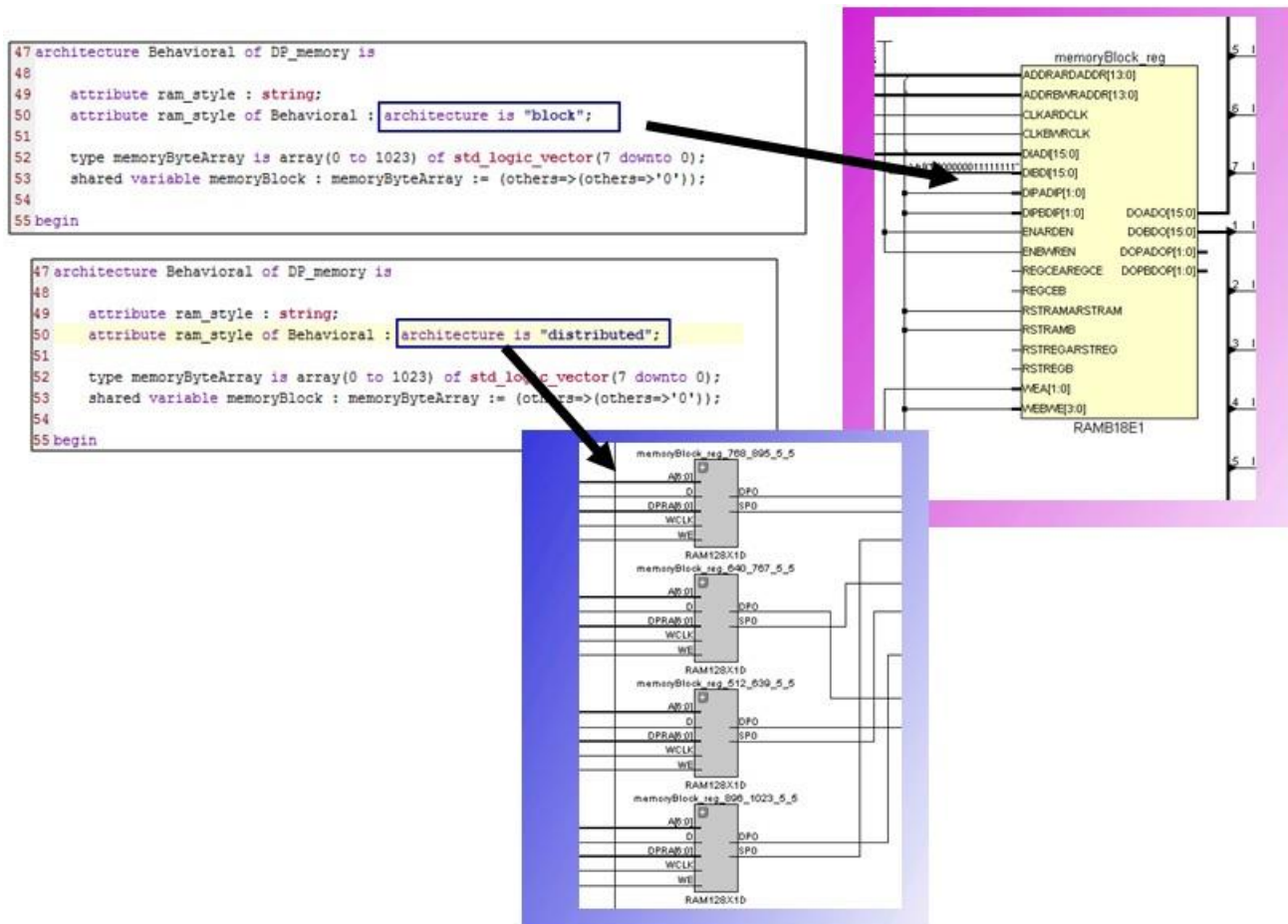
➤ Single and dual-port RAMs

- With optional parity
- True dual-port utilizes only block RAMs

➤ FIFOs

- Controls the address and data lines to/from the memory
 - But does not infer the hardware FIFO controller
- Requires the use of shared variables
 - Covered in the *Advanced VHDL* course

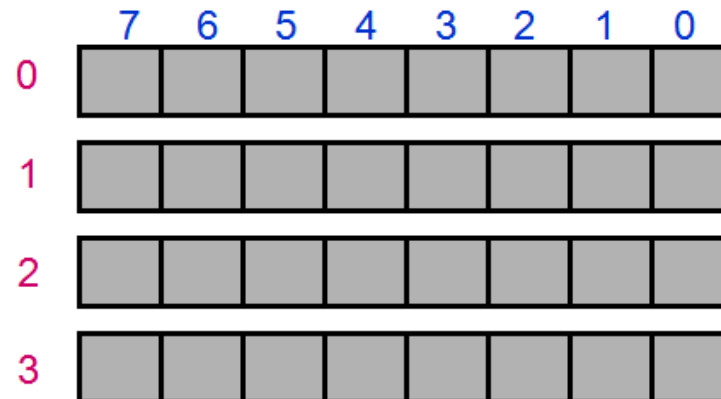
Synthesis Options Impact Results



Memories are Arrays

- When modeling memory structures, creating a two-dimensional array structure is necessary
 - Typically done by creating a one-dimensional array of `std_logic_vector`
 - Commonly used so that data can be managed by word, not bit
- Effectively an array of arrays

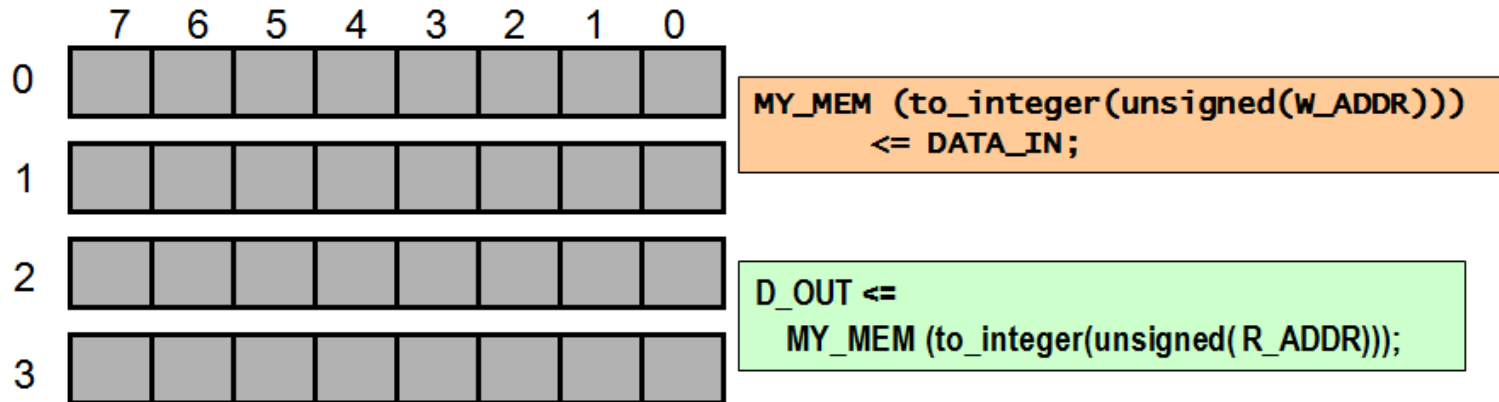
```
type MEM_ARRAY is array (0 to 3) of std_logic_vector (7 downto 0);  
signal MY_MEM : MEM_ARRAY := (others=>(others=>'0'));
```



Getting Data In and Out of Memory

- For most memory applications, the read and write address vector is converted to integer, referencing an element within the array
 - The `to_integer` function is located in the `ieee.numeric_std` package

```
type MEM_ARRAY is array ( 0 to 3 ) of std_logic_vector(7 downto 0);  
signal MY_MEM : MEM_ARRAY ;  
  
signal R_ADDR, W_ADDR : std_logic_vector(1 downto 0) ;
```



Initializing a ROM Array

- For ROMs, an aggregate is a convenient means for initializing the array

```
type ROM_ARRAY is array (0 to 3) of std_logic_vector (7 downto 0);  
constant MY_ROM : ROM_ARRAY := ( 0 => (others => '1'),  
                                  1 => "10100010",  
                                  2 => "00001111",  
                                  3 => "11110000" );
```

	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	1
1	1	0	1	0	0	0	1	0
2	0	0	0	0	1	1	1	1
3	1	1	1	1	0	0	0	0

Synchronous ROM

- For synchronous ROM, the memory access occurs within a clocked VHDL process
 - This example contains an enable signal

```
type ROM_ARRAY is array ( 0 to 3 ) of  
std_logic_vector ( 7 downto 0 );  
constant MY_ROM : ROM_ARRAY := (0 => (others => '1'),  
1 => "10100010",  
2 => "00001111",  
3 => "11110000" ) ;
```

```
process ( CLK )  
begin  
    if rising_edge ( CLK ) then  
        if ( READ_EN = '1' ) then  
            D_OUT <= MY_ROM(to_integer(unsigned(ADDR)));  
        end if;  
    end if;  
end process;
```

Initializing Memory

- RAMs and ROMs can be pre-loaded with an initial set of values on power-up

- Three common mechanisms for initializing memory (both RAM and ROM)

- In the signal definition
 - With aggregates

```
type MEM_ARRAY is array ( 0 to 3 ) of std_logic_vector ( 7 downto 0 );  
constant MY_MEM : MEM_ARRAY := ( 0 => (others => '1'), 1 => "10100010",  
                                   2 => "00001111", 3 => "11110000" );
```

- In code
 - RAM memory can be filled by a function or explicit value settings
- From a text file
 - Requires the use of the TEXTIO library and the file type
 - Covered in the *Advanced VHDL* course

Initialization in Code

➤ Especially useful when true dual-port memory is used

- But only one side is used by the application
- Second side can be used for loading data by a "memory initialization process"

➤ Can be done with a single process

- Issue becomes signal conflicts if multiple processes use the memory
 - One process can initialize memory and another access the memory, but signal conflicts must be managed via a multiplexer or equivalent circuitry
- Simpler approach is to initialize the memory with the reset statement
 - Once reset is released at the "system" level, the process must remain in reset until the memory is completely initialized.

Single Process Memory Initialization Example

```
doMemory: process (clk)
    type MEM_ARRAY is array ( 0 to 255 ) of std_logic_vector ( 7 downto 0);
    variable MY_MEM : MEM_ARRAY := (others=>(others=>'U'));
    variable initializingMemory : std_logic := 'U';
    variable initializationAddress : unsigned (ADDR_WIDTH-1 downto 0);
    variable valueToLoad : integer range 0 to 255;
begin
    syncEvents: if rising_edge(clk) then
        if (reset = '1') then
            initializingMemory := '1';
            initializationAddress := (others=>'0');
        elsif (initializingMemory = '1') then
            valueToLoad := to_integer(initializationAddress) * 3 + 1;
            MY_MEM(to_integer(initializationAddress)) :=
                std_logic_vector(to_unsigned(valueToLoad,DATA_WIDTH));
            initializationAddress := initializationAddress + 1;
            testDone: if (initializationAddress = 0) then
                initializingMemory := '0';
            end if testDone;
        else
            -- normal memory access...
```

Summary

- Review
- Coding to Instantiate Memory
- Coding to Infer Memory
- **Summary**



Apply Your Knowledge

1. What is the best way to create memory?
2. How else can memory be created inside an FPGA?

Summary

- **Many types of memory can be inferred**
 - For those that cannot be, use the IP catalog
- **The basic data structure to infer memory is an array of `std_logic_vectors`**
 - The "depth" of the memory is specified by the length of the array
 - The "width" of the memory is controlled by the length of the `std_logic_vector`
- **More complex memory types, such as FIFOs, LIFOs, or multi-port, for example, can be built from this basic data structure**
 - Even though the memory structure can be built, it may not use the hardware resources such as the built-in hardware FIFO in the block RAM tiles

What's Next?

➤ Now you know

- Several ways to create memory in an FPGA

➤ What do you do now?

- First, a lab where memory will be inferred, then
- An introduction to Finite State Machines (FSMs)