



# CprE 381: Computer Organization and Assembly Level Programming

Lab Week 4 VHDL

Henry Duwe  
Electrical and Computer Engineering  
Iowa State University

# Modeling Memory in VHDL

```
architecture rtl of mem is

    -- Build a 2-D array type for the RAM
    subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
    type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;

    -- Declare the RAM signal and specify a default value. Quartus Prime
    -- will load the provided memory initialization file (.mif).
    signal ram : memory_t;

begin

    process(clk)
    begin
        if(rising_edge(clk)) then
            if(we = '1') then
                ram(to_integer(unsigned(addr))) <= data;
            end if;
        end if;
    end process;

    q <= ram(to_integer(unsigned(addr)));

end rtl;
```

# Modeling Memory in VHDL

```
architecture rtl of mem is
```

```
-- Build a 2-D array type for the RAM
subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;
```

Defining data type

```
-- Declare the RAM signal and specify a default value. Quartus Prime
-- will load the provided memory initialization file (.mif).
```

```
signal ram : memory_t;
```

Instantiation

```
begin
```

```
process(clk)
begin
if(rising_edge(clk)) then
    if(we = '1') then
        ram(to_integer(unsigned(addr))) <= data;
    end if;
end if;
end process;
```

```
q <= ram(to_integer(unsigned(addr)));
```

```
end rtl;
```

# Modeling Memory in VHDL

```
architecture rtl of mem is

    -- Build a 2-D array type for the RAM
    subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
    type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;

    -- Declare the RAM signal and specify a default value. Quartus Prime
    -- will load the provided memory initialization file (.mif).
    signal ram : memory_t;

begin

    process(clk)
    begin
        if(rising_edge(clk)) then
            if(we = '1') then
                ram(to_integer(unsigned(addr))) <= data;
            end if;
        end if;
    end process;

    q <= ram(to_integer(unsigned(addr)));

end rtl;
```

Synchronous write

Asynchronous read



# Word Vs. Byte Addressable memory

## word Addressable

- This is the format provided in lab
- Must read/write 4 bytes at a time
- Only 4 addresses (0x00-0x03)
- Big Endian format

<b>00</b>	0x14120900
<b>01</b>	0x44220215
<b>10</b>	0xFFFFFFFF
<b>11</b>	0x00000000

## Byte Addressable

- This is the format RISC-V uses
- Can read/write to individual bits
- 16 addresses (0x00-0x0F)

	<b>00</b>	<b>01</b>	<b>10</b>	<b>11</b>
<b>00</b>	0x00	0x09	0x12	0x14
<b>01</b>	0x15	0x02	0x22	0x44
<b>10</b>	0xFF	0xFF	0xFF	0xFF
<b>11</b>	0x00	0x00	0x00	0x00

# Simulation Hierarchy

```
entity tb_dffg is
  generic(gCLK_HPER : time := 50 ns);
end tb_dffg;
```

← Simulated Entity

```
DUT: dffg
port map(i_CLK => s_CLK,
         i_RST => s_RST,
         i_WE  => s_WE,
         i_D   => s_D,
         o_Q   => s_Q);
```

```
architecture mixed of dffg is
  signal s_D : std_logic; -- Multiplexed input to the FF
  signal s_Q : std_logic; -- Output of the FF
```

# Simulation Hierarchy

The screenshot displays a simulation tool interface with three main panels:

- sim - Default:** A tree view showing the simulation hierarchy. The 'tb\_dffg' instance is highlighted with a red circle, and a red arrow points from it to the 'Simulated Entity' text below. Below 'tb\_dffg' are its sub-instances: 'DUT', 'line\_40', 'line\_44', 'line\_51', 'P\_CLK', and 'P\_TB'.
- Objects:** A table listing the objects in the simulation.
- Wave - Default:** A waveform viewer showing signals over time.

Name	Value	Signal	Internal
s_Q	0	Signal	Internal
s_D	0	Signal	Internal
o_Q	0	Signal	Out
i_WE	0	Signal	In
i_RST	0	Signal	In
i_D	1	Signal	In
i_CLK	0	Signal	In

Signal	Value
/tb_dffg/DUT/s_Q	0
/tb_dffg/DUT/s_D	0

Simulated Entity

# Simulation Hierarchy

```
entity tb_dffg is
```

```
    generic(gCLK_HPER : time := 50 ns);  
end tb_dffg;
```

← Simulated Entity

```
DUT: dffg
```

```
    port map(i_CLK => s_CLK,  
            i_RST => s_RST,  
            i_WE  => s_WE,  
            i_D   => s_D,  
            o_Q   => s_Q);
```

← Component Instantiation

```
architecture mixed of dffg is
```

```
    signal s_D : std_logic;    -- Multiplexed input to the FF  
    signal s_Q : std_logic;    -- Output of the FF
```



# Simulation Hierarchy

The screenshot displays the simulation hierarchy in a tool like ModelSim. The 'Instance' window on the left shows a tree structure where 'tb\_dffg' is circled in red. A red arrow points from this instance to the text 'Component Instantiation'. Below this, the text 'Simulated Entity' is shown. The 'Objects' window in the center lists various signals and their values: s\_Q (0), s\_D (0), o\_Q (0), i\_WE (0), i\_RST (0), i\_D (1), and i\_CLK (0). The 'Wave' window on the right shows the timing waveforms for the signals /tb\_dffg/DUT/s\_Q and /tb\_dffg/DUT/s\_D.

Name	Value	Signal	Internal
s_Q	0	Signal	Internal
s_D	0	Signal	Internal
o_Q	0	Signal	Out
i_WE	0	Signal	In
i_RST	0	Signal	In
i_D	1	Signal	In
i_CLK	0	Signal	In

Wave - Default	Msgs
/tb_dffg/DUT/s_Q	0
/tb_dffg/DUT/s_D	0

Component Instantiation

Simulated Entity

# Simulation Hierarchy

```
entity tb_dffg is
```

```
    generic(gCLK_HPER : time := 50 ns);  
end tb_dffg;
```

Simulated Entity

```
DUT: dffg
```

Component Instantiation

```
    port map(i_CLK => s_CLK,  
            i_RST => s_RST,  
            i_WE  => s_WE,  
            i_D   => s_D,  
            o_Q   => s_Q);
```

```
architecture mixed of dffg is
```

```
    signal s_D : std_logic; -- Multiplexed input to the FF  
    signal s_Q : std_logic; -- Output of the FF
```

Internal Signals

# Simulation Hierarchy

Internal Signals

The screenshot displays three panels from a simulation tool. The left panel, titled 'sim - Default', shows a project tree with 'tb\_dffg' and 'DUT' highlighted. The middle panel, titled 'Objects', shows a table of internal signals and ports. The right panel, titled 'Wave - Default', shows a waveform viewer with signals from the 'DUT' component.

Name	Value	Signal	Internal
s_Q	0	Signal	Internal
s_D	0	Signal	Internal
o_Q	0	Signal	Out
i_WE	0	Signal	In
i_RST	0	Signal	In
i_D	1	Signal	In
i_CLK	0	Signal	In

Component Instantiation

Internal Ports

Simulated Entity

- You can find the address of a component by following this hierarchy
- Ex, the address of the internal signal s\_Q is "/tb\_dffg/DUT/s\_Q"

# Initializing memory

The screenshot displays a simulation tool interface. On the left, the 'Instance' pane shows a hierarchy: 'tb\_mem' (Design unit: tb\_mem, Architecture: Architectur) contains 'DUT' (Design unit: mem, Architecture: Architectur). 'DUT' contains several components: 'line\_39' (Process), 'line\_48' (Process), 'P\_CLK' (Process), 'P\_TB' (Process), 'standard' (Package), 'textio' (Package), and 'std\_logic\_1164' (Package). On the right, the 'Objects' pane shows a table of signals and their values:

Name	Value	Signal	Direction
we	0	Signal	In
ram	{32'hXXXXXXXX} {...	Signal	Internal
q	32'hXXXXXXXX	Signal	Out
DATA_WIDTH	32'h20	Gene...	In
data	32'h00000000	Signal	In
clk	0	Signal	In
ADDR_WIDTH	32'hA	Gene...	In
addr	10'h000	Signal	In

mem load -infile {path to memory file} – format hex {path in simulated hierarchy}

- Our goal in this case is to initialize the RAM signal, so we should use the address
  - /tb\_mem/DUT/RAM