Memories

Last updated 5/15/20

Counters

These slides review the design for several types of memories

Upon completion: You should be able to design ROMs and RAMS of various sizes and register configurations

Memories

- ROM – mux based with memory values stored as constants

Memories

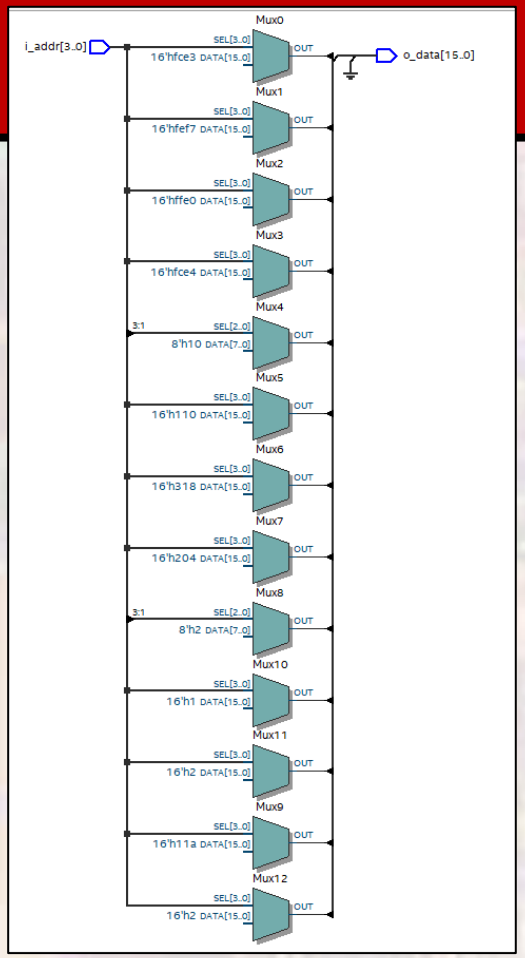
- ROM – mux based

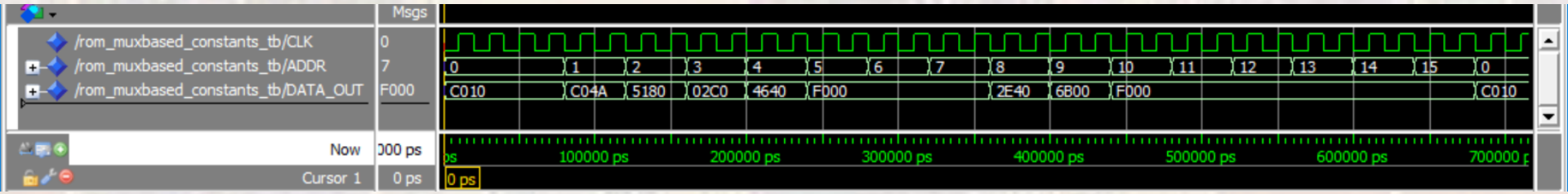
```
--
-- rom_muxbased_constants.vhdl
--
-- created 4/25/17
-- tj
--
-- rev 0
----------------------------------------
--
-- Mux based rom with constants for values
--
----------------------------------------
-- inputs: addr
-- outputs: data
----------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity rom_muxbased_constants is
   generic(
      mem_width:  positive := 16;
      mem_depth:  positive := 16
   );
   port(
         i_addr:  in    std_logic_vector(((integer(ceil(log2(real(mem_depth))))) - 1) downto 0);
         o_data:  out   std_logic_vector((mem_width - 1) downto 0)
   );
end entity;
```

```
architecture behavioral of rom_muxbased_constants is

   -- ROM structure
   type rom_type is array (0 to (mem_depth - 1)) of std_logic_vector ((mem_width - 1) downto 0);

-- ROM contents
   constant my_ROM: rom_type:=(
      0 =>  X"C010",
      1 =>  X"C04A",
      2 =>  X"5180",
      3 =>  X"02C0",
      4 =>  X"4640",

      8 =>  X"2E40",
      9 =>  X"6B00",
      10 => X"F000",

      others => X"F000"
   );

begin
   o_data <= my_ROM(to_integer(unsigned(i_addr)));

end architecture;
```

Memories

- ROM – mux based





Memories

- ROM – inferred with memory values stored in a xx.mif file

Memories

- ROM – inferred w/ mif file

```
----------------------------------------
--
-- rom_inferred.vhdl
--
-- created 4/25/17
-- tj
--
-- rev 0
----------------------------------------
--
-- 16B synchronous ROM loaded from file
--
----------------------------------------
--
-- Inputs:  clk, addr
-- Outputs: data
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity rom_inferred is
   port(
      i_clk:   in std_logic;
      i_addr:  in std_logic_vector(3 downto 0);
      o_data:  out std_logic_vector(7 downto 0)
   );
end;
```

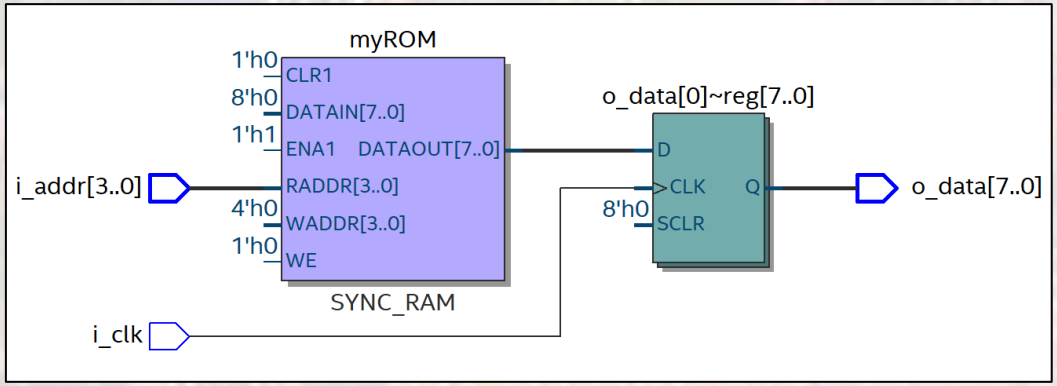
```
architecture behavioral of rom_inferred is

   type rom_type is array (0 to 15) of std_logic_vector (7 downto 0);

   -- ROM
   signal myROM: rom_type;

   -- ROM contents
   ATTRIBUTE ram_init_file: string;
   ATTRIBUTE ram_init_file of myROM: signal is "rom_init.mif";


   begin

      process (i_clk)
      begin
         if (rising_edge(i_clk)) then
            o_data <= myROM(to_integer(unsigned(i_addr)));
         end if;
      end process;

      -- Output signals

end behavioral;
```

rom_init.mif

```
WIDTH=8;
DEPTH=16;

ADDRESS_RADIX=UNS;
DATA_RADIX=HEX;

CONTENT BEGIN
        0   :   03;
        1   :   99;
        2   :   2A;
        3   :   26;
        4   :   9A;
        5   :   49;
        6   :   60;
        [7..8]  :   4F;
        9   :   60;
        10  :   49;
        11  :   9A;
        12  :   26;
        13  :   2A;
        14  :   99;
        15  :   03;
END;
```

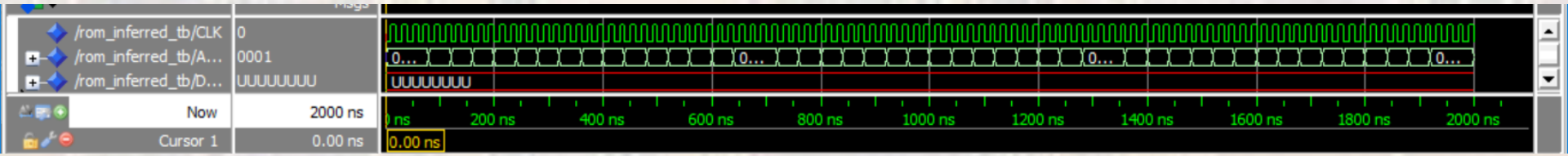
Memories

- ROM – inferred w/ mif file



Flow Status	Successful - Sun May 17 16:5
Quartus Prime Version	19.1.0 Build 670 09/22/2019
Revision Name	Class_Examples
Top-level Entity Name	rom_inferred
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	0
Total registers	0
Total pins	13
Total virtual pins	0
Total memory bits	128
Embedded Multiplier 9-bit elements	0
Total PLLs	0
UFM blocks	0
ADC blocks	0

- Cannot be simulated



Memories

- SRAM – synchronous write – generic
  - No inferred memory

Memories

- SRAM – register based

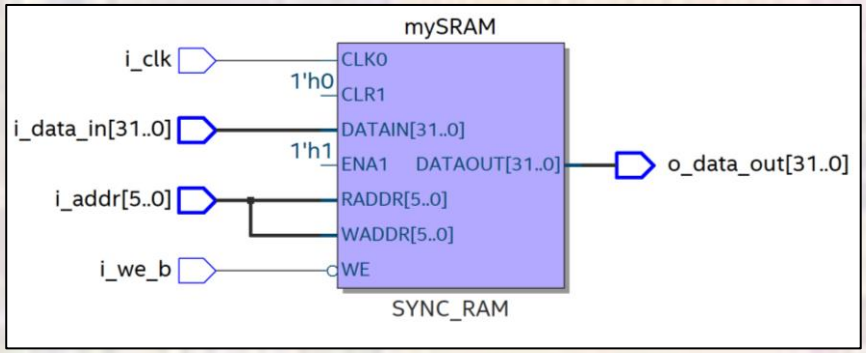
```
----------------------------------------
--
-- sram_regbased.vhdl
--
-- created 4/25/17
-- tj
--
-- rev 0
----------------------------------------
--
-- synchronous RAM built with registers
--
----------------------------------------
--
-- Inputs:  clk, addr, we_b, data_in
-- Outputs: data_out
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity sram_regbased is
   generic(
      mem_width:  positive := 32;
      mem_depth:  positive := 64
   );
   port(
      i_clk:      in    std_logic;
      i_we_b:     in    std_logic;
      i_addr:     in    std_logic_vector(((integer(ceil(log2(real(mem_depth))))) - 1) downto 0);
      i_data_in:  in    std_logic_vector((mem_width - 1) downto 0);
      o_data_out: out   std_logic_vector((mem_width - 1) downto 0)
   );
end entity;
```

```
architecture behavioral of sram_regbased is

   --
   -- create type
   --
   type sram_type is array (0 to (mem_depth - 1)) of std_logic_vector ((mem_width - 1) downto 0);
   --
   -- create memory
   --
   signal mySRAM: sram_type;

   begin

      --
      -- SRAM write process
      --
      process(i_clk)
      begin
         if (rising_edge(i_clk)) then
            -- write logic
            if(i_we_b ='0') then
               mySRAM(to_integer(unsigned(i_addr))) <= i_data_in;
            end if;
         end if;
      end process;

      --
      -- SRAM asynchronous read
      --
      o_data_out <= mySRAM(to_integer(unsigned(i_addr)));

end behavioral;
```

32 bits x 64 words
4 bytes x 64 words
8x4x64 bits (registers) = 2048

Memories

- SRAM – register based



Flow Summary

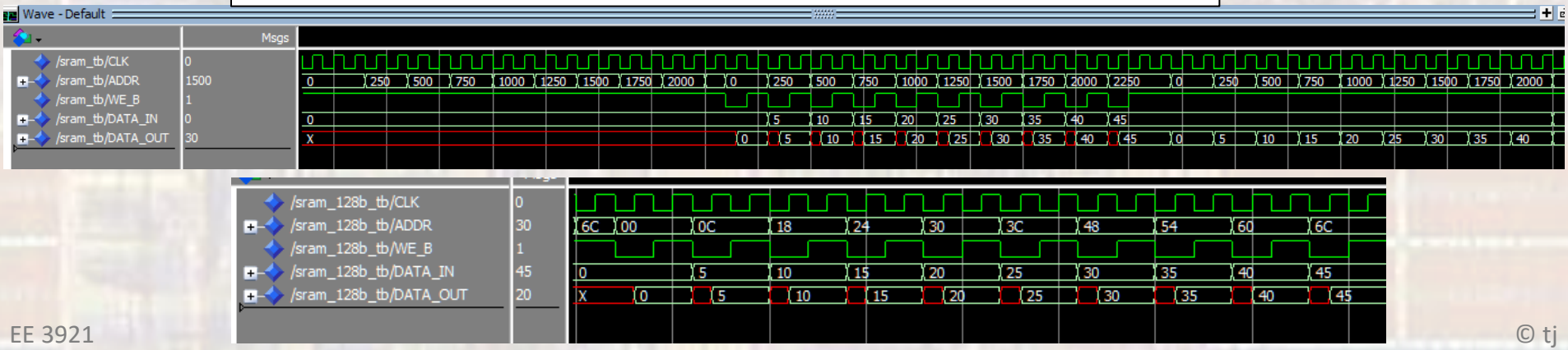
<<Filter>>

Flow Status	Successful - Fri May 15 11:1
Quartus Prime Version	19.1.0 Build 670 09/22/201
Revision Name	Class_Examples
Top-level Entity Name	sram_regbased
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	3,472
Total registers	2048
Total pins	72
Total virtual pins	0
Total memory bits	0
Embedded Multiplier 9-bit elements	0
Total PLLs	0
UFM blocks	0
ADC blocks	0

Memories

- SRAM – register based

```
-- Run Process
run: Process      -- note - no sensitivity list allowed
begin
   -- Initalize values
   ADDR <= (others => '0');
   DATA_IN <= (others => '0');
   WE_B <= '1';

   -- Read from a few addresses
   for i in 0 to 9 loop
      wait for 2*PER;
      ADDR <= std_logic_vector(to_unsigned(i*250,(integer(ceil(log2(real(mem_depth)))))));
   end loop;

   -- Write to a few addresses
   for i in 0 to 9 loop
      wait for 1*PER;
      ADDR <= std_logic_vector(to_unsigned(i*250,(integer(ceil(log2(real(mem_depth)))))));
      DATA_IN <= std_logic_vector(to_unsigned(i*5, mem_width));
      WE_B <= '0';
      wait for 1*PER;
      WE_B <= '1';
   end loop;

      -- Read from a few addresses
   for i in 0 to 9 loop
      wait for 2*PER;
      ADDR <= std_logic_vector(to_unsigned(i*250,(integer(ceil(log2(real(mem_depth)))))));
   end loop;
end process run;
```



Memories

- SRAM – synchronous read/write – generic
  - Inferred memory

Memories

- SRAM – inferred memory

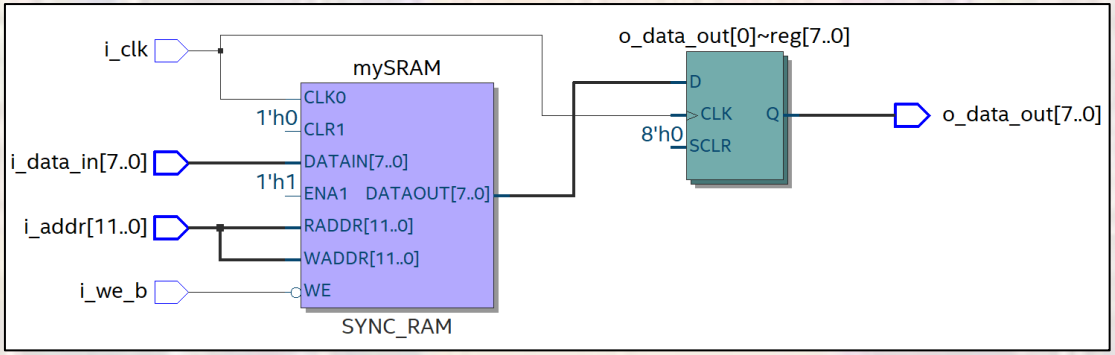
```
--------------------------------------
--
-- sram_inferred.vhdl
--
-- created 4/25/17
-- tj
--
-- rev 0
--------------------------------------
--
-- synchronous RAM using inferred memories
--
--------------------------------------
--
-- Inputs:  clk, addr, we_b, data_in
-- Outputs: data_out
--
--------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

entity sram_inferred is
   generic(
      mem_width:  positive := 8;
      mem_depth:  positive := 4096
   );
   port(
      i_clk:      in    std_logic;
      i_we_b:     in    std_logic;
      i_addr:     in    std_logic_vector(((integer(ceil(log2(real(mem_depth))))) - 1) downto 0);
      i_data_in:  in    std_logic_vector((mem_width - 1) downto 0);
      o_data_out: out   std_logic_vector((mem_width - 1) downto 0)
   );
end;
```

```
architecture behavioral of sram_inferred is

   --
   -- create type
   --
   type sram_type is array (0 to (mem_depth - 1)) of std_logic_vector ((mem_width - 1) downto 0);
   --
   -- create memory
   --
   signal mySRAM: sram_type;

   begin

      --
      -- SRAM write process
      --
      process (i_clk)
      begin
         if (rising_edge(i_clk)) then
            -- read logic
            if(i_we_b ='0') then
               mySRAM(to_integer(unsigned(i_addr))) <= i_data_in;
            end if;
            --registered output
            o_data_out <= mySRAM(to_integer(unsigned(i_addr)));
         end if;
      end process;

end behavioral;
```

8 bits x 4096 words = 32,768 bits

Memories

- SRAM – inferred memory



Flow Status	Successful - Fri May 15 11:2
Quartus Prime Version	19.1.0 Build 670 09/22/201
Revision Name	Class_Examples
Top-level Entity Name	sram_inferred
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	0
Total registers	0
Total pins	30
Total virtual pins	0
Total memory bits	32,768
Embedded Multiplier 9-bit elements	0
Total PLLs	0
UFM blocks	0
ADC blocks	0

Memories

- SRAM – inferred memory