VHDL Memories

Last updated 9/19/23

VHDL Memories

- Four major VHDL memory solutions
  - Mux based
    - Only applicable for ROMs
  - FlipFlop based
    - Very large – only acceptable for very small memories
  - Inferred
    - Memory is implemented in a pre-built memory block
      - Memory block must exist in the platform
      - Tightly coupled memory – small but very fast
      - General memory – large and not as fast
  - External
    - The memory interface is implemented
    - The memory itself is a separate chip

VHDL Memories

N words x M bits/word

N array elements x SLV

- VHDL solution for memories
  - An array of std_logic_vectors
  - Coded just like the non-optimized long array of data words
  - Array construct
    - New type, that has array type as its basis

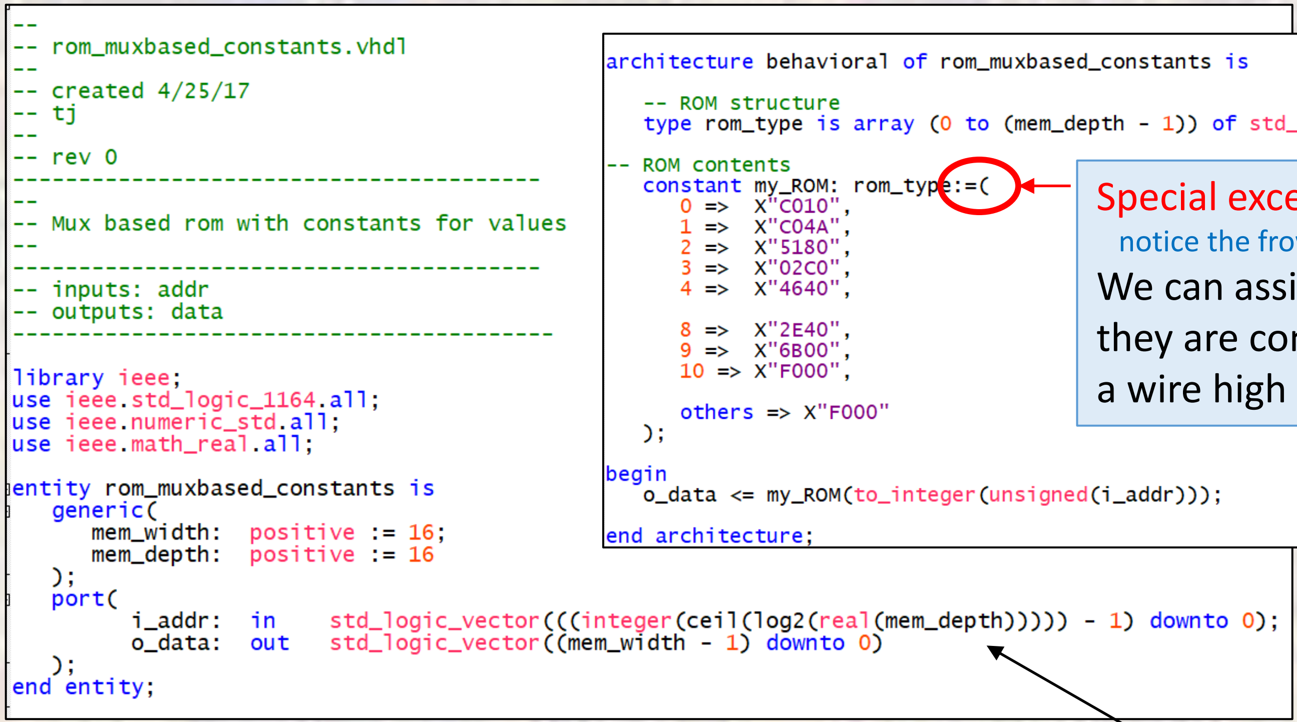
      type my_new_type is array (0 to depth) of some_vhdl_type
  - Memory construct
    - Uses std_logic_vector
      - No understanding of the values (signed/unsigned) is assumed, just bits

```
type my_memory is array (0 to depth) of std_logic_vector((wordwidth - 1) downto 0);
```

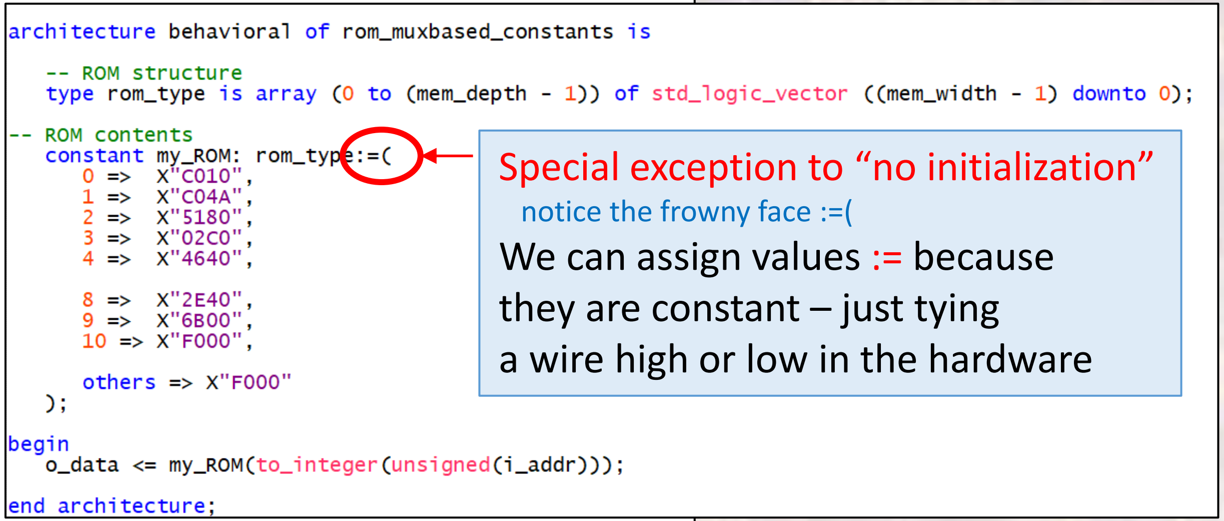
VHDL Memories

- ROM – mux based
  - Read only
  - Memory values stored as constants

16 word, 16b/w (2B/w) ROM



```vhdl
--
-- 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;
```



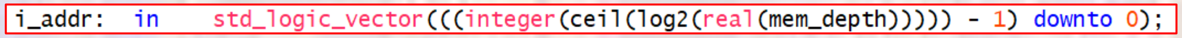
```vhdl
architecture behavioral of rom_muxbased_constants is

   -- 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;
```

Special exception to “no initialization”
notice the frowny face :=(
We can assign values := because they are constant – just tying a wire high or low in the hardware

Calculating the # of address bits based on the mem-depth
 - see next slide

VHDL Memories

- ROM – mux based
  - Address bit calculation



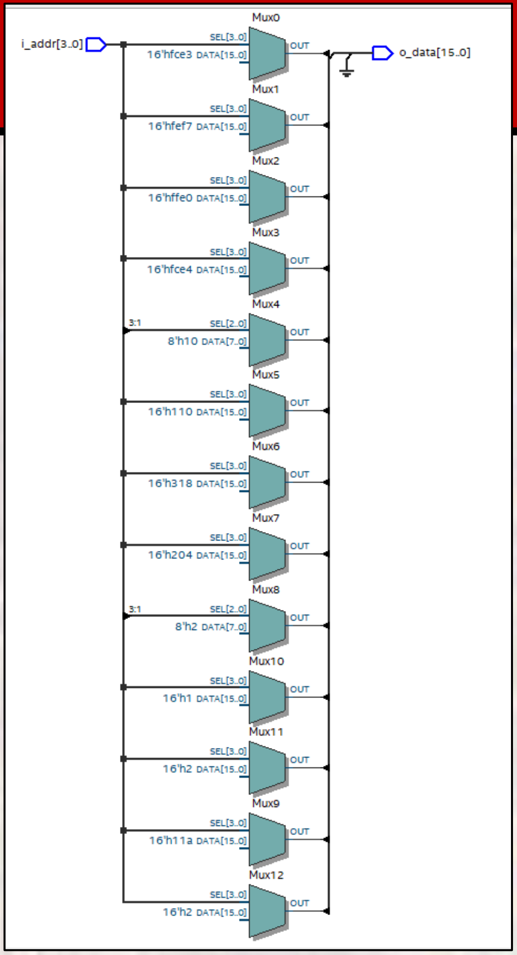
```
i_addr:   in     std_logic_vector(((integer(ceil(log2(real(mem_depth)))) - 1) downto 0);
```

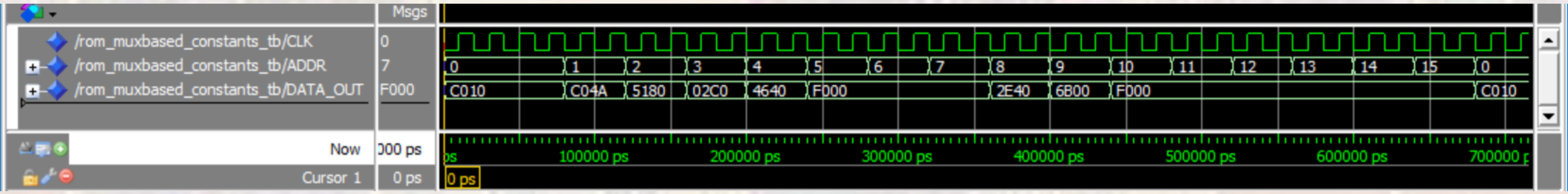
mem_depth	only makes sense to be an integer
real(mem_depth)	turns it into a real number (not an int)
log2(real(mem_depth))	calculates the log base 2 requires a real input provides a real output
ceil(log2(real(mem_depth)))	rounds up (next largest whole real number) provides support for non-2^N sizes 24 → 4.585 → 5.0
integer(ceil(log2(real(mem_depth))))	converts the real to an integer must be integer to use as index

VHDL Memories

- ROM – mux based
  - Memory values stored as constants

Modelled as 1 mux per bit in the word




Msgs
/rom_muxbased_constants_tb/CLK
/rom_muxbased_constants_tb/ADDR
/rom_muxbased_constants_tb/DATA_OUT
0
7
F000
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0
C010 C04A 5180 02C0 4640 F000 2E40 6B00 F000 C010
Now
000 ps
Cursor 1
0 ps
100000 ps
200000 ps
300000 ps
400000 ps
500000 ps
600000 ps
700000 ps

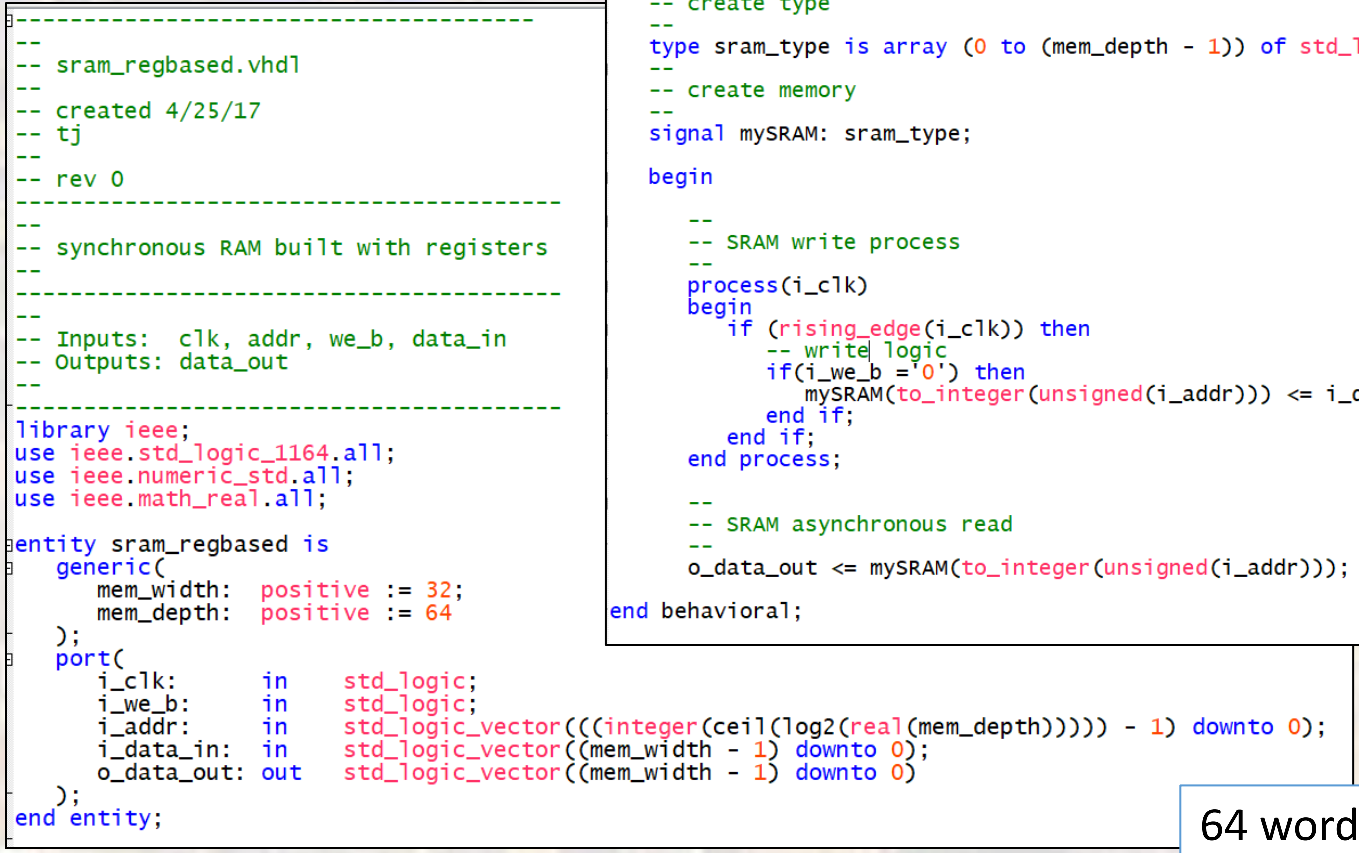
VHDL Memories

- SRAM – flipflop based
  - Using flipflops as our memory storage element
  - The inferred memories on our FPGA all require synchronous read paths
    - To force a flipflop based memory the read path must be asynchronous
  - Since we want flipflops, we must have some clock controlling the memory
    - Make the write path synchronous
    - Outputs of the flipflops are always available

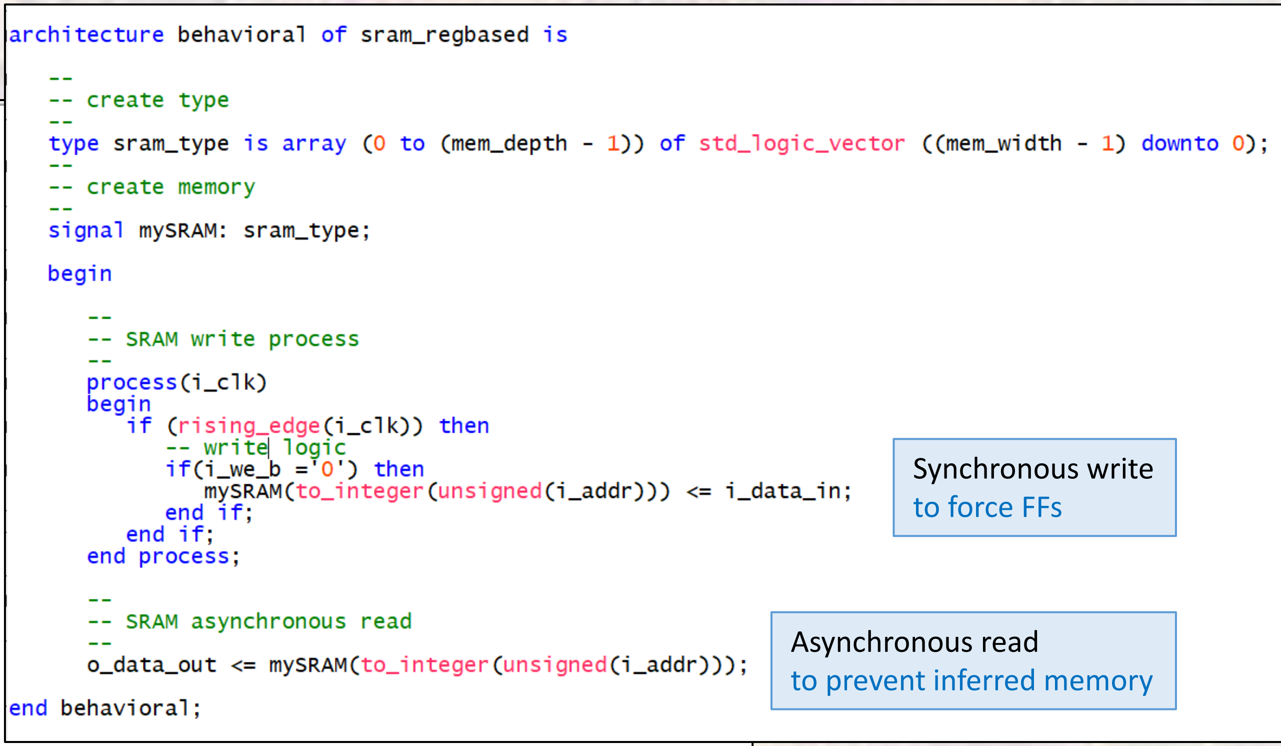
VHDL Memories

- SRAM – flipflop based
  - 64 x 32b

64 word, 32b/w (4B/w) FF based SRAM



```vhdl
-----------------------------------------
--
-- 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;
```



```vhdl
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;
```

Synchronous write
to force FFs

Asynchronous read
to prevent inferred memory

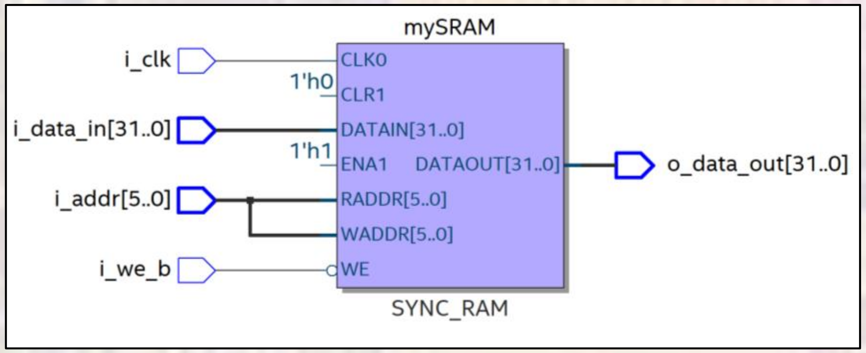
64 words X 32 bits
64 words X 4 Bytes
64x4x8 bits = 2048 flipflops

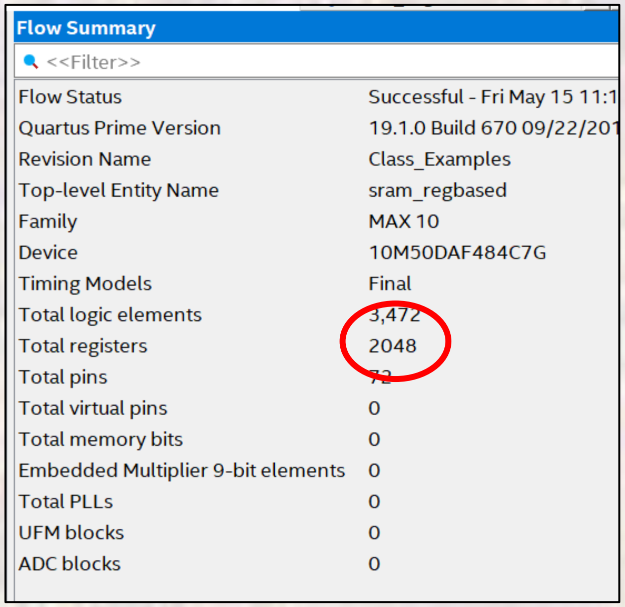
VHDL Memories

- SRAM – register based
  - 64x32b

The implementation is in flipflops

There is an RTL model for this memory




Flow Summary

<<Filter>>

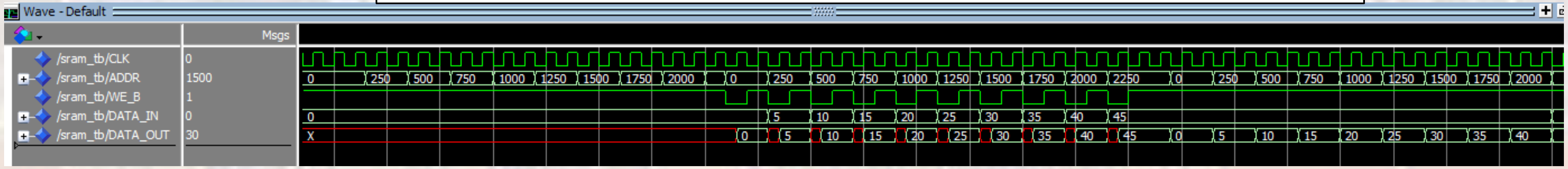
Flow Status	Successful - Fri May 15 11:1
Quartus Prime Version	19.1.0 Build 670 09/22/201
Revision Name	Class_Examples
Top-level Entity Name	sram_regbased
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	3,472
Total registers	2048
Total pins	72
Total virtual pins	0
Total memory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

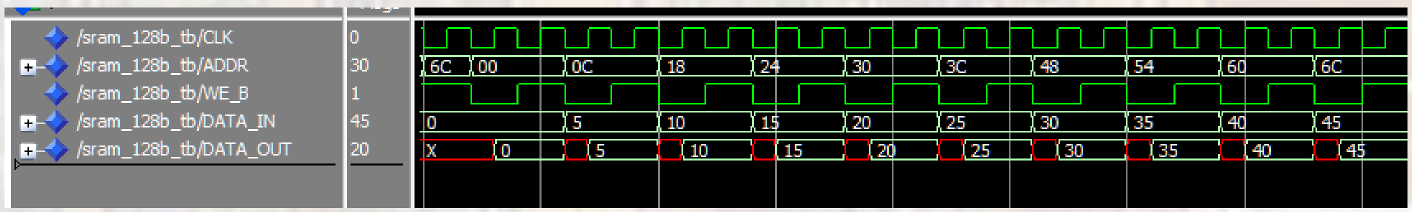
VHDL Memories

- SRAM – register based – test bench
  - 64x32b

```
-- 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;
```

Not all addresses tested
Not all bit values tested





VHDL Memories

- Memory Test Benches
  - A proper memory testbench would test:
    - All addresses
    - All bits 0 and 1
    - Read ROMs, R/W for RAMs
    - Write_enable_bar functionality