VHDL Memories Inferred

Last updated 2/19/25

VHDL Memories – Inferred

- 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 – Inferred

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

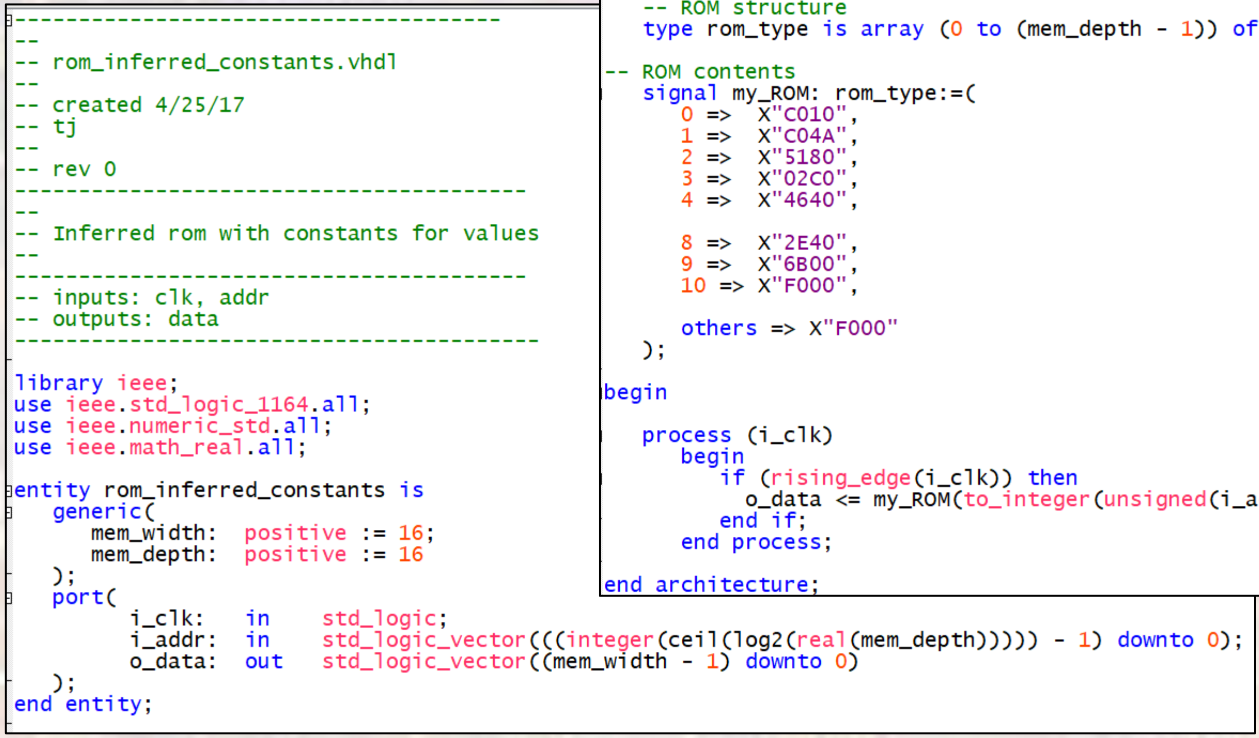
```
type my_memory is array (0 to depth) of std_logic_vector((wordwidth - 1) downto 0);
```

VHDL Memories – Inferred

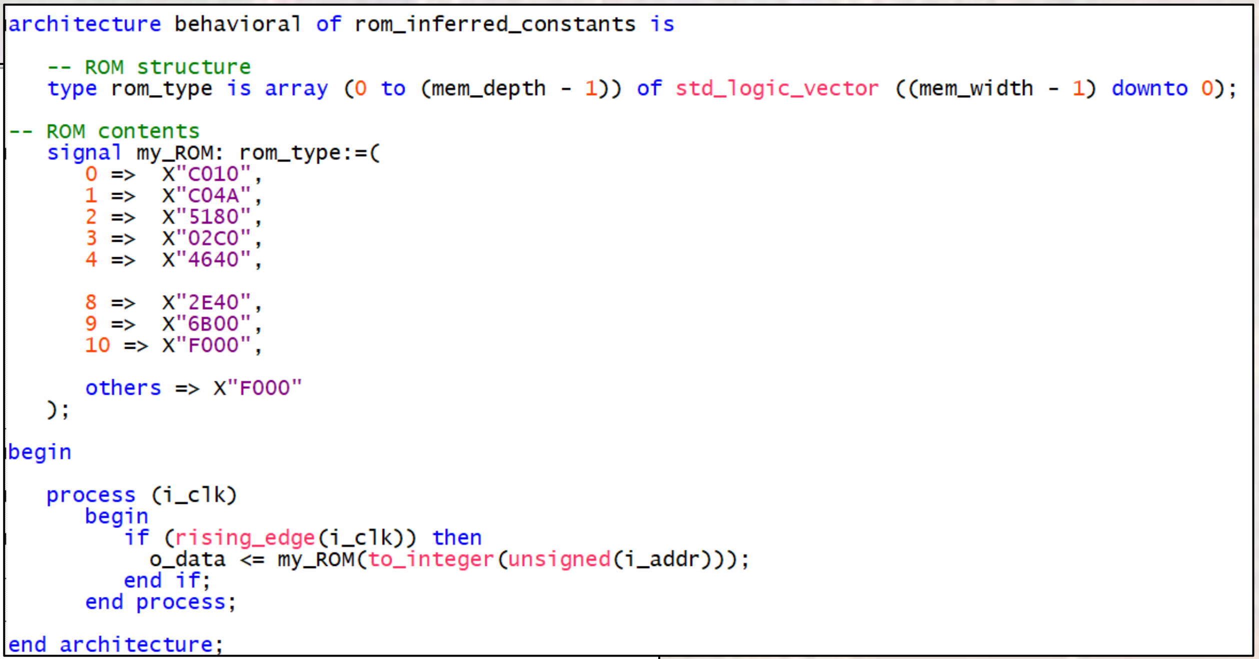
- ROM – Inferred
  - Using SRAM integrated on the FPGA as our memory storage element – configured with no write path
  - The inferred memories on our FPGA all require synchronous read paths
    - To force an inferred memory the read path must be synchronous

VHDL Memories – Inferred

- ROM - Inferred
  - Fixed values in the Memory signal
  - Synchronous read path
  - No write path



```
----------------------------------------
--
-- rom_inferred_constants.vhdl
--
-- created 4/25/17
-- tj
--
-- rev 0
------------------------------------------
--
-- Inferred rom with constants for values
--
------------------------------------------
-- inputs: clk, addr
-- outputs: data
-------------------------------------------

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

entity rom_inferred_constants is
   generic(
      mem_width:  positive := 16;
      mem_depth:  positive := 16
   );
   port(
        i_clk:   in    std_logic;
        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_inferred_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
   signal 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

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

end architecture;
```

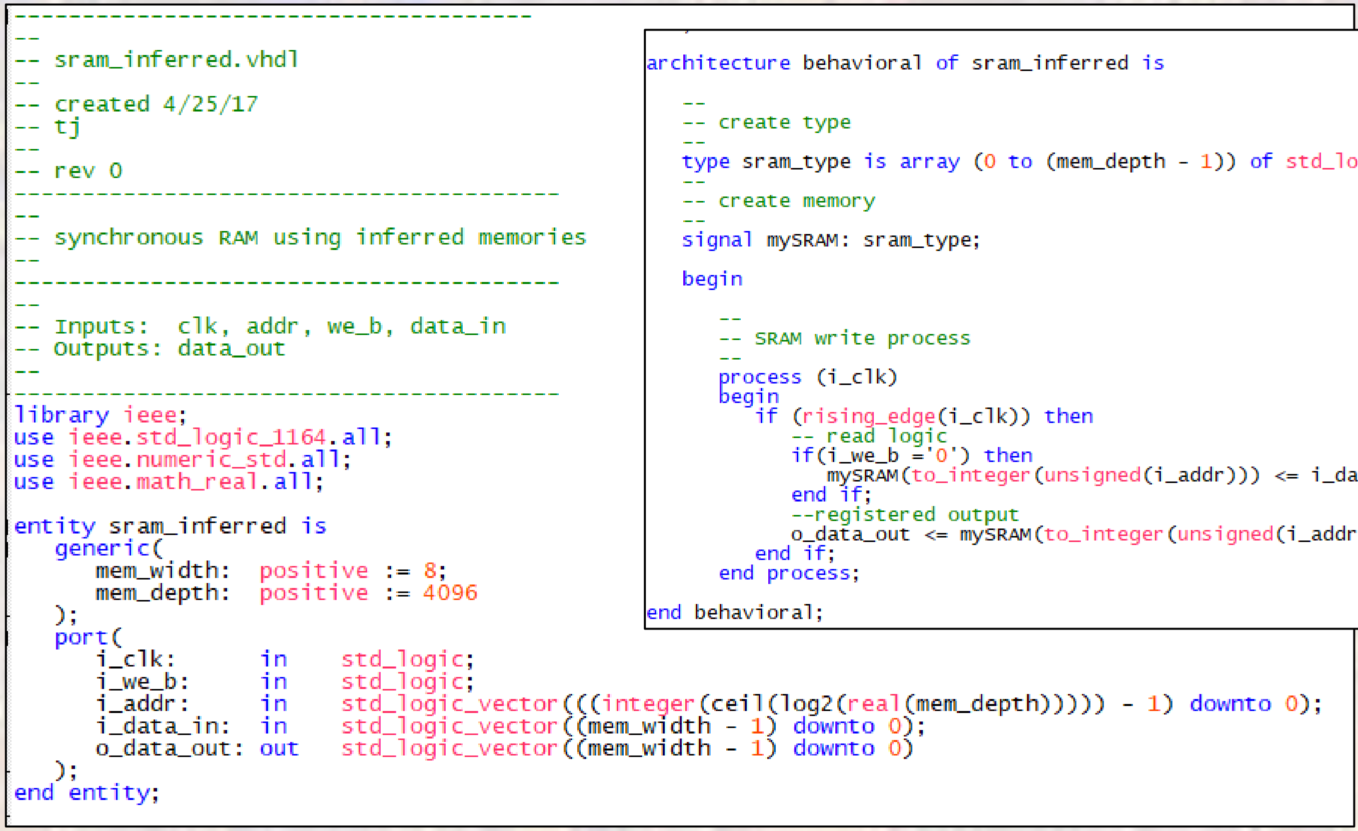
VHDL Memories – Inferred

- SRAM – Inferred
  - Using SRAM integrated on the FPGA as our memory storage element
  - The inferred memories on our FPGA all require synchronous read paths
    - To force an inferred memory the read path must be synchronous

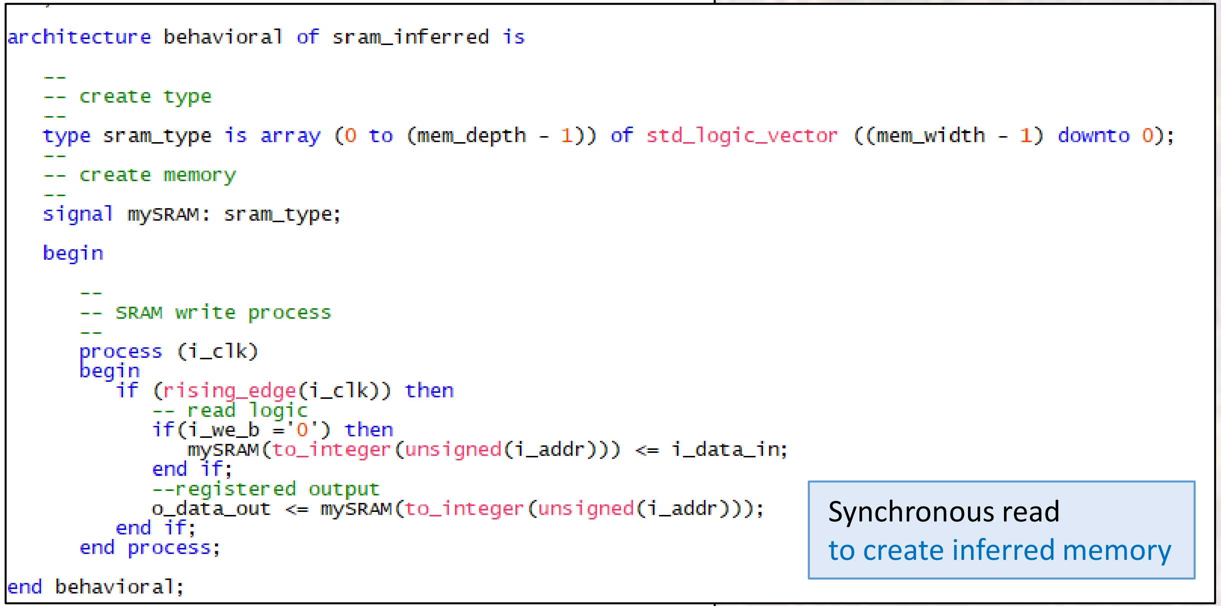
VHDL Memories – Inferred

4K word, 1B/w inferred SRAM

- SRAM – inferred
  - 4K x 8



```
-----------------------------------------
--
-- 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 entity;
```



```
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;
```

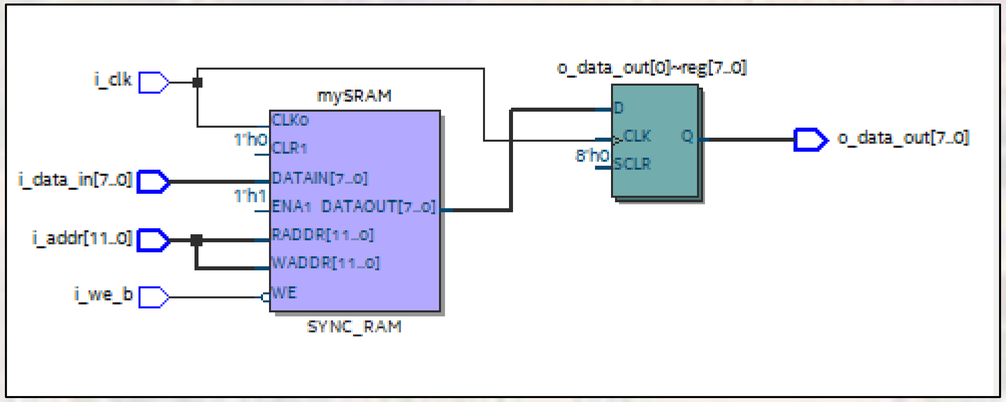
Synchronous read
to create inferred memory

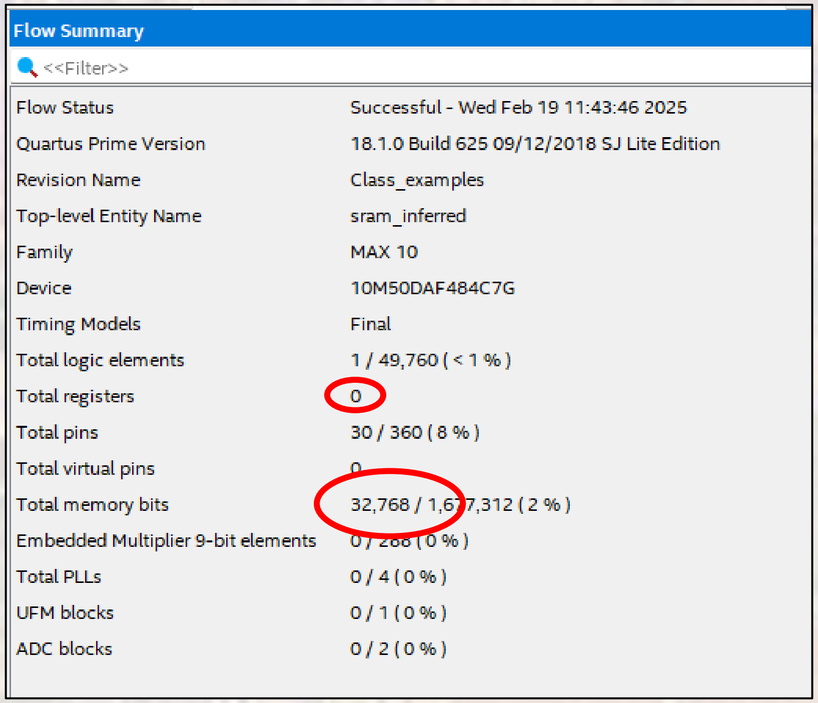
VHDL Memories – Inferred

- SRAM – inferred
  - 4K x 8

The implementation used inferred SRAM on the FPGA

There is an RTL model for this memory





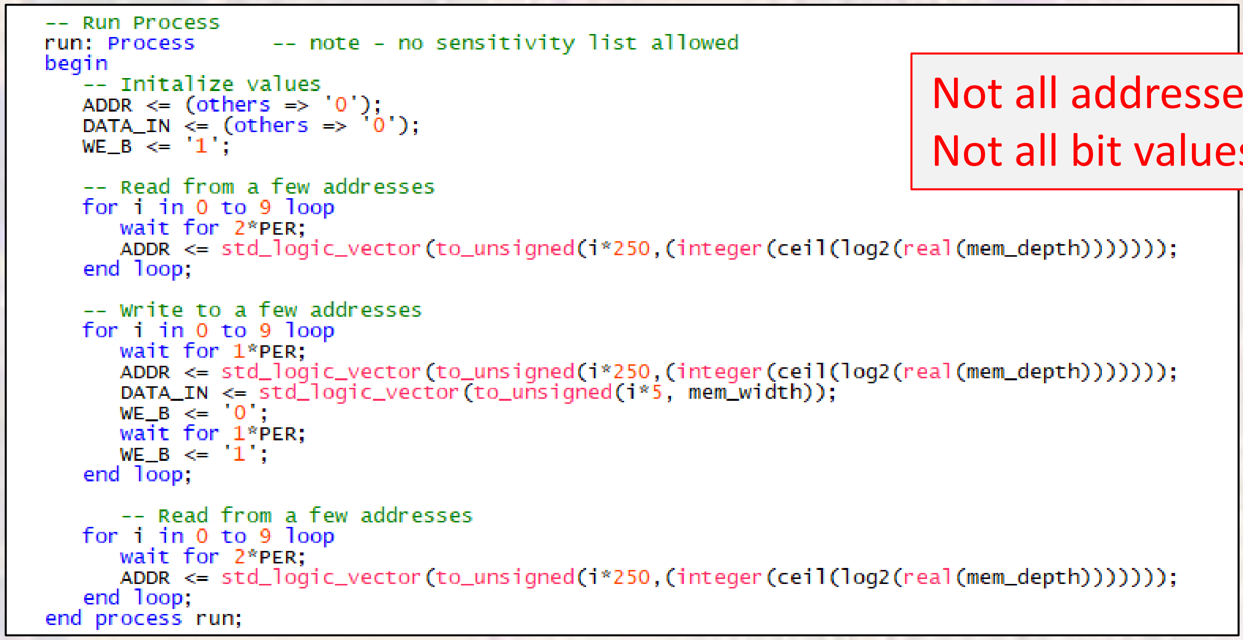
Flow Summary

<<Filter>>

Flow Status	Successful - Wed Feb 19 11:43:46 2025
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	Class_examples
Top-level Entity Name	sram_inferred
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Final
Total logic elements	1 / 49,760 (< 1 %)
Total registers	0
Total pins	30 / 360 (8 %)
Total virtual pins	0
Total memory bits	32,768 / 1,677,312 (2 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)
UFM blocks	0 / 1 (0 %)
ADC blocks	0 / 2 (0 %)

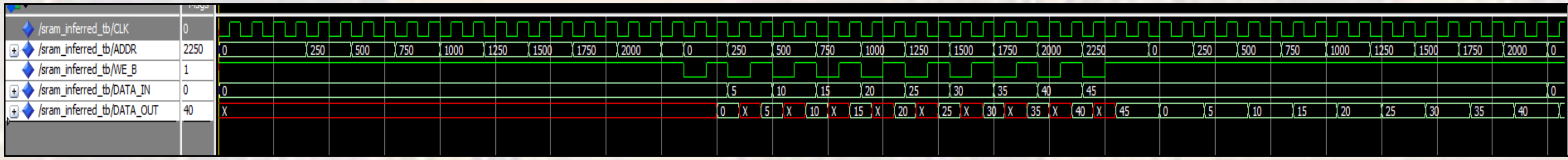
VHDL Memories – Inferred

- SRAM – Inferred – test bench
  - 4K x 8

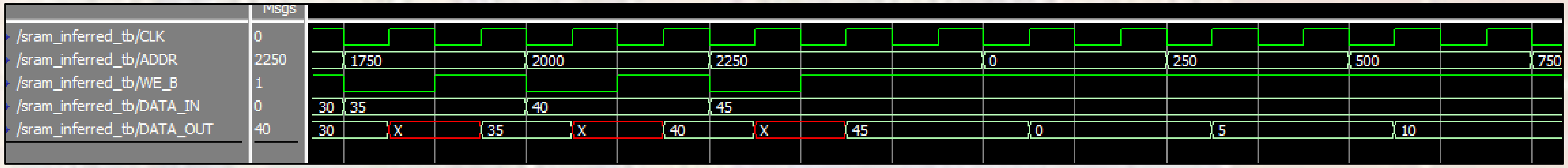


```
-- Run Process
run: Process        -- note - no sensitivity list allowed
begin
   -- Initalize values
   ADDR <= (others => '0');
   DATA_IN <= (others => '0');
   WE_B <= '1';

   -- 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



/sram_inferred_tb/CLK 0
/sram_inferred_tb/ADDR 2250
/sram_inferred_tb/WE_B 1
/sram_inferred_tb/DATA_IN 0
/sram_inferred_tb/DATA_OUT 40



Msgs
/sram_inferred_tb/CLK 0
/sram_inferred_tb/ADDR 2250
/sram_inferred_tb/WE_B 1
/sram_inferred_tb/DATA_IN 0
/sram_inferred_tb/DATA_OUT 40

VHDL Memories – Inferred

- 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