Register Files - HDL

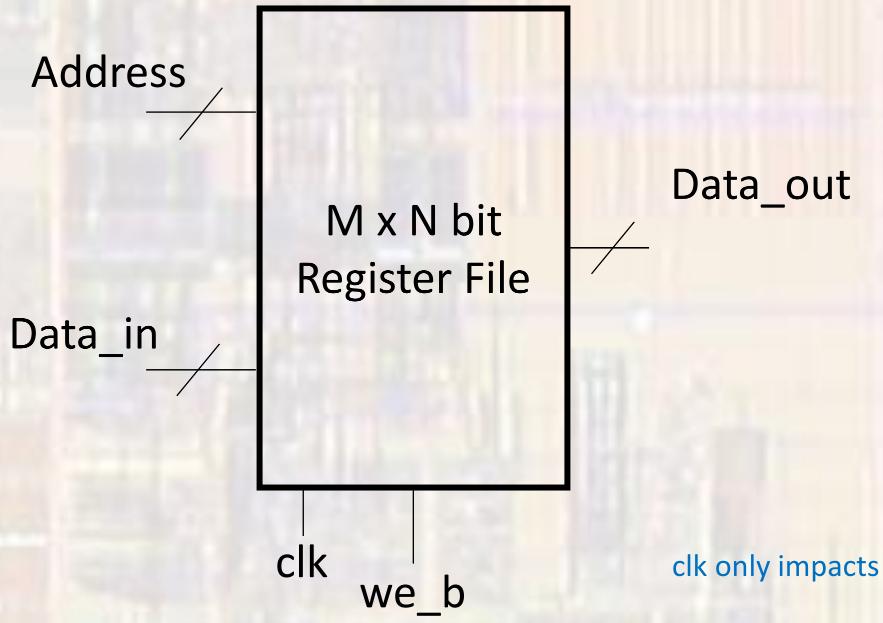
Last updated 2/22/21

Register Files - HDL

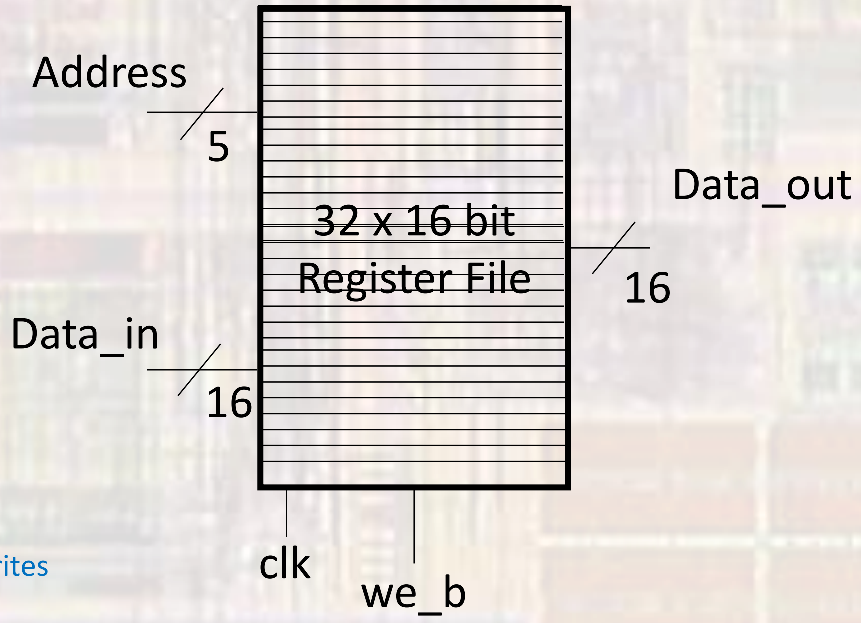
- These slides introduce VHDL based Register Files
- Upon completion: You should be able design and use Register Files

Register Files - HDL

- Register File – simple
  - Group of n-bit registers
  - Accessed via a multiplexor (address)



clk only impacts writes



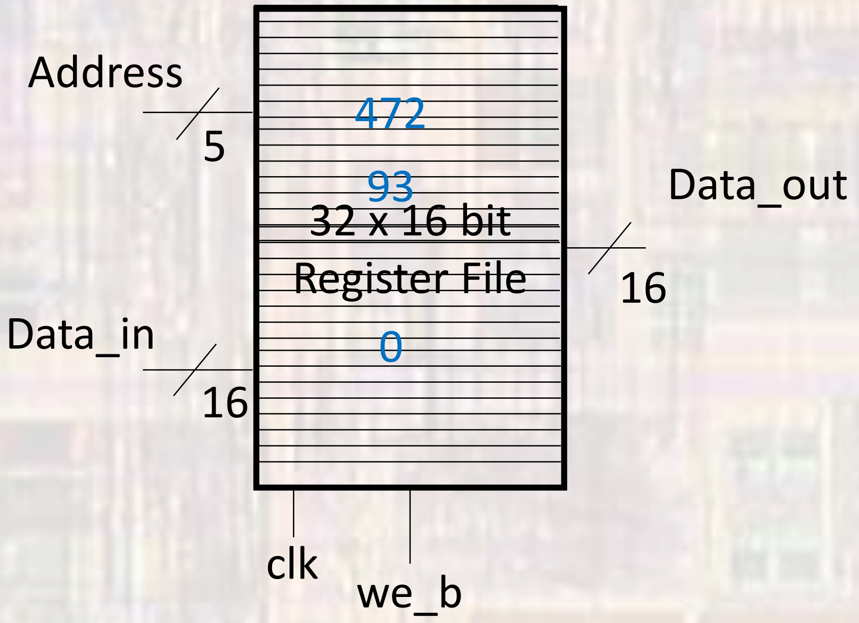
Register Files - HDL

- Register File – simple
  - Output data (from reg at Address) – always available
  - Write requires we_b low and clk ↑

	we_b	addr	data_in	data_out
	1	7	472	?
	0	7	472	?
clk ↑ →				
	1	7	472	472
	1	7	93	472
	1	12	93	?
	0	12	93	93
clk ↑ →				
	1	12	93	?
	1	23	0	?
	0	23	0	0
clk ↑ →				
	1	23	0	472
	1	7	55	93
	1	12	55	

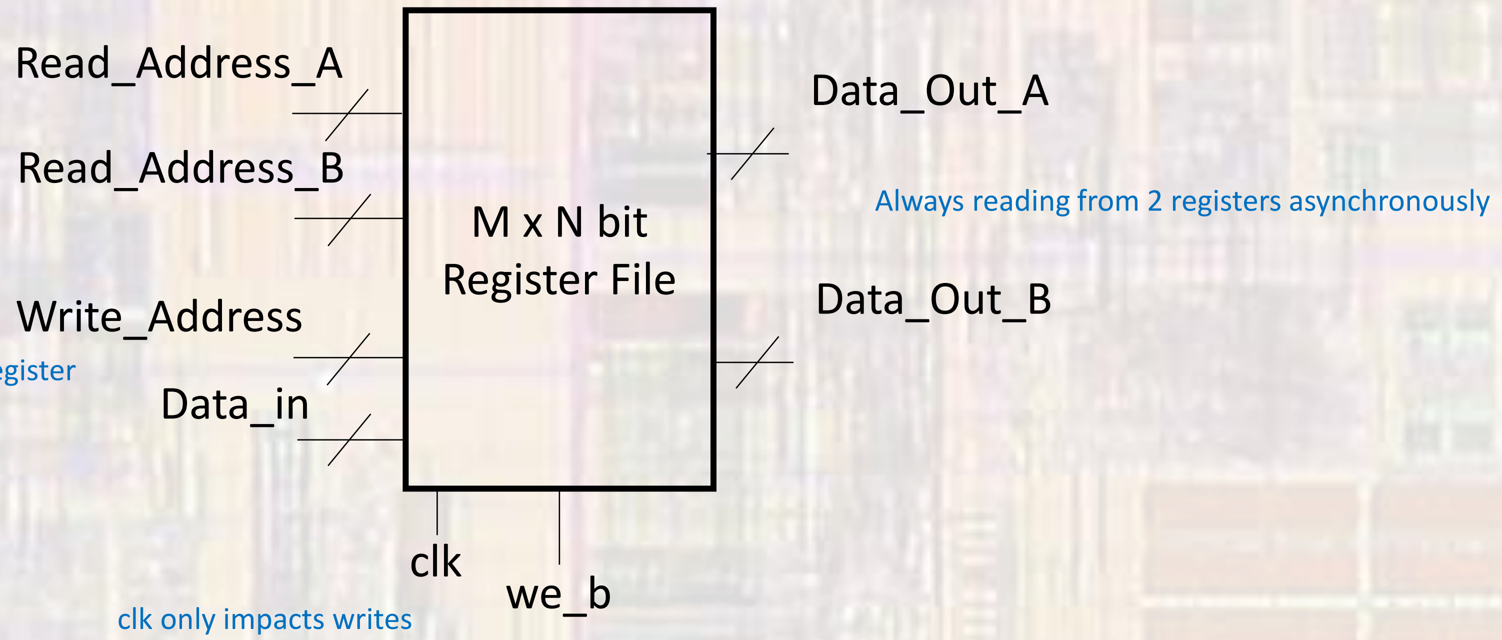
clk only required for writes, reads are asynchronous

In normal application, clock would always be running



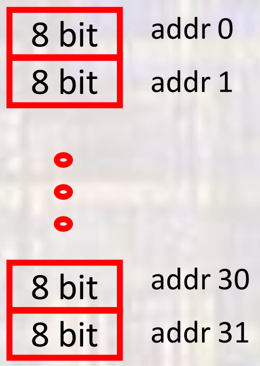
Register Files - HDL

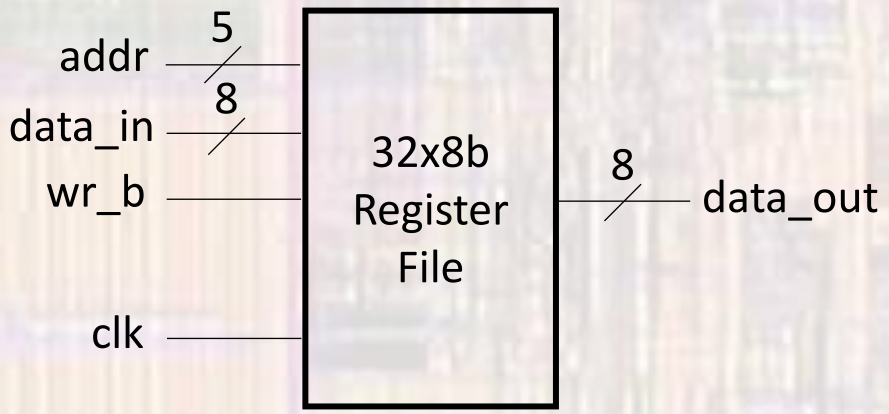
- Register File – multi-port
  - Group of n-bit registers
  - Accessed via a multiplexor
  - Can have multiple inputs and outputs



Register Files - HDL

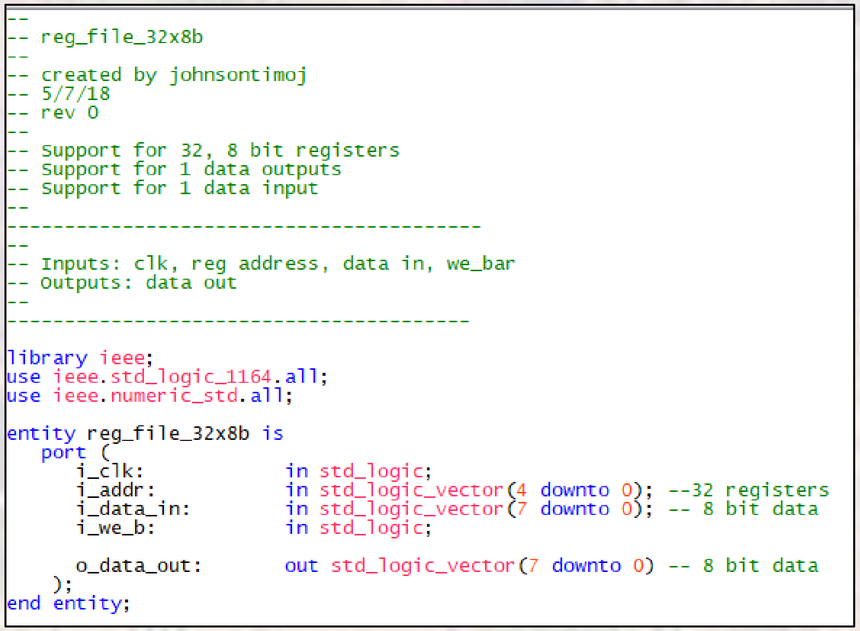
- 32 x 8 bit register file





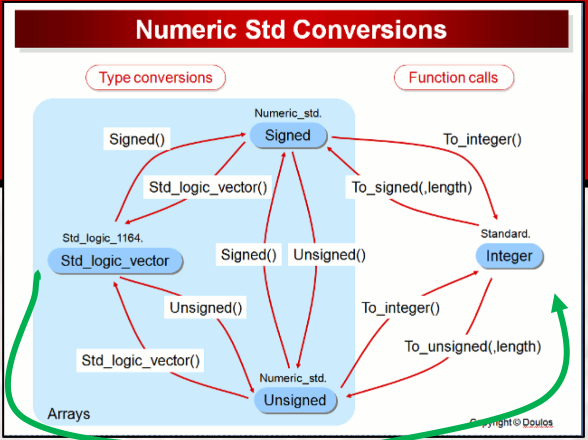
Register Files - HDL

- 32 x 8 bit register file



```
--
-- reg_file_32x8b
--
-- created by johnsontimoj
-- 5/7/18
-- rev 0
--
-- Support for 32, 8 bit registers
-- Support for 1 data outputs
-- Support for 1 data input
--
-----------------------------------------
--
-- Inputs: clk, reg address, data in, we_bar
-- Outputs: data out
--
-----------------------------------------

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

entity reg_file_32x8b is
   port (
      i_clk:          in std_logic;
      i_addr:         in std_logic_vector(4 downto 0); --32 registers
      i_data_in:      in std_logic_vector(7 downto 0); -- 8 bit data
      i_we_b:         in std_logic;

      o_data_out:     out std_logic_vector(7 downto 0) -- 8 bit data
    );
end entity;
```

Register Files - HDL



- 32 x 8 bit register file

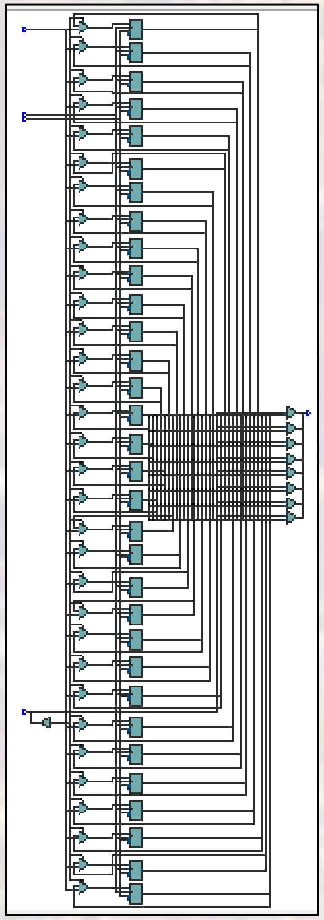
Create a new 'type'
The new 'type' is an array of std_logic_vector

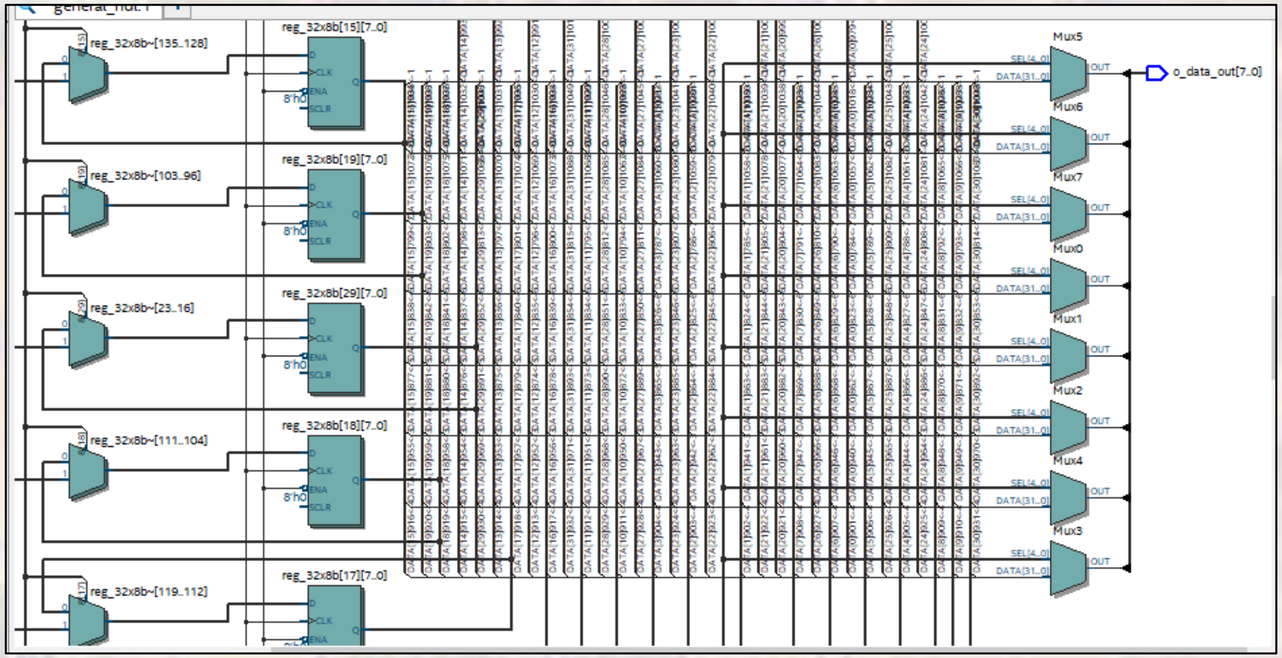
```
architecture behavioral of reg_file_32x8b is
    --
    -- Create an "TYPE" for an array of 8 bit registers
    --
    type REG_FILE is array(0 to 31) of std_logic_vector(7 downto 0);
    --
    -- Create a signal of the array type
    --
    signal reg_32x8b: REG_FILE;
    --
    -- Special command to stop Quartus from inferring a RAM
    -- Must also turn off assignments / settings / compiler settings/
    -- adavanced settings (synthesis) / auto RAM replacement --> off
    attribute ramstyle : string;
    attribute ramstyle of reg_32x8b : signal is "logic";

begin
    --
    -- register logic
    --
    process(i_clk)
    begin
        if(rising_edge(i_clk)) then
            if(i_we_b = '0') then
                reg_32x8b(to_integer(unsigned(i_addr))) <= i_data_in;
            end if;
        end if;
    end process;

    --
    -- output logic
    --
    process(all)
    begin
        o_data_out <= reg_32x8b(to_integer(unsigned(i_addr)));
    end process;

end behavioral;
```

Need the address as an integer to index the array.
Address is SLV coming in – need to pass through unsigned along the way

Register Files - HDL

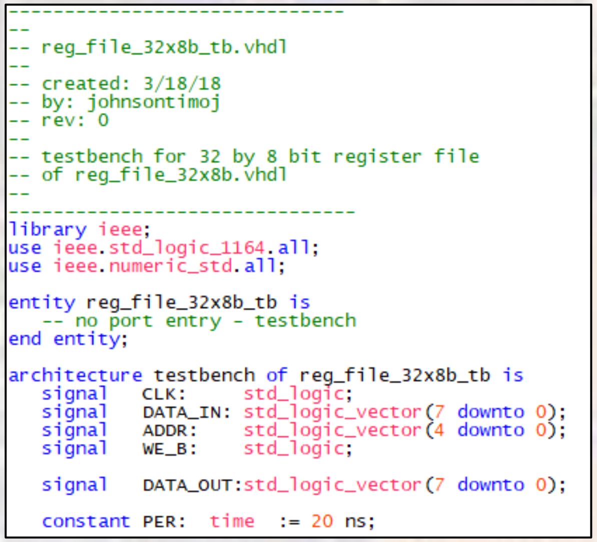
- 32 x 8 bit register file



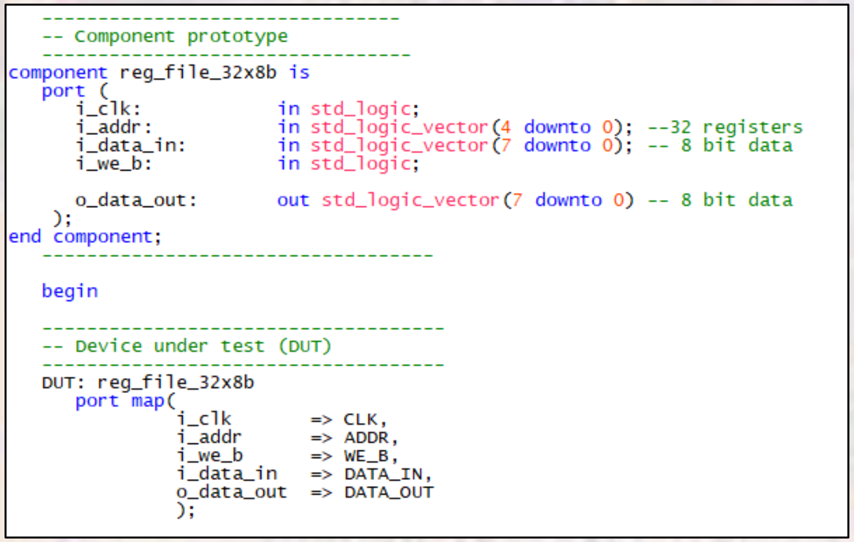


Register Files - HDL

- 32 x 8 bit register file - testbench



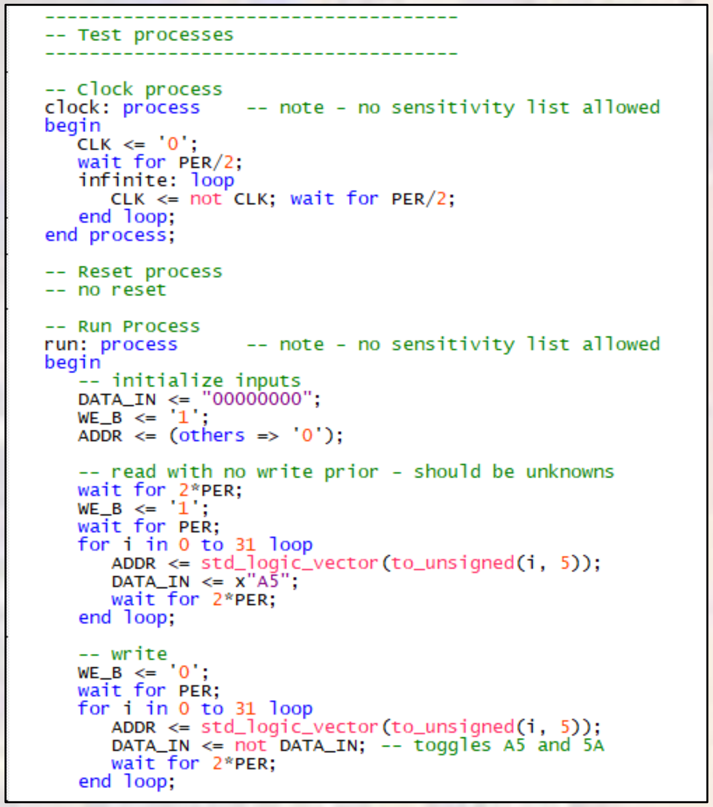
```
------------------------------
--
-- reg_file_32x8b_tb.vhdl
--
-- created: 3/18/18
-- by: johnsontimoj
-- rev: 0
--
-- testbench for 32 by 8 bit register file
-- of reg_file_32x8b.vhdl
--
-------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity reg_file_32x8b_tb is
    -- no port entry - testbench
end entity;

architecture testbench of reg_file_32x8b_tb is
   signal   CLK:     std_logic;
   signal   DATA_IN: std_logic_vector(7 downto 0);
   signal   ADDR:    std_logic_vector(4 downto 0);
   signal   WE_B:    std_logic;

   signal   DATA_OUT:std_logic_vector(7 downto 0);

   constant PER:  time  := 20 ns;
```



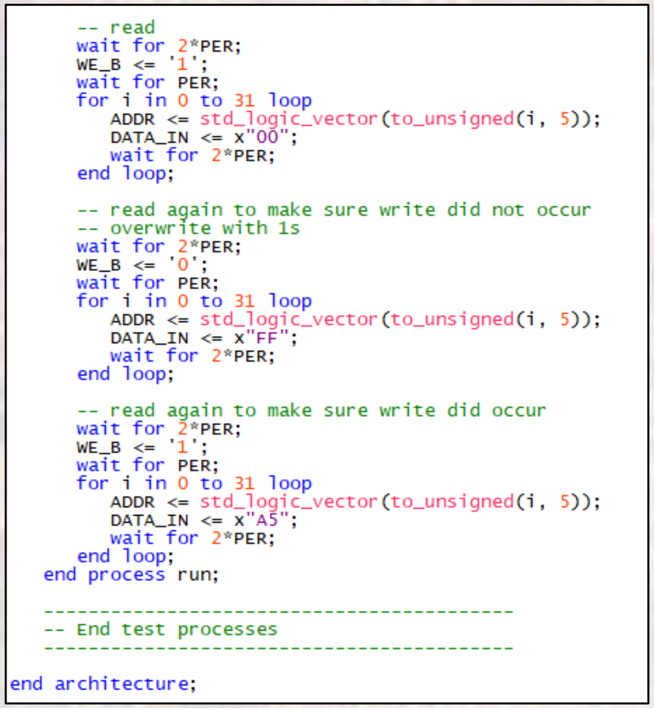
```
   --------------------------------
   -- Component prototype
   ---------------------------------
component reg_file_32x8b is
   port (
      i_clk:            in std_logic;
      i_addr:           in std_logic_vector(4 downto 0); --32 registers
      i_data_in:        in std_logic_vector(7 downto 0); -- 8 bit data
      i_we_b:           in std_logic;

      o_data_out:       out std_logic_vector(7 downto 0) -- 8 bit data
    );
end component;
   -----------------------------------

   begin

   ------------------------------------
   -- Device under test (DUT)
   ------------------------------------
   DUT: reg_file_32x8b
      port map(
               i_clk       => CLK,
               i_addr      => ADDR,
               i_we_b      => WE_B,
               i_data_in   => DATA_IN,
               o_data_out  => DATA_OUT
               );
```

Register Files - HDL

- 32 x 8 bit register file - testbench



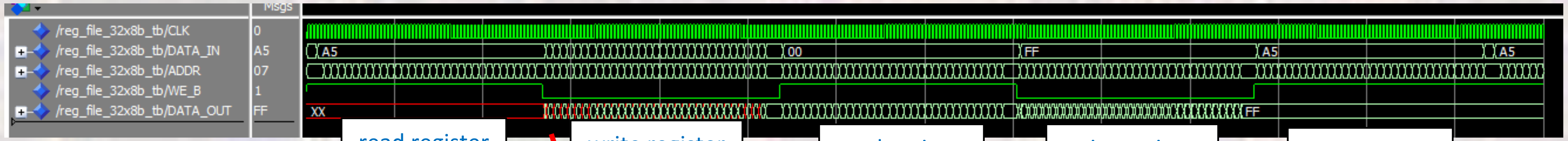
```
   -------------------------------------
   -- Test processes
   -------------------------------------

   -- Clock process
   clock: process    -- note - no sensitivity list allowed
   begin
      CLK <= '0';
      wait for PER/2;
      infinite: loop
         CLK <= not CLK; wait for PER/2;
      end loop;
   end process;

   -- Reset process
   -- no reset

   -- Run Process
   run: process      -- note - no sensitivity list allowed
   begin
      -- initialize inputs
      DATA_IN <= "00000000";
      WE_B <= '1';
      ADDR <= (others => '0');

      -- read with no write prior - should be unknowns
      wait for 2*PER;
      WE_B <= '1';
      wait for PER;
      for i in 0 to 31 loop
         ADDR <= std_logic_vector(to_unsigned(i, 5));
         DATA_IN <= x"A5";
         wait for 2*PER;
      end loop;

      -- write
      WE_B <= '0';
      wait for PER;
      for i in 0 to 31 loop
         ADDR <= std_logic_vector(to_unsigned(i, 5));
         DATA_IN <= not DATA_IN; -- toggles A5 and 5A
         wait for 2*PER;
      end loop;
```



```
      -- read
      wait for 2*PER;
      WE_B <= '1';
      wait for PER;
      for i in 0 to 31 loop
         ADDR <= std_logic_vector(to_unsigned(i, 5));
         DATA_IN <= x"00";
         wait for 2*PER;
      end loop;

      -- read again to make sure write did not occur
      -- overwrite with 1s
      wait for 2*PER;
      WE_B <= '0';
      wait for PER;
      for i in 0 to 31 loop
         ADDR <= std_logic_vector(to_unsigned(i, 5));
         DATA_IN <= x"FF";
         wait for 2*PER;
      end loop;

      -- read again to make sure write did occur
      wait for 2*PER;
      WE_B <= '1';
      wait for PER;
      for i in 0 to 31 loop
         ADDR <= std_logic_vector(to_unsigned(i, 5));
         DATA_IN <= x"A5";
         wait for 2*PER;
      end loop;
   end process run;

   ------------------------------------------
   -- End test processes
   ------------------------------------------

end architecture;
```

Register Files - HDL

- 32 x 8 bit register file - verification



Msgs
/reg_file_32x8b_tb/CLK 0
/reg_file_32x8b_tb/DATA_IN A5
/reg_file_32x8b_tb/ADDR 07
/reg_file_32x8b_tb/WE_B 1
/reg_file_32x8b_tb/DATA_OUT FF
A5
00
FF
A5
A5
XX
FF

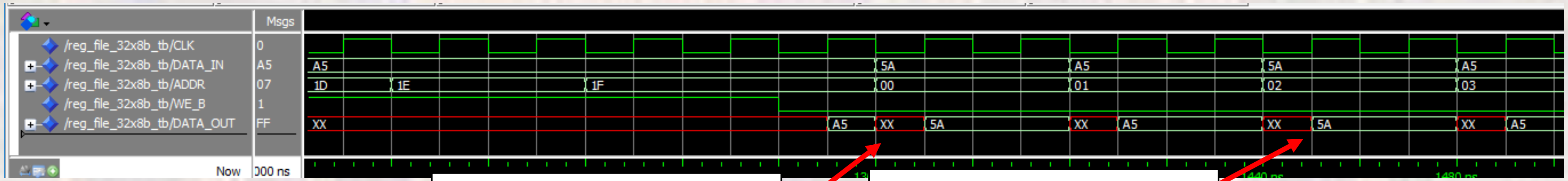
read register unknown

write register pattern

read register no write

write register data_in = FF

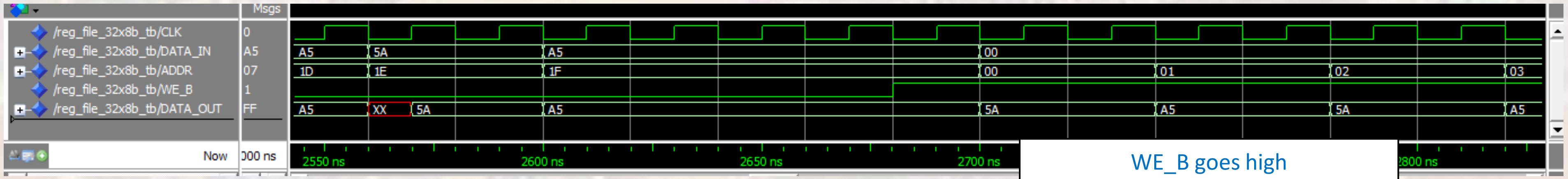
read register no write



Msgs
/reg_file_32x8b_tb/CLK 0
/reg_file_32x8b_tb/DATA_IN A5
/reg_file_32x8b_tb/ADDR 07
/reg_file_32x8b_tb/WE_B 1
/reg_file_32x8b_tb/DATA_OUT FF
Now
A5
5A
A5
5A
A5
1D
1E
1F
00
01
02
03
XX
A5
XX
5A
XX
A5
XX
5A
XX
A5

Addr 0
Initially unknown
Loads input on rising clock edge

Addr 2
Initially unknown
Loads input on rising clock edge

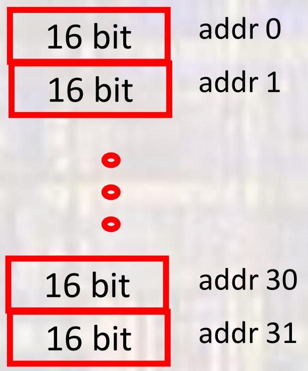


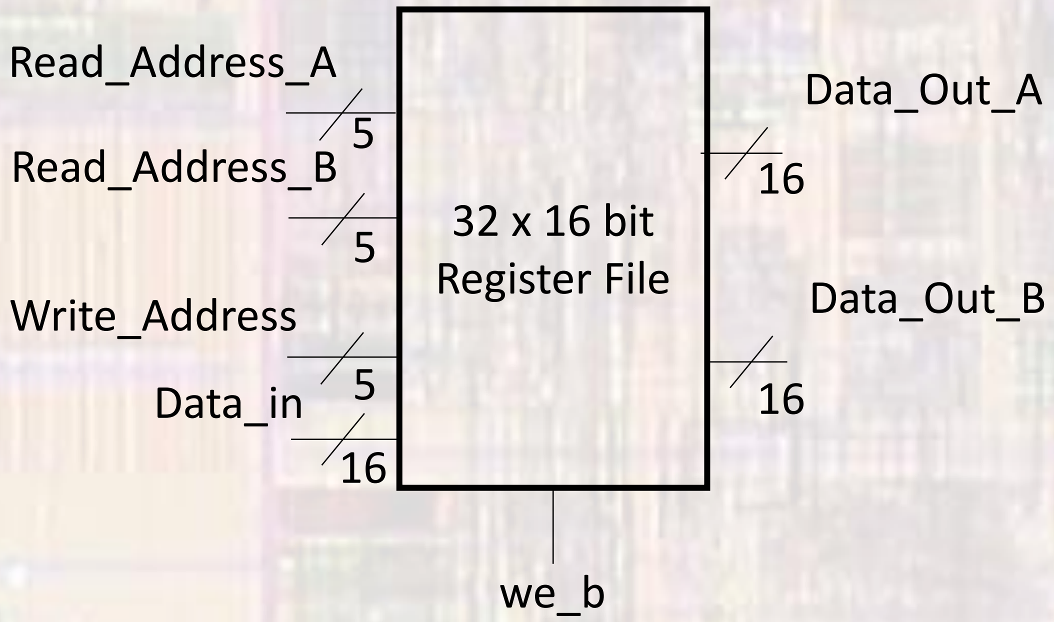
Msgs
/reg_file_32x8b_tb/CLK 0
/reg_file_32x8b_tb/DATA_IN A5
/reg_file_32x8b_tb/ADDR 07
/reg_file_32x8b_tb/WE_B 1
/reg_file_32x8b_tb/DATA_OUT FF
Now
A5
5A
A5
00
1D
1E
1F
00
01
02
03
A5
XX
5A
A5
5A
A5
5A
A5
2550 ns
2600 ns
2650 ns
2700 ns
2800 ns

WE_B goes high
Still reading the old data – no write

Register Files - HDL

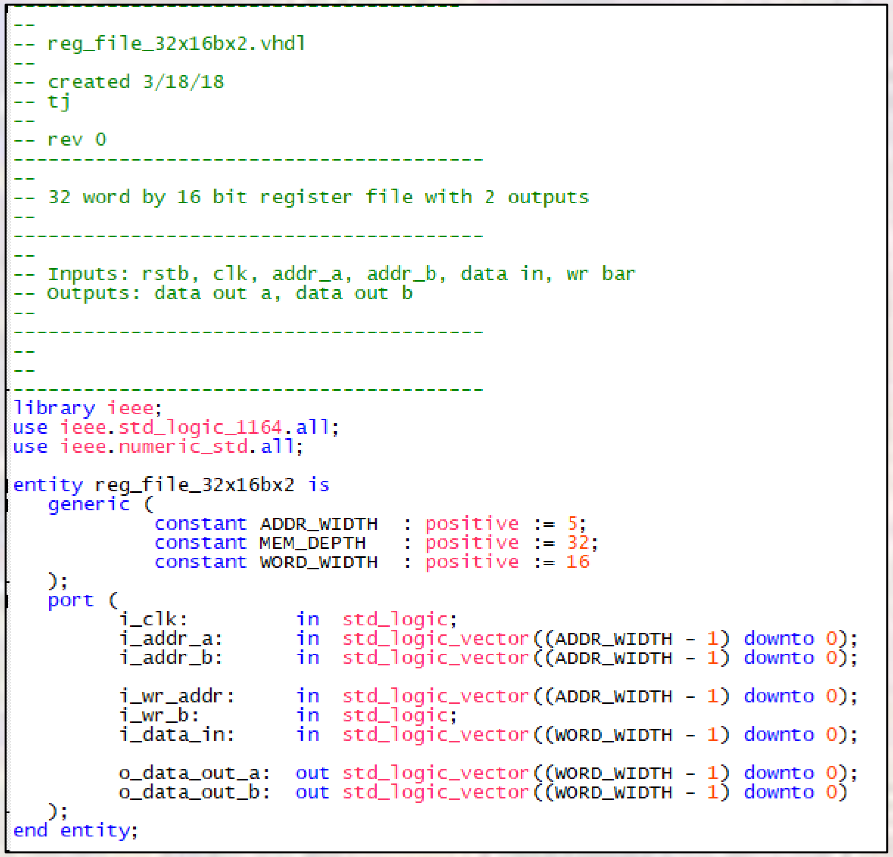
- 32 x 16 bit register file , 2 outputs





Register Files - HDL

- 32 x 16 bit register file, 2 outputs

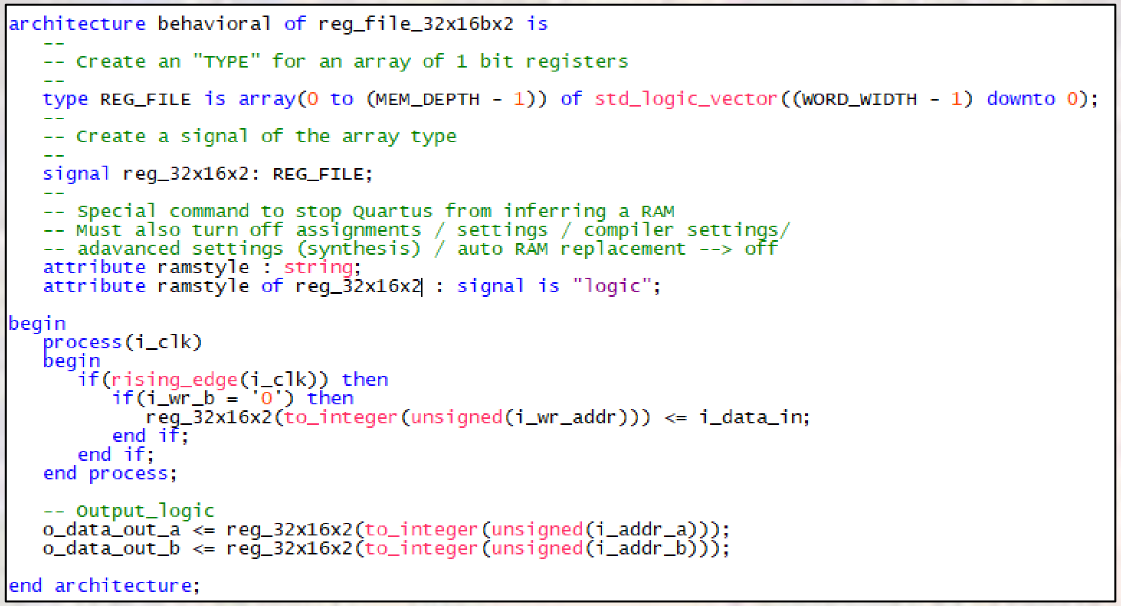


```
----------------------------------------
--
-- reg_file_32x16bx2.vhdl
--
-- created 3/18/18
-- tj
--
-- rev 0
----------------------------------------
--
-- 32 word by 16 bit register file with 2 outputs
--
----------------------------------------
--
-- Inputs: rstb, clk, addr_a, addr_b, data in, wr bar
-- Outputs: data out a, data out b
--
----------------------------------------
--
--
----------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity reg_file_32x16bx2 is
   generic (
            constant ADDR_WIDTH  : positive := 5;
            constant MEM_DEPTH   : positive := 32;
            constant WORD_WIDTH  : positive := 16
   );
   port (
         i_clk:         in  std_logic;
         i_addr_a:      in  std_logic_vector((ADDR_WIDTH - 1) downto 0);
         i_addr_b:      in  std_logic_vector((ADDR_WIDTH - 1) downto 0);

         i_wr_addr:     in  std_logic_vector((ADDR_WIDTH - 1) downto 0);
         i_wr_b:        in  std_logic;
         i_data_in:     in  std_logic_vector((WORD_WIDTH - 1) downto 0);

         o_data_out_a:  out std_logic_vector((WORD_WIDTH - 1) downto 0);
         o_data_out_b:  out std_logic_vector((WORD_WIDTH - 1) downto 0)
   );
end entity;
```

Register Files - HDL

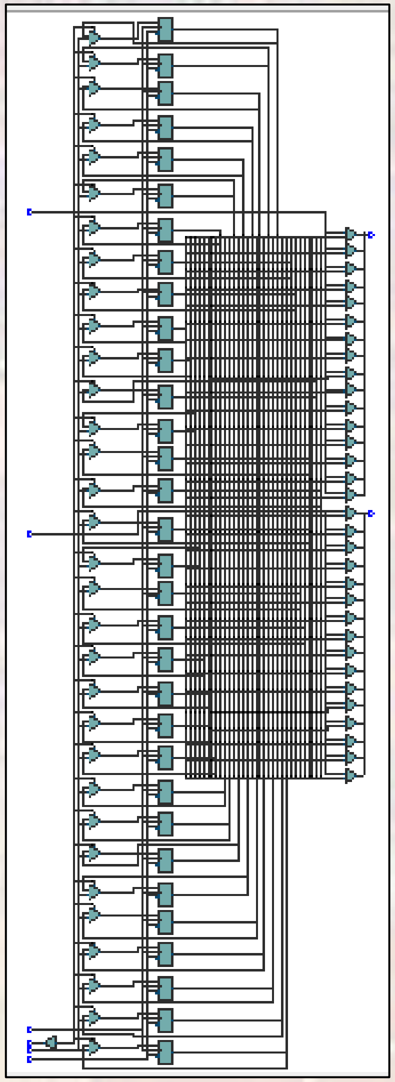
- 32 x 16 bit register file, 2 outputs

Create a new 'type'
The new 'type' is an array of std_logic_vector



```
architecture behavioral of reg_file_32x16bx2 is
   --
   -- Create an "TYPE" for an array of 1 bit registers
   --
   type REG_FILE is array(0 to (MEM_DEPTH - 1)) of std_logic_vector((WORD_WIDTH - 1) downto 0);
   --
   -- Create a signal of the array type
   --
   signal reg_32x16x2: REG_FILE;
   --
   -- Special command to stop Quartus from inferring a RAM
   -- Must also turn off assignments / settings / compiler settings/
   -- adavanced settings (synthesis) / auto RAM replacement --> off
   attribute ramstyle : string;
   attribute ramstyle of reg_32x16x2 : signal is "logic";

begin
   process(i_clk)
   begin
      if(rising_edge(i_clk)) then
         if(i_wr_b = '0') then
            reg_32x16x2(to_integer(unsigned(i_wr_addr))) <= i_data_in;
         end if;
      end if;
   end process;

   -- output_logic
   o_data_out_a <= reg_32x16x2(to_integer(unsigned(i_addr_a)));
   o_data_out_b <= reg_32x16x2(to_integer(unsigned(i_addr_b)));

end architecture;
```

Need the address as an integer to index the array.
Address is SLV coming in – need to pass through unsigned along the way



Register Files - HDL

- 32 x 16 bit register file, 2 outputs – testbench
  - Normal clock process
  - No reset process
  - Read A and B outputs opposite directions(unknown)
  - Write to all locations (1-0 pattern)
  - Read A and B outputs opposite directions(1-0 pattern)
  - Write to all locations (0-1 pattern)
  - Read A and B outputs opposite directions(0-1 pattern)
  - Exercise Write address and data but with no wr_b
  - Read A and B outputs opposite directions(no change)