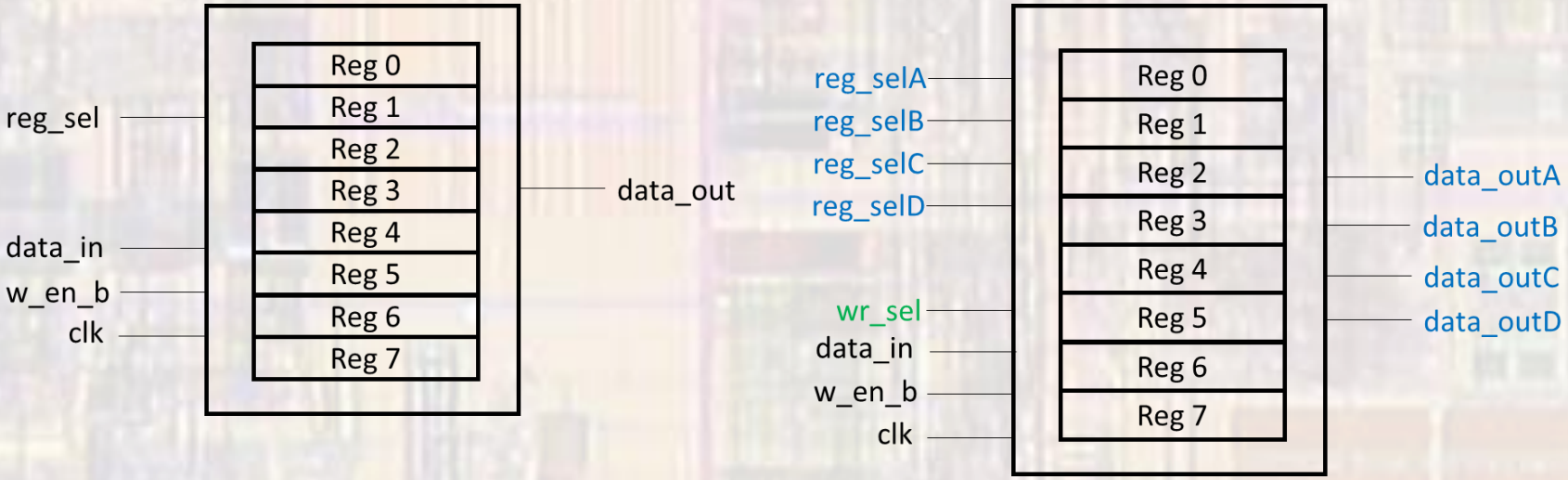
VHDL Register Files

Last updated 1/22/25

Register Files

- Register File
  - A register file is a collection of registers treated as a single entity
  - Commonly used in CPUs (e.g. RA, RB, …)



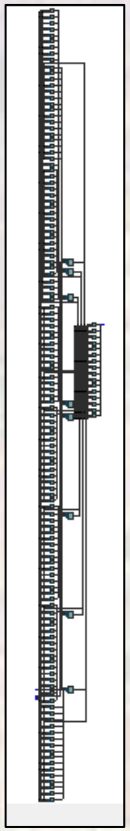
VHDL Register Files

- Register File – 8, 16bit registers – 1 in/out

```
--
-- regfile_16x8x1
--
-- 8 - 16 bit registers
-- single input/output
--
-- created by johnsontimoj
-- 5/7/18
-- rev 0
--
-----------------------------------------
--
-- Inputs: clk, reg_sel, we_bar, data_in
-- Outputs: data_out
--
-- note - reg files do not have a reset
--
-----------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity regfile_16x8x1 is
   port (
         i_clk :         in std_logic;
         i_reg_sel :     in std_logic_vector(2 downto 0);
         i_data_in :     in std_logic_vector(15 downto 0);
         i_we_b:         in std_logic;

         o_data_out:     out std_logic_vector(15 downto 0)
      );
end entity;
```

There are more efficient ways to code this – using an array

```
architecture behavioral of regfile_16x8x1 is
   --
   -- register signals
   --
   signal reg_0: std_logic_vector(15 downto 0);
   signal reg_1: std_logic_vector(15 downto 0);
   signal reg_2: std_logic_vector(15 downto 0);
   signal reg_3: std_logic_vector(15 downto 0);
   signal reg_4: std_logic_vector(15 downto 0);
   signal reg_5: std_logic_vector(15 downto 0);
   signal reg_6: std_logic_vector(15 downto 0);
   signal reg_7: std_logic_vector(15 downto 0);

begin
   --
   -- register creation and write logic
   --
   process(i_clk)
   begin
      if(rising_edge(i_clk)) then
         if(i_we_b = '0') then
            case(i_reg_sel) is
               when "000"  => reg_0 <= i_data_in;
               when "001"  => reg_1 <= i_data_in;
               when "010"  => reg_2 <= i_data_in;
               when "011"  => reg_3 <= i_data_in;
               when "100"  => reg_4 <= i_data_in;
               when "101"  => reg_5 <= i_data_in;
               when "110"  => reg_6 <= i_data_in;
               when "111"  => reg_7 <= i_data_in;
               when others => reg_0 <= i_data_in; -- arbitrary
            end case;
         end if;
      end if;
   end process;

   --
   -- output logic
   --
   process(all)
   begin
      case(i_reg_sel) is
         when "000" => o_data_out <= reg_0;
         when "001" => o_data_out <= reg_1;
         when "010" => o_data_out <= reg_2;
         when "011" => o_data_out <= reg_3;
         when "100" => o_data_out <= reg_4;
         when "101" => o_data_out <= reg_5;
         when "110" => o_data_out <= reg_6;
         when "111" => o_data_out <= reg_7;
         when others => o_data_out <= reg_0; -- arbitrary
      end case;
   end process;

end behavioral;
```



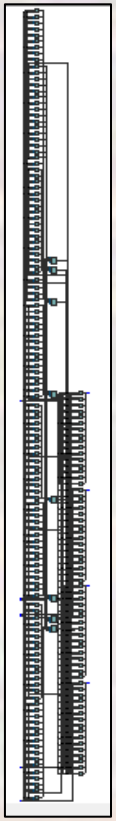
VHDL Register Files

- Register File – 8, 16bit registers – 4 outputs

```
--
-- regfile_16x8x4
--
-- 8 - 16 bit registers
-- 4 input/output
--
-- created by johnsontimoj
-- 5/7/18
-- rev 0
--
-----------------------------------------
--
-- Inputs: clk, reg_sel, wr_sel, we_bar, data_in
-- Outputs: data_out a, b, c, d
--
-- note - reg files do not have a reset
--
-----------------------------------------
library ieee;
use ieee.std_logic_1164.all;

entity regfile_16x8x4 is
    port (
          i_clk :          in std_logic;
          i_reg_selA :     in std_logic_vector(2 downto 0);
          i_reg_selB :     in std_logic_vector(2 downto 0);
          i_reg_selC :     in std_logic_vector(2 downto 0);
          i_reg_selD :     in std_logic_vector(2 downto 0);
          i_wr_sel :       in std_logic_vector(2 downto 0);
          i_data_in :      in std_logic_vector(15 downto 0);
          i_we_b:          in std_logic;

          o_data_outA:     out std_logic_vector(15 downto 0);
          o_data_outB:     out std_logic_vector(15 downto 0);
          o_data_outC:     out std_logic_vector(15 downto 0);
          o_data_outD:     out std_logic_vector(15 downto 0)
        );
end entity;
```

```
architecture behavioral of regfile_16x8x4 is
    --
    -- register signals
    --
    signal reg_0: std_logic_vector(15 downto 0);
    signal reg_1: std_logic_vector(15 downto 0);
    signal reg_2: std_logic_vector(15 downto 0);
    signal reg_3: std_logic_vector(15 downto 0);
    signal reg_4: std_logic_vector(15 downto 0);
    signal reg_5: std_logic_vector(15 downto 0);
    signal reg_6: std_logic_vector(15 downto 0);
    signal reg_7: std_logic_vector(15 downto 0);

begin
    --
    -- write and register creation logic
    --
    process(i_clk)
    begin
        if(rising_edge(i_clk)) then
            if(i_we_b = '0') then
                case(i_wr_sel) is
                    when "000"  => reg_0 <= i_data_in;
                    when "001"  => reg_1 <= i_data_in;
                    when "010"  => reg_2 <= i_data_in;
                    when "011"  => reg_3 <= i_data_in;
                    when "100"  => reg_4 <= i_data_in;
                    when "101"  => reg_5 <= i_data_in;
                    when "110"  => reg_6 <= i_data_in;
                    when "111"  => reg_7 <= i_data_in;
                    when others => reg_0 <= i_data_in; -- arbitrary
                end case;
            end if;
        end if;
    end process;
```

```
    --
    -- output logic
    --
    process(all)
    begin
        case(i_reg_selA) is
            when "000"  => o_data_outA <= reg_0;
            when "001"  => o_data_outA <= reg_1;
            when "010"  => o_data_outA <= reg_2;
            when "011"  => o_data_outA <= reg_3;
            when "100"  => o_data_outA <= reg_4;
            when "101"  => o_data_outA <= reg_5;
            when "110"  => o_data_outA <= reg_6;
            when "111"  => o_data_outA <= reg_7;
            when others => o_data_outA <= reg_0; -- arbitrary
        end case;
        case(i_reg_selB) is
            when "000"  => o_data_outB <= reg_0;
            when "001"  => o_data_outB <= reg_1;
            when "010"  => o_data_outB <= reg_2;
            when "011"  => o_data_outB <= reg_3;
            when "100"  => o_data_outB <= reg_4;
            when "101"  => o_data_outB <= reg_5;
            when "110"  => o_data_outB <= reg_6;
            when "111"  => o_data_outB <= reg_7;
            when others => o_data_outB <= reg_0; -- arbitrary
        end case;
            case(i_reg_selC) is
            when "000"  => o_data_outC <= reg_0;
            when "001"  => o_data_outC <= reg_1;
            when "010"  => o_data_outC <= reg_2;
            when "011"  => o_data_outC <= reg_3;
            when "100"  => o_data_outC <= reg_4;
            when "101"  => o_data_outC <= reg_5;
            when "110"  => o_data_outC <= reg_6;
            when "111"  => o_data_outC <= reg_7;
            when others => o_data_outC <= reg_0; -- arbitrary
        end case;
            case(i_reg_selD) is
            when "000"  => o_data_outD <= reg_0;
            when "001"  => o_data_outD <= reg_1;
            when "010"  => o_data_outD <= reg_2;
            when "011"  => o_data_outD <= reg_3;
            when "100"  => o_data_outD <= reg_4;
            when "101"  => o_data_outD <= reg_5;
            when "110"  => o_data_outD <= reg_6;
            when "111"  => o_data_outD <= reg_7;
            when others => o_data_outD <= reg_0; -- arbitrary
        end case;
    end process;

end behavioral;
```



There are more efficient ways to code this – using an array