



# ECE 545—Digital System Design with VHDL

## Lecture 13

VHDL Modeling of Microprocessors (cont'd)

Final Review

12/2/08

1

### Outline

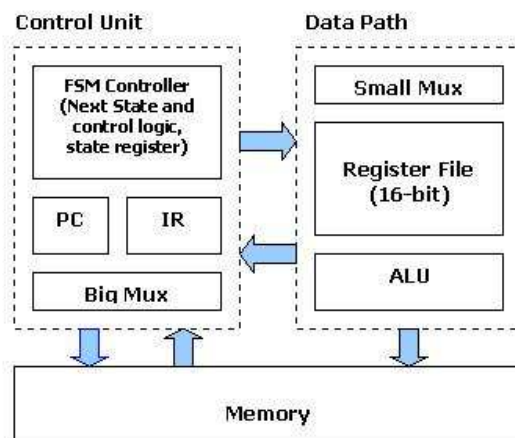
- VHDL Modeling of Microprocessors
  - VHDL Code for Fibonacci Sequence on Simple Microprocessor
- Final Review

2

## Simple Microprocessor Design cont'd

3

## Simple Microprocessor: Architecture



Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

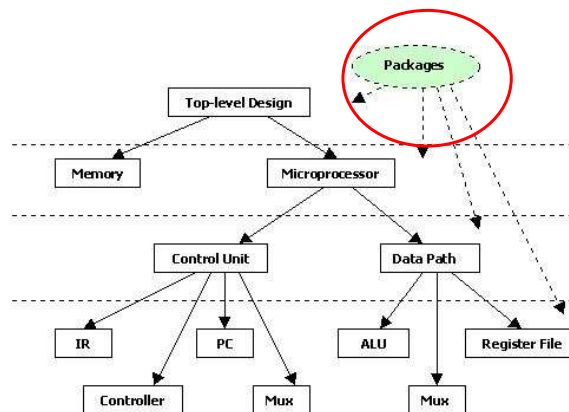
4

## Source of Code

- The following code is from the following source:
  - <http://esd.cs.ucr.edu/labs/tutorial/>
- I made slight modifications to the code
- This code is helpful because:
  - This code illustrates the difference between behavioral non-synthesizable code and synthesizable code
  - This code helps to learn how to read others' code: an important design skill!

5

## Simple Microprocessor: VHDL Hierarchy



Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

6

## constant\_lib.vhd

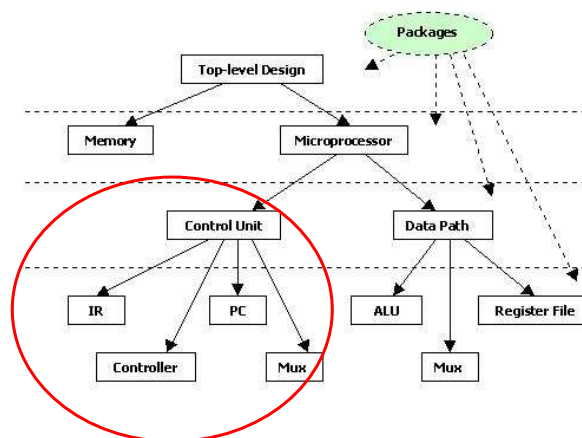
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

PACKAGE constant_lib IS
CONSTANT HIRES : std_logic_vector(15 downto 0) := "ZZZZZZZZZZZZZZZZ";

CONSTANT ZERO:          std_logic_vector(15 downto 0) :=
    "0000000000000000";
CONSTANT mov1 : std_logic_vector(3 downto 0) := "0000";
CONSTANT mov2 : std_logic_vector(3 downto 0) := "0001";
CONSTANT mov3 : std_logic_vector(3 downto 0) := "0010";
CONSTANT mov4 : std_logic_vector(3 downto 0) := "0011";
CONSTANT add :          std_logic_vector(3 downto 0) := "0100";
CONSTANT sub :          std_logic_vector(3 downto 0) := "0101";
CONSTANT jz :           std_logic_vector(3 downto 0) := "0110";
CONSTANT readm:         std_logic_vector(3 downto 0) := "0111";
CONSTANT halt :        std_logic_vector(3 downto 0) := "1111";
END constant_lib;
```

7

## Simple Microprocessor: VHDL Hierarchy



Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

8

## ir.vhd

```
-----  
-- Simple Microprocessor Design (ESD Book Chapter 3)  
-- Copyright 2001, Weijun Zhang  
--  
-- Instruction Register  
-- IR.vhd  
-----  
  
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;  
  
entity IR is  
port(  IRin:      in std_logic_vector(15 downto 0);  
      IRld:      in std_logic;  
      dir_addr:  out std_logic_vector(15 downto 0);  
      IRout:     out std_logic_vector(15 downto 0)  
);  
end IR;
```

## ir.vhd

```
architecture behv of IR is  
  
begin  
  process(IRld, IRin)  
  begin  
    if IRld = '1' then  
      IRout <= IRin;  
      dir_addr <= "00000000" & IRin(7 downto 0);  
    end if;  
  end process;  
end behv;
```

Non-synthesizable model



## smallmux.vhd

```
-----  
-- Simple Microprocessor Design  
--  
-- multiplexor of control unit  
-- three 16 bit inputs and one 16 bit output  
-- bigmux.vhd  
-----  
  
library ieee;  
use ieee.std_logic_1164.all;  
  
entity smallmux is  
port(   I0:      in std_logic_vector(15 downto 0);  
        I1:      in std_logic_vector(15 downto 0);  
        I2:              in std_logic_vector(15 downto 0);  
        Sel:      in std_logic_vector(1 downto 0);  
        O:              out std_logic_vector(15 downto 0)  
);  
end smallmux;
```

## smallmux.vhd

```
architecture behv of smallmux is  
  
begin  
  process(I0, I1, I2, Sel)  
  begin  
    case Sel is  
      when "00" => O <= I0;  
      when "01" => O <= I1;  
      when "10" => O <= I2;  
      when others =>  
    end case;  
  end process;  
end behv;
```

## pc.vhd

```
-----  
-- Simple Microprocessor Design  
--  
-- Program Counter  
-- PC.vhd  
-----  
  
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;  
use work.constant_lib.all;  
  
entity PC is  
port(  --clock: in std_logic;  
      PCld:  in std_logic;  
      PCinc: in std_logic;  
      PCclr: in std_logic;  
      PCin:  in std_logic_vector(15 downto 0);  
      PCout: out std_logic_vector(15 downto 0)  
);  
end PC;
```

## pc.vhd

```
architecture behv of PC is  
  
  signal tmp_PC: std_logic_vector(15 downto 0);  
  
  begin  
    process(PCclr, PCinc, PCld, PCin)  
      begin  
        if PCclr='1' then  
          tmp_PC <= ZERO;  
        elsif (PCld'event and PCld = '1') then  
          --elsif PCld = '1' then  
            tmp_PC <= PCin;  
        elsif (PCinc'event and PCinc = '1') then  
          --elsif PCinc = '1' then  
            tmp_PC <= tmp_PC + 1;  
        end if;  
      end process;  
  
      PCout <= tmp_PC;  
    end behv;
```

Non-synthesizable model

## controller.vhd

```
-----  
-- Simple Microprocessor Design (ESD Book Chapter 3)  
-- Copyright 2001 Weijun Zhang  
--  
-- Controller (control logic plus state register)  
-- VHDL FSM modeling  
-- controller.vhd  
-----
```

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;  
use work.constant_lib.all;
```

## controller.vhd

```
entity controller is  
port( clock: in std_logic;  
rst: in std_logic;  
IR_word: in std_logic_vector(15 downto 0);  
RFs_ctrl: out std_logic_vector(1 downto 0);  
RFwa_ctrl: out std_logic_vector(3 downto 0);  
RFr1a_ctrl: out std_logic_vector(3 downto 0);  
RFr2a_ctrl: out std_logic_vector(3 downto 0);  
RFwe_ctrl: out std_logic;  
RFr1e_ctrl: out std_logic;  
RFr2e_ctrl: out std_logic;  
  
ALUs_ctrl: out std_logic_vector(1 downto 0);  
jmpen_ctrl: out std_logic;  
PCinc_ctrl: out std_logic;  
PCclr_ctrl: out std_logic;  
IRld_ctrl: out std_logic;  
Ms_ctrl: out std_logic_vector(1 downto 0);  
Mre_ctrl: out std_logic;  
Mwe_ctrl: out std_logic;  
oe_ctrl: out std_logic  
);  
end controller;
```

## controller.vhd

```
architecture fsm of controller is

    type state_type is ( S0,S1,S1a,S1b,S2,S3,S3a,S3b,S4,S4a,S4b,S5,S5a,S5b,
                        S6,S6a,S7,S7a,S7b,S8,S8a,S8b,S9,S9a,S9b,S10,S11,S11a);
    signal state: state_type;

begin

    process(clock, rst, IR_word)

        variable OPCODE: std_logic_vector(3 downto 0);

    begin

        if rst='1' then
            Ms_ctrl <= "10";
            PCclr_ctrl <= '1';                                -- Reset State
            PCinc_ctrl <= '0';
            IRld_ctrl <= '0';
            Rfs_ctrl <= "00";
            Rfwe_ctrl <= '0';
            Mre_ctrl <= '0';
            Mwe_ctrl <= '0';
            jmpen_ctrl <= '0';
            oe_ctrl <= '0';
            state <= S0;
        end if;
    end process;
end architecture;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

17

## controller.vhd

```
elsif (clock'event and clock='1') then

    case state is

        when S0 => PCclr_ctrl <= '0';                        -- Reset State
                    state <= S1;

        when S1 => PCinc_ctrl <= '0';                        -- Fetch
                    Instruction
                    RFwe_ctrl <= '0';
                    Ms_ctrl <= "10";
                    IRld_ctrl <= '1';
                    Mwe_ctrl <= '0';
                    Mre_ctrl <= '1';
                    jmpen_ctrl <= '0';
                    oe_ctrl <= '0';
                    state <= S1a;

        when S1a => PCinc_ctrl <= '1';
                    state <= S1b;

        when S1b => PCinc_ctrl <= '0';                        -- Fetch end ..
                    state <= S2;
    end case;
end if;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

18

## controller.vhd

```
when S2 => OPCODE := IR_word(15 downto 12);
        case OPCODE is
            when mov1 => state <= S3;
            when mov2 => state <= S4;
            when mov3 => state <= S5;
            when mov4 => state <= S6;
            when add => state <= S7;
            when subt => state <= S8;
            when jz => state <= S9;
            when halt => state <= S10;
            when readm => state <= S11;
            when others => state <= S1;
        end case;

when S3 => RFwa_ctrl <= IR_word(11 downto 8); -- RF[rn] <= mem[direct]
        RFs_ctrl <= "01";
        Ms_ctrl <= "01";
        Mre_ctrl <= '1';
        Mwe_ctrl <= '0';
        state <= S3a;
when S3a => RFwe_ctrl <= '1';
        state <= S3b;
when S3b => RFwe_ctrl <= '0';
        state <= S1;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

19

## controller.vhd

```
when S4 => RFr1a_ctrl <= IR_word(11 downto 8); --
mem[direct] <= RF[rn]
        RFr1e_ctrl <= '1';
        Ms_ctrl <= "01";
        ALUs_ctrl <= "00";
        IRld_ctrl <= '0';
        state <= S4a; -- read value from RF
when S4a => Mre_ctrl <= '0';
        Mwe_ctrl <= '1';
        state <= S4b; -- write into memory
when S4b => Ms_ctrl <= "10";
        Mwe_ctrl <= '0';
        state <= S1;

when S5 => RFr1a_ctrl <= IR_word(11 downto 8); -- mem[RF[rn]]
<= RF[rm]
        RFr1e_ctrl <= '1';
        Ms_ctrl <= "00";
        ALUs_ctrl <= "01";
        RFr2a_ctrl <= IR_word(7 downto 4); -- set address & data
        RFr2e_ctrl <= '1';
        state <= S5a;
when S5a => Mre_ctrl <= '0'; -- write into memory
        Mwe_ctrl <= '1';
        state <= S5b;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

20

## controller.vhd

```
when S5b => Ms_ctrl <= "10";           -- return
           Mwe_ctrl <= '0';
           state <= S1;

when S6 => RFWa_ctrl <= IR_word(11 downto 8); -- RF[rn] <= imm.
           RFwe_ctrl <= '1';           -- done
           RFS_ctrl <= "10";
           IRld_ctrl <= '0';
           state <= S6a;
when S6a => state <= S1;

when S7 => RFR1a_ctrl <= IR_word(11 downto 8);
           -- RF[rn] <= RF[rn] + RF[rm]
           RFR1e_ctrl <= '1';
           RFR2a_ctrl <= IR_word(7 downto 4);
           RFR2e_ctrl <= '1';
           ALUs_ctrl <= "10";
           state <= S7a;
when S7a => RFR1e_ctrl <= '0';
           RFR2e_ctrl <= '0';
           RFS_ctrl <= "00";
           RFWa_ctrl <= IR_word(11 downto 8);
           RFwe_ctrl <= '1';
           state <= S7b;
when S7b => state <= S1;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

21

## controller.vhd

```
when S8 => RFR1a_ctrl <= IR_word(11 downto 8);
           -- RF[rn] <= RF[rn] - RF[rm]
           RFR1e_ctrl <= '1';
           RFR2a_ctrl <= IR_word(7 downto 4);
           RFR2e_ctrl <= '1';
           ALUs_ctrl <= "11";
           state <= S8a;
when S8a => RFR1e_ctrl <= '0';
           RFR2e_ctrl <= '0';
           RFS_ctrl <= "00";
           RFWa_ctrl <= IR_word(11 downto 8);
           RFwe_ctrl <= '1';
           state <= S8b;
when S8b => state <= S1;

when S9 => jmpen_ctrl <= '1';
           RFR1a_ctrl <= IR_word(11 downto 8);           -- jz if R[rn] =
0
           RFR1e_ctrl <= '1';
           ALUs_ctrl <= "00";
           state <= S9a;
when S9a => state <= S9b;
when S9b => state <= S1;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

22

## controller.vhd

```
when S10 =>          state <= S10;          -- halt

when S11 =>  Ms_ctrl <= "01";              -- read memory
             Mre_ctrl <= '1';
             Mwe_ctrl <= '0';
             state <= S11a;
when S11a =>  oe_ctrl <= '1';
             state <= S1;

when others =>

end case;

end if;

end process;

end fsm;
```

## ctrl\_unit.vhd

```
-----
-- Simple Microprocessor Design (ESD Book Chapter 3)
-- Copyright Spring 2001 Weijun Zhang
--
-- Control Unit composed of Controller, PC, IR and multiplexor
-- VHDL structural modeling
-- ctrl_unit.vhd
-----

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
```

## ctrl\_unit.vhd

```
entity ctrl_unit is
port(   clock_cu:      in      std_logic;
      rst_cu:         in      std_logic;
      PCld_cu:        in      std_logic;
      mdata_out:      in      std_logic_vector(15 downto 0);
      dpdata_out:     in      std_logic_vector(15 downto 0);
      maddr_in:       out     std_logic_vector(15 downto 0);
      immdata:        out     std_logic_vector(15 downto 0);
      RFS_cu:         out     std_logic_vector(1 downto 0);
      RFwa_cu:        out     std_logic_vector(3 downto 0);
      RFr1a_cu:       out     std_logic_vector(3 downto 0);
      RFr2a_cu:       out     std_logic_vector(3 downto 0);
      RFwe_cu:        out     std_logic;
      RFr1e_cu:       out     std_logic;
      RFr2e_cu:       out     std_logic;
      jpen_cu:        out     std_logic;
      ALUs_cu:        out     std_logic_vector(1 downto 0);
      Mre_cu:         out     std_logic;
      Mwe_cu:         out     std_logic;
      oe_cu:          out     std_logic
);
end ctrl_unit;
```

## ctrl\_unit.vhd

```
architecture struct of ctrl_unit is

component controller is
port(   clock:      in std_logic;
      rst:         in std_logic;
      IR_word:     in std_logic_vector(15 downto 0);
      RFS_ctrl:    out std_logic_vector(1 downto 0);
      RFwa_ctrl:   out std_logic_vector(3 downto 0);
      RFr1a_ctrl:  out std_logic_vector(3 downto 0);
      RFr2a_ctrl:  out std_logic_vector(3 downto 0);
      RFwe_ctrl:   out std_logic;
      RFr1e_ctrl:  out std_logic;
      RFr2e_ctrl:  out std_logic;

      ALUs_ctrl:   out std_logic_vector(1 downto 0);
      jmpen_ctrl:  out std_logic;
      PCinc_ctrl:  out std_logic;
      PCclr_ctrl:  out std_logic;
      IRld_ctrl:   out std_logic;
      Ms_ctrl:     out std_logic_vector(1 downto 0);
      Mre_ctrl:    out std_logic;
      Mwe_ctrl:    out std_logic;
      oe_ctrl:     out std_logic
);
end component;
```

## ctrl\_unit.vhd

```
component IR is
port(   IRin:           in std_logic_vector(15 downto 0);
      IRld:           in std_logic;
      dir_addr:       out std_logic_vector(15 downto 0);
      IRout:          out std_logic_vector(15 downto 0)
);
end component;

component PC is
port(   PCld:   in std_logic;
      PCinc:   in std_logic;
      PCclr:   in std_logic;
      PCin: in std_logic_vector(15 downto 0);
      PCout:   out std_logic_vector(15 downto 0)
);
end component;

component bigmux is
port(   Ia:   in std_logic_vector(15 downto 0);
      Ib:   in std_logic_vector(15 downto 0);
      Ic:   in std_logic_vector(15 downto 0);
      Id:   in std_logic_vector(15 downto 0);
      Option: in std_logic_vector(1 downto 0);
      Muxout: out std_logic_vector(15 downto 0)
);
end component;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

27

## ctrl\_unit.vhd

```
signal IR_sig: std_logic_vector(15 downto 0);
signal PCinc_sig, PCclr_sig, IRld_sig: std_logic;
signal Ms_sig: std_logic_vector(1 downto 0);
signal PC2mux: std_logic_vector(15 downto 0);
signal IR2mux_a, IR2mux_b: std_logic_vector(15 downto 0);

begin

  IR2mux_a <= "00000000" & IR_sig(7 downto 0);
  IR2mux_b <= "000000000000" & IR_sig(11 downto 8);
  immdata <= IR2mux_a;

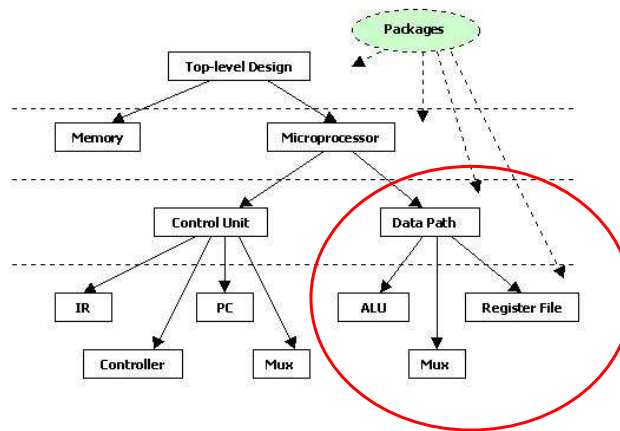
  U0: controller port map( clock_cu,rst_cu,IR_sig,RFs_cu,RFwa_cu,
                          RFr1a_cu,RFr2a_cu,RFwe_cu,RFrie_cu,
                          RFr2e_cu,ALUs_cu,jpen_cu,PCinc_sig,
                          PCclr_sig,IRld_sig,Ms_sig,Mre_cu,Mwe_cu,oe_cu);
  U1: PC port map(PCld_cu, PCinc_sig, PCclr_sig, IR2mux_a, PC2mux);
  U2: IR port map(mdata_out, IRld_sig, IR2mux_a, IR_sig);
  U3: bigmux port map(dpdata_out,IR2mux_a,PC2mux,IR2mux_b,Ms_sig,maddr_in);

end struct;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

28

## Simple Microprocessor: VHDL Hierarchy



Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

29

## obuf.vhd

```
-----  
-- Simple Microprocessor Design (ESD Book Chapter 3)  
-- Copyright 2001 Weijun Zhang  
--  
-- Output buffer of Data Path  
-- obuf.vhd  
-----  
  
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use work.constant_lib.all;  
  
entity obuf is  
port(  
    O_en:          in std_logic;  
    obuf_in:       in std_logic_vector(15 downto 0);  
    obuf_out:      out std_logic_vector(15 downto 0)  
);  
end obuf;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

30

## obuf.vhd

```
architecture behv of obuf is
begin

    process (O_en, obuf_in)
    begin
        if O_en = '1' then
            obuf_out <= obuf_in;
        else
            obuf_out <= HIRES;
        end if;
    end process;

end behv;
```

## regfile.vhd

```
-- Simple Microprocessor Design (ESD Book Chapter 3)
-- Copyright 2001 Weijun Zhang
--
-- Register Files (16*16) of datapath compsed of
-- 4-bit address bus; 16-bit data bus
-- reg_file.vhd
-----
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.constant_lib.all;

entity reg_file is
port ( clock      :          in std_logic;
      rst         :          in std_logic;
      RFwe        :          in std_logic;
      RFr1e       :          in std_logic;
      RFr2e       :          in std_logic;
      RFwa        :          in std_logic_vector(3 downto 0);
      RFr1a       :          in std_logic_vector(3 downto 0);
      RFr2a       :          in std_logic_vector(3 downto 0);
      RFw         :          in std_logic_vector(15 downto 0);
      RFr1        :          out std_logic_vector(15 downto 0);
      RFr2        :          out std_logic_vector(15 downto 0));
end reg_file;
```

## regfile.vhd

```
architecture behv of reg_file is

    type rf_type is array (0 to 15) of
        std_logic_vector(15 downto 0);
    signal tmp_rf: rf_type;

begin

    write: process(clock, rst)
    begin
        if rst='1' then
            tmp_rf <= (tmp_rf'range => ZERO);           -- high active
        else
            if (clock'event and clock = '1') then
                if RFwe='1' then
                    tmp_rf(conv_integer(RFwa)) <= RFw;
                end if;
            end if;
        end if;
    end process;

end architecture;
```

## regfile.vhd

```
read1: process(clock, rst)
begin
    if rst='1' then
        RFr1 <= ZERO;
    else
        if (clock'event and clock = '1') then
            if RFr1e='1' then

                RFr1 <= tmp_rf(conv_integer(RFr1a));
            end if;
        end if; end if;
    end process;

read2: process(clock, rst)
begin
    if rst='1' then
        RFr2 <= ZERO;
    else
        if (clock'event and clock = '1') then
            if RFr2e='1' then

                RFr2 <= tmp_rf(conv_integer(RFr2a));
            end if;
        end if;
    end process;
end behv;
```

## bigmux.vhd

```
-----  
-- Simple Microprocessor Design (ESD Book Chapter 3)  
-- Copyright 2001 Weijun Zhang  
--  
-- big multiplexor of control unit has  
-- four 16-bit inputs and one 16-bit output  
-- bigmux.vhd  
-----  
  
library ieee;  
use ieee.std_logic_1164.all;  
  
entity bigmux is  
port(   Ia:      in std_logic_vector(15 downto 0);  
       Ib:      in std_logic_vector(15 downto 0);  
       Ic:      in std_logic_vector(15 downto 0);  
       Id:      in std_logic_vector(15 downto 0);  
       Option:  in std_logic_vector(1 downto 0);  
       Muxout:  out std_logic_vector(15 downto 0)  
);  
end bigmux;
```

## bigmux.vhd

```
architecture behv of bigmux is  
  
begin  
  
    process(Ia, Ib, Ic, Id, Option)  
    begin  
        case Option is  
        when "00" => Muxout <= Ia;  
        when "01" => Muxout <= Ib;  
        when "10" => Muxout <= Ic;  
        when "11" => Muxout <= Id;  
        when others =>  
            end case;  
        end process;  
  
end behv;
```

## alu.vhd

```
-----  
-- Simple Microprocessor Design  
--  
-- alu has functions of bypass, addition and subtraction  
-- alu.vhd  
-----  
  
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;  
use work.constant_lib.all;  
  
entity alu is  
port ( num_A: in std_logic_vector(15 downto 0);  
       num_B: in std_logic_vector(15 downto 0);  
       jpsign: in std_logic;  
       ALUs: in std_logic_vector(1 downto 0);  
       ALUz: out std_logic;  
       ALUout: out std_logic_vector(15 downto 0)  
);  
end alu;
```

## alu.vhd

```
architecture behv of alu is  
signal alu_tmp: std_logic_vector(15 downto 0);  
begin  
  process(num_A, num_B, ALUs)  
  begin  
    case ALUs is  
      when "00" => alu_tmp <= num_A;  
      when "01" => alu_tmp <= num_B;  
      when "10" => alu_tmp <= num_A + num_B;  
      when "11" => alu_tmp <= num_A - num_B;  
      when others =>  
        end case;  
    end process;  
  
    process(jpsign, alu_tmp)  
    begin  
      if (jpsign = '1' and alu_tmp = ZERO) then  
        ALUz <= '1';  
      else  
        ALUz <= '0';  
      end if;  
    end process;  
    ALUout <= alu_tmp;  
  end behv;
```

## datapath.vhd

```
-----  
-- Simple Microprocessor Design (ESD Book Chapter 3)  
-- Copyright 2001 Weijun Zhang  
--  
-- DATAPATH composed of Multiplexor, Register File and ALU  
-- VHDL structural modeling  
-- datapath.vhd  
-----  
  
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;  
  
entity datapath is  
port(  clock_dp:      in      std_logic;  
      rst_dp:        in      std_logic;  
      imm_data:      in      std_logic_vector(15 downto 0);  
      mem_data:      in      std_logic_vector(15 downto 0);  
      RFs_dp:        in      std_logic_vector(1 downto 0);  
      RFwa_dp:       in      std_logic_vector(3 downto 0);  
      RFr1a_dp:      in      std_logic_vector(3 downto 0);  
      RFr2a_dp:      in      std_logic_vector(3 downto 0);  
      RFwe_dp:       in      std_logic;  
      RFr1e_dp:      in      std_logic;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

39

## datapath.vhd

```
      RFr2e_dp:      in      std_logic;  
      jp_en:        in      std_logic;  
      ALUs_dp:      in      std_logic_vector(1 downto 0);  
      oe_dp:        in      std_logic;  
      ALUz_dp:      out     std_logic;  
      RFlout_dp:    out     std_logic_vector(15 downto 0);  
      ALUout_dp:    out     std_logic_vector(15 downto 0);  
      bufout_dp:    out     std_logic_vector(15 downto 0)  
);  
end datapath;  
  
architecture struct of datapath is  
  
  component smallmux is  
  port(  I0:      in  std_logic_vector(15 downto 0);  
        I1:  in  std_logic_vector(15 downto 0);  
        I2:  in  std_logic_vector(15 downto 0);  
        Sel: in  std_logic_vector(1 downto 0);  
        O:   out std_logic_vector(15 downto 0)  
  );  
  end component;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

40

## datapath.vhd

```
component reg_file is
port ( clock      :      in std_logic;
      rst       :      in std_logic;
      RFwe      :      in std_logic;
      RFr1e     :      in std_logic;
      RFr2e     :      in std_logic;
      RFwa      :      in std_logic_vector(3 downto 0);
      RFr1a     :      in std_logic_vector(3 downto 0);
      RFr2a     :      in std_logic_vector(3 downto 0);
      RFw       :      in std_logic_vector(15 downto 0);
      RFr1      :      out std_logic_vector(15 downto 0);
      RFr2      :      out std_logic_vector(15 downto 0)
);
end component;

component alu is
port ( num_A:    in std_logic_vector(15 downto 0);
      num_B:    in std_logic_vector(15 downto 0);
      jpsign:   in std_logic;
      ALUs:     in std_logic_vector(1 downto 0);
      ALUz:     out std_logic;
      ALUout:   out std_logic_vector(15 downto 0)
);
end component;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

41

## datapath.vhd

```
component obuf is
port( O_en:      in std_logic;
      obuf_in:   in std_logic_vector(15 downto 0);
      obuf_out:  out std_logic_vector(15 downto 0)
);
end component;

signal mux2rf, rf2alu1: std_logic_vector(15 downto 0);
signal rf2alu2, alu2memmux: std_logic_vector(15 downto 0);
begin

U1: smallmux port map(alu2memmux, mem_data,
                     imm_data, RFs_dp, mux2rf);
U2: reg_file port map(clock_dp, rst_dp, RFwe_dp,
                     RFr1e_dp, RFr2e_dp,
                     RFwa_dp, RFr1a_dp, RFr2a_dp,
                     mux2rf, rf2alu1, rf2alu2 );
U3: alu port map( rf2alu1, rf2alu2, jp_en, ALUs_dp,
                 ALUz_dp, alu2memmux);
U4: obuf port map(oe_dp, mem_data, bufout_dp);

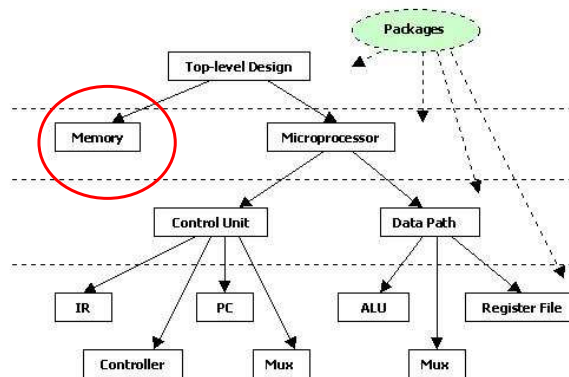
ALUout_dp <= alu2memmux;
RF1out_dp <= rf2alu1;

end struct;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

42

## Simple Microprocessor: VHDL Hierarchy



Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

43

## memory.vhd

```
-----  
-- Simple Microprocessor Design  
--  
-- memory 256*16  
-- 8 bit address; 16 bit data  
-- memory.vhd  
-----  
  
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;  
use work.constant_lib.all;  
  
entity memory is  
port ( clock      : in std_logic;  
      rst         : in std_logic;  
      Mre         : in std_logic;  
      Mwe         : in std_logic;  
      address     : in std_logic_vector(7 downto 0);  
      data_in     : in std_logic_vector(15 downto 0);  
      data_out    : out std_logic_vector(15 downto 0)  
);  
end memory;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

44

## memory.vhd

```
architecture behv of memory is

    type ram_type is array (0 to 255) of
        std_logic_vector(15 downto 0);
    signal tmp_ram: ram_type;

begin

    write: process(clock, rst)
    begin
        if rst='1' then
            tmp_ram <= (

                0 => "0011000000000000",      -- mov R0, #0
                1 => "0011000100000001",      -- mov R1, #1
                2 => "0011001000110100",      -- mov R2, #52
                3 => "0011001100000001",      -- mov R3, #1
                4 => "0001000000110010",      -- mov M[50], R0
                5 => "0001000100110011",      -- mov M[51], R1
                6 => "0001000101100100",      -- mov M[100], R1
                7 => "0100000100000000",      -- add R1, R0
                8 => "0000000001100100",      -- mov M[100], R0
                9 => "0010001000010000",      -- mov M[R2], R1

            );
        end if;
    end process;

end architecture;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

45

## memory.vhd

```
10 => "0100001000110000",      -- add R2, R3
11 => "0000010000111011",      -- mov R4, M[59]
12 => "0110010000000101",      -- jz R4, #6
13 => "0111000000110010",      -- output M[50]
14 => "0111000000110011",      --           ,, M[51]
15 => "0111000000110100",      --           ,, M[52]
16 => "0111000000110101",      --           ,, M[53]
17 => "0111000000110110",      --           ,, M[54]
18 => "0111000000110111",      --           ,, M[55]
19 => "0111000000111000",      --           ,, M[56]
20 => "0111000000111001",      --           ,, M[57]
21 => "0111000000111010",      --           ,, M[58]
22 => "0111000000111011",      --           ,, M[59]
23 => "1111000000000000",      -- halt
others => "0000000000000000");
    else
        if (clock'event and clock = '1') then
            if (Mwe = '1' and Mre = '0') then
                tmp_ram(conv_integer(address)) <= data_in;
            end if;
        end if;
    end process;

end architecture;
```

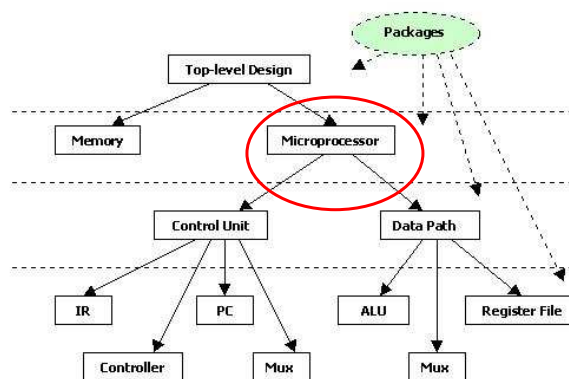
Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

46

## memory.vhd

```
read: process(clock, rst)
begin
    if rst='1' then
        data_out <= ZERO;
    else
        if (clock'event and clock = '1') then
            if (Mre ='1' and Mwe ='0') then
                data_out <=
                    tmp_ram(conv_integer(address));
            end if;
        end if;
    end if;
end process;
end behv;
```

## Simple Microprocessor: VHDL Hierarchy



## microprocessor.vhd

```
-----  
-- Simple Microprocessor Design  
--  
-- Microprocessor composed of  
-- Ctrl_Unit, Data_Path and Memory  
-- structural modeling  
-- microprocessor.vhd  
-----  
  
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_arith.all;  
use ieee.std_logic_unsigned.all;  
use work.constant_lib.all;  
  
entity microprocessor is  
port(   cpu_clk: in std_logic;  
        cpu_rst: in std_logic;  
        cpu_output: out std_logic_vector(15 downto 0)  
);  
end microprocessor;  
  
architecture structure of microprocessor is
```

## microprocessor.vhd

```
component datapath is  
port(   clock_dp: in std_logic;  
        rst_dp: in std_logic;  
        imm_data: in std_logic_vector(15 downto 0);  
        mem_data: in std_logic_vector(15 downto 0);  
        RFs_dp: in std_logic_vector(1 downto 0);  
        RFwa_dp: in std_logic_vector(3 downto 0);  
        RFr1a_dp: in std_logic_vector(3 downto 0);  
        RFr2a_dp: in std_logic_vector(3 downto 0);  
        RFwe_dp: in std_logic;  
        RFr1e_dp: in std_logic;  
        RFr2e_dp: in std_logic;  
        jp_en: in std_logic;  
        ALUs_dp: in std_logic_vector(1 downto 0);  
        oe_dp: in std_logic;  
        ALUz_dp: out std_logic;  
        RFlout_dp: out std_logic_vector(15 downto 0);  
        ALUout_dp: out std_logic_vector(15 downto 0);  
        bufout_dp: out std_logic_vector(15 downto 0)  
);  
end component;
```

## microprocessor.vhd

```
component ctrl_unit is
port(
  clock_cu: in std_logic;
  rst_cu: in std_logic;
  PCld_cu: in std_logic;
  mdata_out: in std_logic_vector(15 downto 0);
  dpdata_out: in std_logic_vector(15 downto 0);
  maddr_in: out std_logic_vector(15 downto 0);
  immdata: out std_logic_vector(15 downto 0);
  RFs_cu: out std_logic_vector(1 downto 0);
  RFwa_cu: out std_logic_vector(3 downto 0);
  RFr1a_cu: out std_logic_vector(3 downto 0);
  RFr2a_cu: out std_logic_vector(3 downto 0);
  RFwe_cu: out std_logic;
  RFr1e_cu: out std_logic;
  RFr2e_cu: out std_logic;
  jpen_cu: out std_logic;
  ALUs_cu: out std_logic_vector(1 downto 0);
  Mre_cu: out std_logic;
  Mwe_cu: out std_logic;
  oe_cu: out std_logic
);
end component;
```

## microprocessor.vhd

```
component memory is
port (
  clock : in std_logic;
  rst : in std_logic;
  Mre : in std_logic;
  Mwe : in std_logic;
  address : in std_logic_vector(7 downto 0);
  data_in : in std_logic_vector(15 downto 0);
  data_out: out std_logic_vector(15 downto 0)
);
end component;

signal addr_bus, mdin_bus, mdout_bus, immd_bus, rfout_bus: std_logic_vector(15
  downto 0);
signal mem_addr: std_logic_vector(7 downto 0);
signal RFwa_s, RFr1a_s, RFr2a_s: std_logic_vector(3 downto 0);
signal RFwe_s, RFr1e_s, RFr2e_s: std_logic;
signal ALUs_s, RFs_s: std_logic_vector(1 downto 0);
signal IRld_s, PCld_s, PCinc_s, PCclr_s: std_logic;
signal Mre_s, Mwe_s, jpz_s, oe_s: std_logic;
```

## microprocessor.vhd

```
begin

    mem_addr <= addr_bus(7 downto 0);

    Unit0: ctrl_unit port map(
        cpu_clk,cpu_rst,PCld_s,mdout_bus,rfout_bus,addr_bus,
                                                immd_bus,
        RFs_s,RFwa_s,RFrla_s,RFr2a_s,RFwe_s,
        RFr1e_s,RFr2e_s,jpz_s,ALUs_s,Mre_s,Mwe_s,oe_s);
    Unit1: datapath port map(
        cpu_clk,cpu_rst,immd_bus,mdout_bus,
        RFs_s,RFwa_s,RFrla_s,RFr2a_s,RFwe_s,RFr1e_s,
        RFr2e_s,jpz_s,ALUs_s,oe_s,PCld_s,rfout_bus,
        mdin_bus,cpu_output);
    Unit2: memory port map(
        cpu_clk,cpu_rst,Mre_s,Mwe_s,mem_addr,mdin_bus,mdout_bus);

end structure;
```

## tb\_mp.vhd

```
-----
-- Simple Microprocessor Design
--
-- Test Bench for Fibonacci Number Generator
-----

library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.constant_lib.all;

entity TB_MP is
end TB_MP;

architecture behv of TB_MP is

    component microprocessor is
    port(
        cpu_clk: in std_logic;
        cpu_rst: in std_logic;
        cpu_output: out std_logic_vector(15 downto 0)
    );
    end component;
```

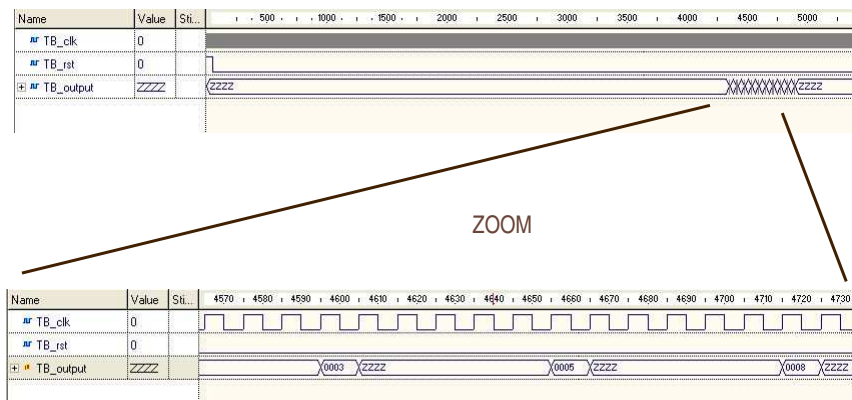
## tb\_mp.vhd

```
signal TB_clk: std_logic;  
signal TB_rst: std_logic;  
signal TB_output: std_logic_vector(15 downto 0);  
  
begin  
  
    Unit: microprocessor port map(TB_clk, TB_rst, TB_output);  
  
    process  
    begin  
        TB_clk <= '0';  
        wait for 5 ns; -- offer clock signal  
        TB_clk <= '1'; -- in cocurrent process  
        wait for 5 ns;  
    end process;  
  
    process  
    begin  
        TB_rst <= '1';  
        wait for 50 ns;  
        TB_rst <= '0';  
        wait for 100000 ns;  
    end process;  
  
end behv;
```

Source: Vahid and Givargis, "Embedded System Design: A Unified Hardware/Software Introduction"

55

## Simulation Waveform



56



## Final Review

57

## Course Objectives

- At the end of this course you should be able to:
  - Code in VHDL for synthesis
  - Decompose a digital system into a controller (FSM) and datapath, and code accordingly
  - Write VHDL testbenches
  - Synthesize and implement digital systems on FPGAs
  - Understand behavioral, non-synthesizable VHDL and its role in modern design
  - Effectively code digital systems for cryptography, signal processing, and microprocessor applications
- This knowledge will come about through homework, exams, and an extensive project
  - The project in particular will help you know VHDL and the FPGA design flow from beginning to end

58

## Final Specifics

- The final will take place in class on Tuesday, December 9, from 4:30 pm – 7:15 pm.
- The final will be open-book, open-notes.
  - You can bring any textbooks
  - No electronic devices (cell phones, PDAs, laptops, etc.) are allowed → you must **PRINT OUT** all notes on paper.
  - Do not come to class with your notes in electronic form on your laptop, as you will not be allowed to use your laptop.
- Bring a simple calculator
- The final will be **comprehensive**, covering all material in the course
  - It will cover all material from lectures, homeworks, projects, hands-on sessions, and textbook readings.
  - Exception: there will be no problems in which you need to use the FPGA CAD tools.
  - We will go over last year's final in class today to give you a better feel for example final problems.

59

## Final Review

- To be done in-class

60