VHDL is an abbreviation for Very **H**igh Speed Integrated Circuit **H**ardware **D**escription **L**anguage, and is used for modeling digital systems. VHDL coding includes *behavior modeling*, *structure modeling* and *dataflow modeling*, After studying the following simple examples, you will quickly become familiar with its syntax and reserved words (in bold). Please note that these codes are only for your reference, NOT the exact solution for EE/CS120A’s Hwk/Lab/Project. However, if you spend some time to understand all of them, it could be very helpful, even for the future class EE/CS120B. Have fun!

(1) D Flip-Flop with Asynchronous Reset

```
library IEEE;
use IEEE.std_logic_1164.all;

entity dff_async_rst is
      port( data, clk, reset: in    std_logic;
            q:                out   std_logic);
end dff_async_rst;

architecture behav of dff_async_rst is
begin
process(clk,reset)
begin
      if (reset=’0’) then
            q<=’0’;
      elsif (clk’event and clk=’1’) then
            q<=data;
      end if;
end process;
end behav;
```

(2) 4:1 Multiplexor

```
library IEEE;
use IEEE.std_logic_1164.all;

entity mux is
      port( c,d,e,f:    in    std_logic;
            s:          in    std_logic_vector(1 downto 0);
            muxout:     out   std_logic);
end mux;

architecture my_mux of mux is
begin
mux1: process(s,c,d,e,f)
begin
      case s is
            when “00”=> muxout <= c;
            when “01”=> muxout <= d;
            when “10”=> muxout <= e;
            when others => muxout <= f;
      end case;
end process mux1;
end my_mux;
```

(3) Shift Register

```
library IEEE;
use IEEE.std_logic_1164.all;

entity shift is
      port( d_in,clk,resetn:  in    std_logic;
            d_out:            out   std_logic );
end shift;

architecture shift_reg of shift is
begin
process(clk,d_in,d_out)
      variable REG: std_logic_vector(2 downto 0):=(‘0’,’0’,’0’);
begin
      if resetn=’0’ then REG (2 downto 0):=(‘0’,’0’,’0’);
      elsif clk ‘event and clk=’1’ then
            REG:= d_in & REG(2 downto 1);
      end if;
      d_out<=REG(0);
end process;
end shift_reg;
```

(4) 4-bit Up Counter with Enable and Asynchronous Reset

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

entity counter4 is
      port( clk,en,rst:       in    std_logic;
            count:            out   std_logic_vector(7 downto 0));
end counter4;

architecture behav of counter4 is
signal cnt: std_logic_vector (3 downto 0);
begin
process(clk,en,cnt,rst)
      begin
      if (rst=’0’) then
            cnt <= (others =>’0’);
      elsif (clk’event and clk=’1’) then
            if (en=’1’) then
                  cnt <= cnt+’1’;
      end if;
end process;
      count <= cnt;
end behav;
```

(5) 2-4 Decoder

```
library IEEE;
use IEEE.std_logic_1164.all;

entity decode is
      port( Ain:  in    std_logic_vector(1 downto 0);
            En:   in    std_logic;
            Yout: out   std_logic(3 downto 0));
end decode;

architecture decode_arch of decode is
begin
process(Ain)
begin
      if (En=’0’) then
            Yout <= (others=>’0’);
      else
            case Ain is
            when “00”=> Yout<=”0001”;
            when “01”=> Yout<=”0010”;
            when “10”=> Yout<=”0100”;
            when “11”=> Yout<=”1000”;
            end case;
      end if;
end process;
end decode_arch;
```

(6) 8-bit Comparator/Comparator Cell

```
library IEEE;
use IEEE.std_logic_1164.all;

entity cmp_cell is
port( a,b:  in    std_logic;
      Cin:  in    std_logic_vector(1 downto 0);
      Cout: out   std_logic_vector(1 downto 0));
end decode;

architecture cmp_cell of cmp_cell is
begin
process(Cin,a,b)
begin
      if Cin=”10” then Cout<=”10”;
      elsif Cin=”01” then Cout<=”01”;
      elsif Cin=”00” and a=b then Cout<=”00”;
      elsif Cin=”00” and a>b then Cout<=”01”;
      elsif Cin=”00” and a<b then Cout<=”10”;
      end if;
end process;
end cmp_cell;


library IEEE;
use IEEE.std_logic_1164.all;
```

```
entity comparator_8 is
port(
      A,B:        in    std_logic_vector(7 downto 0);
      CMPIN:      in    std_logic_vector(1 downto 0);
      CMPOUT:     out   std_logic_vector(1 downto 0));
end comparator_8;

architecture cmp_8 of comparator_8 is
 component cmp_cell
 port(      a,b:  in    std_logic;
            Cin:  in    std_logic_vector(1 downto 0);
            Cout: out   std_logic_vector(1 downto 0));
 end component;
signal OUT0,OUT1,OUT2,OUT3,OUT4,OUT5,OUT6:std_logic_vector(1 downto 0);
begin
      cell0:cmp_cell port map(A(0),B(0),CMPIN,OUT0);
      cell1:cmp_cell port map(A(1),B(1),OUT0,OUT1);
      cell2:cmp_cell port map(A(2),B(2),OUT1,OUT2);
      cell3:cmp_cell port map(A(3),B(3),OUT2,OUT3);
      cell4:cmp_cell port map(A(4),B(4),OUT3,OUT4);
      cell5:cmp_cell port map(A(5),B(5),OUT4,OUT5);
      cell6:cmp_cell port map(A(6),B(6),OUT5,OUT6);
      cell7:cmp_cell port map(A(7),B(7),OUT6,CMPOUT);
end cmp_8;
```

(7) Finite State Machine

```
library IEEE;
use IEEE.std_logic_1164.all;

entity sequence is
port( clk,resetn,data_in:     in    std_logic;
      data_out:               out   std_logic));
end sequence;

architecture state_machine of SEQUENCE is
      type StateType is (ST0,ST1,ST2,ST3,ST4,ST5,ST6,ST7);
      signal Current_State:StateType;
begin
state_comb: process(clk,resetn, Current_State)
      variable REG: std_logic_vector(2 downto 0);
begin
      if clk ‘event and clk=’1’ and resetn=’1’ then
        case Current_State is
        when ST0 =>
            if data_in=’0’ then Current_State<=ST0;
            else Current_State<=ST1;
            end if;
        when ST1 =>
            if data_in=’0’ then Current_State<=ST2;
            else Current_State<=ST3;
            end if;
       when ST2 =>
            if data_in=’0’ then Current_State<=ST4;
            else Current_State<=ST5;
            end if;
       when ST3 =>
            if data_in=’0’ then Current_State<=ST6;
            else Current_State<=ST7;
            end if;
       when ST4 =>
            if data_in=’0’ then Current_State<=ST0;
            else Current_State<=ST1;
            end if;
       when ST5 =>
            if data_in=’0’ then Current_State<=ST0;
            else Current_State<=ST1;
            end if;
       when ST6 =>
            if data_in=’0’ then Current_State<=ST4;
            else Current_State<=ST5;
            end if;
       when ST7 =>
            if data_in=’0’ then Current_State<=ST6;
            else Current_State<=ST7;
            end if;
       end case;
        REG:=data_in&REG(2 downto 1);
      elsif resetn=’0’ then
            Current_State<=ST0;
            REG(2 downto 0):=(‘0’,’0’,’0’);
      end if;
      data_out<=REG(0);
end process state_comb;
end architecture state_machine;
```