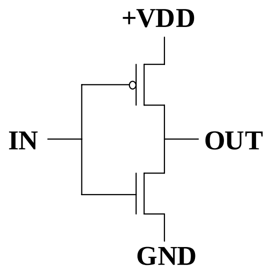
The University of Alabama in Huntsville
ECE Department
CPE 426 01
Final Exam Solution
Spring 2005

1. (5 points) Draw the transistor-level diagram of a CMOS inverter.



2. (10 points) Write a VHDL entity (3 points) and architecture (7 points) of a two-input AND gate with the generics, TIPDA and TIPDB, which reflect the delay on the routing to the inputs A and B, respectively of the AND gate.

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

entity AND_TIP is
  generic (TIPDA, TIPDB : time);
  port (A, B : in std_logic;
        C : out std_logic);
end AND_TIP ;

architecture BEHAV of AND_TIP is
  signal A_TEMP, B_TEMP : std_logic;
begin
  A_TEMP <= transport A after TIPDA;
  B_TEMP <= transport B after TIPDB;
  C <= A_TEMP and B_TEMP;
end BEHAV;
```

3. (1 point) _VITAL_ models provide blanks for timing information that is provided by place and route tools.

4. (1 point) A(n) _ASIC_ is an integrated circuit produced for a specific application and produced in relatively small volumes.

5. (1 point) All sequential circuits should have a(n) __reset_ input.

6. (1 point) A(n) __ entity, package, configuration __ is a primary design unit.

7. (5 points) If the NRE costs for FPGA and CBIC circuits are \$41,000 and \$237,000, respectively, and the cost of individual parts for FPGA and CBIC circuits are \$18 and \$5, respectively, what is the break-even manufacturing volume for these two types of circuits?

$18x + $41,000 = $5x + $237,000
13x = 196,000
x = 15076.9 = 15077

8. (5 points) What kind of hardware element will be inferred by a synthesis tool from the following model?

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

entity WIDGET is
  Port (A, B : in SIGNED (0 to 2);
        CLK, RESET : in std_logic;
        Z : out SIGNED(0 to 2));
end WIDGET;

architecture EXAMPLE of WIDGET is
begin
  process (CLK, RESET)
  begin
    if (RESET = '1') then
      Z <= '0';
    elsif (CLK = '1') then
        Z <= A nor B;
    end if;
  end process;
end EXAMPLE;
```

Answer: A latch with asynchronous reset

9. (1 point) _BIT, Boolean_ is a predefined enumerated type in VHDL.

10. (1 point) A(n) ___ if, case, loop, wait _ is an example of a sequential VHDL statement.

11. (10 points) For the data lifetime chart shown, use the left edge algorithm to obtain an efficient register allocation.

	A	B	C	D	E	F	G	H	I	J	K	L
S1			X	X			X				X	
S2		X	X				X			X	X	
S3		X				X	X		X	X	X	
S4	X	X				X			X		X	X
S5	X					X		X	X		X	X
S6	X				X			X	X			X
S7					X				X			X

	D	C	G	K	J	B	F	I	A	L	H	E
S1	X	X	X	X								
S2		X	X	X	X	X						
S3			X	X	X	X	X	X				
S4				X		X	X	X	X	X		
S5				X			X	X	X	X	X	
S6								X	X	X	X	X
S7								X		X		X

	R1	R2	R3	R4	R5	R6
S1	D	C	G	K		
S2	J	C	G	K	B	
S3	J	F	G	K	B	I
S4	A	F	L	K	B	I
S5	A	F	L	K	H	I
S6	A	E	L		H	I
S7		E	L			I

12. (6 points) (a) (4 points) Translate the following statement to an `if-then-else` statement:

```
transmit <= signal_a when state = idle else
            signal_b when state =  incoming else
            signel_c when state = outgoing else
            signal_d;
```

(b) (2 points) Can these two forms be used interchangeably? Why or why not?

a)

```
if (state = idle) then
  transmit <= signal_a;
elsif (state = incoming) then
  transmit <= signal_b;
elsif (state = outgoing) then
  transmit <= singal_c then
else
  transmit <= signal_d;
end if;
```

(b) No, one is a concurrent statement and one is a sequential statement. The sequential statement can only occur inside a process, procedure or function.

13. (10 points) Build a 16-bit down counter with synchronous load and asynchronous reset. The outputs are three-state outputs, controlled by two separate signals – one for the lower 8 bits and one for the upper 8-bits. Inputs: clk, reset, load, data[15:0], upper_enable, lower_enable; output: count[15:0] (a) (3 points) entity (b) (7 points) architecture Include any necessary library or use statements.

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

```
entity UP_DOWN_COUNTER is
  port (CLK, RESET, LOAD, UPPER_EN, LOWER_EN : in std_logic;
        DATA_IN : in UNSIGNED(15 downto 0);
        DATA_OUT : out UNSIGNED(15 downto 0));
end UP_DOWN_COUNTER;

architecture BEHAV of UP_DOWN_COUNTER is
  signal COUNT : UNSIGNED(15 downto 0);
begin
  process (CLK, RESET)
  begin
    if (RESET = '0') then
      COUNT <= "0000000000000000";
    elsif (CLK'event and CLK = '1') then
      if (LOAD = '1') then
        COUNT <= DATA_IN;
      elsif (COUNT = UNSIGNED'("0000000000000000")) then
        COUNT <= "1111111111111111";
      else
        COUNT <= COUNT - "1";
      end if;
    end if;
  end process;
  process (COUNT, UPPER_EN, LOWER_EN)
  begin
    if (UPPER_EN = '1') then
      DATA_OUT(15 downto 8) <= COUNT(15 downto 8);
    else
      DATA_OUT(15 downto 8) <= "ZZZZZZZZ";
    end if;
    if (LOWER_EN = '1') then
       DATA_OUT(7 downto 0) <= COUNT(7 downto 0);
    else
       DATA_OUT(7 downto 0) <= "ZZZZZZZZ";
    end if;
  end process;
end BEHAV;
```

14. (12 points) (a) (6 points) Write a single VHDL model which represents an exclusive-OR gate with an arbitrary number of inputs, N. (b) (6 points) Use that model as a component in an entity that represents a three input exclusive-OR gate with inputs a, b, c and output f

(a)

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

entity XORN is
  generic (N : integer);
  port (I : in std_logic_vector(N-1 downto 0);
        O : out std_logic);
end XORN;
```

```
architecture BEHAVE of XORN is
begin
  process (I)
    variable TEMP : std_logic;
  begin
    TEMP := '0';
    for J in 0 to N-1 loop
      TEMP := TEMP xor I(J);
    end loop;
    O <= TEMP;
  end process;
end BEHAVE;
```

(b)

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

entity XOR3 is
  port(A, B, C : in std_logic;
       F : out std_logic);
end XOR3;

architecture GEN of XOR3 is
  component XORN is
    generic (N : integer);
    port (I : in std_logic_vector(N-1 downto 0);
          O : out std_logic);
  end component;
  signal TEMP : std_logic_vector(N-1 downto 0);
begin
  TEMP <= A&B&C;
  U1 : XORN generic map (3)
            port map (TEMP, F);
end GEN;
```

15. (6 points) Translate the following VHDL to two `with-select-when` statements:

```
case state is
  when idle => a <= “11”; b <= “00”;
  when terminate | increase => a <= “01”; b <= “--”;
  when maintain | decrease => a <= “10”; b <= “11”;
  when others => a <= “11”; b <= “01”;
end case;
```

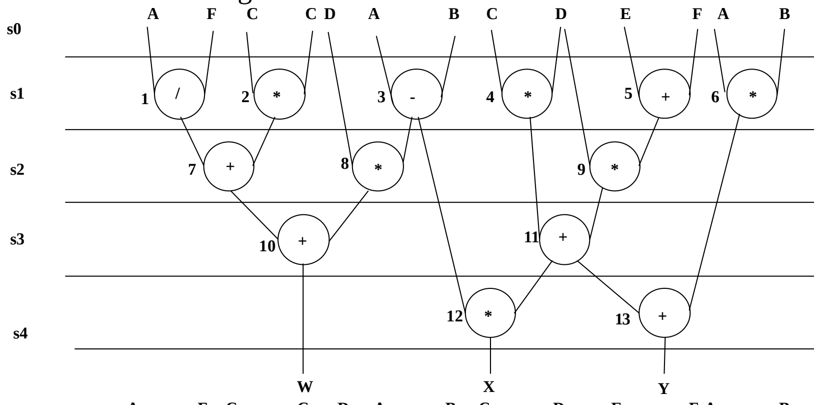
```
with STATE select
    a <= "11" when IDLE,
         "01" when TERMINATE | INCREASE,
         "10" when MAINTAIN | DECREASE,
         "11" when others;
  with STATE select
    b <= "00" when IDLE,
         "--" when TERMINATE | INCREASE,
         "11" when MAINTAIN | DECREASE,
         "01" when others;
```

Consider the following VHDL code:

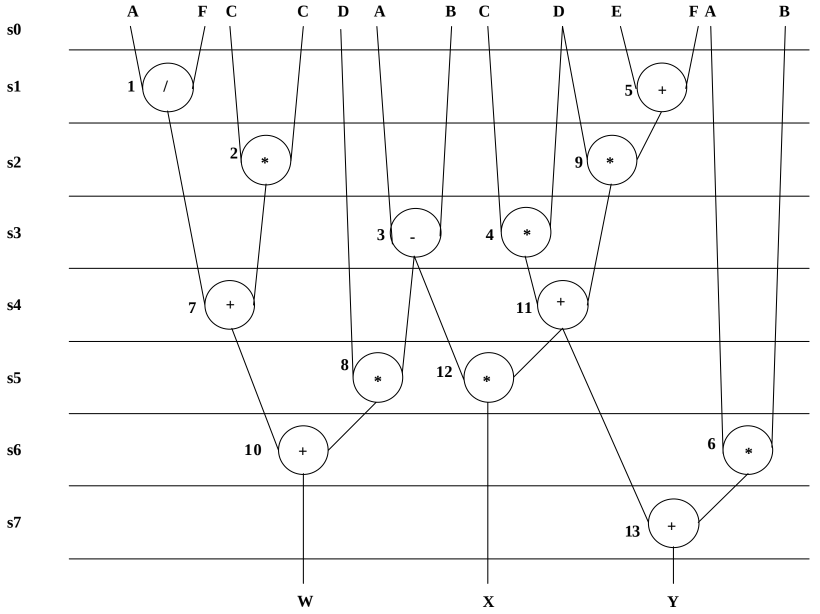
```
entity SCHED2 is
  port (A, B, C, D, E, F: in INTEGER;
        CLK : in BIT;
        W, X, Y: out INTEGER);
end SCHED2;

architecture HIGH_LEVEL of SCHED2 is
  signal Z: INTEGER;
begin
  X <= (A – B) * Z;
  Y <= (A * B) + Z;
  Z <= (C * D) + D * (E + F);
  W <= A/F + C*C + D* (A – B);
end HIGH_LEVEL;
```

16. (15 points) The following task refers to the VHDL code above. Assume that all operations are done in an ALU module and there are two ALU modules available. Derive a schedule for the operations using the freedom-directed algorithm.

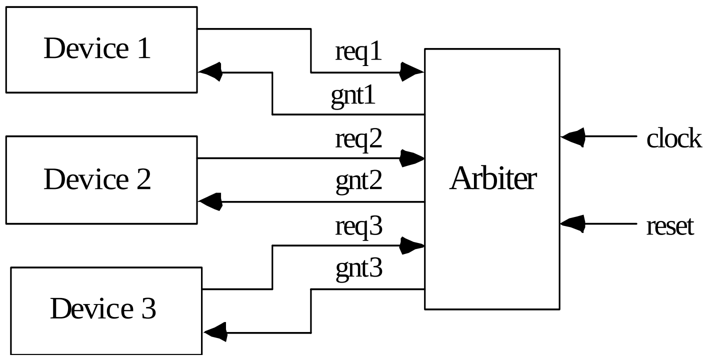


Op	Earliest ASAP	Latest ALAP	Range
1	1	2	2
2	1	2	2
3	1	2	2
4	1	2	2
5	1	1	1
6	1	3	3
7	2	3	2
8	2	3	2
9	2	2	1
10	3	4	2
11	3	3	1
12	4	4	1
13	4	4	1



Step 1: Ready {1 ,2 ,3, 4, 5, 6}
    Schedule {5, 1}
Step 2: Ready {2 , 3, 4, 6, 9}
    Schedule {9, 2}
Step 3: Ready {3, 4, 6, 7}
    Schedule {3, 4}
Step 4: Ready {6, 7, 8, 11}
    Schedule {11, 7}
Step 5: Ready {6, 8, 12}
    Schedule {12, 8}
Step 6: Ready {6, 10}
    Schedule {6, 10}
Step 7: Ready {13}
    Schedule {13

17. (10 points) Model an arbiter in VHDL that is arbitrating between three devices. Device 1 has the highest priority while device 3 has the lowest priority. This arbiter controls access by various devices to a shared resource in a given system, such as a bus. Only one device can use the resource at a time. Assume that the active clock edge is the positive edge. Each device provides one input to the controller, called a request, and the controller produces a separate output for each device, called a grant. A device indicates its need to use the resource by asserting its request signal. Whenever the shared resource is not already in use, the controller considers all requests that are active. Based on a priority scheme, it selects one of the requesting devices and asserts its grant signal. When the device is finished using the resource, it deasserts its request signal. Make sure that there are no implied latches in your model.



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

entity ARBITER is
  port (CLK, RESET, REQ1, REQ2, REQ3 : in std_logic;
        GNT1, GNT2, GNT3 : out std_logic);
end ARBITER;

architecture BEHAV of ARBITER is
  type STATE_TYPE is (S0, S1, S2, S3);
  signal STATE, NEXT_STATE : STATE_TYPE;
begin
  process (CLK, RESET)
  begin
    if (RESET = '0') then
      STATE <= S0;
    elsif (CLK'event and CLK = '1') then
      STATE <= NEXT_STATE;
    end if;
  end process;
  process (STATE, REQ1, REQ2, REQ3)
  begin
    case STATE is
      when S0 => GNT1 <= '0'; GNT2 <= '0'; GNT3 <= '0';
                 if (REQ1 = '1') then
                   NEXT_STATE <= S1;
                 elsif (REQ2 = '1') then
                   NEXT_STATE <= S2;
                 elsif (REQ3 = '1') then
                   NEXT_STATE <= S3;
                 else
                   NEXT_STATE <= S0;
                 end if;
      when S1 => GNT1 <= '1'; GNT2 <= '0'; GNT3 <= '0';
                 if (REQ1 = '0') then
                   if (REQ2 = '1') then
                     NEXT_STATE <= S2;
                   elsif (REQ3 = '1') then
                     NEXT_STATE <= S3;
                   else
                     NEXT_STATE <= S0;
                   end if;
                 else
                   NEXT_STATE <= S1;
                 end if;
      when S2 => GNT1 <= '0'; GNT2 <= '1'; GNT3 <= '0';
                 if (REQ2 = '0') then
                   if (REQ1 = '1') then
                     NEXT_STATE <= S1;
                   elsif (REQ3 = '1') then
                     NEXT_STATE <= S3;
                   else
                    NEXT_STATE <= S0;
                  end if;
                else
                  NEXT_STATE <= S2;
                end if;
      when S3 => GNT1 <= '0'; GNT2 <= '0'; GNT3 <= '1';
                   if (REQ3 = '0') then
                     if (REQ1 = '1') then
                       NEXT_STATE <= S1;
                     elsif (REQ2 = '1') then
                       NEXT_STATE <= S2;
                     else
                       NEXT_STATE <= S0;
                     end if;
                   else
                     NEXT_STATE <= S3;
                   end if;
    end case;
  end process;
end BEHAV;
```