EE 449 - Microprocessor System Design
Test #1

Instructor: Sunil P Khatri

Fall 2012

NAME:

Problem	maximum points	your points
Problem 1	10	
Problem 2	30	
Problem 3	25	
Problem 4	35	
Total	100	

For all your answers which involve Verilog code or C code, you should provide comments in your code for full credit.

For other answers, you must provide comments about *how* you obtained the answer for full credit.

1. (a) Consider the two Verilog modules shown below:

Module 1:

```
module AND_ONE(z, a, b, c)
input a, b, c;
output z;
wire p;
assign p = a & b;
assign z = p & c;
endmodule
```

Module 2:

```
module AND_TWO(z, a, b, c)
input a, b, c;
output z;
wire p;
and g0(p, a, b);
and g1(z, p, c);
endmodule
```

i. What style of Verilog is used in the modules AND_ONE and AND_TWO?

ii. Suppose I was to synthesize AND_ONE. Will the resulting circuit have the same structure as AND_TWO? Why?

Solution:

(a) i. AND_ONE: dataflow model
The module is using a dataflow operator (&) with assign statements.
AND_TWO: structural model
The module is explicitly declaring and gates.

ii. Not necessarily. The synthesizer will try to optimize the dataflow model in a certain way and it is not guaranteed that it will result in the same structure as the structural model.

2. (a) Write a C program which accepts (via user input) two integers `n` and `m`, such that `n` $\leq$ `m`. The program prints (to the display monitor) all numbers between `n` and `m` which are powers of 2, one per line. You must also perform relevant error checking in your program.

(b) Write a C program which accepts two separate strings `S1` and `S2` from a file which contains these strings, one per line. The file is called "`IN.txt`", and is in the same directory as the program. The program then prints the number of letters (which can be chosen from `a-z, A-Z, 0-9`) that are common between `S1` and `S2`, along with a space-separated listing of these letters. For example if `S1` is `jungle` and `S2` is `nuns`, then the result is "`2 n u`" or "`2 u n`" (since the two common letters are `n` and `u`). You must also perform relevant error checking in your program.

Solution:

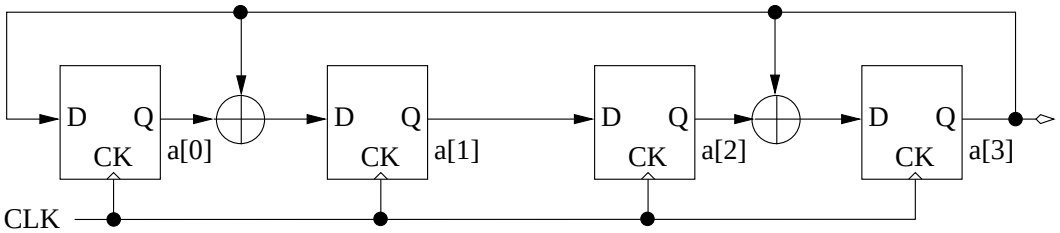
(a)
```
#include <stdio.h>
```

```
int main()
{
int m,n,temp;
printf("Enter n:");
scanf("%d",&n);
printf("Enter m:");
scanf("%d",&m);
if (n > m)         //if n>m swap n,m
{
temp = m;
m = n;
n = temp;
}                 // n is guaranteed to be less than or equal to m
int power = 1;
while(power < n)//set power to the first power of 2 that is greater than or equal to n
power = power * 2;
printf("The numbers are:\n");
while (power <= m)  //print all powers of 2 between n and m (inclusive of n and m)
{
printf("%d\n",power);
power = power * 2;
}
return 0;
}
```

(b)
```
#include <stdio.h>

int main()
{
char S1[100];
char S2[100];
char result[201];     //The result string which will have space separated
                              //common letters in S1 and S2
int numbers[10] = {0};  // Hash tables for each type of characters, 0 initialized
int caps[26] = {0};
int smalls[26] = {0};
int i;
int count = 0; //Count of common characters
FILE * fp;
fp = fopen("IN.txt","r");
if (fp == NULL)
{
printf("Error opening file");
return 1;
}
if (fgets(S1,100,fp) == NULL)
{
printf("Error reading S1");
```

```
return 1;
}
if (fgets(S2,100,fp) == NULL)
{
printf("Error reading S2");
return 1;
}
for (i = 0; i < strlen(S1); i++)
{
if ((S1[i] >= '0') && (S1[i] <= '9')) //Set the value of all found valid characters
numbers[S1[i] - '0'] = 1; //in S1 to 1
if ((S1[i] >= 'a') && (S1[i] <= 'z'))
smalls[S1[i] - 'a'] = 1;
if ((S1[i] >= 'A') && (S1[i] <= 'Z'))
caps[S1[i] - 'A'] = 1;
}
for (i = 0; i < strlen(S2); i++)
{
if ((S2[i] >= '0') && (S2[i] <= '9') && (numbers[S2[i] - '0']==1))
//If the S2 character is valid and was found in S1 only (value = 1)
{
numbers[S2[i] - '0']++; //increment the value in the hash table (to avoid repetition)
result[2*count]=S2[i]; //add the character to the final result
result[2*count+1]=' '; //augment the space
count++;         //increment the number of common characters
}
if ((S2[i] >= 'a') && (S2[i] <= 'z') && (smalls[S2[i] - 'a']==1))
     //same as above for lower case letters
{
smalls[S2[i] - 'a']++;
result[2*count]=S2[i];
result[2*count+1]=' ';
count++;
}

if ((S2[i] >= 'A') && (S2[i] <= 'Z') && (caps[S2[i] - 'A']==1))
//same as above for capital letters
{
caps[S2[i] - 'A']++;
result[2*count]=S2[i];
result[2*count+1]=' ';
count++;
}
}
result[2*count] = 0;           //set the last character of result string to NULL
printf("%d %s",count,result); //print the number of common letters followed by them
return 0;
}
```

3. (a) Consider the circuit shown below. In this circuit, we have four D flip-flops, and two 2-input XOR gates (shown as circles). This circuit is called a Linear Feedback Shift Register (LFSR) and is used as a pseudo-random number generator in VLSI design.



   i. Write a behavioral module in Verilog to perform the functionality of the above circuit. The input to this circuit is `CLK`. The only output of the circuit is `a[3]`. The circuit should start from state where all the signals `a[0]`, `a[1]`, `a[2]` and `a[3]` are 1.

   ii. Simulate the circuit above, and print out the values of `a[3], a[2], a[1], a[0]` over 18 clock cycles, assuming that the circuit is started from the state where all the signals `a[0]`, `a[1]`, `a[2]` and `a[3]` are 1. You do not need to perform event driven simulation, but rather, just present the values of the signals after each clock cycle.

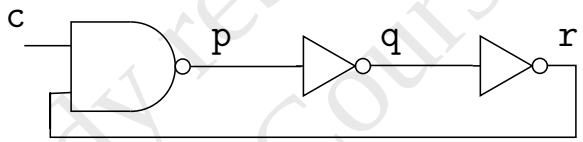
Solution:

(a) i.
```
module LFSR(a[3],clk);
input clk;
output a[3];
reg a[3:0];

initial
begin
a[0] = 1;
a[1] = 1;
a[2] = 1;
a[3] = 1;
end

always @ (posedge clk)
begin
a[0] <= a[3];
a[1] <= a[0] ^ a[3];
a[2] <= a[1];
a[3] <= a[2] ^ a[3];
end
endmodule
```

   ii. Simulation:

Cycle	a[3]	a[2]	a[1]	a[0]
0	1	1	1	1
1	0	1	0	1
2	1	0	1	0
3	1	1	1	1
4	0	1	0	1
5	1	0	1	0
6	1	1	1	1
7	0	1	0	1
8	1	0	1	0
9	1	1	1	1
10	0	1	0	1
11	1	0	1	0
12	1	1	1	1
13	0	1	0	1
14	1	0	1	0
15	1	1	1	1
16	0	1	0	1
17	1	0	1	0
18	1	1	1	1

4. (a) Consider the circuit shown below. We would like to perform event-driven timing simulation for this circuit. Assume that the signal values of (`c`, `p`, `q` and `r`) are `0`, `1`, `0` and `1` respectively, before time 0. At time 0, `c` becomes 1. The delays of the NAND and INV gates are 1 each.



   i. Perform event-driven timing simulation on this circuit, and write down the entries of the stack as simulation proceeds. When an event a generates an event b, draw an arrow between a and b. For each entry of the stack, write down the signal name, time stamp, and the value of the signal before and after the time stamp.

   ii. Based on your answer above, draw the waveform of signal r starting from time 0.

   iii. Based on your answer above, what is the functionality of the circuit?

   (b) Consider an adder A, which adds a 64-bit number `a` and another 64-bit number `b` to create a result `c`. The adder takes 11ns to perform its computation. The signals `a` and `b` are driven from a series of D flip-flops which are clocked using a 500Mhz clock. At the rising edge of this clock, we drive fresh values of `a` and `b`. The flip-flops have a setup time $T_s = 0.0ns$, a hold time $T_h = 0.0ns$, and a clock-to-output delay $T_{co} = 0.0ns$.

   i. How many bits wide is the number `c`?

   ii. Why will the above adder not operate correctly at a 500MHz clock rate?

   iii. I would like to make the adder operate at a 500MHz clock rate using pipelining. How many pipeline stages will I need? Why?

   iv. Using the number of pipeline stages from the previous part, what is the fastest clock rate that the pipelined circuit can be operated at?

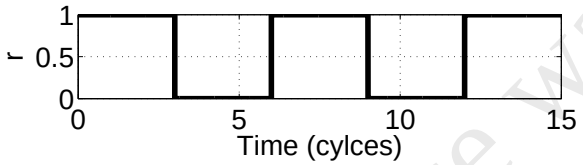
Solution:

(a) i. Simulation:

Time	Signal	Value
0	c	$0 \rightarrow 1$
1	p	$1 \rightarrow 0$
2	q	$0 \rightarrow 1$
3	r	$1 \rightarrow 0$
4	p	$0 \rightarrow 1$
5	q	$1 \rightarrow 0$
6	r	$0 \rightarrow 1$
7	p	$1 \rightarrow 0$

And so on.

ii. Waveform:



iii. A clock generator, with a time period = 2× total gates delay (3) = 6 time units, and an enable signal (c). The clock oscillates only when (c=1).

(b) i. 65 bits. You will need 1 extra bit to account for the case when you add the two maximum values of a and b (to avoid overflow error).

ii. f = 500 MHz → Time period = 2 ns.
New values of a and b arrive every 2 ns, but the adder takes 11 ns to finish each operation. Therefore, the adder will not operate correctly.
For the circuit to operate correctly: $(T_{period} > T_{CombinationalLogicDelay} + T_{setup} + T_{CO})$

iii. You will need to make the delay of each pipeline stage $\leq$ 2 ns. Thus, you need 6 pipeline stages.

iv. If we can make the pipeline stages perfectly balanced:
Delay of each stage $= \frac{11}{6} = 1.833ns$
Max Frequency $= \frac{1}{Delayofeachstage} = 545.45MHz$