Problem 1 (15 points): The structure of an unsigned parallel multiplier is based on the observation that partial products in the multiplication process can be computed in parallel. For example we can consider the following unsigned binary integers:

$$X = \sum_{i=0}^{m-1} x_i 2^i \quad ; MULTIPLICAND$$
$$Y = \sum_{j=0}^{n-1} y_j 2^i \quad ; MULTIPLIER$$

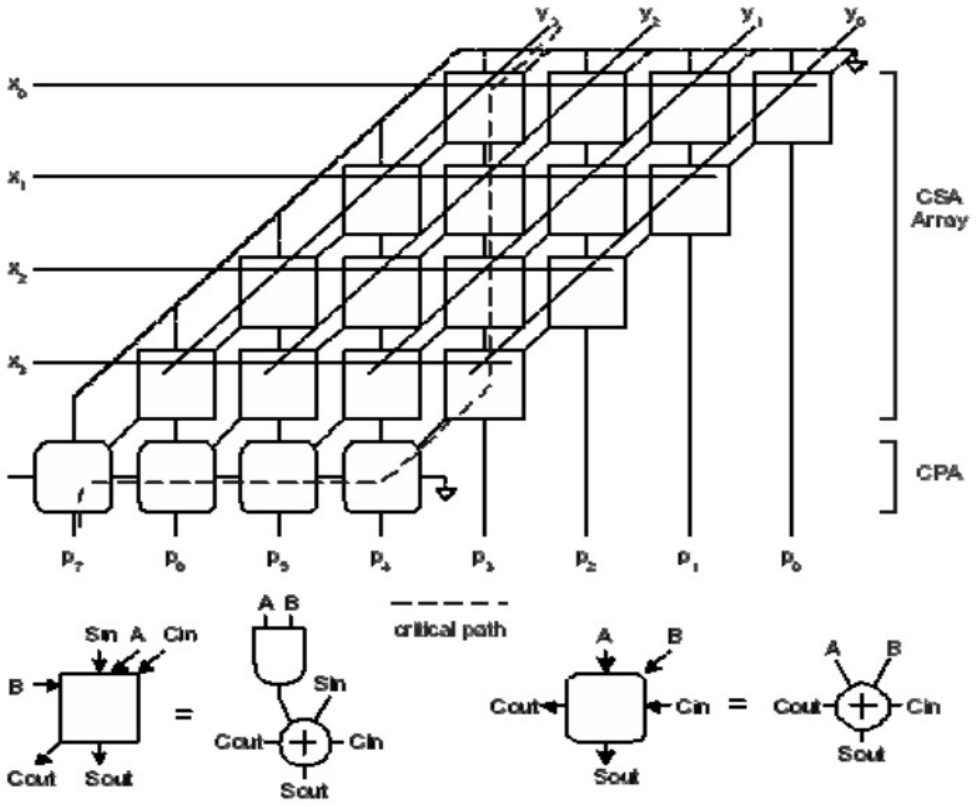
This product is found by:

$$P_r = X_r Y_r = (\sum_{i=0}^{m-1} x_i 2^i)(\sum_{j=0}^{n-1} y_j 2^j) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (x_i y_j) 2^{i+j} = \sum_{k=0}^{m+n-1} P_k 2^k$$

For a 4 by 4 multiplier the expression can be expanded as follows:

				X3 Y3	X2 Y2	X1 Y1	X0 Y0	MULTIPLICAND MULTIPLIER
				X3Y0	X2Y0	X1Y0	X0Y0	
			X3Y1	X2Y1	X1Y1	X0Y1		
		X3Y2	X2Y2	X1Y2	X0Y2			
	X3Y3	X2Y3	X1Y3	X0Y3				
P7	P6	P5	P4	P3	P2	P1	P0	PRODUCT

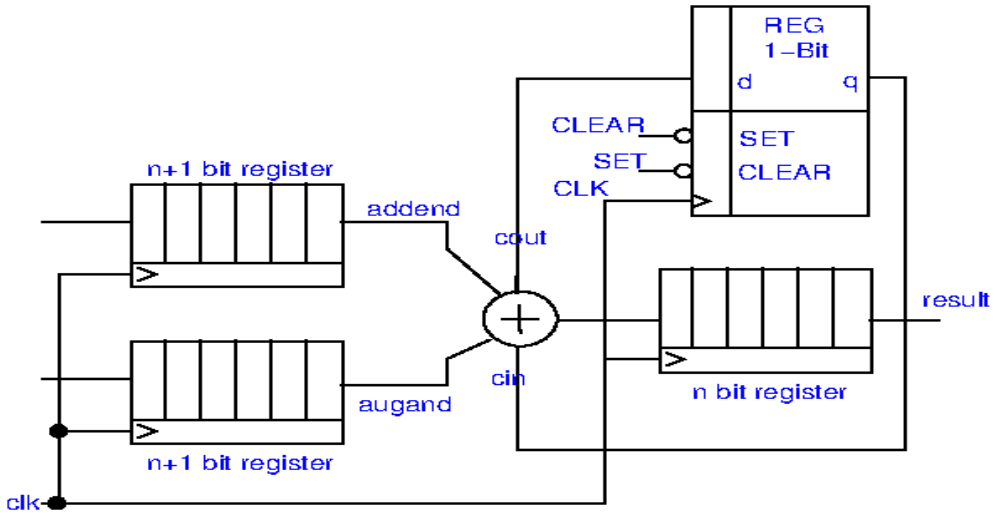
The basic structure is presented in the diagram below:



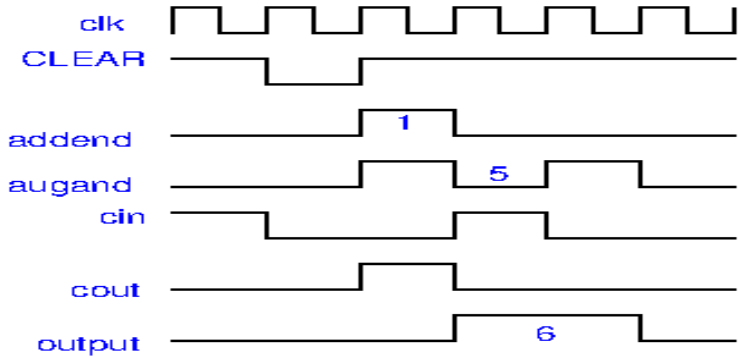
a) Write a VHDL model to describe this basic structure.
b) Use ModelSim to compile your multiplier model.
c) Use ModelSim to simulate your multiplier with the following cases: (2*4, 15*3)

Problem 2(20 points): Bit-serial architectures have been used successfully for a variety of signal-processing applications. Figure 4(a) shows a bit-serial adder. This uses a single adder and constructs the SUM sequentially. At time t, the SUM is calculated and the CARRY stored in a register. At time t+1, the sum uses CARRY[t] to calculate a new SUM.

The two inputs to the adder are stored in n-bit registers. The SUM output is stored in an n-bit result register. An illustrative add cycle is shown in Figure 4(b). Addition is commenced by clearing the carry register. Then the operands are serially applied to the inputs of the adder, the least significant bit first. The example shows 1 added to 5 to form 6 at the output. It takes n clock cycles to complete an n-bit add. In a serial adder, equal SUM and CARRY delays are advantageous, because these delays determine the fastest clock frequency at which the adder can operate.



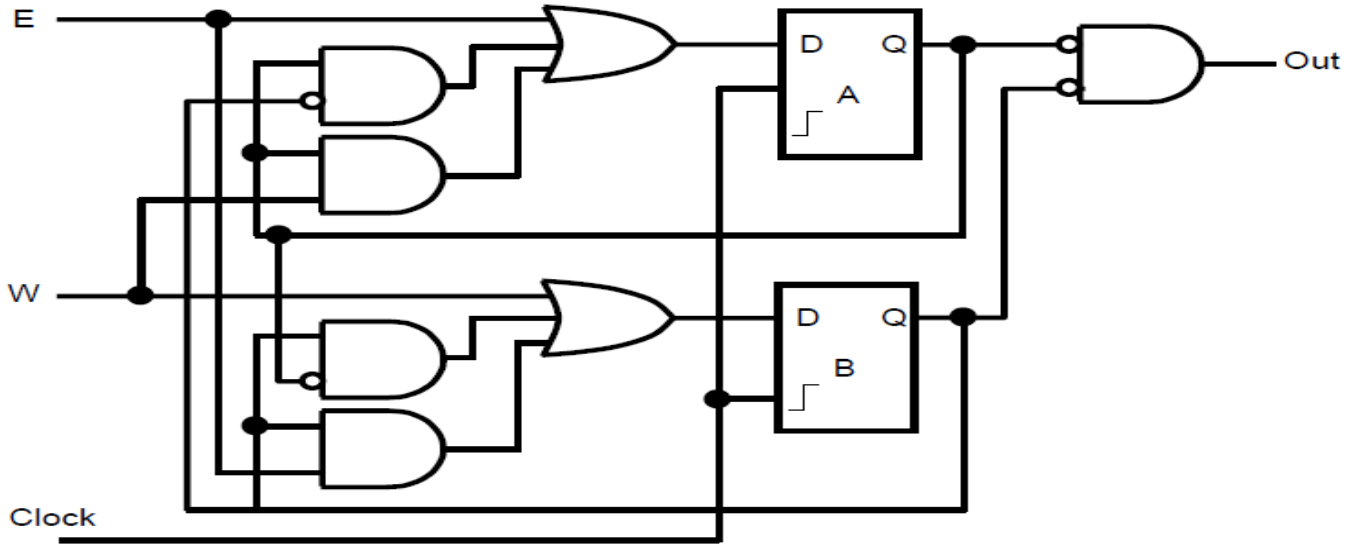
(a) Circuit design



(b) Adding cycle

a) Write a VHDL file to model the above behavior for 8 bits adder.
b) Use ModelSim to compile your VHDL file.
c) Use ModelSim to simulate your adder circuit with the following cases: 7+3, 6+4.

Problem 3(15 points): Analyze the logic circuit given below.



Derive a state diagram for the circuit.

a) Write a structural VHDL Model for this design and simulate your circuit to verify correctness. (Explain your approach)
b) Write a behavioral VHDL model based on the derived state-diagram and simulate to verify correctness.
c) Using ModelSim, can you verify that the model in a) and c) are equivalent (Explain).

Problem 4 (30 points): You are asked to implement a processor with ten instructions**: load, store, add, complement, xor, shift, rotate, nop, halt, and branch**. The instruction set and format for this processor are shown below.

Instructions: **ADD R1,R2** means R1 = R1 + R2 and **CMP R1, R2** means R1 = ~(R2)

Name	Mnemonic	Opcode	format(Instdst,Src)	
NOP	NOP	0	NOP	
LOAD	LD	1	LD reg, mem1	set PSR,PSR[0]=0
STORE	STR	2	STR mem, src	clear
BRANCH	BRA	3	BRA mem, CC	
XOR	XOR	4	XOR reg, src	set PSR,PSR[0]=0
ADD	ADD	5	ADD reg, src	set PSR
ROTATE	ROT	6	ROT reg, cnt	set PSR
SHIFT	SHF	7	SHF reg, cnt	set PSR
HALT	HLT	8	HLT	
COMPLEMEN	CMP	9	CMP reg, src	set PSR,PSR[0]=0

Condition Code (CC):		
Name	Means	code
A	Always	0
P	Parity	1
E	Even	2
C	carry	3
N	Negative	4
Z	Zero	5
NC	No carry	6
PO	Positive	7

Operand Addressing	
Mem	Memory address
mcm1	Memory address or immediate value
Reg	Any register index
Src	Any register index, or immediate value
CC	condition code
Cnt	Shift/Rotate cnt, cnt > 0 means right. < 0 means left.+/-16

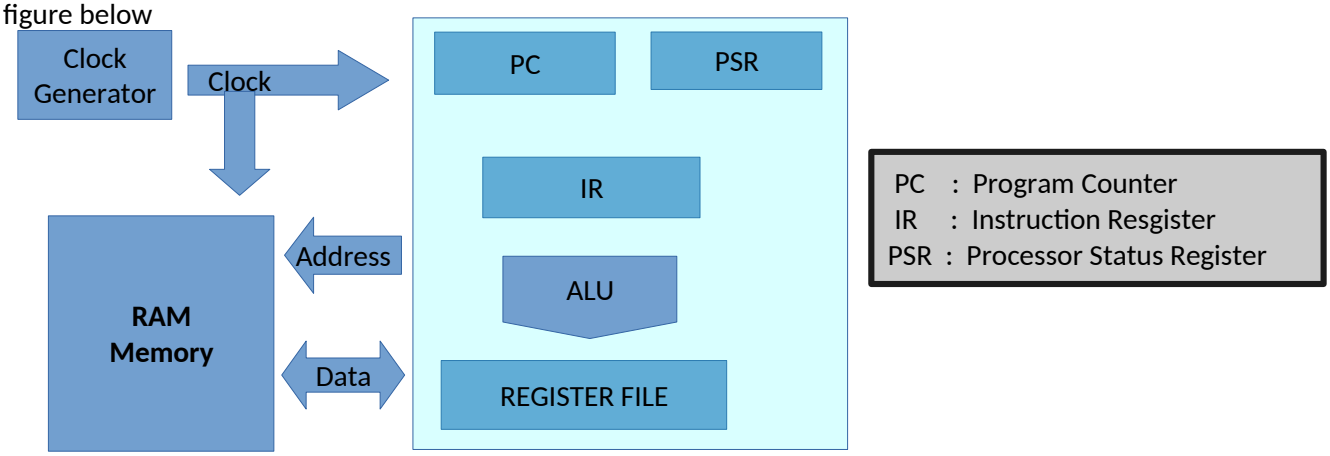
Instruction Register (IR) Format:	
IR[31:28]	Opcode
IR[27:24]	CC
IR[27]	source type
IR[26]	destination type 0=reg,
IR[23:12]	Source address
IR[23:12]	shift/rotate count
IR[11:0]	destination address

Processor Status Register (PSR)	
PSR[0]	Carry
PSR[1]	Parity
PSR[2]	Even
{SR[3]	Negative
PSR[4]	Zero

Implementation of the corresponding hardware to execute these instructions can be described at the behavioral level. For example, an add instruction may simply be modeled as:

```
assign {carry,sum} = A + B;
```

Without going into detail of whether the addition be performed using a ripple-carry adder or a carry-look-ahead adder. Similarly, we treat memory as a large set of registers directly visible to the processor. The processor structure is shown in the figure below



Write a Verilog model for this processor (Assume that the width of the data-path is 32 bits, the size of the address field is 12 bits, and the register file contains 16 registers).

To verify your model write a program to read a positive number 'N' stored at memory location zero and compute the two's complement representation of '-N'. The number -N in two's complement is to be stored in memory location one. Test your code for N=6.

Problem 5 (10points): To verify your model write a program to count the number of 1's in memory location zero. The number of ones is to be stored in memory location one.

Problem 6 (10points): Write a code to multiply two 4 bit numbers A and B and store the results in C. A, B, and C are stored in memory location 0, 1 and 2 respectively.