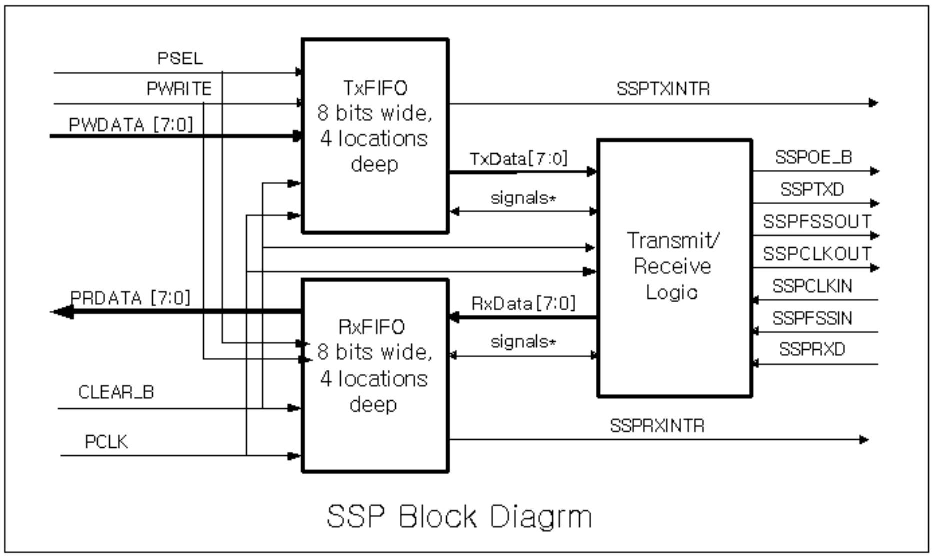
Problem 1 (35 points): Synchronous Serial Port Specifications: Now that you are familiar with the tools and reference materials, let's try to understand the specifications of the modules and start designing a system.

Overview: What is the SSP module? You can consider it to be a synchronous version of the serial port you can find in your desktop computer. A word of (data or instruction) is stored at each address of your computer's memory. When this data is transmitted to outside of your computer through a serial port (for example, a modem which is connected to your serial port), each bit of the word is transmitted sequentially. When there is incoming data through the serial port (for example, data from the modem), each bit of the incoming data is received sequentially by the serial port, and an entire word has been read in, your computer stores it at a memory location. These operations can be performed synchronously or asynchronously; your task is to design a synchronously operating serial port.

For simplicity, our SSP module operates on 8-bit words. In essence, the SSP needs to perform parallel-to-serial conversion on data received from the processor, and serial-to-parallel conversion on data received from a peripheral device. Your design should provide buffering capability on both transmit and receive logic; specifically, use FIFOs to allow up to 4 8-bit words to be stored independently in both TX and RX modes. An example SSP block diagram is shown below.



Unnamed signals or signals denoted by signals* in this diagram are internal signals that you can define as you need. Your SSP module must have a transmit FIFO, a receive FIFO, and transmit/receive logic. Your SSP module can have more modules and signals other than these if you need; you are not to introduce more input or output ports. Now we will discuss each module of the SSP:

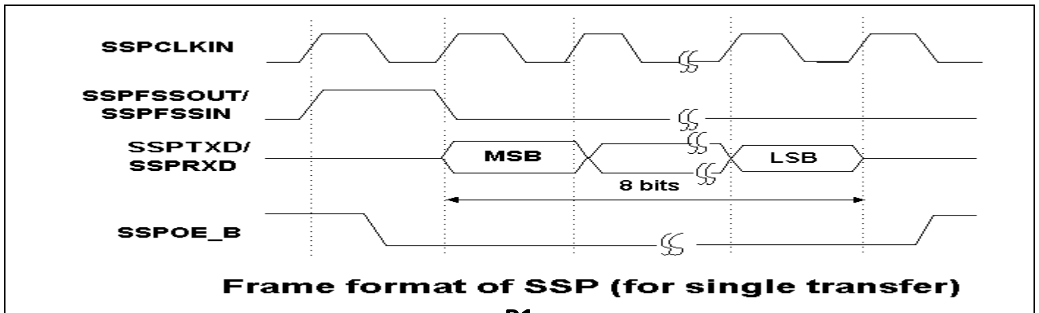
Transmit FIFO (TxFIFO): The transmit FIFO is an 8-bit wide, 4-location deep, first-in-first-out memory buffer. Data written to SSP module will be stored in the buffer until it is read out by the transmit logic. If this FIFO is full, it should pull the SSPTXINTR interrupt signal high, and it should not accept any additional data while it is full. When the FIFO is no longer full, it should lower the SSPTXINTR signal. For simplicity we do not consider the case of a read request when the FIFO is empty. Data written to the FIFO should be transferred to transfer logic in as few cycles as possible so that the transfer rate of the data should be maximized.

- **FIFO (RxFIFO):** The receive FIFO is an 8-bit wide, 4-location deep, first-in-first-out memory buffer. Receive data from the serial interface are stored in the buffer until read out by the processor. If this FIFO is full, it should generate the SSPRXINTR interrupt signal, and refuse to accept any additional data. The same comments for the TxFIFO hold here.
- **Transmit Logic:** The transmit logic successively reads words from the transmit FIFO and performs parallel to serial conversion on the word; it then sends the serial data stream and frame control signals, synchronized to SSPCLKOUT, through the SSPTXD pin and SSPFSSOUT pin.
- **Receive Logic:** The SSPCLKIN clock is provided by an attached peripheral and used to synchronize its reception sequences. Receive logic performs serial to parallel conversion on the incoming synchronous SSPRXD data stream, extracting and storing values into the receive FIFO, to be read subsequently by the processor.

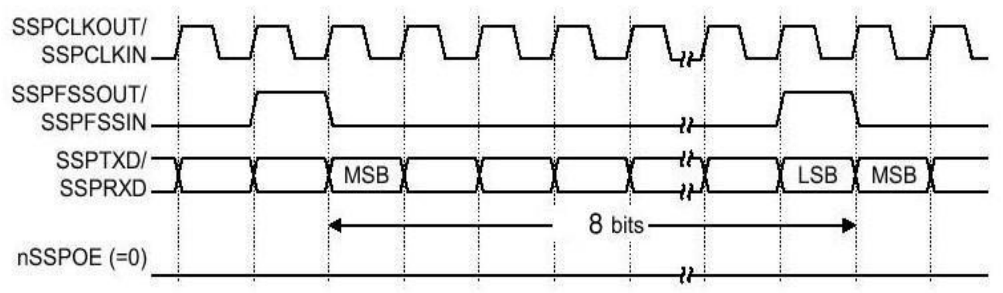
Port description: Your SSP module should follow input/output ports specifications. In other words, you should use exactly the names shown below. Also, name your top module as 'ssp'.

input ports: PCLK, CLEAR_B, PSEL, PWRITE, PWDATA[7:0], SSPCLKIN, SSPFSSIN, SSPRXD
output ports: PRDATA[7:0], SSPOE_B, SSPTXD, SSPCLKOUT, SSPFSSOUT, \\SSPTXINTR, SSPRXINTR

P1 Shows the frame format timing diagram for single transfer mode and P2 shows the frame format timing diagram for continuous transfer mode of SSP. Data entered consecutively into SSP should be transmitted back to back as shown below.



Frame format of SSP (for single transfer)
P1



P2

- ❖ **PCLK :** clock for SSP. All the operations of the interface block and FIFOs should be synchronized to this signal.
- ❖ **CLEAR_B**: Low active clear signal. Use it to initialize your SSP.
- ❖ **SSPCLKIN, SSPCLKOUT:** Synchronization clock for reception and transmission of data respectively. Because we will loop back the transmitted data to SSPRXD, SSPCLKOUT need to be connected to SSPCLKIN. Make SSPCLKOUT twice slower than PCLK.
- ❖ **PSEL:** Chip select signal. (Indicating this SSP is selected for data transfer.) Whenever the SSP is accessed, this signal should be set to 1. In other words, data can enter into (or get out of) the SSP module only when this signal is 1. This applies ONLY to data being read and written to the SSP from and into the FIFOs through the PWDATA and PRDATA lines. All data already in the FIFO should be sent and anything being received should be processed.
- ❖ **PWRITE:** Read/Write signal. If it is 1, it is write (to SSP) signal, and if it is 0, it is read (out of SSP) signal.
- ❖ **PWDATA:** 8-bit data which should be transmitted.
- ❖ **PRDATA:** 8-bit data which was received.
- ❖ **SSPFSSOUT:** A frame control signal for transmission. Once the bottom entry of the transmit FIFO contains data, SSPFSSOUT is pulsed HIGH for one SSPCLKOUT period. The value to be transmitted is also transferred from the transmit FIFO to the serial shift register of the transmit logic. On the next rising edge of SSPCLKOUT, the MSB of the 8-bit data frame is shifted out on the SSPTXD pin.
- ❖ **SSPFSSIN:** A frame control signal for reception. Serial data start to be received at the next rising edge of SSPCLKIN after this signal is asserted. No data should be received without reception of this signal.
- ❖ **SSPTXD:** 1-bit serial data output.
- ❖ **SSPRXD:** 1-bit serial data input.
- ❖ **SSPOE_B:** Low active output enable signal. Make it low from the negative edge of the SSPCLKOUT just before the valid data transmission to the negative edge of the clocks just after the valid data transmission.

Because we don't have any other peripheral to send serial data to the SSP module, we will loop back the serial data from its transmission pin to its reception pin. This direct connection will be made at test bench module. Do not connect them within SSP module. You can refer **ssptest.v** (from the class web site) file to see this connection.

So the data flow of the whole system will be:
(Written by the processor)→SSP transmit FIFO→SSPTXD→SSPRXD →SSP receive FIFO → (Read by the processor).

EECS 318 ***Due: Oct 8***

Problem 2 (35 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 0=reg(mem),1=imm
IR[26]	destination type 0=reg, 1=imm
IR[23:12]	Source address
IR[23:12]	shift/rotate count
IR[11:0]	destination address

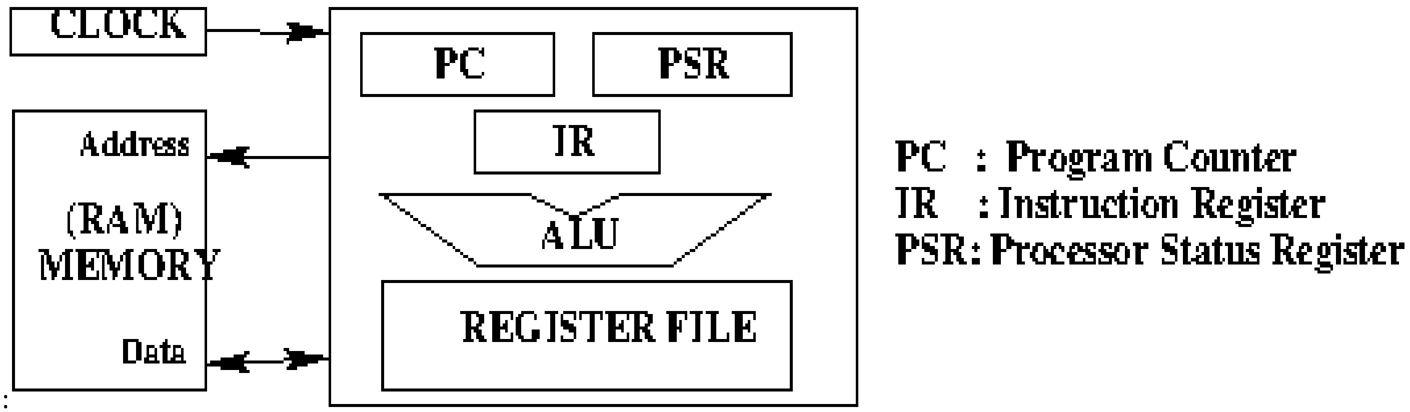
Processor Status Register (PSR)

PSR[0]	Carry
PSR[1]	Parity
PSR[2]	Even
PSR[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 3 (15points): 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 4 (15points): 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.