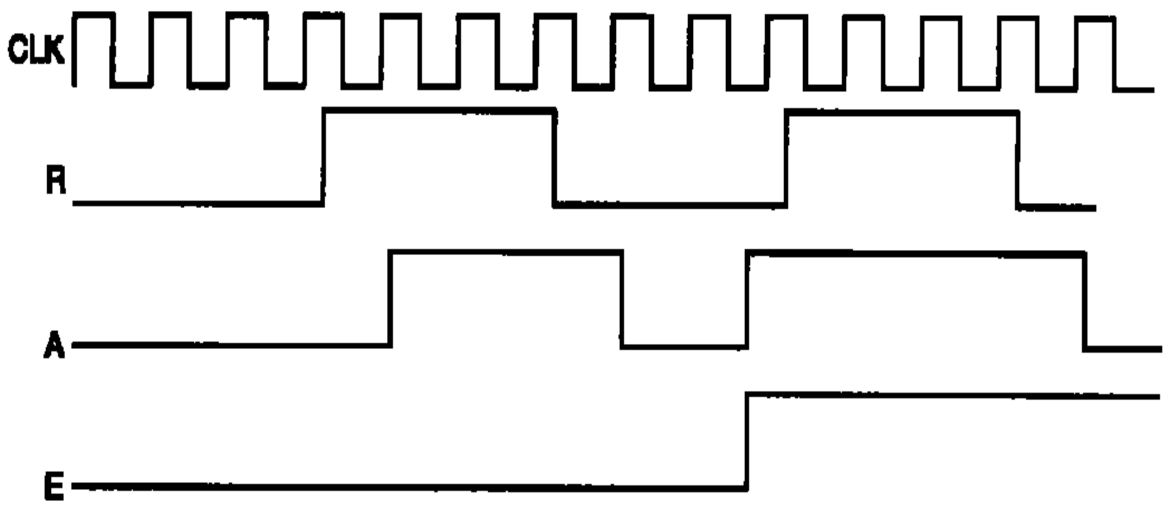
Problem 1(20 points): A pair of signals Request (R) and Acknowledge(A) are used to coordinate transactions between a CPU and its I/O system. The interaction of these signals is often referred to as a "handshake." These signals are synchronous with the clock and, for a transaction, are to always have their transitions appear in the order shown in below. A handshake checker is to be designed that will verify the transition order. The checker has inputs, R and A, asynchronous reset signal, RESET, and has output, Error(E). If the transitions in a handshake are in order, E = 0. If the transitions are out of order, then E becomes 1 and remains at 1 until the an asynchronous reset signal (RESET = 1) is applied to the CPU.



a) Find the state diagram for the handshake checker.
b) Find the state table for the handshake checker.
c) Write VHDL structural model for the above design. Simulate using ModelSim to verify the correctness of your design (explain).
d) Write VHDL behavioral model based on the described function. Simulate using ModelSim to verify the correctness and compare with the model derived in part (c).

1. Typical Cache Memory

To illustrate the concept of cache memory, we assume a very small cache of eight 32-bit words and a small main memory with 1 KB (256 words), as shown in Figure 1. Both of these are too small to be realistic, but their size makes illustration of the concepts easier. The cache address contains 3 bits, the memory address 10. Out of the 256 words in main memory, only 8 at a time may lie in the cache.

In order for the CPU to address a word in the cache, there must be information in the cache to identify the address of the word in main memory. If we consider the example of for loop, clearly, we find it desirable to contain the entire loop data required within the cache, so that all of the instructions can be fetched from the cache while the program is executing most of the passes through the loop. The instructions in the loop lie in consecutive word addresses. Thus, it is desirable for the cache to have words from consecutive addresses in main memory present simultaneously. A simple way to facilitate this feature is to make bits 2 through 4 of the main memory address be the cache address. We refer to these bits as the index, as shown in Figure 1. Note that the data from address 0000001100 in main memory must be stored in cache address 011. The upper 5 bits of the main memory address, called the tag, are stored in the cache along with the data. Continuing the example, we find that for main memory address 0000001100, the tag is 00000. The tag combined with the index (or cache address) and byte field identify an address in main memory.

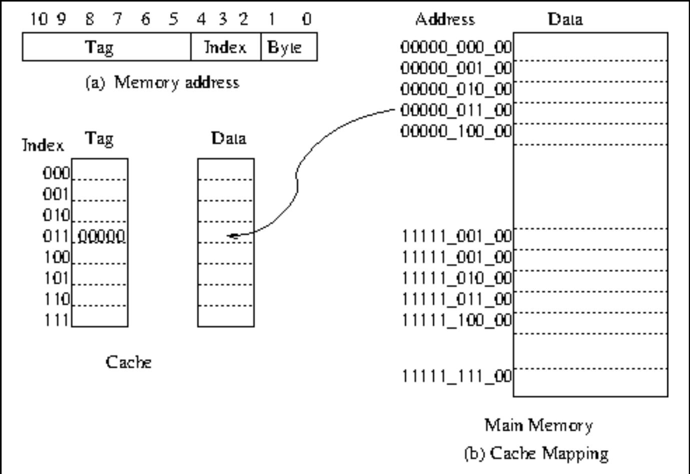


Figure 1: Direct Mapped Cache

Suppose that the CPU is to fetch an instruction from location 000001100 in main memory. This instruction may actually come from either the cache or main memory. The cache separates the tag 00000 from the cache address 011, internally fetches the tag and the stored word from location 011 in the cache memory, and compares the tag fetched with the tag portion of the address from the CPU. If the tag fetched is 00000, then the tags match, and the stored word fetched from cache memory is the desired instruction. Thus, the cache control places this word on the bus to the CPU, completing the fetch operation. This case in which the memory word is fetched from cache is called a cache hit. If the tag fetched from cache memory is not 00000, then there is a tag mismatch, and the cache control notifies main memory that it must provide the memory word, which is not available in the cache. This situation is called a cache miss. For a cache to be effective, the slower fetches from main memory must be avoided as much as possible, making considerably more cache hits than cache misses necessary.
When a cache miss occurs on a fetch, the word from main memory is not placed just on the bus for the CPU. The cache also captures the word and its tag and stores them for future access. In our example, the tag 00000 and the word from memory will be written in cache location 011 in anticipation of future accesses to the same memory address.

2. Interfaces

A cache system typically lies between the processor and the main system bus. The following block diagram Figure 2 describes the signals and buses that the cache needs to communicate with the processor and the system.

Note that in our system the system bus is synonymous with main memory.

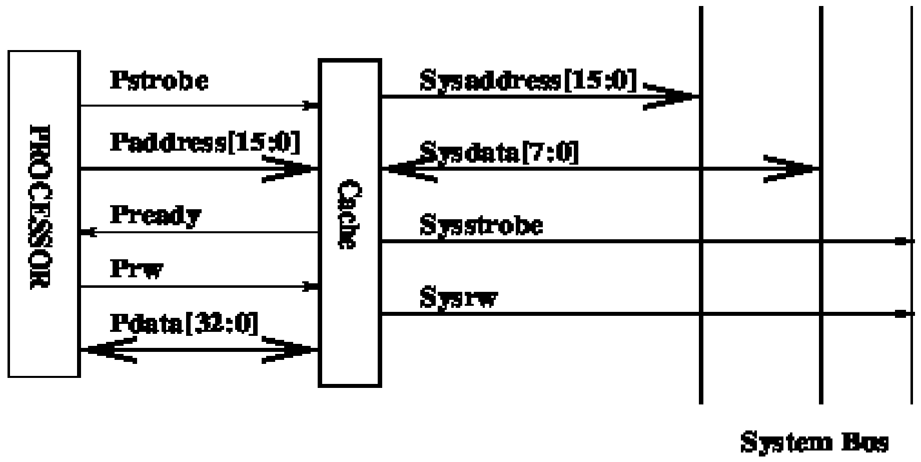


Figure 2: System Interface

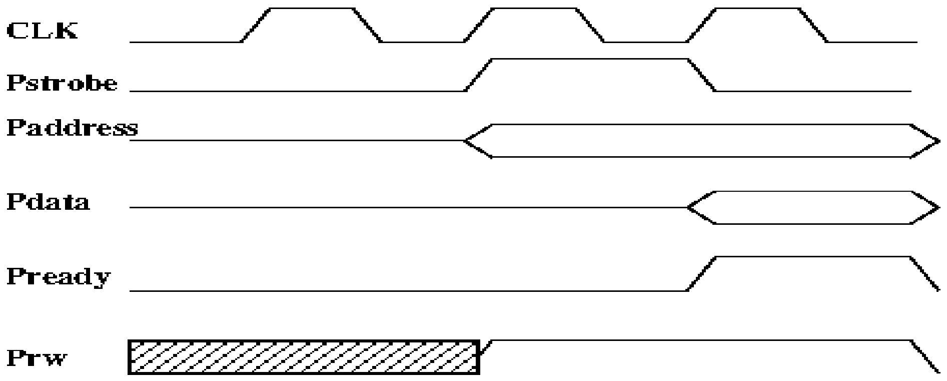


Figure 3: Processor Interface

2.1 Processor Interface

The processor interface consists of the processor address bus, Paddresst[15:0], the processor data bus, Pdata[32:0], and control signals Pstrobe and Pready. The Pstrobe is asserted when the processor is starting a bus transaction and a valid address is on the Paddress bus. Pready is used to signal to the processor that the bus transaction is completed. The timing diagram in Figure 3 demonstrates a simple read cycle. The Prw signal is high for a read and low for a write.

2.2 System Bus Interface

For our cache model we assume a simple bus model. For a read operation, the Sysaddress is first presented to the bus along with the Sysstrobe signal and the Sysrw. The Sysrw signal is high for read operations and low for write operations. After a set number of wait states (in our case it is four clock cycles), the data is returned. A write operation is similar, but the data is driven onto the Pdata bus immediately and then waits for the set number of wait states (in our case it is four clock cycles) before issuing another write operation. Figure 4 shows a system read with one wait state.

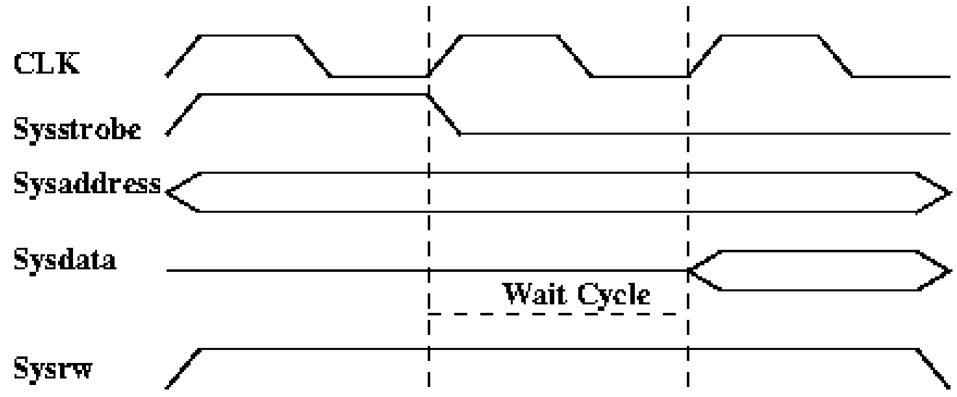


Figure 4: System Interface

2.3 Cache Architecture

The direct-mapped cache is the simplest of all cache architectures. A direct mapped cache consists of a single tag RAM, cache RAM, and a simple controller. You have to model each of these parts separately and then bring them together in the final model.

Problem 2(40 points): Assume that the processor has a 16-bit address composed of the following fields:

1. The byte field (bits [1:0]),
2. The index field (bits [9:2]),
3. The tag filed (bits [10:15]).

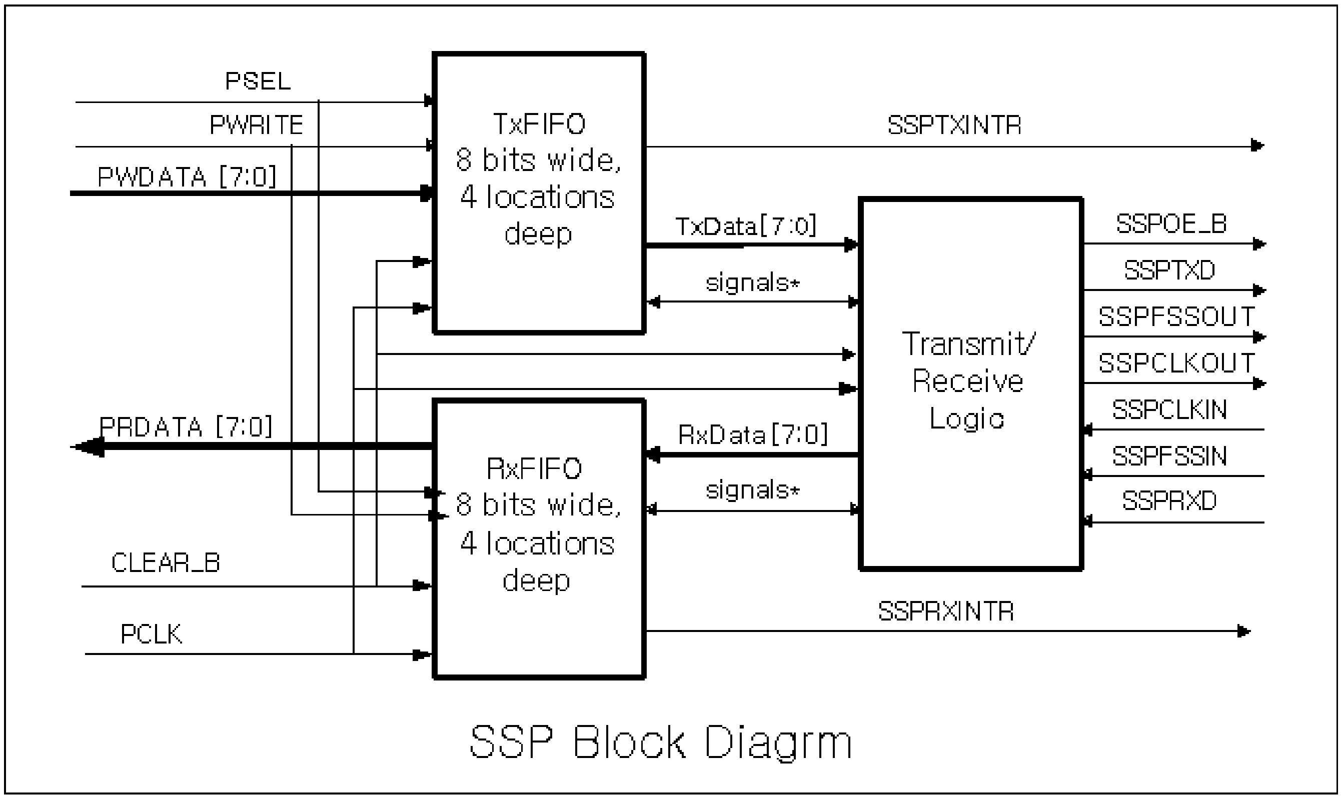
To simplify the cache, all writes from the processor update both the cache and main memory. This is called write-through mode operation. In write-through mode, the cache memory is always kept coherent with main memory.

Assume the following that a bus should be tristate at the end of a bus transaction, memory write and cache write can be performed in parallel, a signal assignment incurs one unit delay, and the clock period is 100 units.

Write a VHDL model for the above system. To test your system, simulate a processor request of the following sequence of Hex addresses: '12','15','12','55','22','75','66','75','59','22'.

Problem 3(40 points): Design a Verilog Synchronous Serial Port (SSP) circuit from the specification below. 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 the transmit/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 in P3.1. 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:

1. **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

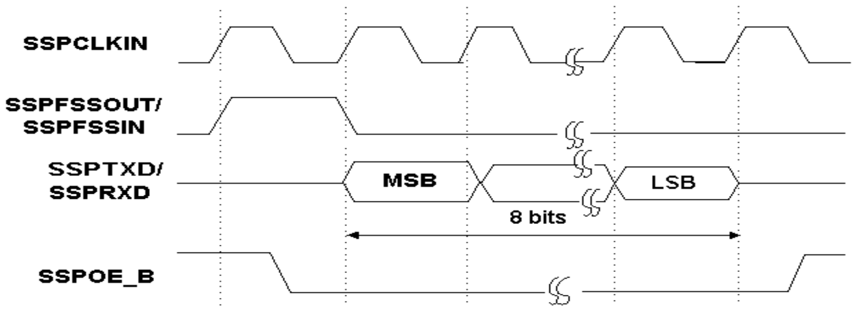
transferred to transfer logic in as few cycles as possible so that the transfer rate of the data should be maximized.

2. **Receive 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.
3. **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.
4. **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'.

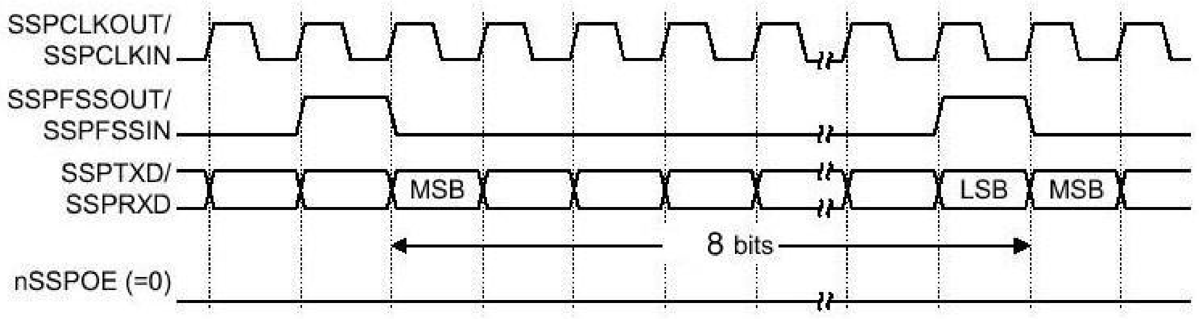
1. input ports: PCLK, CLEAR_B, PSEL, PWRITE, PWDATA[7:0], SSPCLKIN, SSPFSSIN, SSPRXD
2. output ports: PRDATA[7:0], SSPOE_B, SSPTXD, SSPCLKOUT, SSPFSSOUT, \\SSPTXINTR, SSPRXINTR

Here is the frame format timing diagram for single transfer mode of SSP.



Frame format of SSP (for single transfer)

Here is 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.



1. PCLK: clock for SSP. All the operations of the interface block and FIFOs should be synchronized to this signal.
2. CLEAR_B: Low active clear signal. Use it to initialize your SSP.
3. 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.
4. 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 logic 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.
5. 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.
6. PWDATA: 8-bit data which should be transmitted.
7. PRDATA: 8-bit data which was received.
8. 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.
9. 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.
10. SSPTXD: 1-bit serial data output.
11. SSPRXD: 1-bit serial data input.
12. 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).