1. The following dataflow graph shows the computation of a 2-level, 1-D wavelet transform. This computation transforms an input block of 4 values into an output block of 4 values. Design a circuit that performs this computation. The circuit should read the input values one at a time in the order D0, D1, D2, D3 and output the results one at a time in the order Q0, Q1, Q2, Q3. That is, your circuit should read one input value per clock cycle and produce one output value per clock cycle. The inputs and outputs are long (infinite) streams of values, but are processed in blocks of 4.

![Dataflow Graph]

The interface to your circuit is shown below:

![Interface Diagram]

The D values are ready starting with the first cycle that reset is 0. Since the output values are delayed with respect to the input data stream by some number of clock cycles, the ready output signal is used to indicate valid output data. This signal will be asserted continuously once outputs start being produced.

Note: There are 4 clock cycles to process each block of 4 inputs and outputs. Since there are 6 arithmetic operations to perform, at most two need to be performed each clock cycle. That is, it would be a waste to use more than two arithmetic units, which can be reused on each of the four clock cycles.

First design the datapath to perform this computation. There will be some number of controls required to share the arithmetic units and to load registers at the right time.

Next design the finite-state machine that provides these controls. Is a Mealy or Moore machine more appropriate here? Explain why. Go through the synthesis steps required to implement this FSM: Use a 1-hot state encoding and generate the next state equations and output equations.
2. In this problem, you will design a simple cache controller for a direct-mapped, read-allocate, write-through cache. The controller sits between the processor and the cache memory, as shown in the figure below, and generates the control signals needed by the processor and the memory. Note that you only have to implement the cache controller. The cache controller has the following I/O signals:

Inputs:
- Read - Processor asserts to request a read.
- Write - Processor asserts to request a write. (Only one of Read/Write can be asserted at once)
- Address - Processor provides the memory address of word being read or written.
- Hit - The cache tag memory asserts Hit if the memory address is in the cache.
- Reset - Resets the cache controller into an initial state.

Outputs:
- Ready - Signal to the processor indicating data is present and processor can continue.
- MemRead - Asserted to tell memory to perform a read operation
- MemWrite - Asserted to tell memory to perform a write operation. Only one of MemRead and MemWrite can be asserted at a time.
- MemAddress - Address to main memory of word being read/ written.

Addresses are word addresses with 4 words per cache line. On each cycle, the processor asserts a read or write request (or neither) to the cache controller along with the memory address of a word. It then waits for the Ready signal to be asserted. A cache tag module checks to see if the address is in the cache. If so, it asserts Hit in the next cycle after the Read/Write request is asserted. (You do not have to design the cache tag module - assume it's there and the Hit signal is asserted appropriately.)

Different things happen depending on whether a Read or Write has been asserted:

**Read** - If Hit is asserted, then Ready should be asserted immediately (same clock cycle as Hit). If Hit is not asserted, the controller must read the cache line from memory, starting with the requested word. From the time MemRead is asserted, it takes 4 cycles before the first word from memory is available. After that, it takes only one cycle per word read from the same cache line. That is, the controller issues MemRead with the address of the first word to be accessed, waits 4 cycles and then issues 3 more reads, one per cycle with the addresses of the remaining words in the cache line. The controller can assert the Ready signal as soon as the data from the requested address is available.
**Write** - Writes are easier since cache lines are not allocated on a write miss. Whether or not Hit is asserted, the controller issues a write to memory for one word. Ready can be asserted immediately, however, without waiting for the write to complete, which takes 4 cycles.

**Read/Write Overlap** - Reads and writes that go to memory cannot overlap. That is, if there are two read misses in a row, the second set of reads to memory must wait until the previous reads have completed. This can happen because the controller asserts Ready before completing all the reads from the first miss. If a read miss or a write is followed by a read hit, however, Ready can be asserted right away since no memory access is required.

Use a combination of schematics and Verilog to design this circuit: schematic to show the datapath and how the modules are connected, and Verilog to describe the state machines. You should think how you might be able to break the design into more than one component to simplify the design.