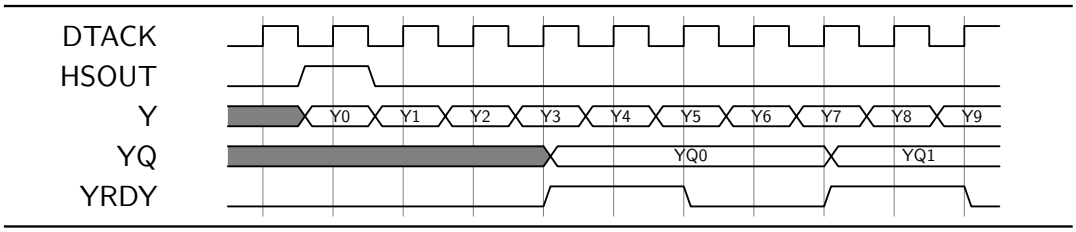
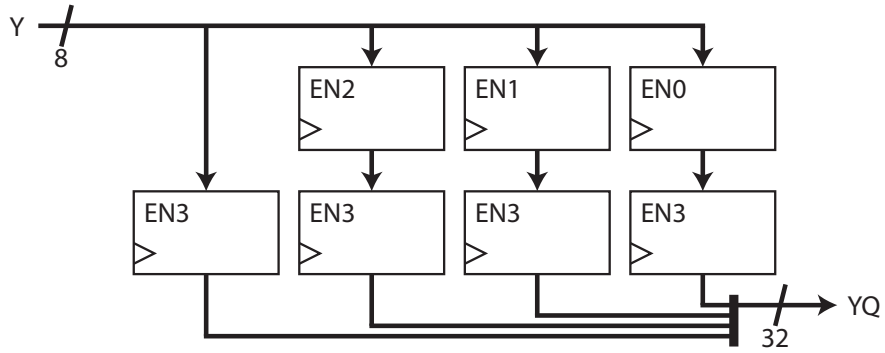
1.1. **Synchronization pt 1 (10 pts).** The AD9980 video decoder on the XUP board sends luminance information for each pixel Y[7:0], and timing information HSOUT and VSOUT using a generated data clock DTACK at 49.5 MHz. For 800x600 at 75 Hz, the nominal DTACK rate is 49.5 MHz. Using the 49.5 MHz clock, 4 consecutive pixels are packed into a 32 bit word YQ[31:0]. This results in YQ(N) = {Y(4N+3),Y(4N+2),Y(4N+1),Y(4N)}.

A FSM (using DTACK as the clock) is used to generate a YRDY signal which lasts for 2 DTACK clock cycles when a new YQ value is ready. YQ should change at 12.375 MHz. For now assume that HSOUT is asserted for the first pixel of each row, and should be used to align the input data. The timing diagram below details this operation.

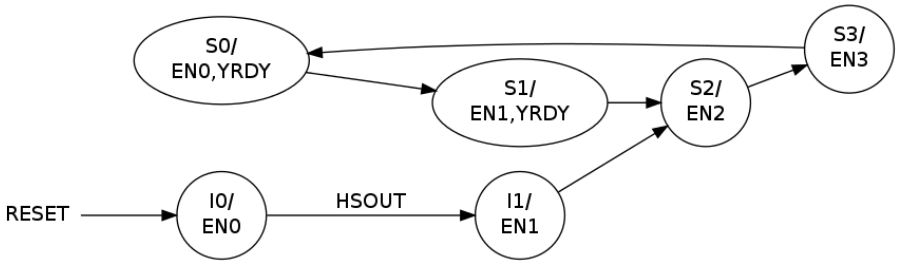


a. Draw a detailed block diagram (to level of registers, muxes, etc.) for the data path which generates YQ from Y.



All registers are clocked on DTACK.

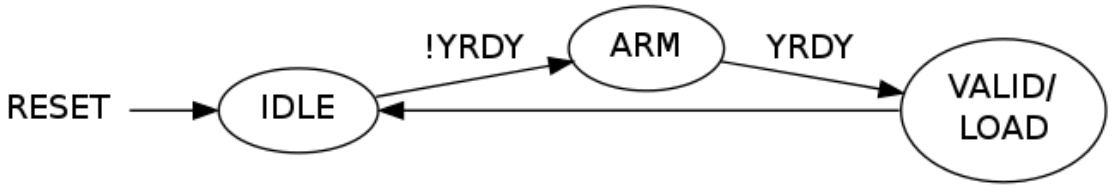
b. Design a simple FSM which generates a glitch free YQ, and glitch free YRDY and controls the data path for the Y to YQ conversion.



YRDY will be glitch-free if the encoding of this FSM is the output of a single FF. State = $Q_2Q_1Q_0$ = {I0, I1, S0, S1, S2, S3} = {000, 001, 010, 011, 100, 100} with YRDY = Q_1 is an encoding that accomplishes this.

1.2. **Synchronization pt 2 (15 pts).** The DTACK clock, YQ, YRDY are asynchronous with respect to the main clock FPGAClk of 100 MHz, which runs the FeatureDetector block. Consider passing YQ to the FeatureDetector, with control signal YRDY. For every new 32 bit word YQ from the AD9980, a control FSM in the FeatureDetector block should load the quad pixel into the image processing pipeline. (You can assume a 32 bit input register with Load enable, clocked at 100 MHz.).

a. How could the FSM generate the Load signal? Draw a state diagram. Explain, with the aid of a timing diagram, whether or not the following schemes would guarantee proper inputs (and hence operation) of the FeatureDetector block.

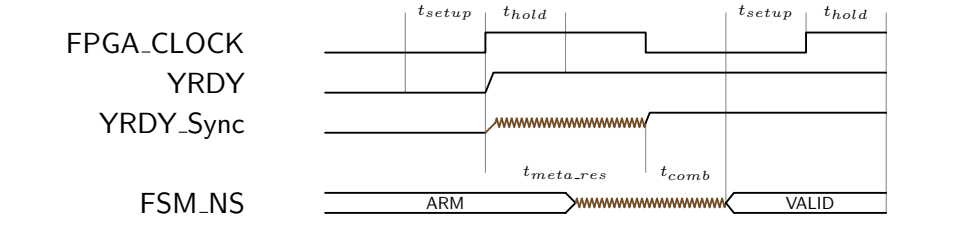


i. YRDY connected directly to FeatureDetector FSM input.

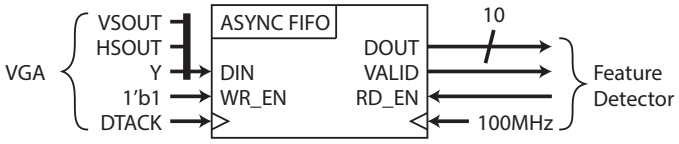
With YRDY connected directly to the FSM input, it is possible with most arbitrary state encodings that more than one FF could have its setup/hold time violated (or possibly even become metastable) when YRDY changes. When this happens the state transition will become inconsistent, and it is not guaranteed the FSM will function as desired. For this simple case, if the states follow a Gray code transition, e.g. 00 → 01 → 11, a missed state transition due to the asynchronous clock would result in a delay, not a bad transition.

ii. YRDY connected to a synchronizing FF, and FF output YRDY_Sync connected to FSM input.

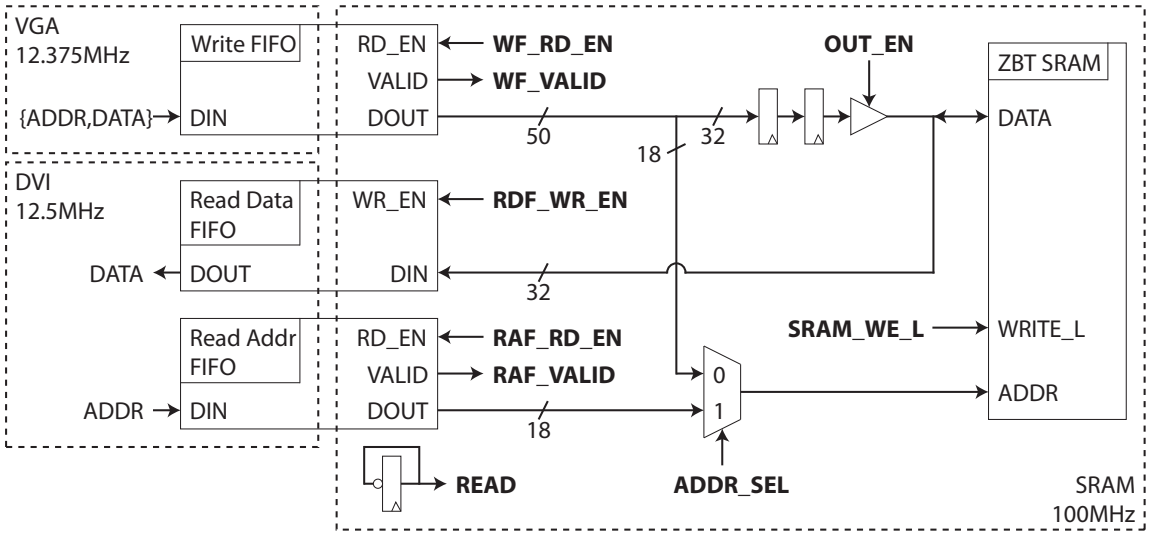
When a synchronizing FF is used to align the asynchronous input YRDY, the possibility of a setup time violation in the FSM next state decoder is eliminated, unless the synchronizer FF goes metastable. If YRDY_Sync is metastable after the rising edge of the FPGA clock, it can be guaranteed that a valid value for the FSM next state will be settled by the setup time of the next clock (if $\frac{1}{f_{FPGA\_CLOCK}} > t_{meta\_res} + t_{comb} + t_{setup}$).



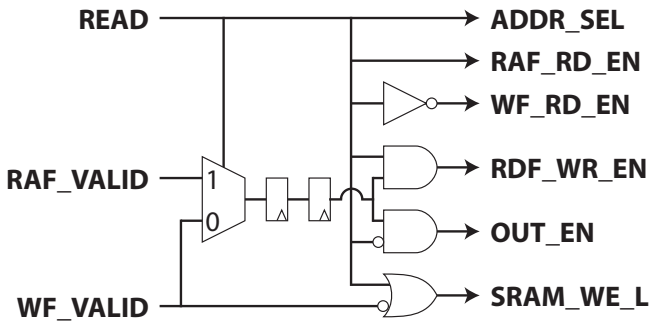
1.3. **Synchronization pt 3 (20 pts).** Now consider the pixels Y[7:0] at the original rate of 49.5 MHz. Draw a block diagram of a general interface between the AD9980 and the FeatureDetector which would guarantee proper reading of Y[7:0], HSOUT and VSOUT by the FeatureDetector. (Hint: FIFO.)



2. **SRAM and FIFOs (55 pts).** In this problem, you will design 2-port arbiter which can share reading to and writing from the IS61NLP25636A-200 "ZBT" synchronous SRAM on the XUP board. This arbiter could enable a simple frame buffer for VGA and DVI interfaces producing and consuming the frames at different rates. In general, these interfaces have different clock speeds and refresh rates, so asynchronous FIFOs are used for communication across clock domains. Assume VGA interface is writing its FIFO at 12.375 MHz, and DVI is reading/writing its FIFOs at 12.5 MHz. The SRAM is clocked at 100 MHz. Please refer to the figure below (notice the shift register and tri-state buffer before SRAM_DATA). FIFO signals operate as described for standard read operation for a FIFO with independent clocks in the section "FIFO Usage and Control" of the Xilinx FIFO Generator v9.1 documentation.



a. Complete the datapath above to result in an arbiter which simply alternates serving requests from the read and write FIFOs. The READ signal functions as a trivial FSM determining which type of operation should be done. If the FIFO for the active operation does not have valid data, the controller should do nothing (equivalent to an SRAM read without putting the data in the Read Data FIFO). Your design should drive all bold inputs above, and use only the bold outputs. You may add simple datapath elements such as shift registers, gates, and muxes. The controller should drive the SRAM with the "Single Read/Write Cycle Timing" shown on p. 20 of the SRAM data sheet.



b. Assume there are 3 read requests and 2 write requests stored in the FIFOs. Draw a timing diagram showing as many cycles as it takes for these 5 requests to be processed. The diagram should begin by processing the first read request.

Include 100MHz clock, READ, SRAM_WE_L, RDF_WR_EN, WF_RD_EN, WF_VALID, RAF_RD_EN, RAF_VALID, OUT_EN, SRAM_DATA, SRAM_ADDR, and ADDR_SEL in the diagram.

Please note that the timing diagram below uses the Read with FWFT (First Word Fall Through) FIFO mode, rather than the standard read mode. (Please see Fig. 5-10 of FIFO generator, or slide 10 of lecture 14.) The reader will accept either FIFO mode.