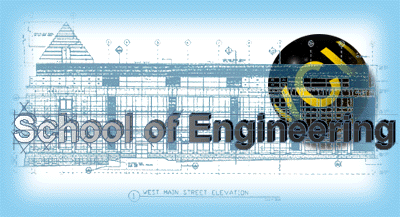
### EGRE 427 Homework No. 4

Title Clock Crossing Boundaries

Name Chad Chapman

##### Date Submitted 13-May-10



I pledge this work to be my own \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Statement of Problem and Objectives**

As discussed in lecture and in the handout called “Caution: Clock Crossing” we already understand that we cannot avoid passing data across clock domains with each having different frequencies. This means we have to pass data from a domain with a slower clock frequency to a domain with a faster clock frequency so we need to be able to achieve the same reliability for both directions. The instructor presented us with a waveform where clock one had a bigger clock cycle and data has to be passed to clock two which has a clock cycle with a smaller frequency.

Thus it makes sense to implement and synthesize a description where data is being passed from a clock domain with a smaller clock cycle to a faster one. We will use a 4-state hand shaking protocol. The objectives are as follows:

* Use the given VHDL Files async\_tb.vhd, data\_gen.vhd, exemplar.vhd, and reg8.vhd along with our descriptions to implement the 4 state hand shaking protocol.
* One of those descriptions is for the Source State Machine which is called source\_fsm.vhd and the second description called dest\_fsm.vhd which resides on the destination side.
* Use rising edge flip-flops to synchronize the signals that will be used for this protocol. These signals are called ack, for acknowledgment, and ready and each of them will require flip-flops.
* Use the ack and ready signals to define the state variables by using all four combinations between these two signals.

**Design Procedure**

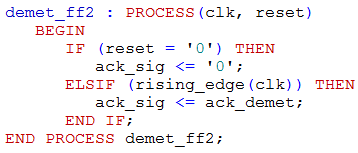
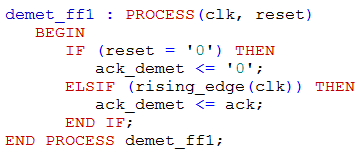
We will look at the source side of the 4-state hand shaking protocol which consists of a Data Generator (data\_gen.vhd), Source Register (reg\_8.vhd), and the Source State Machine (source\_fsm.vhd). By inspection of the test bench (async\_tb.vhd) we already know the port declarations for the source\_fsm. This description is shown below.



**Figure 1** – source\_fsm port inputs and outputs.

The signal data\_valid is driven by the Data Generator. When data\_valid is asserted we know that we have new data that is considered valid and that data will be on a put on an 8-bit bus called data\_packet. This data is comes from the Data Generator and will be loaded into the Source Register. A signal called ready will be the source\_fsm’s only output. The signal that provides acknowledgement originates from the destination finite state machine and is called ack.

The source\_fsm will not directly get the ack signal directly from the destination. Instead we have to use two flip-flops which will synchronize the signal with respect to the clock on the source side, called c1, by preventing metastability. The descriptions of these flip flops are as follows:



**Figure 2** – Metastabilty flip-flops for the Source Finite State Machine.

We use two flop-flops to prevent metastability because a flip flop has a very short decision window. If a signal is asserted in this time frame there exist a chance that the output will be neither a ‘1’ nor ‘0’.

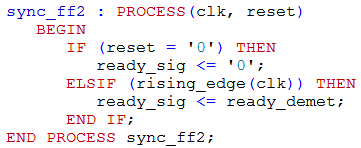
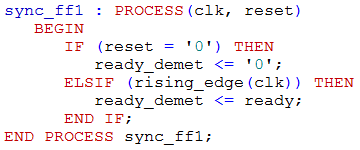
The destination side has a finite state machine has a similar setup that uses flip-flops. First we need to discuss the input and output ports for the dest\_fsm. The port map is shown below in **figure 3**.



**Figure 3 –** dest\_fsm port inputs and outputs.

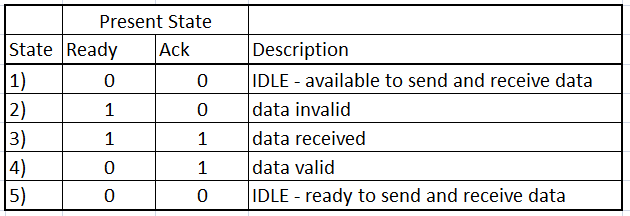
Now ready is an input and ack is an output. The dest\_hold is a third control signal that is used to determine states for the dest\_fsm. dest\_hold originates from the Data Generator and will need to be asserted when the Destination Register is not ready to accept new data. When the Destination State Machine must finish its current data transfer if there is one in progress. If there is no data transfer in progress then it cannot start a new data transfer until dest\_hold is set back to ‘0’.

The data\_latch signal is an output that will control the Destination Register. When the Destination Register is loaded with the new data and after that clock cycle for   
Clock – c2 then the ack signal is asserted. How these signals change depends on the ready signal whose value will determine the next state for the dest\_fsm. Therefore Destination State Machine also needs metastability flip-flops before it can use the ready signal. The flip flops on the destination side are shown below:



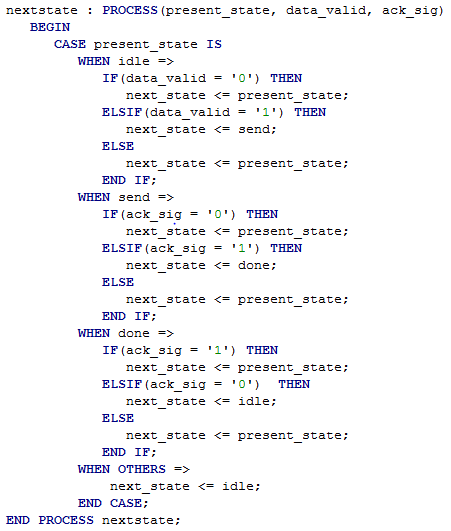
**Figure 4** – Metastabilty flip-flops for the Destination Finite State Machine.

Note how similar the metastability flip-flops are and how they are the only two signals besides data\_trans that actually crosses the clock boundary. These are the signals that we used to describe our 4-state handshake protocol. The four possible states are described in the table below.



**Table 1 –** State descriptions for the 4-state handshake protocol.

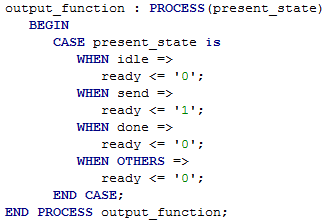
These four states for the 4-state handshaking protocol are a combination outputs from the Source State Machine and Destination State Machine. First we will discuss the description of the Source State Machine whose nextstate process is shown below.



**Figure 5 –** Source State Machine’s next state and output process.

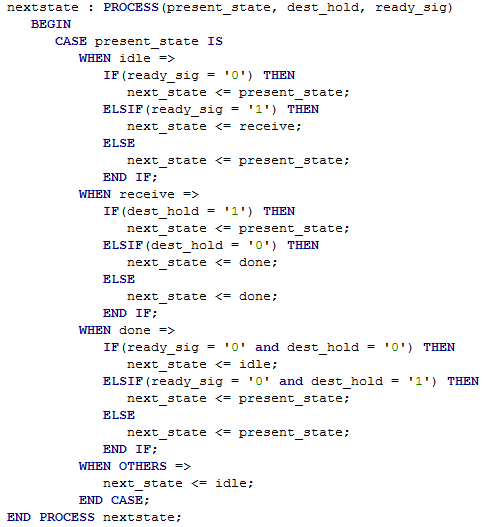
Although the entire description is a 4-state implementation, we only needed to use three states in the Source State Machine. These states are idle, send, and done. Since we already know that implies that there is new data that needs to be transferred we can make a transition to the send state, otherwise we remain in the state labeled idle.

Once we are in the send state we remain there until which will cause the ack\_signal to become asserted. In other words, when we are finished sending data and we now begin acknowledgement. Once we go from the make a transition from the state send to the state done. When the the ready signal is deserted in the output process. The following figure shows the output process.



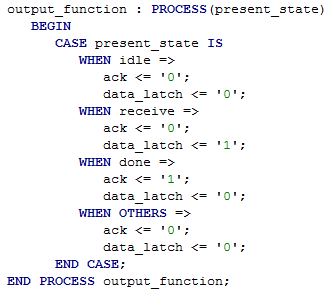
**Figure 6** – Source State Machine output process.

The one thing that may seem confusing is that we use a different signal name since we want to distinguish between the original signal and the signal two clock cycles later as it reaches the Destination State Machine. As discussed earlier the Destination State Machine performs in a similar fashion except it is more dependent on the dest\_hold signal. The hardware description for the next state process in the Destination State Machine is shown in **Figure 7**.



**Figure 6** – Destination State Machine next state process.

The next state process for dest\_fsm has three states called idle, receive, and done. The idle state is similar to the one for dest\_fsm. The only difference is we stay in idle if the signal ready\_sig is 0, otherwise we move to the receive state. When we are in the receive state and the dest\_hold signal is asserted we need to make sure that the current data transmission is completed. Thus while we are in the receive state we keep and . Once dest\_hold goes to ‘0’ we transition to the state called done where and . When ack is equal to 1 we know that the data transmission was completed successfully. These output values are shown below in **Figure 7**.



**Figure 7** – Destination State Machine output process.

The done state was a little tricky to see until we were completely what happens when is suppose to implies. Thus we found it necessary to break the logic into two cases along with ready\_sig. For and we know the best thing to do is go to the idle state. But if and we need to stay in the done state because if there is a new data transition coming up real soon and we just move to the idle state we may run into some trouble. This also makes us stay in the fourth state with respect to the 4-state protocol where and . That way we also force while so that we can be sure that a new data transition does not happen when the Destination Register is not ready for new data.

**Results and Conclusions**

Initial Clock Period Results: Case when dest\_hold = 0

We ran simulations for various times to inspect the behavior when the clock cycle was shortened by 5 no at time. For the clock cycles that were initially given by the instructor in the test bench we observed how for most data transmissions the waveforms matched directly with the one in the hand out.

Initial Clock Period Results: When dest\_hold = 1 at [46,000 ns – 50,000 ns]

But for when dest\_hold is asserted for the first time the Destination State Machine is in the idle state at 46,000 ns. A data transmission was about to start at that time and the data stayed invalid since and after the data\_valid signal went to 0 and the data\_latch stayed asserted until . Once this happened data\_latch stayed asserted long enough for the Destination Register to get the new data.

Initial Clock Period Results: When dest\_hold = 1 at [64,000 ns – 68,000 ns]

The second time is at where and , so the current state is ‘State 4 – data valid’ and the current state for the Destination State Machine is PRESENT\_STATE = DONE. Since the Destination State Machine currently is at PRESENT\_STATE = DONE, so and until when . On the next rising edge of Clock – c2 the signal so the 4-state handshaking protocol goes into ‘State 1 – IDLE’ since and .

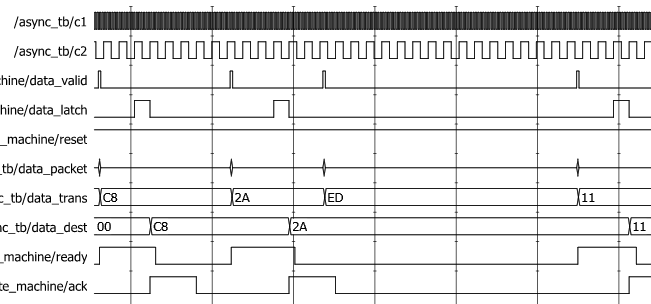
Therefore when we guarantee that the current data transmission is complete and the acknowledgement signals assume the correct values. As a consequence the transmission of new data is prevented since the Destination Register is not ready to until .

Fastest Working Timing Simulation Where   
and

In this simulation the time interval in the printout is with 6 different data transmissions where the set of data transmissions where the data transmission is always successful for but not necessarily the exact length of time for each data transmission

First Timing Simulation Where Simulation Did Not Work Properly   
Where and

In this simulation the time interval in the printout is with 6 different data transmissions where the set of data transmissions where that did not finish successfully. The following figure shows which data transmission that did not complete successfully.



**Figure 8** – Section of waveform that shows which data transmission did not complete successfully.

The following is a list of VHDL source code and waveforms in the order that they appear:

1. async\_tb.vhd
2. source\_fsm.vhd
3. dest\_fsm.vhd
4. data\_gen.vhd
5. reg8.vhd
6. Waveform that resembles the example in class.
7. Waveform showing the complete simulation with all transmissions.
8. A waveform showing a zoom-in of one data transmission as shown above.
9. The fastest working time simulation.
10. First timing simulation where the simulation did not function correctly.