Project Status Report – RMC75E Test Bench

Date: July 7, 2023

Subject: Progress Update

Current Phase:

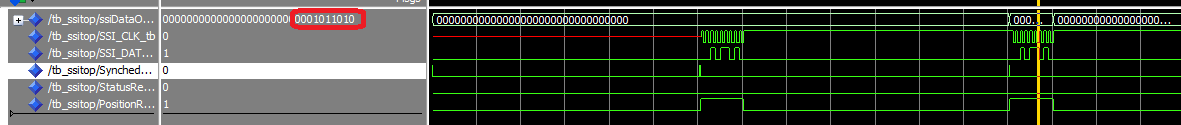
Refining Test Units for Quad and QuadXface Modules

Work to Date:

Last week I was able to work out the conditions that needed to be emulated for the SSI\_CLK signal to be correctly defined during the simulation of the SSITop module. There are now nine negative pulses on the SSI\_CLK line that are generated in response to any loop tick after the initial tick, given that either the position read or status read signals are asserted correctly, along with the SSI select signal. Note that these signals have their own conditions that need to be met as well.

Just yesterday I was able to successfully write an 8-bit vector to the SSI\_DATA line by synchronizing the write operations with the series of pulses that are triggered by the loop tick. The write operation is synchronized to the rising edge of the pulses, with one bit being written per negative pulse, and the last negative pulse reserved for a status / error bit.

Fig. 1



After the write operation, the written vector appears in the SSI data output line on the next position or status read assertion. This is the behavior we expect to see and demonstrates the correct functionality of the module.

One small edit to the source code was made to allow this. In the source code there are initial values for most signals that have been commented out, which to my understanding is because these initial assignments applied only to the previous FPGA family (Xilinx Spartan) and not the IGLOO2 family of FPGA currently in use. Ergo, the initial value assignments are not required.

However, one signal seemed to never be assigned a value during the simulation, and that was the ShiftOn internal signal. This undefined signal seemed to cause the SSI\_CLK to not function properly without an initial value. When given an initial value, this problem cleared up. The initial value seems to be irrelevant, if it is zero, operation will commence normally, if it is set to one, it will be set back to 0 at system start-up. Only when it is undefined at simulation start-up was there an issue.

Fig. 2

A computer screen shot of code

Description automatically generated



To date, three source modules have had their source code edited with some minor tweaks.

* Controloutput.vhd
* ExpModuleLED.vhd
* SSITop.vhd

Fig. 3

A screenshot of a computer

Description automatically generated

The modifications to the controloutput and expmoduleled modules consisted of adding a catch-all statement to the state encoding logic to resolve any weird states that might pop up, ergo:  
  
IF (A state):  
do this thing

IF (B state):  
do this other thing

IF (Any other state): 🡨 This is our catch all logic  
go to IDLE MODE

Fig. 4

A screen shot of a computer program

Description automatically generated

Challenges:

Hunting down the problem signals and figuring out how to massage the internal signals just right so that the SSI\_CLK signal came to life in the SSITop module was fairly time consuming, but also informative.

Once I had the SSI\_CLK signal working correctly, I created more problems for myself by trying to make the test bench more elegant and efficient. I wrapped up two operations, the synchronization sequence that causes SSI\_CLK to send its series of pulses, and the data write operation that writes a vector while the pulses are active, each within their own procedure to reduce repeated code. A procedure is a construct used to group a sequence of statements and encapsulate a specific functionality. I haven’t used them at all in the test benches so far. I thought implementing this new data structure would be fairly straight forward, but I had difficulty managing the scope of each procedure as it related to the rest of the program. Long story short, undoing this change ended up costing way more time than it was worth. My takeaway from this was don’t try to perfect something or make it pretty before I’ve even finished the rough draft.

Next Steps:

Following a similar procedure as with the MDT and SSI modules to set up the Quad and Quadxface modules.

Fig. 5

A white board with writing on it

Description automatically generated

Some whiteboarding RE setting up the Quad module correctly.

A screenshot of a computer

Description automatically generated

All modules and corresponding test units are compiling happily in ModelSim. The work now is to ensure that each of these test units properly emulates the conditions required for correct functionality of the DUT.