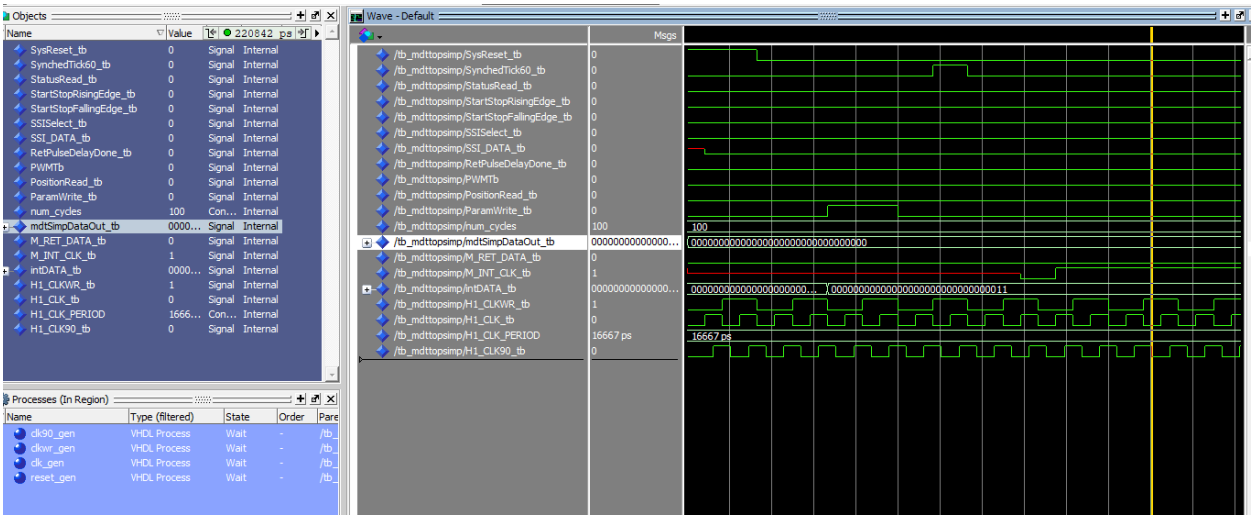
Project Status Report – RMC75E Test Bench

Date: June 14, 2023

Subject: Progress Update: VHDL Testbench for MDTTopSimp Entity

Problem Summary:

Currently, my primary concern is the absence of data on the MDTSimpDataOut output line, which is intended to receive data from the PositionRead and StatusRead signals. An in-depth analysis of the VHDL source code for the MDTTopSimp entity and its associated test bench led us to identify a set of conditions that must be satisfied for this data transfer to occur. These conditions themselves have their own conditions, and emulating all of these conditions is proving to be a challenge.



As of right now, this is my understanding of the conditions that need to be met for the data output to receive information from the PositionRead and StatusRead signals:

1. PositionRead signal must be asserted ('1').

2. SSISelect signal must be deasserted ('0').

3. Either StartStopRisingEdge, StartStopFallingEdge, or PWM signals must be asserted ('1').

4. The RetPulseDelayDone signal must be asserted ('1').

5. All other signals (StatusRead, ParamWrite, SysReset, etc.) should be deasserted ('0').

6. The second SynchedTick pulse must have been driven so that a correct data value is latched, and not just the garbage initial value that is captured on the first pulse.

Note that there may be additional sub-conditions that need to be met for these conditions to be emulated correctly.

Additionally, the internal state machine in the MDTTopSimp entity controls the sequence of operations. It transitions through different states to generate and handle the MDT counters' count sequence. The

state machine should be properly configured to ensure the desired behavior and enable the transfer of data to mdtSimpDataOut.

The RetPulseDelayDone signal indicates that a return pulse has not been received within 50us of the start of the interrogation pulse, which triggers the setting of the NoXducer bit. The combination of these conditions will enable the transfer of data to mdtSimpDataOut.

mdtSimpDataOut can receive data only after the 50us mark has been reached: The RetPulseDelayDone signal, which is set when the Delay counter reaches the value specified by the RetPulseDelay constant (indicating that a return pulse has not been received within 50us), triggers the setting of the NoXducer bit. This condition, in combination with other factors, enables the transfer of data to mdtSimpDataOut. Therefore, mdtSimpDataOut can receive data only when RetPulseDelayDone is asserted, indicating that the return pulse delay has exceeded 50us.

mdtSimpDataOut's data transfer depends on the SynchedTick60 pulse, RetPulseDelay, RetPulseDelayEnable signals, and IntPulseLength: The RetPulseDelay constant specifies the time duration in clock cycles that represents a 50us delay. The RetPulseDelayEnable signal is used to enable the Delay counter, allowing it to count and measure the delay. The IntPulseLength constant represents the duration of the interrogation pulse. The combination of these factors determines the timing and synchronization required for mdtSimpDataOut to receive data correctly.

Progress to Date:

Test units are built for all the modules in the source code, and all test units are able to compile and simulate without error. However, whether or not each test unit correctly emulate all the conditions needed to verify the correct functionality and behavior of the module under test is another matter. For some of the smaller test benches, I think they’re probably set up correctly. For some of the larger modules, like the MDTTop, work still needs to be done to ensure the test benches are set up correctly.

Comprehensive analysis of the VHDL source code for all modules is still under way, including the MDTTopSimp entity and the existing test bench to understand their operation and identify key signals and conditions.

Pinpointed the key conditions required for the MDTSimpDataOut line to receive data from PositionRead, involving signals like PositionRead, SSISelect, StartStopRisingEdge, StartStopFallingEdge, PWM, RetPulseDelayDone, and the internal state machine of the MDTTopSimp entity. Resolving the interaction required between these signals to allow meaningful output from the data line is ongoing.

Identified that the RetPulseDelayDone signal, indicating that a return pulse has not been received within 50us of the start of the interrogation pulse, plays a crucial role in enabling data transfer to MDTSimpDataOut.

Gained insight into the dependencies between signals, e.g., the StartStopRisingEdge, StartStopFallingEdge, and PWM signals depend on the value of TransducerSelect(1 downto 0).

Reviewed the testbench code and made necessary adjustments to ensure the appropriate conditions are met in the correct sequence for data transfer to occur.

Accomplishments

On Thursday I was able to emulate all the conditions necessary to correctly define M_INT_CLK by carefully navigating through the complex chain of conditions in the MDTTopSimp entity. I followed the logic and dependencies of various signals and ensured they were properly set up in the test bench to reflect the expected behavior of the system.

Specifically, I focused on the following:

1. Ensuring that the conditions for setting StartInterrogation to '1' are met. This involves verifying the values of SynchedTick60, RetPulseDelayEnable, CounterOverFlowRetrigger, and MDTSelect. By properly configuring these signals in the testbench, I was able to trigger the StartInterrogation signal and initiate the desired behavior.

2. Controlling the inputs to the state machine to drive it through the required states for data transfer to MDTSimpDataOut. By carefully setting the values of the signals that control the state machine, such as StartInterrogation, ClearCounter, and SendInterrogationPulse, I ensured that the state machine progressed as expected, enabling the transfer of data to MDTSimpDataOut.

Earlier this week I solved an issue involved with ModelSim not recognizing the Smartfusion2 libraries that the clock generator modules in the source code relied upon. This was leading to the clock modules not propagating the clock signals throughout the system correctly. Initially I solved this problem by producing my own drop-in clock generator that did not depend on the smartfusion2 library.

Shortly after that, I resolved the dependency issue in ModelSim that was keeping it from recognizing the smartfusion2 library. Ultimately, it was a matter of the smartfusion2 library instance in my project file not being mapped to the correct source library, which counterintuitively resides in a subdirectory in Libero and not ModelSim. So ultimately, I was finally able to use the original clock modules in the design, which is ideal, as it removes the chance of errors coming from shortcomings in my drop-in clock module.

I have generated quite a lot of documentation regarding the source code and the test benches themselves. I have expanded the docstring descriptions in each of the source modules to further clarify how the module in question functions. I have several documents in various stages of completion that outline the functionality of individual modules or provide a high-level overview on how various modules interact with each other.

I have made some minor changes in two of the source code modules that were causing compilation issues in ModelSim. In both cases the issue was related to a missing catch-all statement to handle odd states that might pop up within the state machine encoding. I am keeping all modified source code in its own directory, ensuring any changes made to the source code are kept track of.

Next Steps:

Continue to investigate the MDTTopSimp module source code until I am able to correctly emulate the conditions required to output data from the mdtSimpDataOut line. The updated testbench should then accurately emulate the desired conditions and generates meaningful and actionable data. I will carefully review the testbench code to ensure all the necessary conditions are properly set up and run multiple simulations to validate the behavior.

After that I can move on to other test benches and apply the same methodology used in the aforenoted module to ensure the test unit in question is ship-shape.

At some point, we can look at removing or modifying source code that may be suboptimal or otherwise extraneous, as per your instructions.

I will continue to document the source code and revised test units, including the rationale behind the changes made, providing clear explanations and instructions for future understanding and maintenance.