Project Status Report – RMC75E Test Bench

Date: July 7, 2023

Subject: Progress Update

Current Phase:

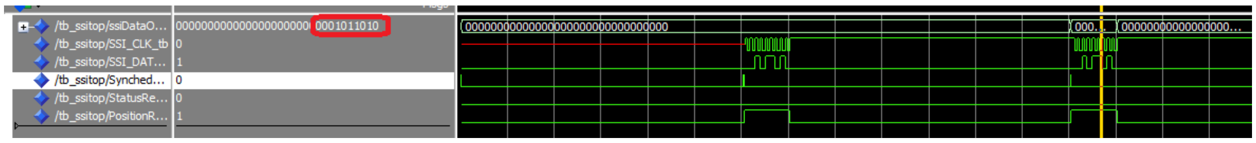
Refining Test Units for Quad and QuadXface Modules

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

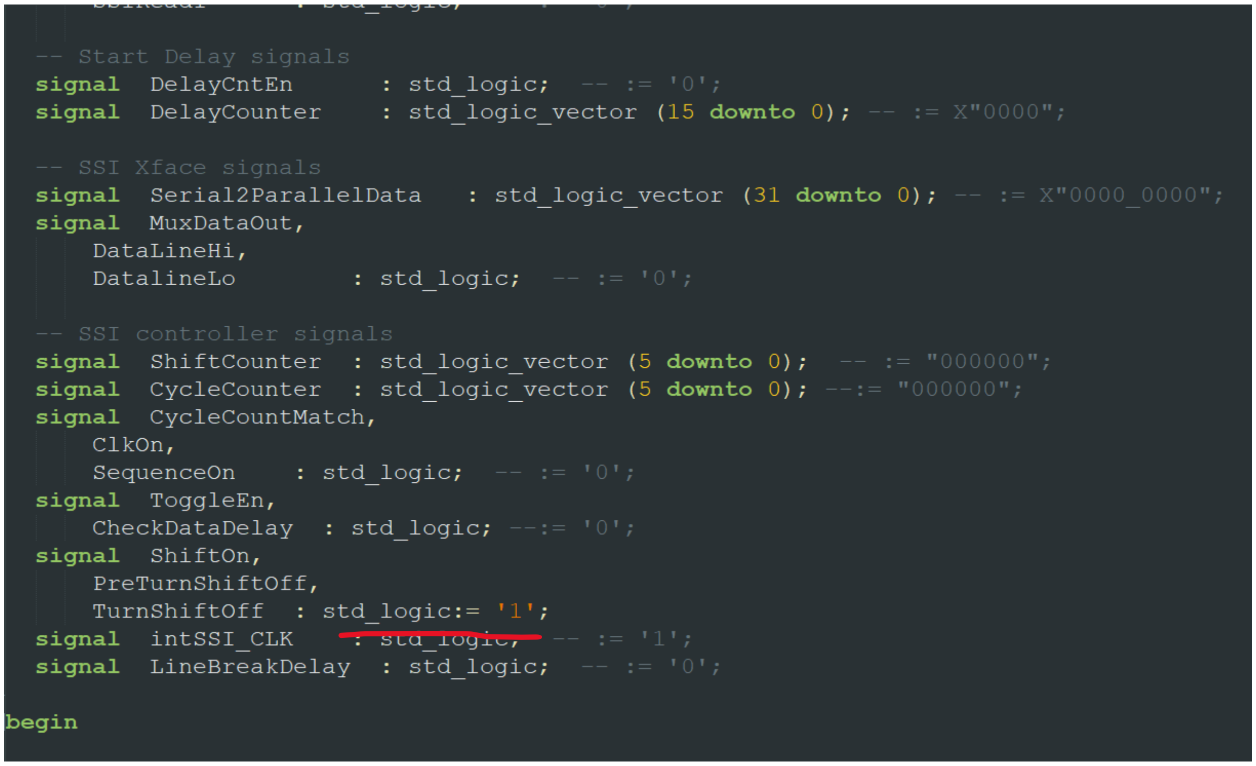


After the write operation, the written vector appears in the SSI data output line on the next position or status read assertion. This is the behavior we expect to see and demonstrates the correct functionality of the module.

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

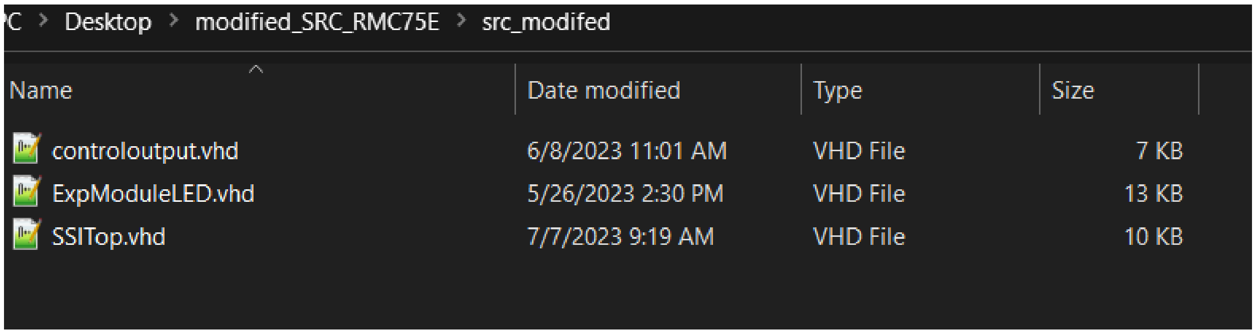


```vhdl
  -- Start Delay signals
  signal  DelayCntEn       : std_logic;  -- := '0';
  signal  DelayCounter     : std_logic_vector (15 downto 0); -- := X"0000";

  -- SSI Xface signals
  signal  Serial2ParallelData   : std_logic_vector (31 downto 0); -- := X"0000_0000";
  signal  MuxDataOut,
      DataLineHi,
      DatalineLo          : std_logic;  -- := '0';

  -- SSI controller signals
  signal  ShiftCounter  : std_logic_vector (5 downto 0);  -- := "000000";
  signal  CycleCounter  : std_logic_vector (5 downto 0); --:= "000000";
  signal  CycleCountMatch,
      ClkOn,
      SequenceOn     : std_logic;  -- := '0';
  signal  ToggleEn,
      CheckDataDelay  : std_logic; --:= '0';
  signal  ShiftOn,
      PreTurnShiftOff,
      TurnShiftOff   : std_logic:= '1';
  signal  intSSI_CLK      : std_logic;  -- := '1';
  signal  LineBreakDelay  : std_logic;  -- := '0';

begin
```

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

- Controloutput.vhd
- ExpModuleLED.vhd
- SSITop.vhd

Fig. 3

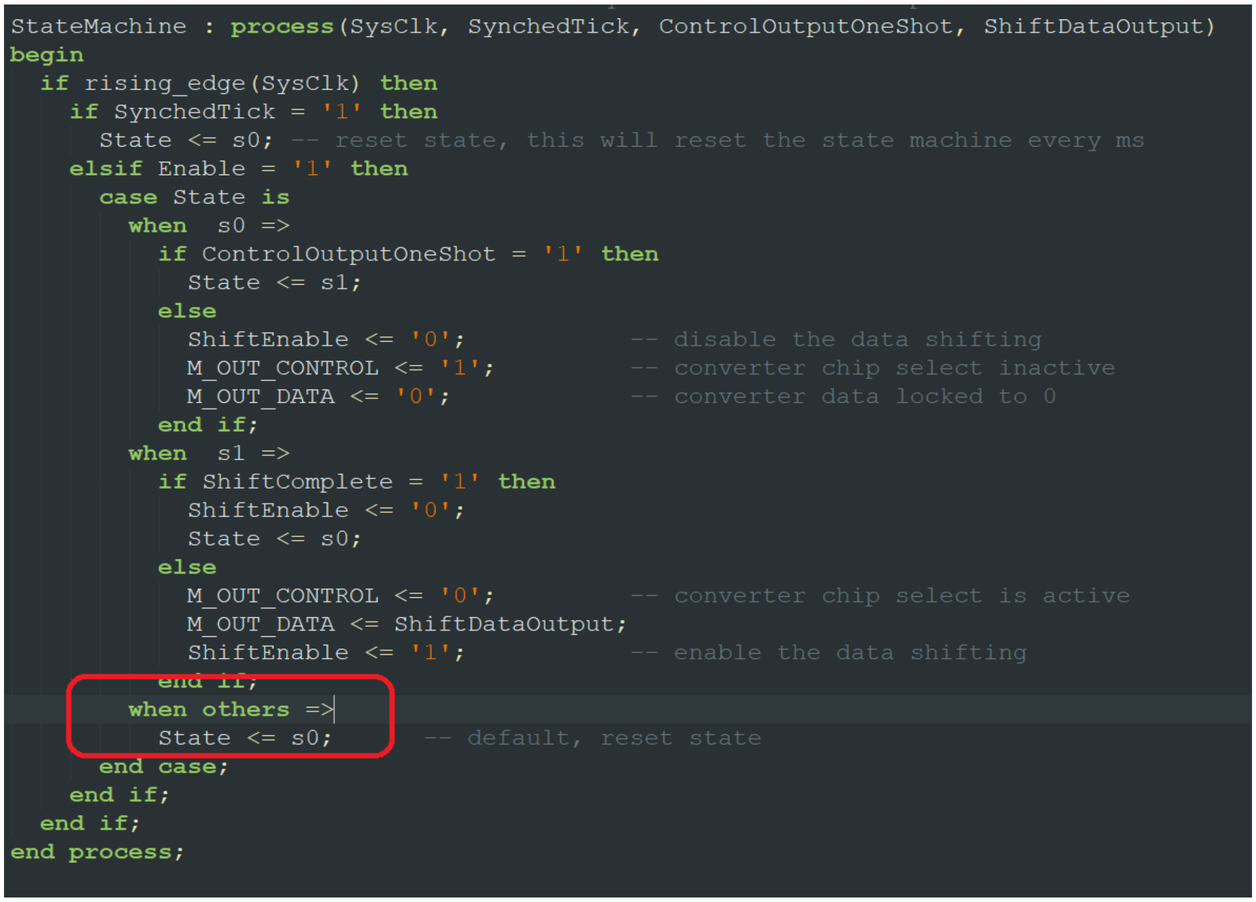


```
'C > Desktop > modified_SRC_RMC75E > src_modifed

Name                    Date modified         Type        Size
controloutput.vhd       6/8/2023 11:01 AM     VHD File     7 KB
ExpModuleLED.vhd        5/26/2023 2:30 PM     VHD File    13 KB
SSITop.vhd              7/7/2023 9:19 AM      VHD File    10 KB
```

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



```
StateMachine : process(SysClk, SynchedTick, ControlOutputOneShot, ShiftDataOutput)
begin
  if rising_edge(SysClk) then
    if SynchedTick = '1' then
      State <= s0; -- reset state, this will reset the state machine every ms
    elsif Enable = '1' then
      case State is
        when  s0 =>
          if ControlOutputOneShot = '1' then
            State <= s1;
          else
            ShiftEnable <= '0';             -- disable the data shifting
            M_OUT_CONTROL <= '1';           -- converter chip select inactive
            M_OUT_DATA <= '0';              -- converter data locked to 0
          end if;
        when  s1 =>
          if ShiftComplete = '1' then
            ShiftEnable <= '0';
            State <= s0;
          else
            M_OUT_CONTROL <= '0';           -- converter chip select is active
            M_OUT_DATA <= ShiftDataOutput;
            ShiftEnable <= '1';             -- enable the data shifting
          end if;
        when others =>
          State <= s0;          -- default, reset state
      end case;
    end if;
  end if;
end process;
```

Challenges:

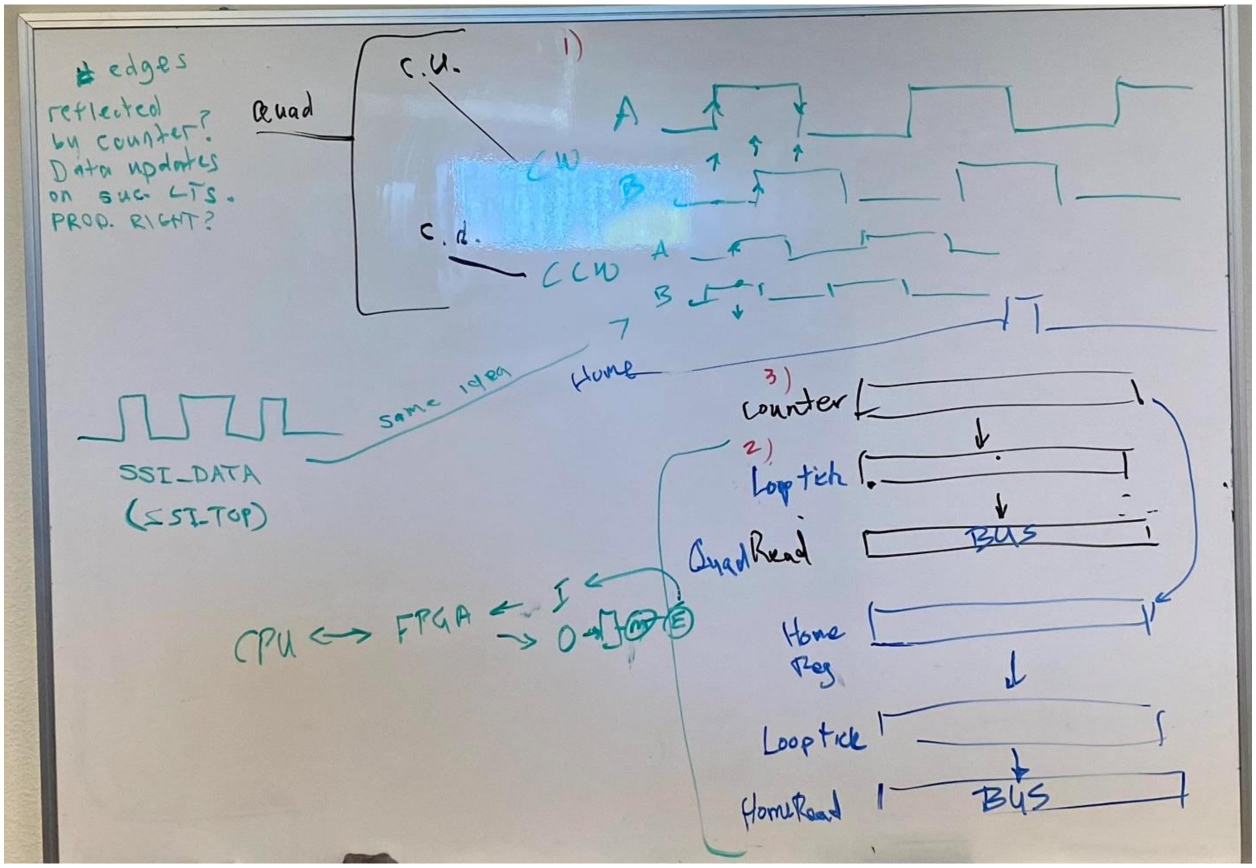
Hunting down the problem signals and figuring out how to massage the internal signals just right so that the SSI_CLK signal came to life in the SSITop module was fairly time consuming, but also informative.

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.

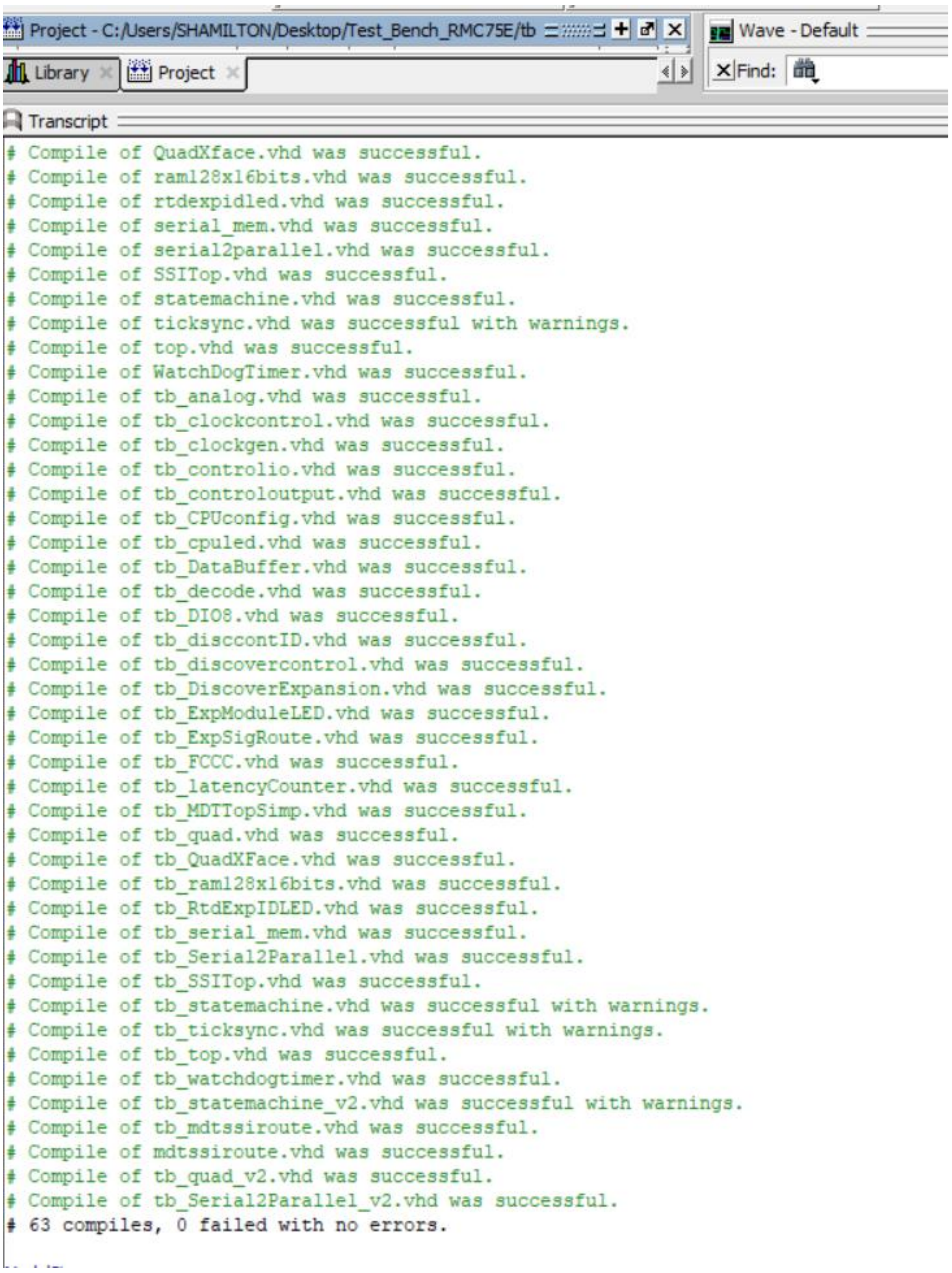
Next Steps:

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

Fig. 5



Some whiteboarding RE setting up the Quad module correctly.



```
# Compile of QuadXface.vhd was successful.
# Compile of ram128x16bits.vhd was successful.
# Compile of rtdexpidled.vhd was successful.
# Compile of serial_mem.vhd was successful.
# Compile of serial2parallel.vhd was successful.
# Compile of SSITop.vhd was successful.
# Compile of statemachine.vhd was successful.
# Compile of ticksync.vhd was successful with warnings.
# Compile of top.vhd was successful.
# Compile of WatchDogTimer.vhd was successful.
# Compile of tb_analog.vhd was successful.
# Compile of tb_clockcontrol.vhd was successful.
# Compile of tb_clockgen.vhd was successful.
# Compile of tb_controlio.vhd was successful.
# Compile of tb_controloutput.vhd was successful.
# Compile of tb_CPUconfig.vhd was successful.
# Compile of tb_cpuled.vhd was successful.
# Compile of tb_DataBuffer.vhd was successful.
# Compile of tb_decode.vhd was successful.
# Compile of tb_DIO8.vhd was successful.
# Compile of tb_disccontID.vhd was successful.
# Compile of tb_discovercontrol.vhd was successful.
# Compile of tb_DiscoverExpansion.vhd was successful.
# Compile of tb_ExpModuleLED.vhd was successful.
# Compile of tb_ExpSigRoute.vhd was successful.
# Compile of tb_FCCC.vhd was successful.
# Compile of tb_latencyCounter.vhd was successful.
# Compile of tb_MDTTopSimp.vhd was successful.
# Compile of tb_quad.vhd was successful.
# Compile of tb_QuadXFace.vhd was successful.
# Compile of tb_ram128x16bits.vhd was successful.
# Compile of tb_RtdExpIDLED.vhd was successful.
# Compile of tb_serial_mem.vhd was successful.
# Compile of tb_Serial2Parallel.vhd was successful.
# Compile of tb_SSITop.vhd was successful.
# Compile of tb_statemachine.vhd was successful with warnings.
# Compile of tb_ticksync.vhd was successful with warnings.
# Compile of tb_top.vhd was successful.
# Compile of tb_watchdogtimer.vhd was successful.
# Compile of tb_statemachine_v2.vhd was successful with warnings.
# Compile of tb_mdtssiroute.vhd was successful.
# Compile of mdtssiroute.vhd was successful.
# Compile of tb_quad_v2.vhd was successful.
# Compile of tb_Serial2Parallel_v2.vhd was successful.
# 63 compiles, 0 failed with no errors.
```

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.