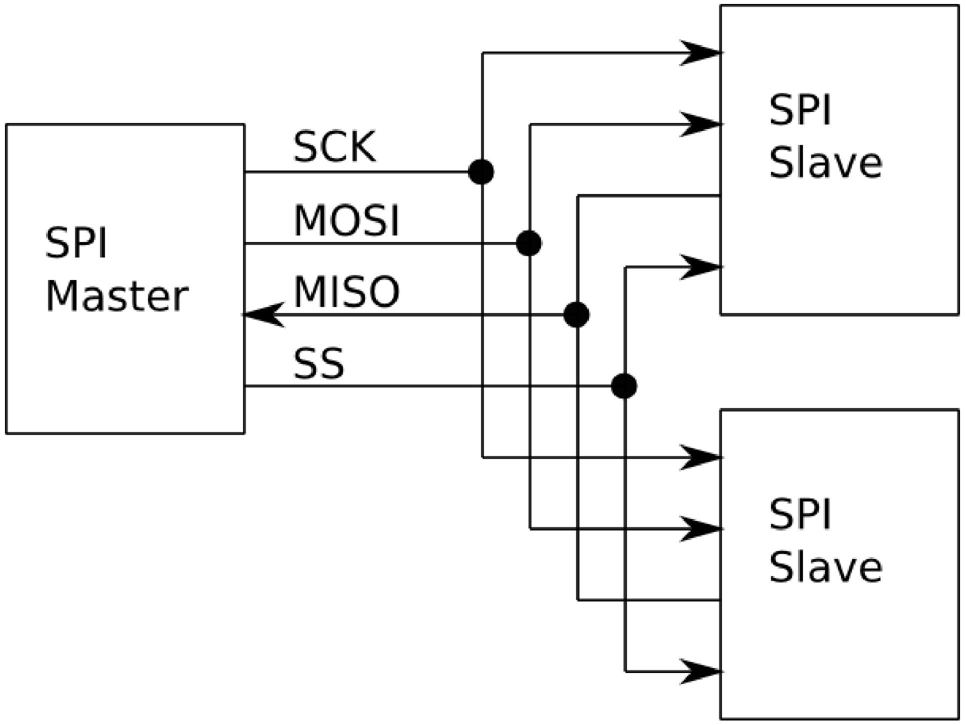
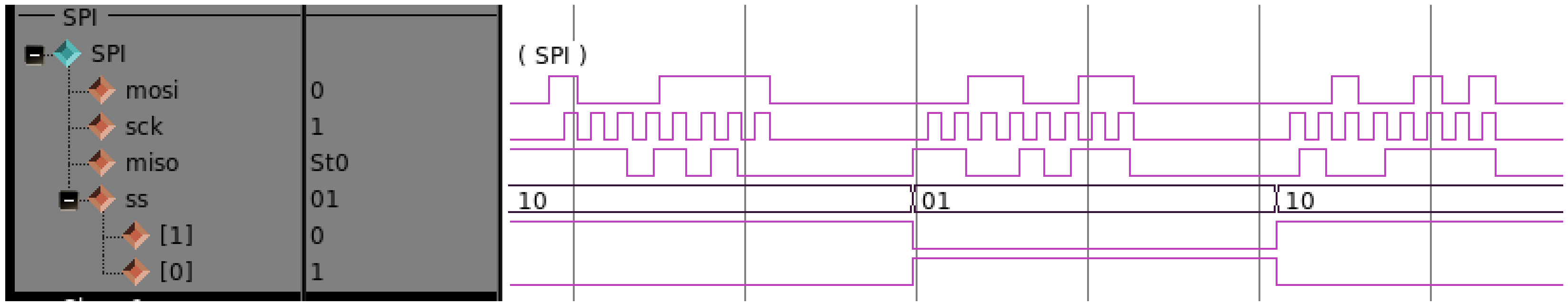
Use of SystemVerilog Interfaces in RTL Design and Test Benches A small example based on SPI

Dr. Mark C. Johnson

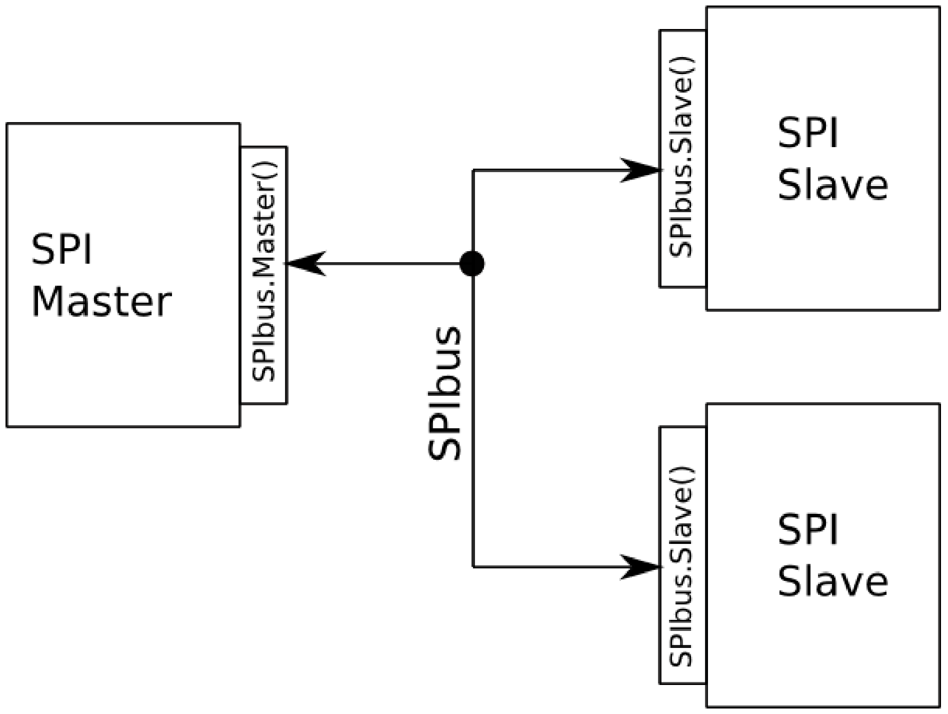
March, 2016

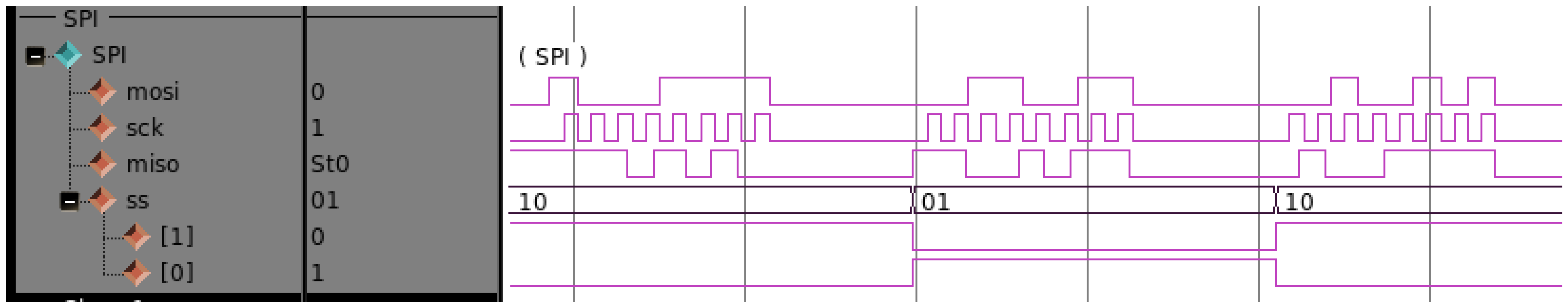


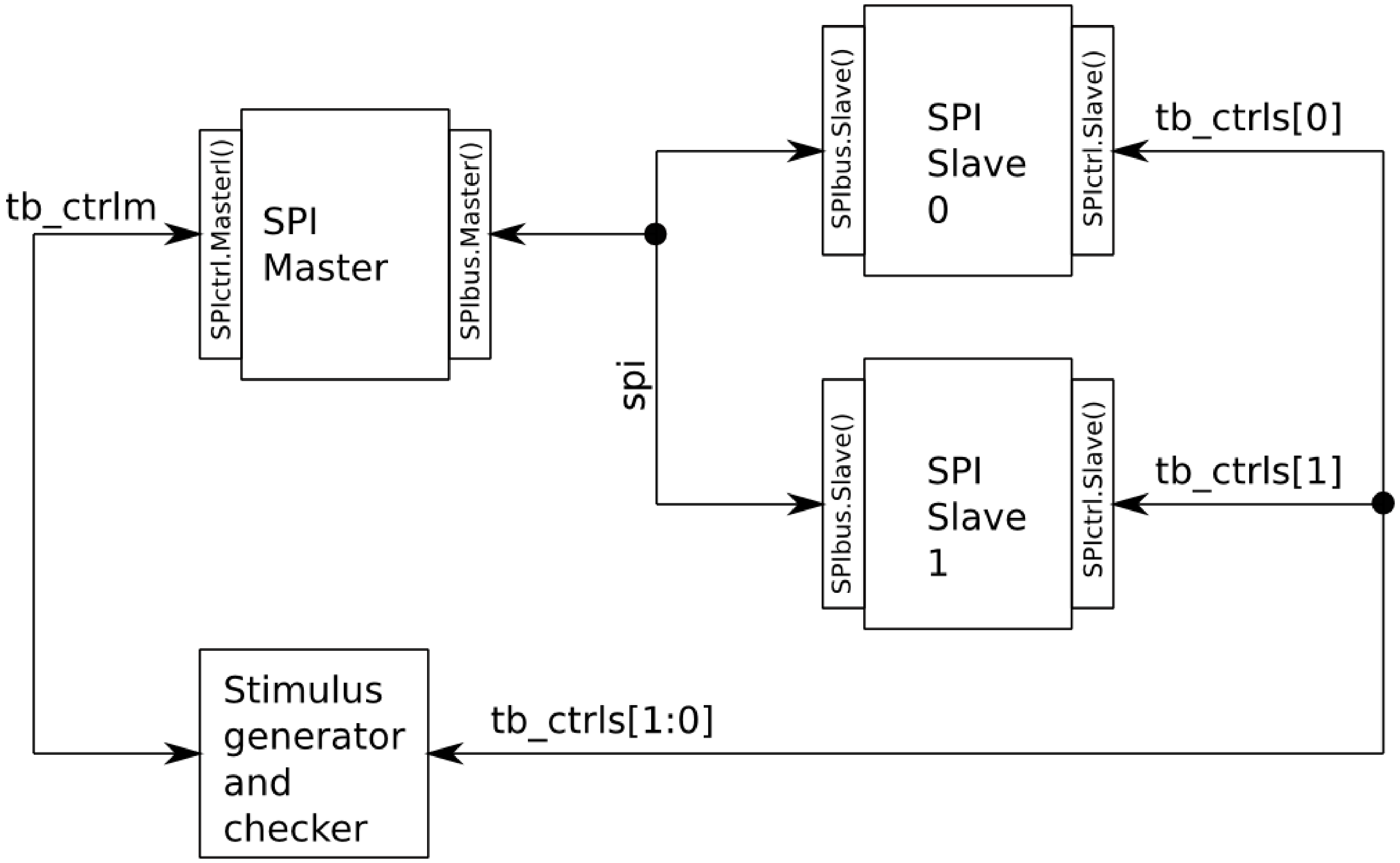
SPI Interconnect without using modular interface definition.



SPI Interconnect with modular interface definition.







Organization of test bench using SV interfaces.

Code to define interface, put in a header file

```
// Simple SPI interface example
// for now only supports Master->Save communication for 2 devices
interface SPIbus;
  logic mosi,sck;
  wire miso;            // logic type won't allow multiple drivers
  logic [1:0] ss;
  modport Master (input miso, output mosi, sck, ss);
  modport Slave (input mosi, sck, ss, output miso);
endinterface

interface SPIctrl;
  logic [7:0] toXmit;   // value to be transmitted by master or slave
  logic [7:0] Rcvd;       // data value recieved by master or slave
  logic strobe;         // tell device that there is data on toXmit available
  logic Ready;            // slave or master indicates data on Rcvd is ready to use
  logic XmitFull;         // = 1 if input buffer already full, not ready for new data
  logic [1:0] ss;       // tell master how to set slave select lines
  logic busy;           // 1 if master or slave already busy transmitting current byte
  modport Master (input toXmit, strobe, ss, output Rcvd, Ready, XmitFull, busy);
  modport Slave  (input toXmit, strobe, output Rcvd, Ready, XmitFull, busy);
  modport System (output toXmit, strobe, input Rcvd, Ready, XmitFull);
endinterface
```

Use of interface in SPI master

```
//`include "source/spi_interface.svh"
```

commented out because already loaded elsewhere

```
module master #(CLKDIV=8'd4)(SPIctrl.Master Ctrl, SPIbus.Master Spim,
                            input Clk_i, Rst_ni);
```

Ctrl is an instance of SPIctrl with input/output as given by master modport. Spim is an instance of SPIbus with Master modport

```
// load shift reg ond ss n strobe
always_comb begin
  buf_nxt = buf_r;
  ss_nxt = ss_r;
  if (Ctrl.strobe) begin
    ss_nxt = Ctrl.ss;
    buf_nxt = Ctrl.toXmit;
  end else if (clkcnt_r == CLKDIV) begin
    buf_nxt = {buf_r[6:0],1'b0};
  end
end
```

References to elements of Ctrl interface as inputs

Use of interface in SPI master

```
assign Spim.mosi = buf_r[7];
assign Spim.sck = sck_r;
assign Spim.ss = ss_r;
```

assign values to elements of an interface as outputs

```
always_ff @(posedge Clk_i, negedge Rst_ni) begin
  if (Rst_ni == 1'b0) begin
    Ctrl.busy = 1'b0;
    end
  else begin
    if (Ctrl.strobe) begin
      Ctrl.busy = 1'b1;
      end
    else if (bitcnt_r == 9) begin
      Ctrl.busy = 1'b0;
      end
    end
  end
```

Use of interface in SPI slave

Very similar to interface use in SPI master (examples)

```
//`include "source/spi_interface.svh"
```

```
module slave #(parameter ID=0) (SPIbus.Slave Spis, SPIctrl.Slave Ctrl,
                                input Clk_i, Rst_ni);
```

```
// determines when receive shift reg takes in another bit
assign rsh_ena = sck1 && !sck2 && (Spis.ss[ID]==1'b1);
```

```
// tell control bus that data is received
always_ff @(posedge Clk_i, negedge Rst_ni) begin
  if (Rst_ni == 1'b0) Ctrl.Ready <= 1'b0;
  else Ctrl.Ready <= done;
  end
```

```
// slave MISO output
assign Spis.miso = (Spis.ss[ID]==1'b1) ? xmit_r[7] : 1'bz;
```

Use of interface in test bench

In this lecture we will just focus on the use of interfaces. In another lecture we will look at many other aspects of the test bench design such as random input generation, use of assertions, etc.

```
`include "source/spi_interface.svh"
```

```
module tb_spi_system#(parameter TCLK=20);

  // interconnections between master, slaves, and tb
  logic tbClkm,tbClks,Rst_n;  // master clock, slave clock, reset
  SPIbus spi();               // SPI bus between master and two slaves
  SPIctrl tb_ctrlm();         // control interfaces for SPI modules
  SPIctrl tb_ctrls[1:0]();    // array of slave control interfaces
```

Interfaces are used as a data type

```
 master MASTER(.Ctrl(tb_ctrlm.Master), .Spim(spi.Master),
   .Clk_i(tbClkm), .Rst_ni(Rst_n));
 slave #(.ID(0)) SLAVE0 (.Ctrl(tb_ctrls[0].Slave), .Spis(spi.Slave),
   .Clk_i(tbClks),.Rst_ni(Rst_n));
 slave #(.ID(1)) SLAVE1 (.Ctrl(tb_ctrls[1].Slave), .Spis(spi.Slave),
   .Clk_i(tbClks),.Rst_ni(Rst_n));
 `endif
```

For course code simulation, interfaces can be used as ports in and out of the device under test. The modports (Slave, Master) are specified since the modports define the direction of elements in the interface.

Use of interface in test bench

```
master MASTER(tb_ctrlm.toXmit,
  tb_ctrlm.strobe,
  tb_ctrlm.ss,
  tb_ctrlm.Rcvd,
  tb_ctrlm.Ready,
  tb_ctrlm.XmitFull,
  tb_ctrlm.busy,
  spi.miso,
  spi.mosi,
  spi.sck,
  spi.ss,
  tbClkm, Rst_n);
```

the mapped netlists from design compiler do not use interfaces as ports, so one must pass each element separately as an input or output

```
// whenever received value at slave changes, save a copy for checking
always @* begin
  if (checkSSm[0]) checkRcvds = tb_ctrls[0].Rcvd;
  else if (checkSSm[1]) checkRcvds = tb_ctrls[1].Rcvd;
  end
```

Interfaces can be used in test bench expressions and assignments just as they were in the Master and Slave RTL code.

Use of interface in test bench

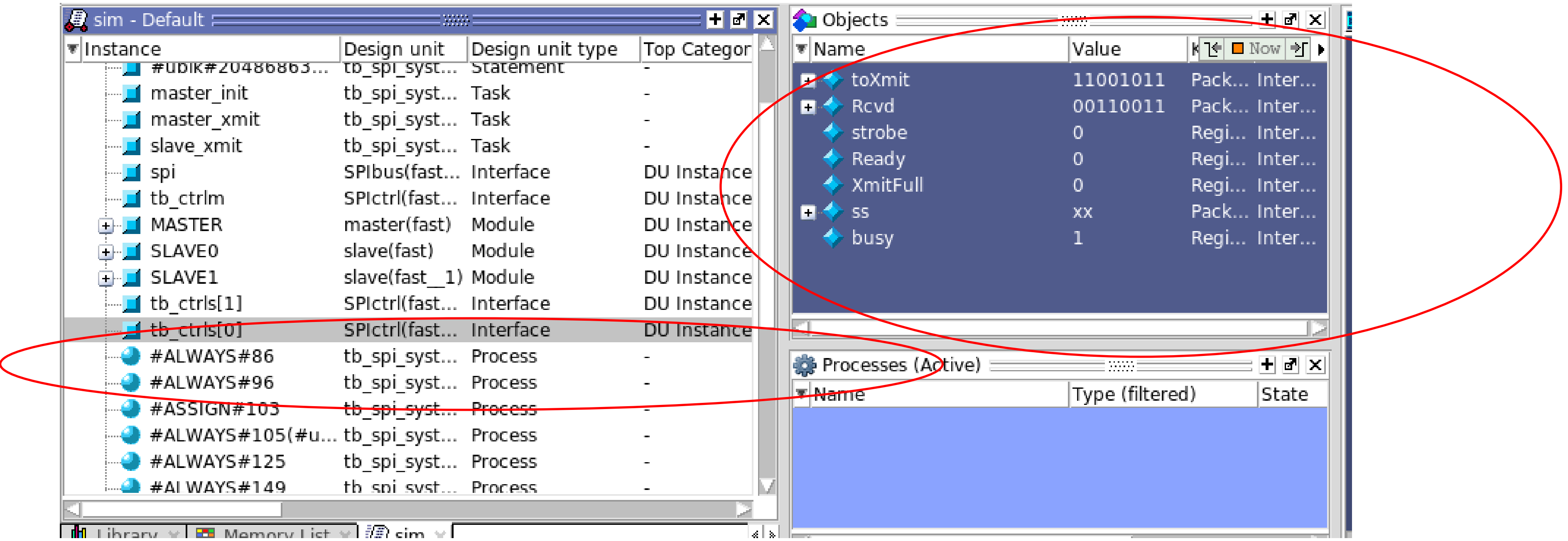
```
// send command to master via control interface
// first provide data and slave selection, then send strobe
task master_xmit (input [7:0] datin, input [1:0] slave_index, input integer delay);
  tb_ctrlm.strobe = 1'b0;
  tb_ctrlm.toXmit = datin;
  tb_ctrlm.ss = 2'b00;
  tb_ctrlm.ss[slave_index] = 2'b01;
  @(posedge tbClkm);
  #(TCLK/2);
  tb_ctrlm.strobe = 1'b1;
  @(posedge tbClkm);
  #(TCLK/2);
  tb_ctrlm.strobe = 1'b0;
  #(TCLK*delay);
endtask
```

Stimulus generation code can assign values to elements of an interface just like you would to a variable or wire that is passed to the device under test.

```
 // if slave 1 was selected and had data to transmit
 else if (tb_ctrlm.ss[1] && slaveFull[1]) begin
   asserts1_pass = (tb_ctrlm.Rcvd == checkSXmit[1]);
   asserts1_fail = ~(tb_ctrlm.Rcvd == checkSXmit[1]);
   assert (tb_ctrlm.Rcvd == checkSXmit[1])
     $display("%t correct value %b received by master from slave 1",
       $time,checkSXmit[1]);
     else
     $display("%t, incorrect value %b received by master, expected %b",
       $time,tb_ctrlm.Rcvd,checkSXmit[1]);
```

Interfaces can be used in assertion expressions

Use of interface in simulation – add to waveforms



sim - Default

Instance	Design unit	Design unit type	Top Categor
#ublk#20486863...	tb_spi_syst...	Statement	-
master_init	tb_spi_syst...	Task	-
master_xmit	tb_spi_syst...	Task	-
slave_xmit	tb_spi_syst...	Task	-
spi	SPIbus(fast...	Interface	DU Instance
tb_ctrlm	SPIctrl(fast...	Interface	DU Instance
MASTER	master(fast)	Module	DU Instance
SLAVE0	slave(fast)	Module	DU Instance
SLAVE1	slave(fast__1)	Module	DU Instance
tb_ctrls[1]	SPIctrl(fast...	Interface	DU Instance
tb_ctrls[0]	SPIctrl(fast...	Interface	DU Instance
#ALWAYS#86	tb_spi_syst...	Process	-
#ALWAYS#96	tb_spi_syst...	Process	-
#ASSIGN#103	tb_spi_syst...	Process	-
#ALWAYS#105(#u...	tb_spi_syst...	Process	-
#ALWAYS#125	tb_spi_syst...	Process	-
#ALWAYS#149	tb_spi_syst...	Process	-

Library | Memory List | sim

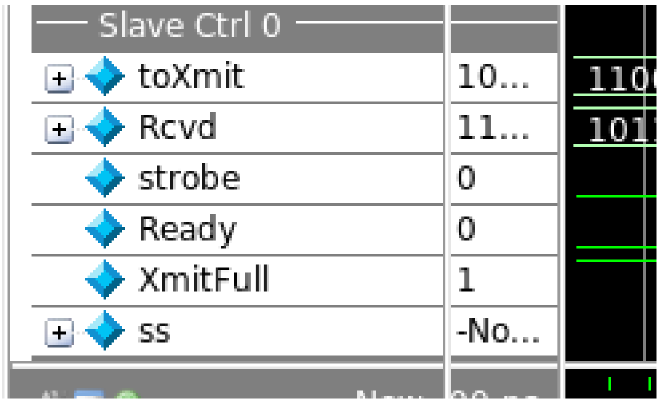
Objects

Name	Value	Now	
toXmit	11001011	Pack...	Inter...
Rcvd	00110011	Pack...	Inter...
strobe	0	Regi...	Inter...
Ready	0	Regi...	Inter...
XmitFull	0	Regi...	Inter...
ss	xx	Pack...	Inter...
busy	1	Regi...	Inter...

Processes (Active)

Name	Type (filtered)	State

All interfaces are listed in Questa/Modelsim the same way as modules, processes, etc. and you add signals to the waveform viewer in the same way.

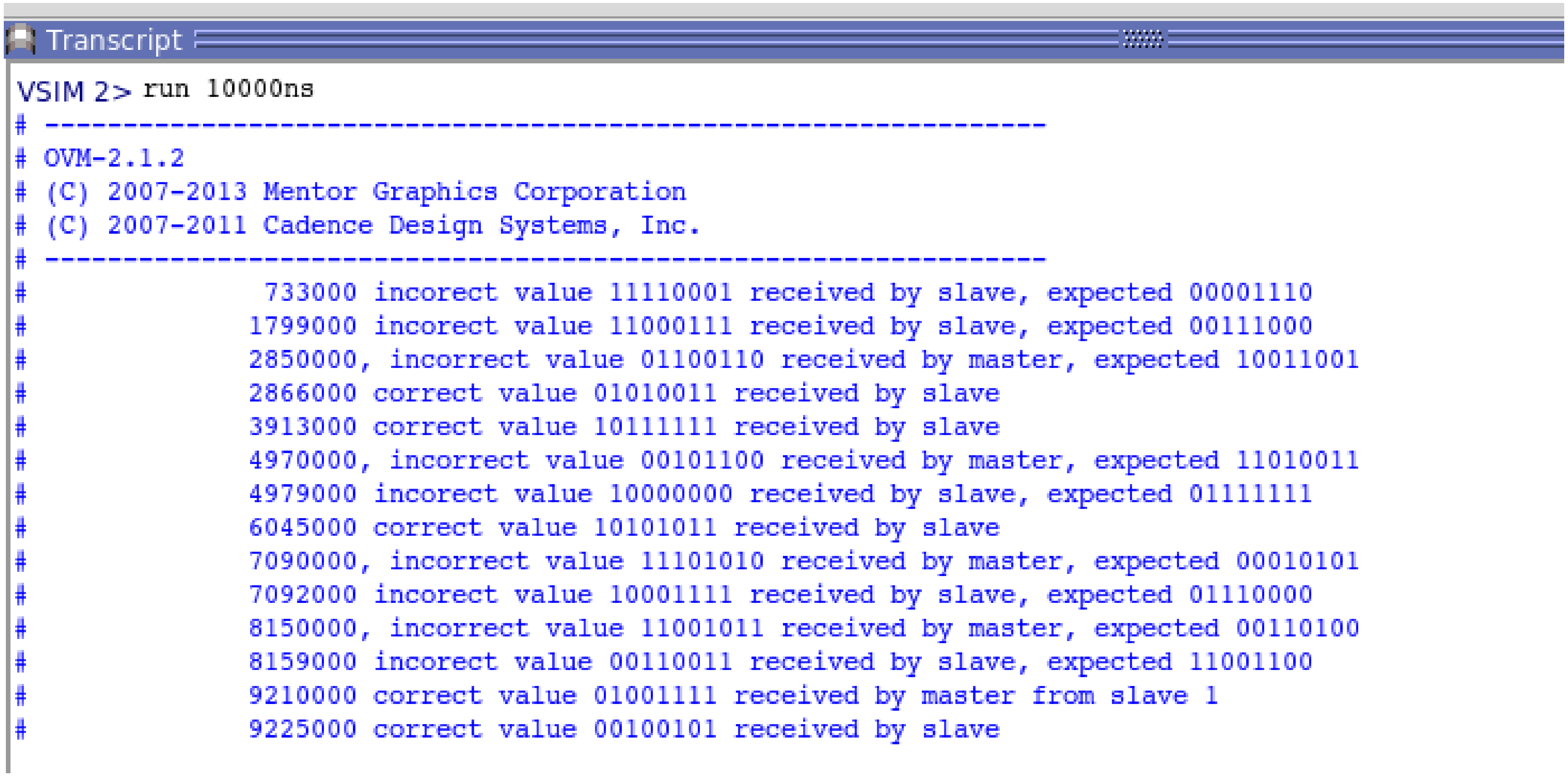


Slave Ctrl 0

toXmit	10...	110
Rcvd	11...	101
strobe	0	
Ready	0	
XmitFull	1	
ss	-No...	

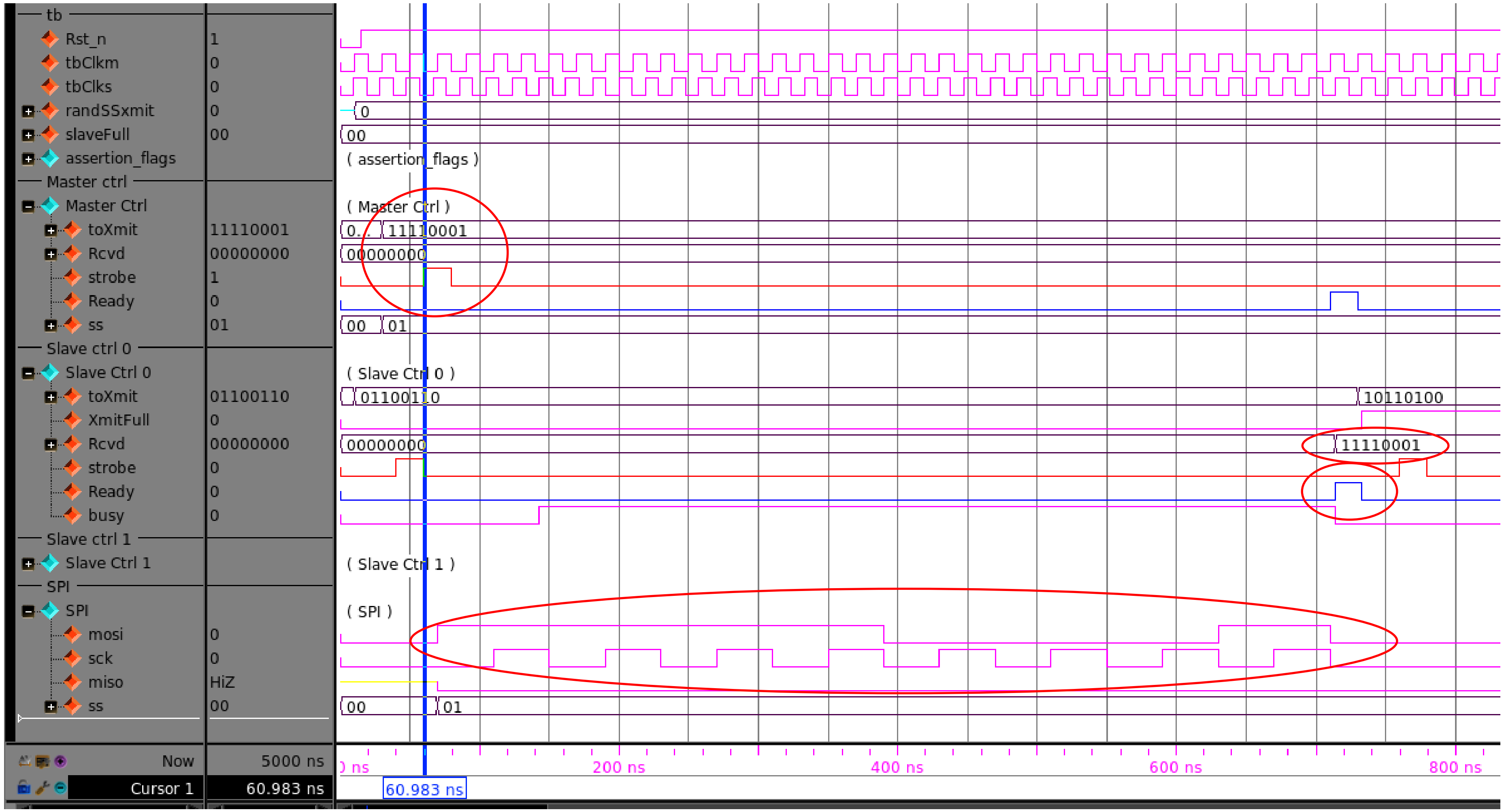
In the waveform viewer, select the signals you added, right click, and select “group” to group the interface signals together.

Simulation results – assertion messages



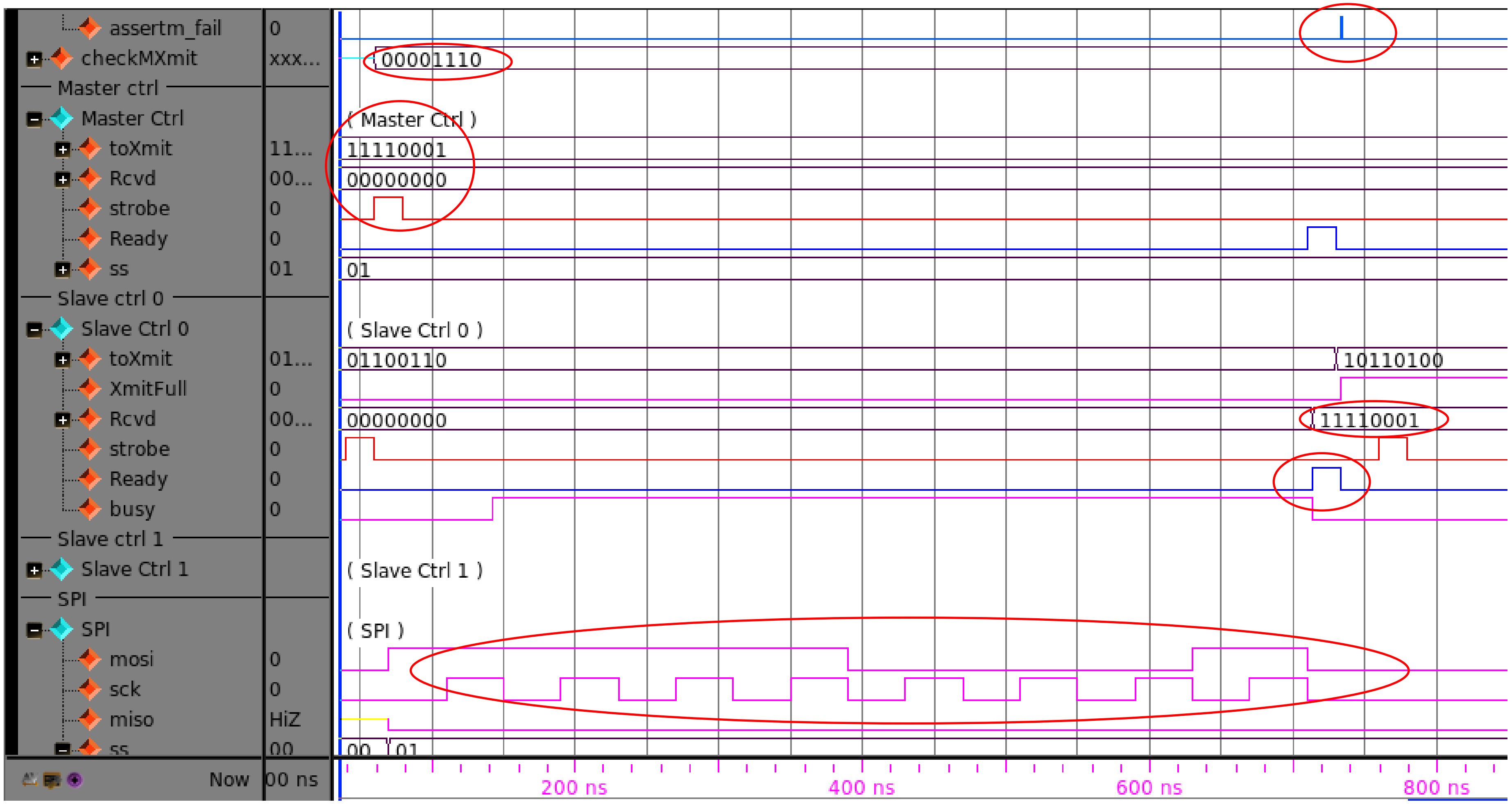
```
VSIM 2> run 10000ns
# ----------------------------------------------------------------
# OVM-2.1.2
# (C) 2007-2013 Mentor Graphics Corporation
# (C) 2007-2011 Cadence Design Systems, Inc.
# ----------------------------------------------------------------
#               733000 incorect value 11110001 received by slave, expected 00001110
#              1799000 incorect value 11000111 received by slave, expected 00111000
#              2850000, incorrect value 01100110 received by master, expected 10011001
#              2866000 correct value 01010011 received by slave
#              3913000 correct value 10111111 received by slave
#              4970000, incorrect value 00101100 received by master, expected 11010011
#              4979000 incorect value 10000000 received by slave, expected 01111111
#              6045000 correct value 10101011 received by slave
#              7090000, incorrect value 11101010 received by master, expected 00010101
#              7092000 incorect value 10001111 received by slave, expected 01110000
#              8150000, incorrect value 11001011 received by master, expected 00110100
#              8159000 incorect value 00110011 received by slave, expected 11001100
#              9210000 correct value 01001111 received by master from slave 1
#              9225000 correct value 00100101 received by slave
```

Simulation – Successful Master->Slave transfer



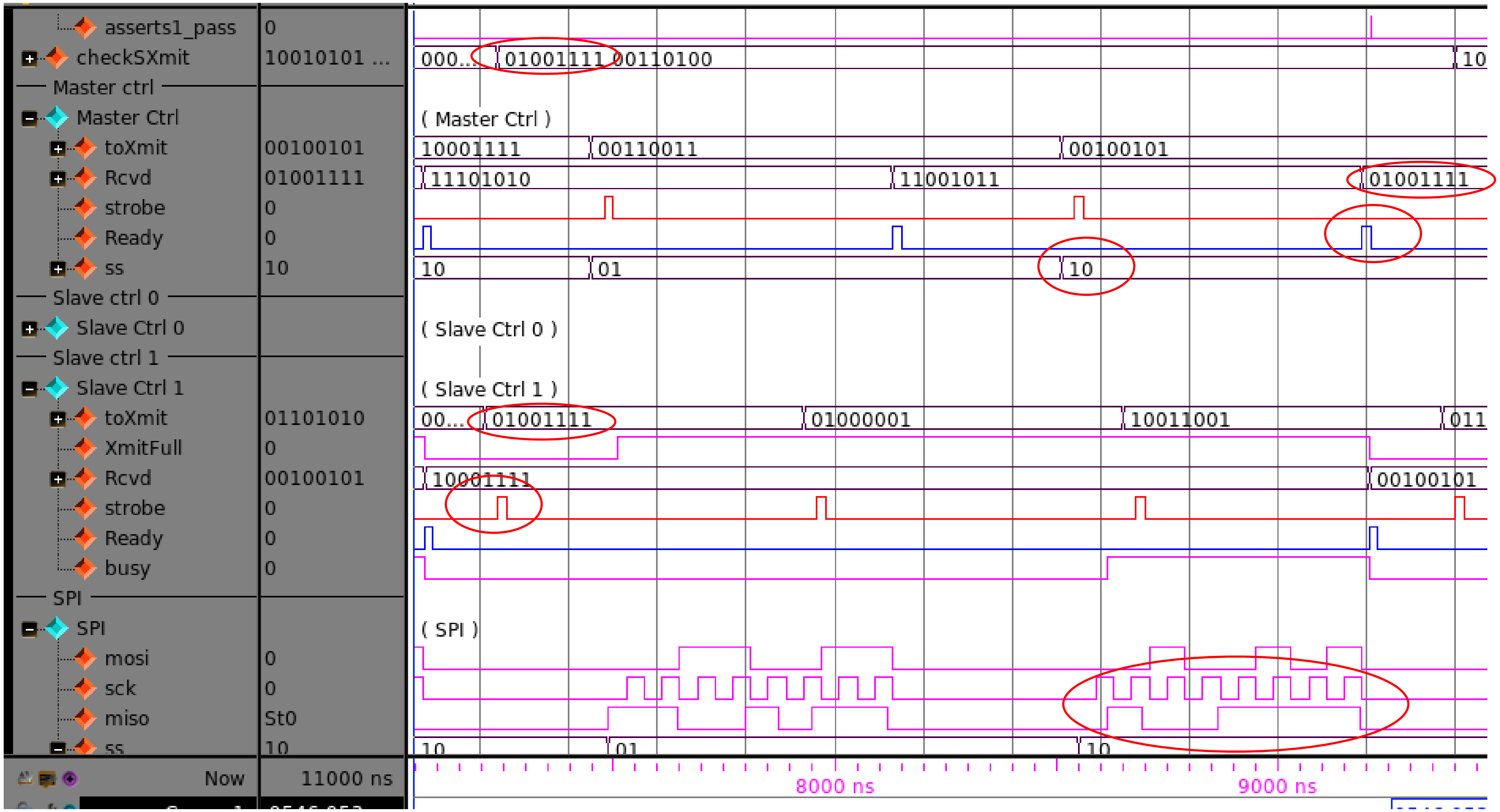
Strobe to master loads 11110001 and slave select[1:0]="01"
SPI sck and mosi proceed to send data to slave 0. Slave 0 Ready indicates data received.
Data appears on Slave 0 Rcvd.

Simulation – Failed Master->Slave transfer



Same sequence as successful transfer except that value saved by test bench is deliberately inverted so that when received value doesn't match, an error is reported.
See checkMXmit and assertm_fail.

Simulation – Successful Slave 1 ->Master transfer



Strobe to slave 1 loads value 01001111 to be transmitted next time slave 1 is selected (i.e., ss = 10) and master cycles sck eight times.

What you have learned (I hope)

- Purpose of System Verilog Interfaces
- Basic interface syntax
- Use of interface in test bench and design code
- Use of interface in simulation and verification
- Benefits of interfaces
  - Improved modularity of code
  - Improved readability of code