Joshua S. Monson
ECEn 620
Assignment 7.2

Overview

In this assignment we were to modify the System Verilog code from the previous assignment to add a transaction generator class and a driver class. These classes are each to run in their own threads and communicate using a mailbox. I created these classes as specified. I also added a clocking block to control synchronization of input driven into the DUT. The clocking block includes the grant signal (as an input) but this is not used in the test bench. Rather, the checking mechanism remained the same as in the previous assignment.

The golden and the dut have three interfaces each. It seemed like good practice to have the signals in each interface synchronized by its own clocking block. To this end, the driver forks a process for each interface; each process synchronized by its own clocking block. Since all clocking blocks trigger off of the same all of the signals are synchronized.

The remainder of this report contains:

1. Source Code for top, package, test, and driver (golden was roughly the same as last time).
2. Waveform showing correct operation.
3. Waveform showing error.
4. Transcripts showing testbench operation with and without errors.

Source Code for Top

```
`include "arb_if.sv"
module top();
   //Clock Generator
   int errors = 0;

   bit clk;
   wire rst;
   always #10ns clk = ~clk;

   arb_if arb0(clk), arb1(clk), arb2(clk);
   arb_if darb0(clk), darb1(clk), darb2(clk);

   test  t(rst,
           arb0.Test,
           arb1.Test,
           arb2.Test,
         darb0.Test,
         darb1.Test,
         darb2.Test,
         errors);
```

```
   golden_arb golden(.clk(clk),
                     .reset(rst),
                     .req0(arb0.req),
                     .req1(arb1.req),
                   .req2(arb2.req),
                   .en0(arb0.en),
                   .en1(arb1.en),
                   .en2(arb2.en),
                   .grant0(arb0.grant),
                   .grant1(arb1.grant),
                   .grant2(arb2.grant)
                  );

   arbiter arb(.clk(clk),
               .reset(rst),
                     .req0(darb0.req),
                     .req1(darb1.req),
                   .req2(darb2.req),
                   .en0(darb0.en),
                   .en1(darb1.en),
                   .en2(darb2.en),
                   .grant0(darb0.grant),
                   .grant1(darb1.grant),
                   .grant2(darb2.grant)
                  );
   always @ (negedge clk) begin
      if(arb0.grant != darb0.grant) begin
      $display("@%0d: Grant0 Mismatch!!", $time);
       errors++;
      end
      if(arb1.grant != darb1.grant) begin
      $display("@%0d: Grant1 Mismatch!!", $time);
       errors++;
      end
      if(arb2.grant != darb2.grant) begin
      $display("@%0d: Grant2 Mismatch!!", $time);
       errors++;
      end
   end


endmodule // top
```

Source Code for Test

```
import tb_pkg::*;

`define SV_RAND_CHECK(r) \
   do begin \
      if (!(r)) begin \
       $display("%s:%0d: Randomization failed \"%s\"", \
                `__FILE__, `__LINE__, `"r`");\
       $finish; \
      end \
```

```
   end while(0)

program automatic test( output bit    rst,
                                arb_if a0,
                                arb_if a1,
                                arb_if a2,
                                arb_if da0,
                                arb_if da1,
                                  arb_if da2,
                    input int errors);

  `include "Driver.sv"
  initial begin
     int count = 100;

     //Declare Genorators, Drivers and Mailboxes
     Generator G;
     Driver D;

     mailbox mbx;

     //Instanstantiate Generators and Drivers
     mbx = new();
     G = new(mbx);
     D = new(mbx);

     //Reset the Circuit
     a0.cb.req <= 0; a1.cb.req <= 0;  a2.cb.req <= 0;
     a0.cb.en <= 0;  a1.cb.en <= 0;   a2.cb.en <= 0;

     da0.cb.req <= 0; da1.cb.req <= 0;  da2.cb.req <= 0;
     da0.cb.en <= 0;  da1.cb.en <= 0;   da2.cb.en <= 0;

     rst = 1;
     @(a0.cb)
     @(a0.cb)
     @(a0.cb)
     rst = 0;
     @(a0.cb)
     //Start Genorators and Drivers
     fork
      G.run(3*count);
      D.run(count);
     join

     $display("Ran for %0d cycles", count);
     $display("Found %0d Errors.", errors);

  end

endprogram
```

Source Code for Package

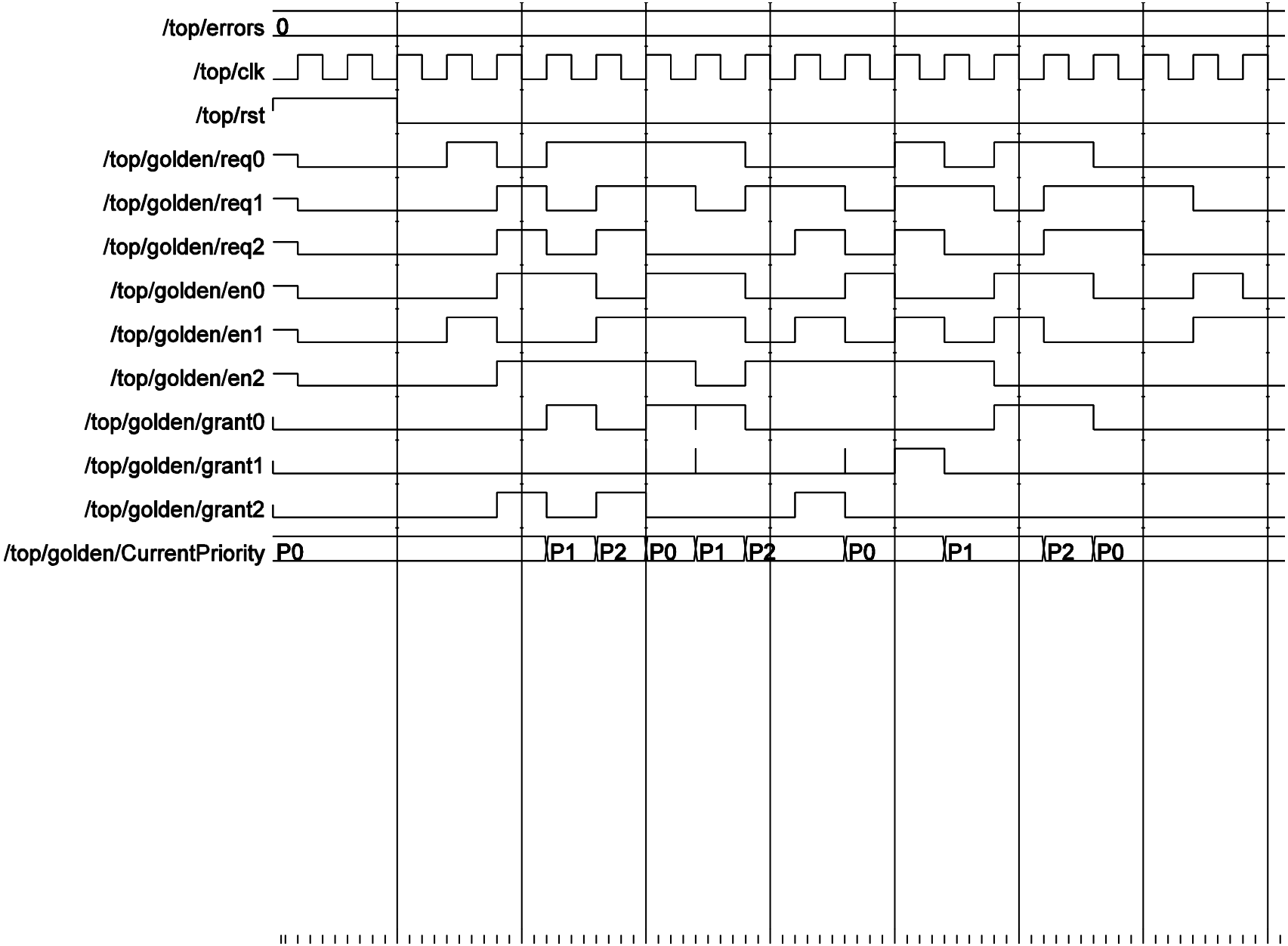
```
package tb_pkg;

class Transaction;
   rand bit req;
   rand bit en;

   /*constraint creq_dist {
      req dist { 0 :/ 30, 1 :/ 70};
   }

   constraint cen_dist {
      en dist { 0 :/ 5, 1 :/95 };
   }*/

endclass // Transaction

class Generator;

   mailbox #(Transaction) mbx;

   function new (mailbox #(Transaction) m);
      mbx = m;
   endfunction // new

   task automatic run (int count);
      Transaction tr;
      repeat (count) begin
       $display("Generating Transaction");
       tr = new();
       tr.randomize();
       mbx.put(tr);
      end
   endtask;

endclass

endpackage
```

Source Code For Driver

```
import tb_pkg::*;

class Driver;
   mailbox #(Transaction) mbx;

   function new (mailbox #(Transaction) m);
      mbx = m;
   endfunction // new

   task run(input int count);
      Transaction tr0, tr1, tr2;
      //Initialize the transaction in case
      // the mailbox is empty on the first try.
```

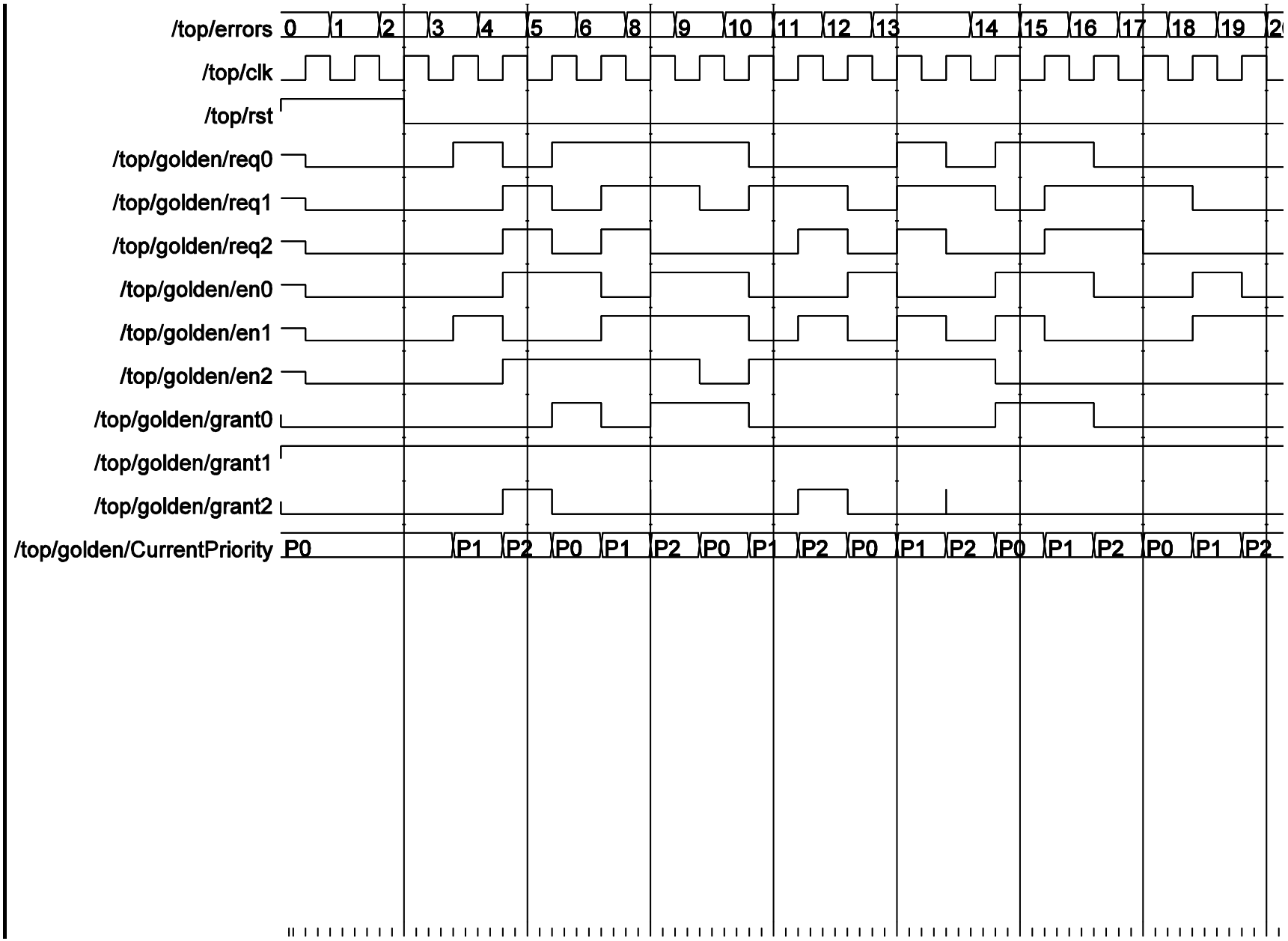
```
      tr0 = new(); tr1 = new(); tr2 = new();
      repeat (count) begin
       //If the Mailbox is empty disable
       //  the port.
       if(!mbx.try_get(tr0)) begin
         tr0.en = 0;
         $display("No transaction in mailbox");
       end
       if(!mbx.try_get(tr1)) begin
         tr1.en = 0;
           $display("No transaction in mailbox");
       end
        if(!mbx.try_get(tr2)) begin
         tr2.en = 0;
         $display("No transaction in mailbox");
       end
       //Drive Golden and DUT Signals
       fork
          begin
             a0.cb.req <= tr0.req;
             a0.cb.en <= tr0.en;
             @(a0.cb);
          end
          begin
             a1.cb.req <= tr1.req;
             a1.cb.en <= tr1.en;
             @(a1.cb);
          end
          begin
             a2.cb.req <= tr2.req;
             a2.cb.en <= tr2.en;
             @(a2.cb);
          end
          begin
             da0.cb.req <= tr0.req;
             da0.cb.en <= tr0.en;
             @(da0.cb);
          end
          begin
             da1.cb.req <= tr1.req;
             da1.cb.en <= tr1.en;
             @(da1.cb);
          end
          begin
             da2.cb.req <= tr2.req;
             da2.cb.en <= tr2.en;
             @(da2.cb);
          end
       join
      end
   endtask // run

endclass
```

Waveform & Transcipt Without Errors


/top/errors
0
/top/clk
/top/rst
/top/golden/req0
/top/golden/req1
/top/golden/req2
/top/golden/en0
/top/golden/en1
/top/golden/en2
/top/golden/grant0
/top/golden/grant1
/top/golden/grant2
/top/golden/CurrentPriority
P0
P1
P2
P0
P1
P2
P0
P1
P2
P0

```
# view wave
# .main_pane.wave.interior.cs.body.pw.wf
# do wave.do
# run -all
# Ran for 100 cycles
# Found 0 Errors.
```

Waveform & Transcript With Errors


/top/errors
0
1
2
3
4
5
6
8
9
10
11
12
13
14
15
16
17
18
19
/top/clk
/top/rst
/top/golden/req0
/top/golden/req1
/top/golden/req2
/top/golden/en0
/top/golden/en1
/top/golden/en2
/top/golden/grant0
/top/golden/grant1
/top/golden/grant2
/top/golden/CurrentPriority
P0
P1
P2
P0
P1
P2
P0
P1
P2
P0
P1
P2
P0
P1
P2
P0
P1
P2

```
# view wave
# .main_pane.wave.interior.cs.body.pw.wf
# do wave.do
# @20: Grant1 Mismatch!!
# @40: Grant1 Mismatch!!
# @60: Grant1 Mismatch!!
# @80: Grant1 Mismatch!!
# @100: Grant1 Mismatch!!
# @120: Grant1 Mismatch!!
# @140: Grant1 Mismatch!!
# @140: Grant2 Mismatch!!
# @160: Grant1 Mismatch!!

…

# @2000: Grant1 Mismatch!!
# @2020: Grant1 Mismatch!!
# @2040: Grant1 Mismatch!!
# @2060: Grant1 Mismatch!!
# Ran for 100 cycles

# Found 108 Errors.
```