Lab 1: Lab 1 Verification

Daniel Gutierrez and Ryan Dosanjh

1. Verifying a Multiplier

Error Reporting:

- **Upon entering the 'DONE' state, if the output signal product_o holds an incorrect product, report a BAD_PRODUCT error.**

```
assert (itf.product == i * j)
                else begin
                    $error ("TB: BAD_PRODUCT error detected, muliplicand =
%d multiplier = %d, product = %d", itf.multiplicand, itf.multiplier,
itf.product);
                    report_error(BAD_PRODUCT);
                end
```

- **If the ready o signal is not asserted after a reset, report a NOT_READY error.**

```
itf.reset_n <= 0;
        ##1;
        assert (itf.rdy == 1'b1)
        else begin
            $error("TB: NOT_READY after reset");
            report_error(NOT_READY);
        end
```

- **If the ready_o signal is not asserted upon completion of a multiplication, report a NOT_READY error.**

```
assert (itf.rdy == 1'b1)
                else begin
                    $error("TB: NOT_READY");
                    report_error(NOT_READY);
                end
```

Multiplier TB Code:

```
import mult_types::*;

module testbench(multiplier_itf.testbench itf);

add_shift_multiplier dut (
    .clk_i          ( itf.clk          ),
    .reset_n_i      ( itf.reset_n      ),
    .multiplicand_i ( itf.multiplicand ),
    .multiplier_i   ( itf.multiplier   ),
    .start_i        ( itf.start        ),
    .ready_o        ( itf.rdy          ),
    .product_o      ( itf.product      ),
    .done_o         ( itf.done         )
);

assign itf.mult_op = dut.ms.op;
default clocking tb_clk @(negedge itf.clk); endclocking

// error_e defined in package mult_types in file types.sv
// Asynchronously reports error in DUT to grading harness
function void report_error(error_e error);
    itf.tb_report_dut_error(error);
endfunction : report_error


// Resets the multiplier
task reset();
    itf.reset_n <= 1'b0;
    ##5;
    itf.reset_n <= 1'b1;
    ##1;
endtask : reset

// DO NOT MODIFY CODE ABOVE THIS LINE

/* Uncomment to "monitor" changes to adder operational state over time */
//initial $monitor("dut-op: time: %0t op: %s", $time, dut.ms.op.name);
```

```
initial itf.reset_n = 1'b0;

initial begin
    reset();
    // itf.reset_n = 0;
    /********************* Your Code Here ***************************/
    $display("TB: TESTING start_i signal asserted while in run states");
    itf.start <= 0;
    itf.multiplicand <= 50;
    itf.multiplier <= 200;
    ##1;
    itf.start <= 1;
    ##1;
    itf.start <= 0;
    ##2;
    if (itf.mult_op == ADD) begin
        $display("TB: start asserted in ADD");
        itf.start <= 1;
        ##1;
        assert (itf.mult_op == SHIFT)
        else begin
            $error("TB: start affected the run");
            report_error(BAD_PRODUCT);
        end
        ##1;
        itf.start <= 0;
    end
    ##1;
    if (itf.mult_op == SHIFT) begin
        $display("TB: start asserted in SHIFT");
        itf.start <= 1;
        ##1;
        assert (itf.mult_op == ADD)
        else begin
            $error("TB: start affected the run");
            report_error(BAD_PRODUCT);
        end
        ##1;
        itf.start <= 0;
```

```
    end
    ##10;
    $display("TB: PASSED start asserted in run state");
    reset();

    $display("TB: Testing reset in run states");
    itf.start <= 0;
    itf.multiplicand <= 50;
    itf.multiplier <= 200;
    ##1;
    itf.start <= 1;
    ##1;
    itf.start <= 0;
    ##2;
    if (itf.mult_op == ADD) begin
        $display("TB: reset asserted in ADD");
        itf.reset_n <= 0;
        ##1;
        assert (itf.rdy == 1'b1)
        else begin
            $error("TB: NOT_READY after reset");
            report_error(NOT_READY);
        end
        itf.reset_n <= 1;
    end

    itf.start <= 1;
    ##1;
    itf.start <= 0;
    ##1;
    if (itf.mult_op == SHIFT) begin
        $display("TB: reset asserted in SHIFT");
        itf.reset_n <= 0;
        ##1;
        assert (itf.rdy == 1'b1)
        else begin
            $error("TB: NOT_READY after reset");
            report_error(NOT_READY);
        end
```

```
        itf.reset_n <= 1;
    end
    $display("TB: PASSED reset asserted in run state");
    reset();

    $display("TB: TESTING every possible comb of muliplicand and
multiplier");
    for (int i=0; i<=8'hff; i++) begin
        for (int j=0; j<=8'hff; j+=itf.rdy) begin
            itf.multiplicand <= i;
            itf.multiplier <= j;
            itf.start <= itf.rdy;
            ##1;
            if (itf.done == 1'b1) begin
                assert (itf.product == i * j)
                else begin
                    $error ("TB: BAD_PRODUCT error detected, muliplicand =
%d multiplier = %d, product = %d", itf.multiplicand, itf.multiplier,
itf.product);
                    report_error(BAD_PRODUCT);
                end
                assert (itf.rdy == 1'b1)
                else begin
                    $error("TB: NOT_READY");
                    report_error(NOT_READY);
                end
            end
        end
    end
    $display("TB: PASSED every possible comb");

    /*****************************************************************/
    itf.finish(); // Use this finish task in order to let grading harness
                  // complete in process and/or scheduled operations
    $error("Improper Simulation Exit");
end

endmodule : testbench
```

2. *Verifying a FIFO*

Error Reporting:

- **Asserting reset_n_i at @(tb_clk) should result in ready_o being high at @(posedge clk_i). If it is not, report the appropriate error.**

```
assert (itf.data_o != 999)
        else begin
            $error ("-----TB: SIMULTANEOUSLY----\n %0d: itf.yumi %0d
itf.data_0 ,error detected", itf.yumi, itf.data_o);
            report_error (RESET_DOES_NOT_CAUSE_READY_O);
```

- **When asserting yumi_i at @(tb_clk) when data is ready, the value on data_o is the CORRECT value. If not, report the appropriate error.**

```
@(posedge itf.clk);
        assert (itf.data_o == i)
            else begin
                $error ("-----TB: BAD DEQUEUE----\n %0d: int i %0d:
itf.yumi %0d itf.data_0 ,error detected", i, itf.yumi, itf.data_o);
                report_error (INCORRECT_DATA_O_ON_YUMI_I);
```

FIFO Testbench Code:

```
`ifndef testbench
`define testbench

import fifo_types::*;

module testbench(fifo_itf itf);

fifo_synch_1r1w dut (
    .clk_i     ( itf.clk     ),
    .reset_n_i ( itf.reset_n ),

    // valid-ready enqueue protocol
    .data_i    ( itf.data_i  ),
    .valid_i   ( itf.valid_i ),
    .ready_o   ( itf.rdy     ),
```

```
    // valid-yumi deqeueue protocol
    .valid_o    ( itf.valid_o ),
    .data_o     ( itf.data_o  ),
    .yumi_i     ( itf.yumi    )
);

// Clock Synchronizer
default clocking tb_clk @(negedge itf.clk); endclocking

task reset();
    itf.reset_n <= 1'b0;
    ##(10);
    itf.reset_n <= 1'b1;
    ##(1);
endtask : reset

function automatic void report_error(error_e err);
    itf.tb_report_dut_error(err);
endfunction : report_error

// DO NOT MODIFY CODE ABOVE THIS LINE

task enqueue();
    $display("Enqueueing");
    //filling up queue
    for (int i = 0; i < cap_p; i++) begin
        itf.data_i <= i; // Incrementing by 1
        itf.valid_i <= 1;
        @(posedge itf.clk);
    end
    itf.valid_i <= 0;
endtask : enqueue


task dequeue();
    //Should release in same order
    $display("Dequeueing");
    //should release queue in same order it was queued originally
    for (int i = 0; i < cap_p; i++) begin
        itf.yumi <= 1;
```

```
        @(posedge itf.clk);
        assert (itf.data_o == i)
            else begin
                $error ("-----TB: BAD DEQUEUE----\n %0d: int i %0d:
itf.yumi %0d itf.data_0 ,error detected", i, itf.yumi, itf.data_o);
                report_error (INCORRECT_DATA_O_ON_YUMI_I);
            end
    end
    itf.yumi <= 0;
endtask : dequeue


task simultaneously();
    $display("simultaneously");
    itf.data_i <= 999;
    itf.valid_i <= 1;
    itf.yumi <= 1;
    @(posedge itf.clk);
    assert (itf.data_o != 999)
        else begin
            $error ("-----TB: SIMULTANEOUSLY----\n %0d: itf.yumi %0d
itf.data_0 ,error detected", itf.yumi, itf.data_o);
            report_error (RESET_DOES_NOT_CAUSE_READY_O);
        end
    itf.valid_i <= 0;
    itf.yumi <= 0;
endtask : simultaneously


initial begin
    reset();
    /********************** Your Code Here **********************/
    // Feel free to make helper tasks / functions, initial / always
blocks, etc.
    $display("Enqueueing");
    enqueue();
    $display("Dequeueing");
    dequeue();
    $display("Together");
    simultaneously();
```

```
    simultaneously();
    simultaneously();
    /*************************************************************/
    // Make sure your test bench exits by calling itf.finish();
    itf.finish();
    $error("TB: Illegal Exit ocurred");
end

endmodule : testbench
`endif
```