R·I·T | KATE GLEASON College of ENGINEERING

Engineering Report

From: Tommy Choephel

Date: December 7, 2021

Subject: Project 3 Engineering Report

Introduction

A "Touch-Tone" phone may use DTMF (Dual Tone Multi Frequency) technology to make calls, navigate in-call menus, or transmit information between network entities. DTMF technology achieves this by sending two tones for each button pushed: one of high frequency and the other of lower frequency. The DTMF receiver is the brain that makes sense of the tones. Within the DTMF, a memory arbiter acts as the entity that arbitrates and grants different device elements (tdsp and dma in this scope) access to the memory under certain conditions. Those conditions are modelled using a state machine. The state machine is modelled using Verilog and a testbench was created for it in the same HDL. Then simulations were performed where fault cases were forced to trigger errors. Then the memory arbiter module was integrated with the greater DTMF device and more simulations were performed, both pre and post synthesis and HUMM and SPI test modes. The arbiter model was a success as all the simulations showed that it was functioning as expected. Simulating the DTMF Receiver in SPI mode revealed the following digits: 1-800-862-4522#.

Memory Access Bus Arbiter (ARB) Module

As stated in the introduction, the memory access bus arbiter determines who and when to give access to entities that demand it. It is a state machine with five states, which are as follows: IDLE, GRANT_DMA, CLEAR, DMA_PRI, and GRANT_TDSP. The inputs dma_breq and tdsp_breq are the request signals. Depending on the sequence those requests are received and with consideration to priority of one over the other depending on machine states, the ARB module grants access to the memory bus and thus to the memory. The test bench created for this ARB simulates different input requests and keeps track of how many grants there should be. A conditionally compiled statement was written to stop the simulation if an expected grant signal wasn't received within a maximum number of clock cycles.

arb.v RTL Source Code

—----------

```
/*
 *
 * Author:  Mark A. Indovina & Tommy Choephel
 *          Rochester, NY, USA
 *
 */

module arb (
            reset,
            clk,
            dma_breq,
            dma_grant,
            tdsp_breq,
            tdsp_grant,
            scan_in0,
            scan_en,
            scan_out0
    );

/*
 *
 * DMA/ TDSP bus arbiter
 *
 */

input
    reset,              // system reset
    clk,                // system clock
    dma_breq,           // dma controller bus request
    tdsp_breq ;         // tdsp bus request

output
    dma_grant,          // dma controller bus grant
    tdsp_grant ;        // tdsp bus grant

input
    scan_in0,           // test scan mode data input
    scan_en;            // test scan mode enable

output
    scan_out0;          // test scan mode data output

reg dma_grant;          //registers for dma, tdsp, state
reg tdsp_grant;
```

```
reg [2:0] state;

//
// explicit state machine states
//
`include "./include/arb.h"

// Tommy Edits Henceforth
// states:    ARB_IDLE ARB_GRANT_TDSP    ARB_GRANT_DMA  ARB_CLEAR ARB_DMA_PRI

// reg [3:0] state;

always @( posedge clk or posedge reset)
   if (reset)
   begin // async reset operation
      dma_grant <= 0;
      tdsp_grant <= 0;
      state <= `ARB_IDLE;
   end
   else
   begin
     dma_grant <= 0;
     tdsp_grant <= 0;

      case (state)
               `ARB_IDLE :
               begin
               if (tdsp_breq && dma_breq) // tdsp priority
               begin
               tdsp_grant <= 1;
               state <= `ARB_GRANT_TDSP;
               end
               else if (tdsp_breq && !dma_breq) // tdsp request
               begin
               tdsp_grant <= 1;
               state <= `ARB_GRANT_TDSP;
               end
               else if (!tdsp_breq && dma_breq) //breq request
               begin
               dma_grant<= 1;
               state <= `ARB_GRANT_DMA;
               end
               else
               state <= `ARB_IDLE;
               end

               `ARB_GRANT_TDSP :
               begin
```

```
if(!tdsp_breq)
begin
tdsp_grant <= 1;
state <= `ARB_CLEAR;
end
else
begin
tdsp_grant <= 1;
state <= `ARB_GRANT_TDSP;
end
end

`ARB_CLEAR :
begin
if (tdsp_breq && dma_breq) // tdsp priority
begin
tdsp_grant <= 1;
state <= `ARB_GRANT_TDSP;
end
else if (dma_breq && !tdsp_breq) // dma request
state   <= `ARB_DMA_PRI;
else if (tdsp_breq && !dma_breq) // tdsp request
begin
tdsp_grant <= 1;
state <= `ARB_GRANT_TDSP;
end
else
begin
tdsp_grant <= 1;
state <= `ARB_CLEAR;
end
end

`ARB_DMA_PRI :
begin
if (!tdsp_breq && !dma_breq) // no requests
state <= `ARB_IDLE;
else if (tdsp_breq && dma_breq) // dma priority
begin
dma_grant <= 1;
state <= `ARB_GRANT_DMA;
end
else if (tdsp_breq && !dma_breq) // tdsp request
begin
tdsp_grant <= 1;
state <= `ARB_GRANT_TDSP;
end
else if (!tdsp_breq && dma_breq) // dma request
```

```
            begin
            dma_grant <= 1;
            state <= `ARB_GRANT_DMA;
            end
            else
            state <= `ARB_DMA_PRI;
            end

            `ARB_GRANT_DMA :
            begin
            if (!dma_breq)
            begin
            tdsp_grant <= 1;
            state <= `ARB_CLEAR;
            end
            else
            begin
            dma_grant <= 1;
            state <= `ARB_GRANT_DMA;
            end
            end

            default :
            begin
            state <= `ARB_IDLE;
            end
    endcase
    end

endmodule // arb
```

—----------

arb_test.v Testbench Source Code

```
/*
 *
 * Author:  Mark A. Indovina & Tommy Choephel
 *              Rochester, NY, USA
 *
 */


`timescale 1ns / 1ns

module test;

wire dma_grant, tdsp_grant;
wire scan_out0;

reg clk, dma_breq, reset, tdsp_breq;
reg scan_in0, scan_en;
reg dma_grant_error, tdsp_grant_error; // TC error flags active high
reg [3:0] dma_clk_cnt, tdsp_clk_cnt;       // TC 4-bti clock counters to count to 15

arb top(
     .reset(reset),
     .clk(clk),
     .dma_breq(dma_breq),
     .dma_grant(dma_grant),
     .tdsp_breq(tdsp_breq),
     .tdsp_grant(tdsp_grant),
     .scan_in0(scan_in0),
     .scan_en(scan_en),
     .scan_out0(scan_out0)
     );


reg [4: 0]
     dma_wait,
     tdsp_wait ;

integer
     i,
     j,
     dma_cnt,
     tdsp_cnt ;
```

```
wire
    grant = dma_grant | tdsp_grant ;

initial
begin
    $timeformat( -9, 2, "ns", 16);
`ifdef SDFSCAN

    $sdf_annotate("sdf/arb_tsmc18_scan.sdf", test.top);
`endif

    clk = 1'b0;
    dma_breq = 1'b0;
    reset = 1'b0;
    tdsp_breq = 1'b0;
    scan_in0 = 1'b0;
    scan_en = 1'b0;
    dma_cnt = 0 ;
    tdsp_cnt = 0 ;
    dma_grant_error = 1'b0; // TC
    tdsp_grant_error = 1'b0; // TC
    dma_clk_cnt = 4'b0; // TC 4-bits just for counting up 15
    tdsp_clk_cnt = 4'b0; // TC
    dma_wait = $random ;
    tdsp_wait = $random ;

    @(negedge clk)
    reset = 1'b1 ;
    repeat (2)
    @(negedge clk) ;
    @(negedge clk)
    reset = 1'b0 ;
    repeat (2)
    @(posedge clk) ;

    repeat (256)
    begin
    @(posedge clk)
    dma_wait <= $random ;
    tdsp_wait <= $random ;
    fork
            dma_request ;
            tdsp_request ;
     dma_check ; // TC DMA check happens here
    tdsp_check ; // TC TDSP check happens here
    join
    repeat (4)
```

```
                @(posedge clk) ;
        end
        repeat (4)
        @(posedge clk) ;
        if (dma_cnt != tdsp_cnt)
        begin
        $display(" ** Fails simulation!");
        $display(" ** 256 Individual Bus request cycles generated,");
        $display(" ** (#tdsp grants == %d) != (#dma grants == %d)", tdsp_cnt, dma_cnt);
        end
        else
        begin
        $display(" ** Passes simulation!");
        $display(" ** 256 Individual Bus request cycles generated,");
        $display(" ** (#tdsp grants == %d) == (#dma grants == %d)", tdsp_cnt, dma_cnt);
        end
        $stop ;
end

always #20
        clk = ~clk ;

task dma_request ;
        begin
        repeat (dma_wait)
                @(posedge clk) ;
        dma_breq <= 1 ;
        $display("%t DMA Bus Request", $time);
        for (i = 0 ; i < (dma_wait + tdsp_wait + 10) ; i = i + 1)
                @(posedge clk)
                if (dma_grant)
                begin
                dma_cnt = dma_cnt + 1 ;
                i = (dma_wait + tdsp_wait + 10) ;
                end
        @(posedge clk)
        dma_breq <= 0 ;
        @(posedge clk);
        end
endtask

// Tommy edits1 begin
task dma_check ;
        begin
        if(!dma_breq) //if dma_breq isn't asserted...
        wait(dma_breq); // wait for it to be asserted

        if (tdsp_grant) //if tdsp_grant asserted
```

```
    wait(!tdsp_grant) // wait for tdsp_grant to be de-asserted and then...

    for (dma_clk_cnt = 0; !dma_grant; dma_clk_cnt = dma_clk_cnt + 1) //start counting c-cycles until dma_grant is asserted
            begin
            @(posedge clk);
                    if (dma_clk_cnt >= 3) // Seems like max c-cycles is 3 from state diagram but tb waveform shows it's 2
                    begin
                    dma_grant_error = 1;
                    $display("%t DMA GRANT ERROR", $time);
                    `define dma_error; //conditionally compile in code to stop sim on dma_error
                    end
                    `ifdef dma_error
                    if (dma_grant_error)
                    $stop;
                    `endif
            end
    end
endtask
// Tommy edits1 end

task tdsp_request ;
    begin
    repeat (tdsp_wait)
            @(posedge clk) ;
    tdsp_breq <= 1 ;
    $display("%t TDSP Bus Request", $time);
    for (j = 0 ; j < (dma_wait + tdsp_wait + 10) ; j = j + 1)
            @(posedge clk)
            if (tdsp_grant)
            begin
            tdsp_cnt = tdsp_cnt + 1 ;
            j = (dma_wait + tdsp_wait + 10) ;
            end
    @(posedge clk)
    tdsp_breq <= 0 ;
    @(posedge clk);
    end
endtask


// Tommy edits2 begin
task tdsp_check ;
    begin
    if(!tdsp_breq) //if tdsp_breq isn't asserted...
    wait(tdsp_breq); // wait for it to be asserted
```

```
    if (dma_grant) //if dma_grant asserted
    wait(!dma_grant) // wait for it to be de-asserted and then...

    for (tdsp_clk_cnt = 0; !tdsp_grant; tdsp_clk_cnt = tdsp_clk_cnt + 1) //start counting c-cycles
until tdsp_grant is asserted
            begin
            @(posedge clk);
                    if (tdsp_clk_cnt >= 3) // Seems like max c-cycles is 3 from state diagram but tb
waveform shows it's 2
                    begin
                    tdsp_grant_error = 1;
                    $display("%t TDSP GRANT ERROR", $time);
                    `define tdsp_error; //conditionally compile in code to stop sim on dma_error
                    end
                    `ifdef tdsp_error
                    if (tdsp_grant_error)
                    $stop;
                    `endif
            end
    end
endtask
// Tommy edits2 end

endmodule
```

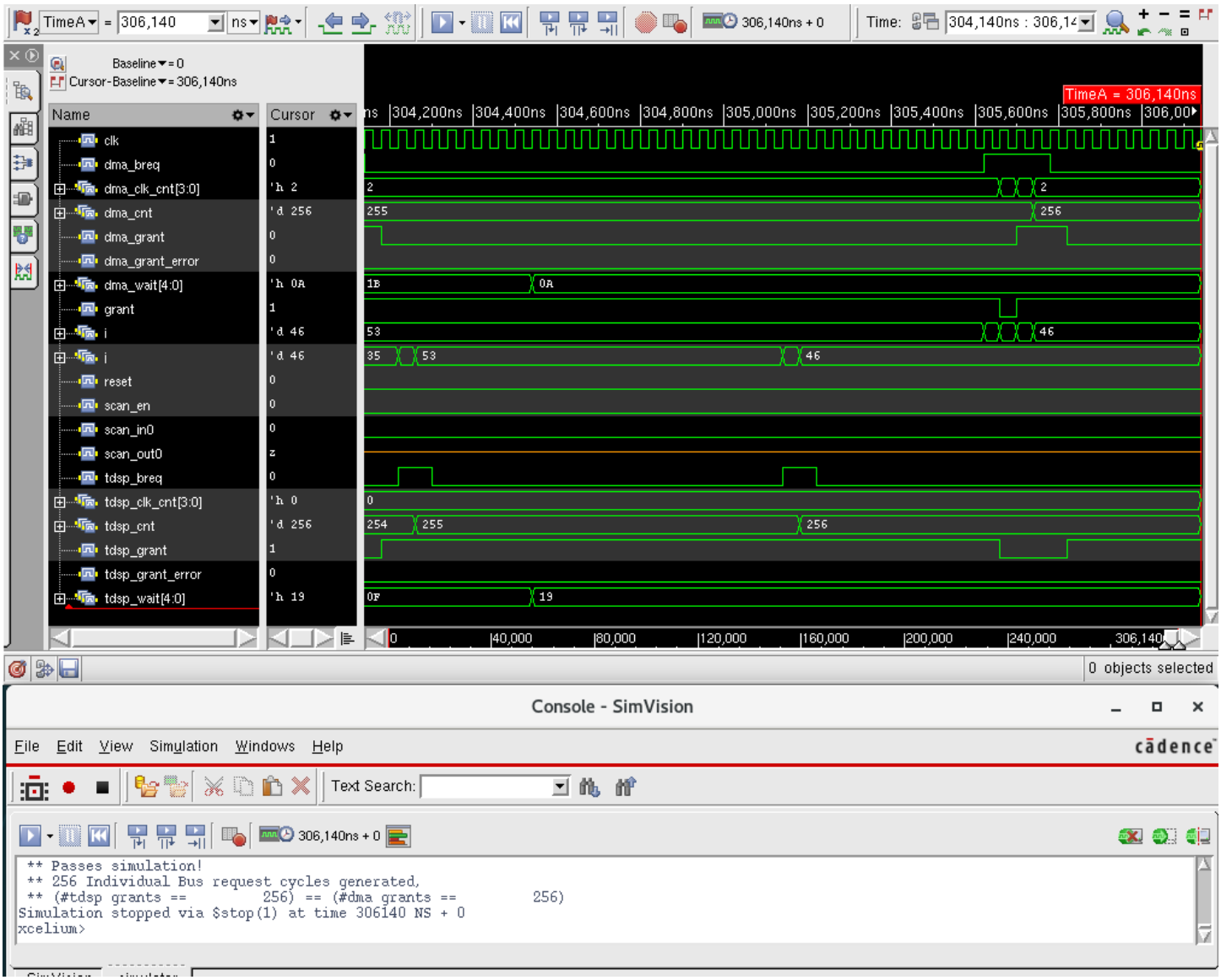

Figure showing memory ARB functioning correctly as the number of requests match the number of grants for both tdsp and dma.

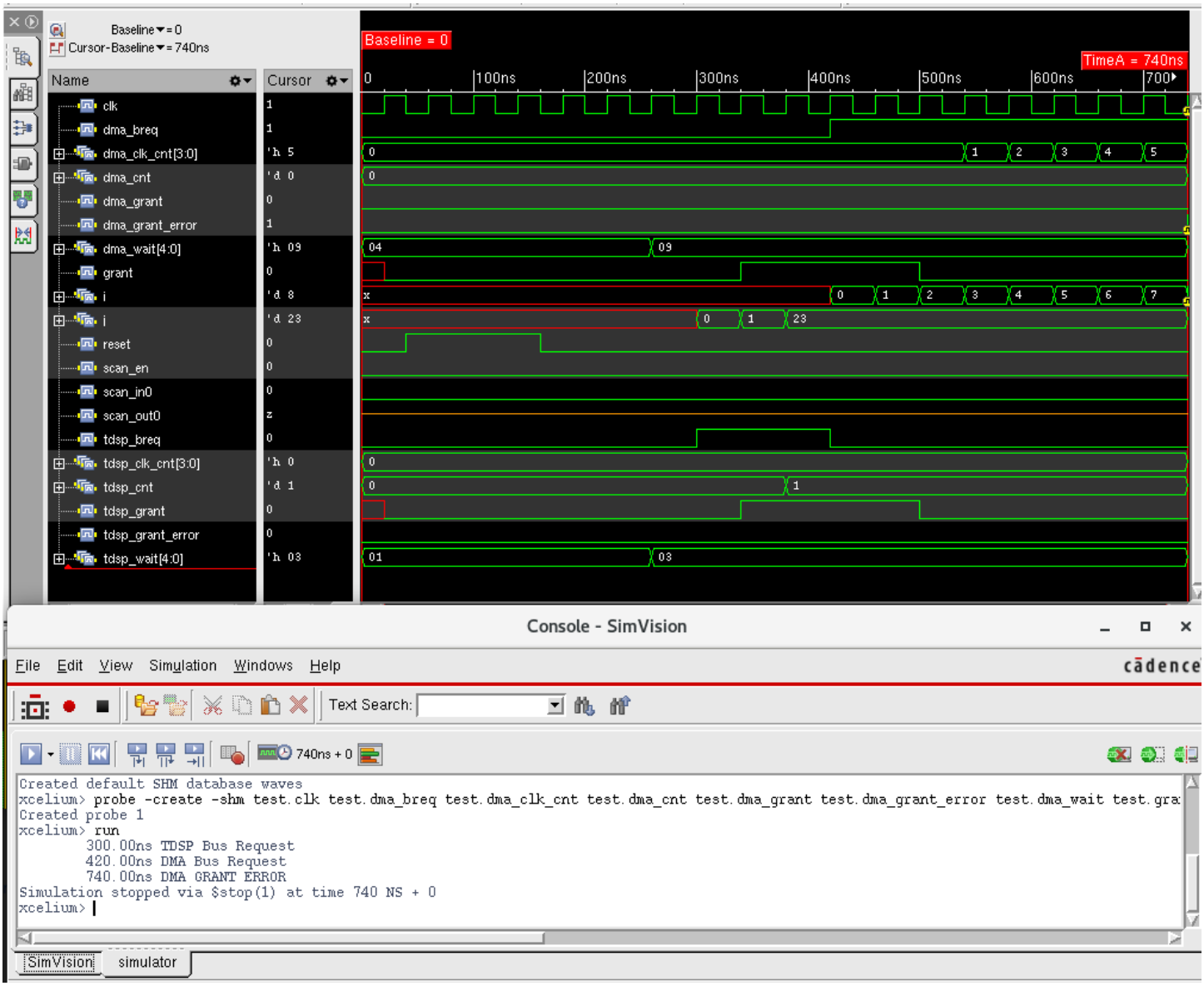

Figure showing intentionally driven to failed simulation as dma_grant was forced to be zero, thus perpetually delaying dma_grant assertion and producing “DMA GRANT ERROR.”

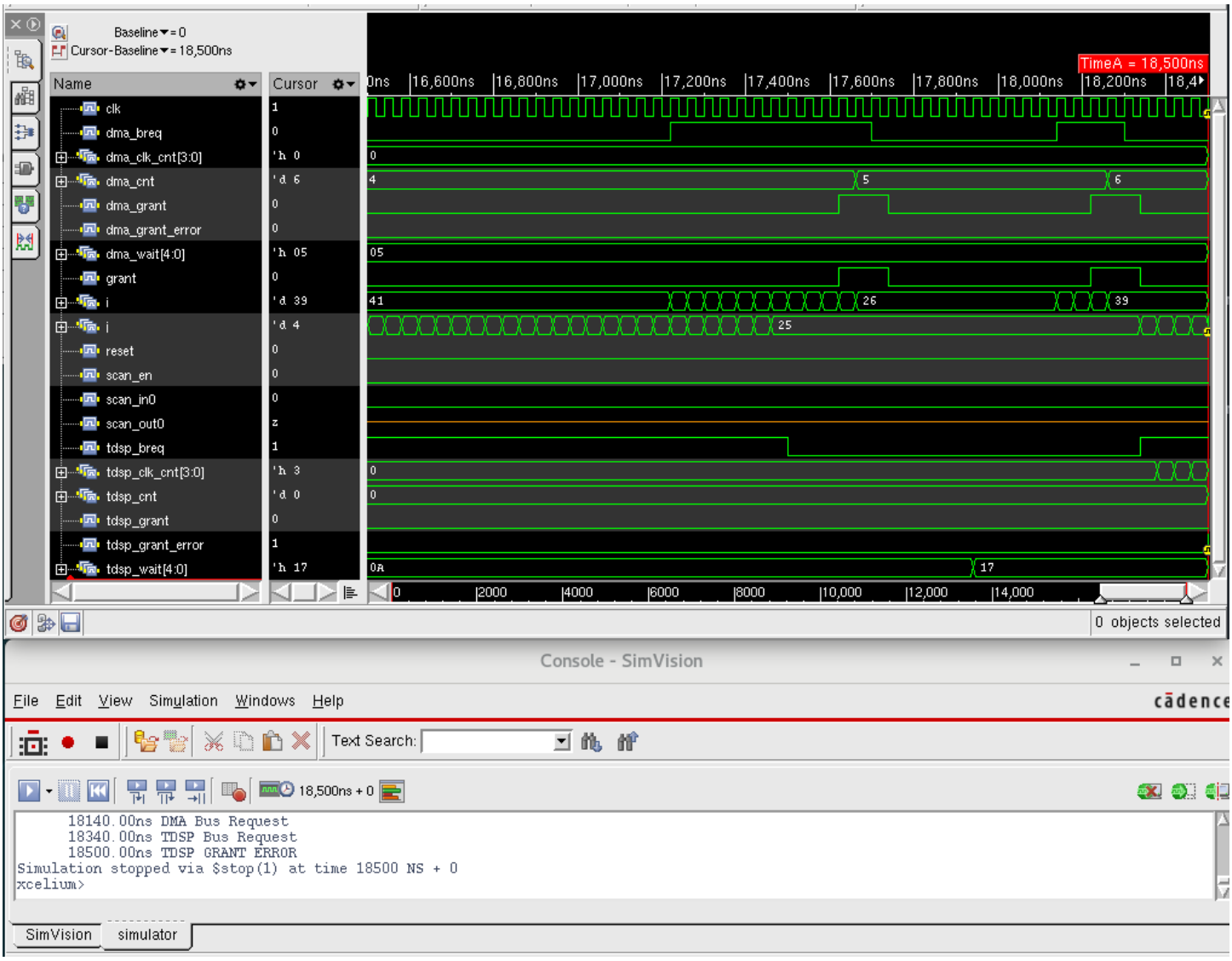

Figure showing intentionally driven to failed simulation as tdsp_grant was forced to be zero, thus perpetually delaying tdsp_grant assertion and producing “TDSP GRANT ERROR.”

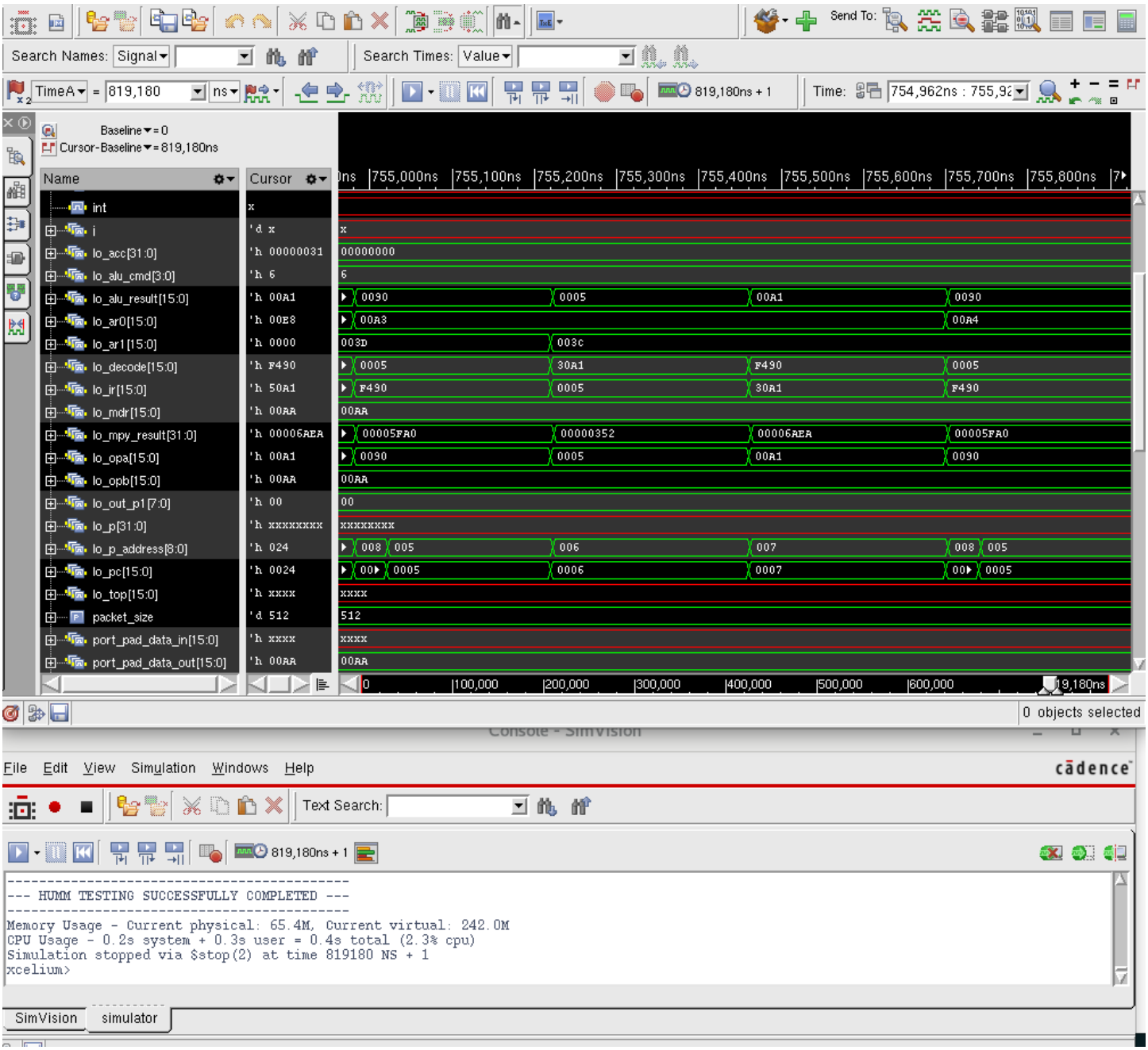

Figure showing pre-synthesis HUMM simulation console and waveform. Note that the console verifies "HUMM TESTING SUCCESSFULLY COMPLETED."

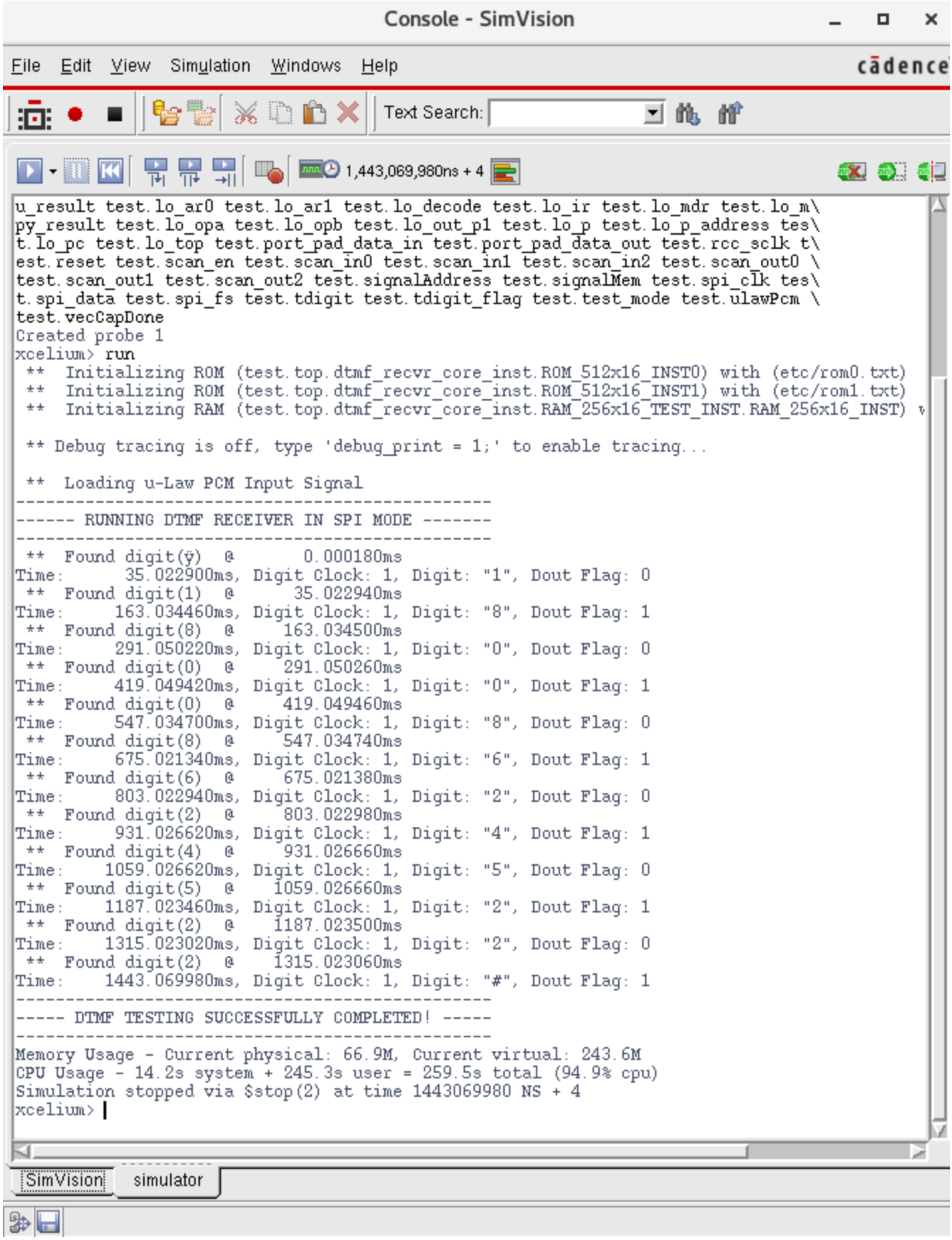

Figure showing pre-synthesis SPI Modde simulation console. The digit is: 1800-862-4522#.

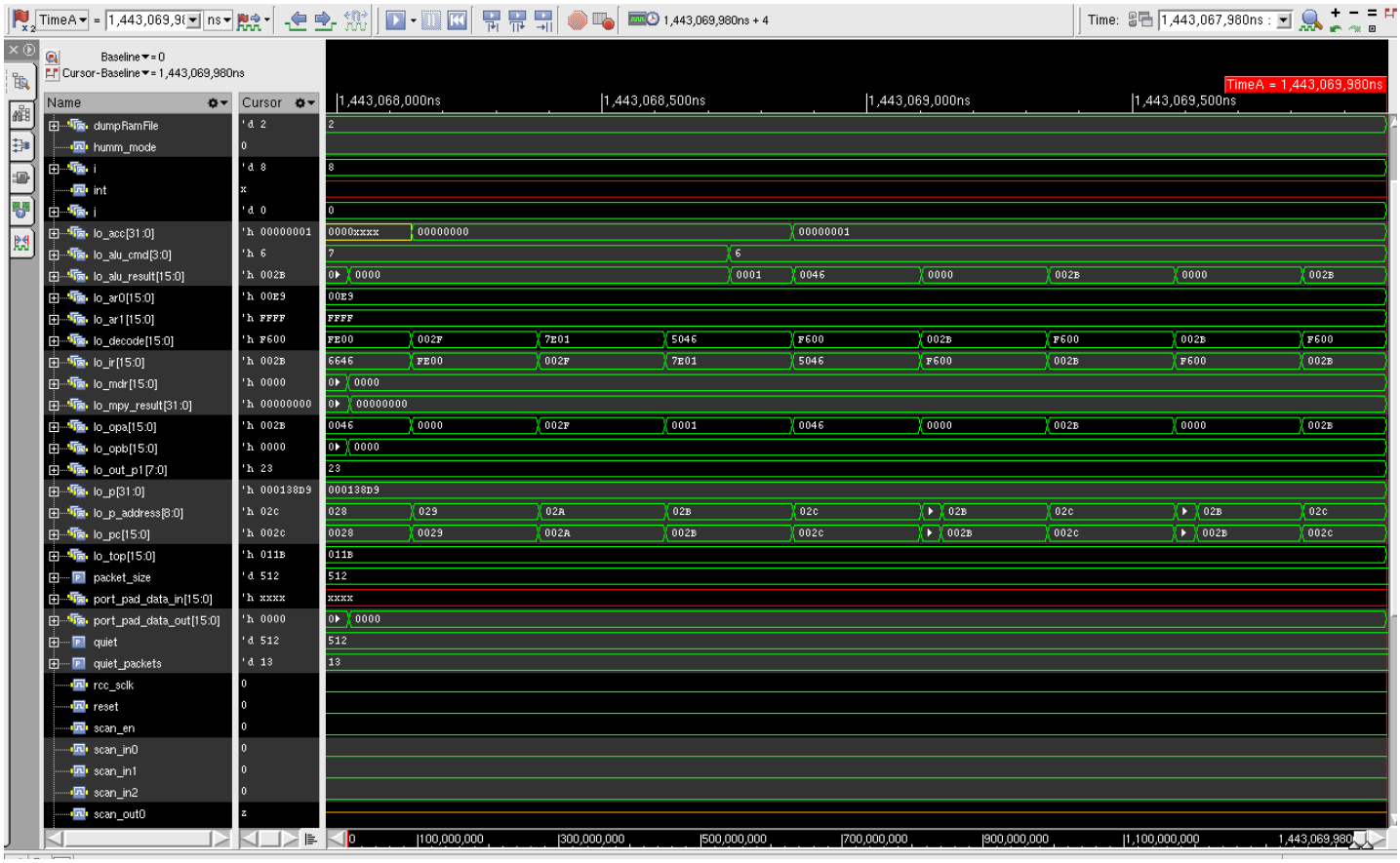

Figure showing pre-synthesis SPI simulation waveform to accompany the console above.

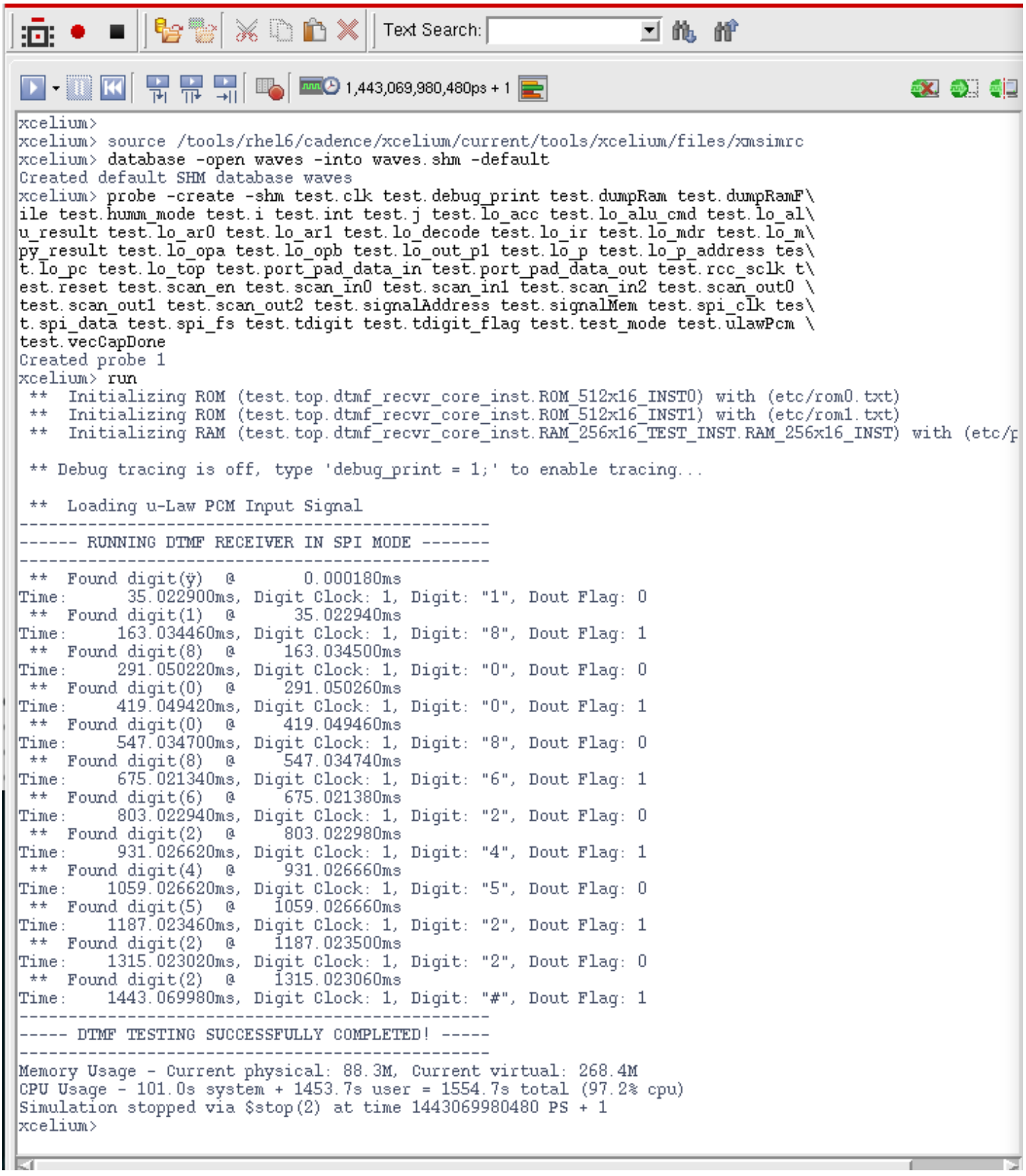

Figure of post-synthesis netlist simulation for the SPI mode console. Though the simulation took significantly longer, the digit found is the same 1800-862-4522#.

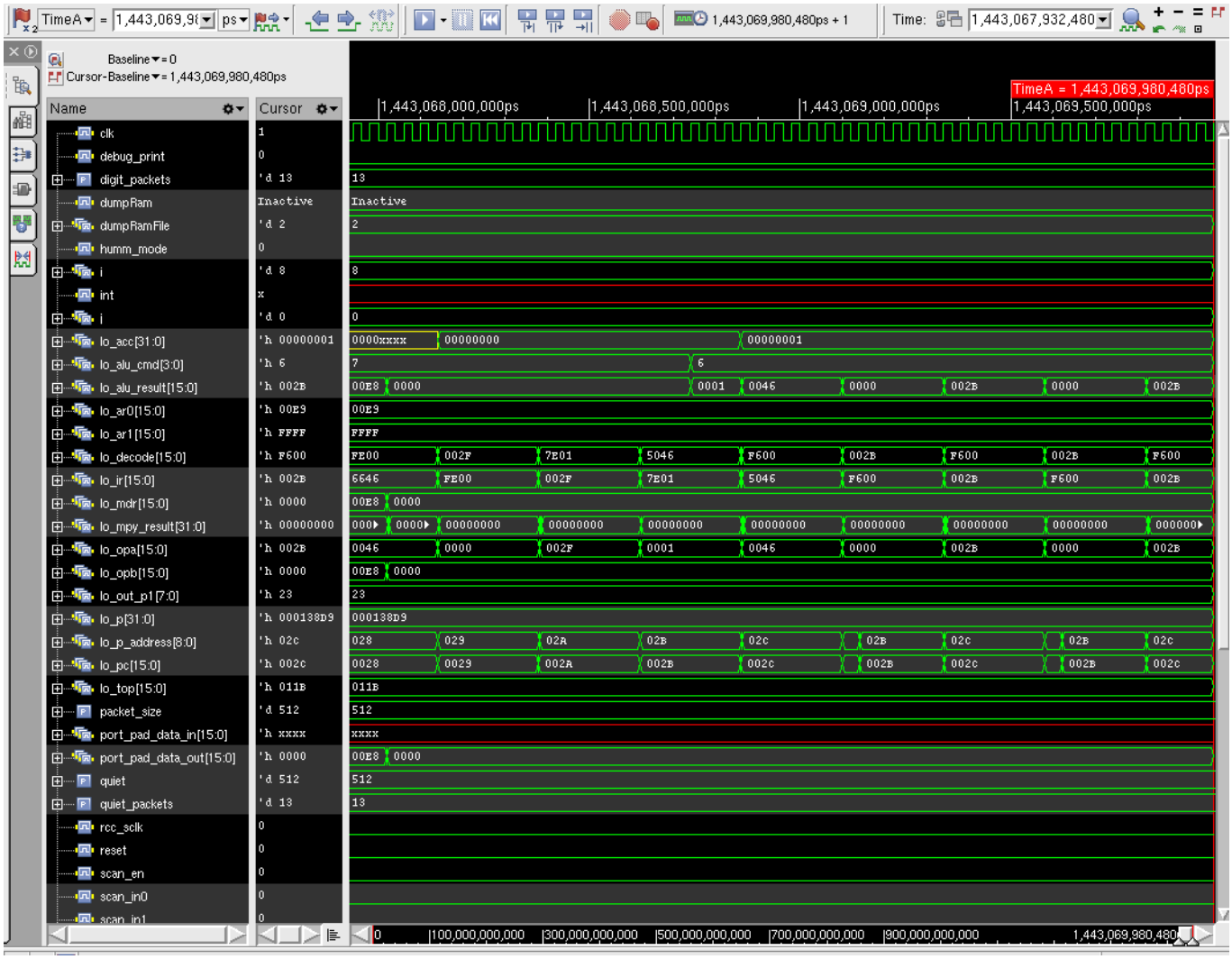

Figure of post-synthesis netlist simulation for the SPI mode waveform. Though the simulation took significantly longer, the waveform is roughly the same.

Summary and Conclusions

This project covered the HDL functional design of a fairly complicated memory arbiter using a state machine. A testbench was also designed for it to test the logic of the access granting process for both request cases. The project started out as quite daunting because of our sporadic and limited experience with Verilog for creating complex devices. However, after inspired and stubborn tries, the state machine proved rather straightforward. The testbench proved to be a greater challenge. Luckily, the virtue of testbenches being that they can be simulated and waveforms observed, I caught the troublesome mistake of assigning a single bit to a variable that should be allowed to count up to 5. This mistake prevented the DUT from reaching the erroneous state to stop the simulation. It was a classic case of being stuck in an infinite loop. Careful examination of the waveform signals show this discrepancy and that was promptly fixed. Other difficulty with this project included missing certain key deliverables such as the assembly code part of the project. This is due to poor time management, substantial workload saved from project 2, as well as my desire to do a thorough job on what is done, rather than shoddily complete all work. Given better time management on my part, less busy work (and reattempts) on prior projects, I am confident that I can complete this project fully.