Systems On-Chip

Lab 3

Synthesis and DFT

Bologna Master Degree in Electrical and Computer Engineering (MEEC)
Instituto Superior Técnico
1st semester, 2nd period

Group 8
David Benjamin Krause | 104898
Duarte Miguel de Aguiar Pinto e Morais Marques | 96523
Eduardo Daniel Roque Martins | 96923

December 19th 2022

1 Introduction

Digital synthesis (and DFT) is necessary to design state machines that are synthesized (implemented with logic gates) automatically. Testing analog and digital circuits requires controllability and observability; for testing purposes, pins are frequently added. **Scan** is an example of a digital test which will be added in this laboratory assignment to the previously designed battery charger controller. In this type of test, an input pin SE alters its value between '1' and '0' in order to alternate between Scan Mode and Normal Mode, respectively. In the former, a state is forced onto the scan register(s), while the latter then determines the resulting state, depending on the current state and the inputs. By successively changing from one mode to another, observability is added to the secondary inputs/outputs and **faults** can be found. These faults are related to the fact that some LSA0 (Line Stuck-At 0) and LSA1 (Line Stuck-At 1) might be undetectable in the circuit being tested. Other types of digital tests include **wire bonding testing** and **multiplexing test pins** - which will also be implemented in this laboratory assignment. In a wire bond test, pairs of digital input and output pins are defined; each digital output pin is driven with the logic value defined at the corresponding digital input pin. A test multiplexer, on the other hand, leads to a reduction in the number of pins required for test purposes, since existing pins are reused for the testing functionality.

Initially, the tutorials regarding the 4 bits counter were repeated. After this, the directory `DIGITAL/SYNTHESIS_SCAN/BATCHARGER` was created, in which the following files were included:

- `BATCHARGERctr_scan.v` - with the same code of `BATCHARGERctr.v` (the new controller design implemented in Lab 1), having the inputs `se` and `si` and the output `so` been added for scan;
- `synth.tcl` - script which calls `synth_initial.tcl` and `synth_final.tcl` in order to perform the synthesis of the circuit;
- `synth_initial.tcl` and `synth_final.tcl`;
- `dft_setup.tcl` - in which the DFT rules are checked, the scan chains are defined an the scan-in/scan-out are created.

Moreover, additional files were added to the directory `DIGITAL/SIMULATION/BATCHARGER`:

- `mux_test.v` - which includes the test mux module and performs the proper connections between its inputs and the added output pin;
- `BATCHARGER_test_with_mux.v` - complete charger description, now including the wire bond test, test multiplexer and scan functionalities, due to the newly created modules `BATCHARGERctr_-scan` and `mux_test`;
- `BATCHARGER_test_with_mux_tb.v` - testbench for the complete charger, in order to test the wire bond and test mux connections;
- `sim_synth` - to perform **logic simulation** of the controller;
- `sim_rtl_scan` - for the simulation performed with the full charger (having the wire bond test and test multiplexer been implemented), using the controller defined at gate level.

2 Adding scan to the charger controller

In order to perform **logic synthesis** with the charger controller, the scripts `synth_initial.tcl`, `synth_final.tcl`, `synth.tcl` and `dft_setup.tcl` (shown in Figures 1 to 4) were initially obtained from the 4-bit counter directory. In the latter two files, no modifications had to be made. In `synth_-final.tcl`, in order to eventually check other aspects of the performance obtained from synthesis, new report commands were added. Additionally, the paths in the code had to be properly modified.

The command `gui_show` can be added in order to visualize the schematic view in the graphical interface after synthesis has been performed. Finally, in `synth_initial.tcl`, the proper paths were also included. Moreover, the outputs described in `BATCHARGERctr_scan.v` (which are `cc`, `tc`, `cv`, `imonen`, `vmonen` and `tmonen`, as described in the previous laboratory assignments) were now included in the code. As mentioned before, this controller file is very similar to the previously used `BATCHARGERctr.v` (having only `si`, `se` and `so` been added for scan), thus it is not included in this report.

```
# Paths
set DIG_DIR "~/DIGITAL"
set RTL_DIR "$DIG_DIR/SIMULATION/BATCHARGER/src"
set LIB_DIR "/opt/ic_tools/pdk/faraday/umc130/HS/fsc0h_d/2009Q1v3.0/GENERIC_CORE/FrontEnd/synopsys"
set script_dir "$DIG_DIR/SYNTHESIS_SCAN/BATCHARGER"

set_db init_lib_search_path $LIB_DIR
set_db init_hdl_search_path $RTL_DIR

# Reads
read_libs fsc0h_d_generic_core_ss1p08v125c.lib
read_hdl -v2001 BATCHARGERctr_scan.v

# Elaboration
elaborate

# Constrains
# --- clock and delays
create_clock -name clk -period 10 [get_ports clk]
# --- wire cap
# Output bus external load 500fF
set_db [get_db ports cc] .external_wire_cap 500
set_db [get_db ports tc] .external_wire_cap 500
set_db [get_db ports cv] .external_wire_cap 500
set_db [get_db ports imonen] .external_wire_cap 500
set_db [get_db ports vmonen] .external_wire_cap 500
set_db [get_db ports tmonen] .external_wire_cap 500

# Synthesis
source ${script_dir}/dft_setup.tcl
syn_generic
syn_map

# Aditional
connect_scan_chain -preview
connect_scan_chain
syn_opt

# output a description of the scan chains in DEF file, will also be done by the write_design command
file mkdir synthDb
write_scandef > synthDb/final.scan.def

# Write the tcl script for ATPG
write_dft_atpg -tcl -library ../../SIMULATION/verilog_libs/fsc0h_d_generic_core_21.lib.src -directory ./atpg_scripts

# the directory ./atpg_scripts is created and modus can be used to source the runmodus.atpg.tcl script
```

Figure 1: File `synth_initial.tcl` created for logic synthesis with scan.

```
# Output paths
set OUT_DIR "./"

# Check obtained performance
report_timing > $OUT_DIR/timing_report
report_power > $OUT_DIR/power_report
report_area > $OUT_DIR/area_report
report_gates > $OUT_DIR/gates_report
report_port * > $OUT_DIR/port_report
report_clocks > $OUT_DIR/clocks_report
report_units > $OUT_DIR/units_report

# Outputs
write_hdl -language v2001 > $OUT_DIR/BATCHARGERctr_synth.v
write_sdc -strict > $OUT_DIR/BATCHARGERctr_synth.sdc
write_db $OUT_DIR/BATCHARGERctr_synth.db

#gui_show
```

Figure 2: File `synth_final.tcl` created for logic synthesis with scan.

```
source ./synth_initial.tcl
source ./synth_final.tcl
```

Figure 3: Script `synth.tcl`, which calls `synth_initial.tcl` and `synth_final.tcl` to perform the synthesis of the circuit, using the command `source synth.tcl` in the genus command line.

```
#---------------------------------------------------------------------------------------------------
# configure the design for scan insertion
# please use write_template -dft to have more information on
# the other options and refer to the DFT userguide
#---------------------------------------------------------------------------------------------------
define_dft shift_enable -lec_value 0 -active high -name se se

# check the dft rules and verify if all the DFF are scanable
check_dft_rules

# define the scan chains and create the scanin/scanout ports when necessary
define_dft scan_chain -sdi si -sdo so -shift_enable se -domain clk -edge rise -non_shared_output

# check the dft rules and verify if all the DFF are scanable
check_dft_rules
```

Figure 4: File `dft_setup.tcl` created for logic synthesis with scan.

Inside the directory `DIGITAL/SYNTHESIS_SCAN/BATCHARGER`, once the search path had been configured and environment variables had been obtained with `source /opt/ic_tools/init/init-genus 19-14-isr4`, the synthesis software **genus** was initialized by running `genus` in the command line. By sourcing the script shown in Figure 3 (thus creating the directory `atpg_scripts`), the schematic view obtained as a result of the logic synthesis was obtained (with `gui_show`). As shown in Figure 6, the controller design was successfully implemented using a considerable number of gates (such as OR, XOR and MUX gates) and interconnections between them.

Once this had been done, the **Automatic Test Pattern Generation (ATPG)** is performed, with the intent of generating stimulus (test vectors) to ensure detection of possible Line Stuck-At (LSA) faults. From this, the value of $\text{Fault Coverage} = \text{Detected Faults}/\text{Total Faults}$ can be obtained. By configuring the search path and other environment variables with `source /opt/ic_-tools/init/init-modus19-12-hf000` and running `modus -no_gui` followed by `source ./atpg_-scripts/runmodus.atpg.tcl` (sourcing the tcl script created by genus), the log file partially shown in Figure 7 was obtained. As seen here, a `fault coverage` of **99.53%** was determined from the circuit shown in Figure 6. This means that, in case this controller design were to be physically fabricated, 0.47% of faults due to LSA0 and LSA1 would never be detected - additional faults will also be present due to the fabrication process itself. These undetectable faults are due to the circuit design itself, thus the code implemented in `BATCHARGERctr_scan.v` could be tweaked in order to possibly obtain an even higher fault coverage. Something else worth noting is that the total number of faults is 1707, which is an **uneven** number (as is the number of detected faults, 1699). At each wire, only LSA0 and LSA1 are possible, which would therefore lead to an even number of faults. However, this uneven number is due to the modus ATPG tools, which does not target the complete possible number of LSA0/LSA1; the fault list is also reduced by fault equivalence. Therefore, the total number of faults is not simply twice the number of nets.

Using the different report files created from the code implemented in `synth_final.tcl`, various information about the performance after synthesis can be obtained. For this laboratory assignment, the most relevant is shown in `gates_report` (included in Figure 5), which gives an estimated area of **2016.000** (most libraries use μm^2 for this parameter). Additionally, the area used for each of the gate instances shown in Figure 6 (in total, 191 instances are used) is included. This values for the area are determined based on the specifications of the library `fsc0h_d_generic_core_ss1p08v125c.lib`, due to the `read_libs` command included in `synth_initial.tcl` (as shown in Figure 1).

```
============================================================
  Generated by:           Genus(TM) Synthesis Solution 19.14-s108_1
  Generated on:           Dec 18 2022  12:24:39 pm
  Module:                 BATCHARGERctr_scan
  Operating conditions:   _nominal_ (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================


   Gate    Instances    Area                Library
-------------------------------------------------------------------
AN2B1CHD           5    38.400    fsc0h_d_generic_core_ss1p08v125c
AN2B1HHD           1    11.520    fsc0h_d_generic_core_ss1p08v125c
AN2EHD             2    12.800    fsc0h_d_generic_core_ss1p08v125c
AN2HHD             1     8.960    fsc0h_d_generic_core_ss1p08v125c
AN2KHD             1    15.360    fsc0h_d_generic_core_ss1p08v125c
AN4B1BHD           3    34.560    fsc0h_d_generic_core_ss1p08v125c
AN4CHD             1    12.800    fsc0h_d_generic_core_ss1p08v125c
AO12CHD            1     8.960    fsc0h_d_generic_core_ss1p08v125c
AO13EHD            1    11.520    fsc0h_d_generic_core_ss1p08v125c
AO2222CHD          1    20.480    fsc0h_d_generic_core_ss1p08v125c
AOI112BHD          2    20.480    fsc0h_d_generic_core_ss1p08v125c
AOI12CHD           1     8.960    fsc0h_d_generic_core_ss1p08v125c
AOI13BHD           1    10.240    fsc0h_d_generic_core_ss1p08v125c
AOI22BHD          11    98.560    fsc0h_d_generic_core_ss1p08v125c
DFZCRBEHD         15   537.600    fsc0h_d_generic_core_ss1p08v125c
INVCKDHD          20    76.800    fsc0h_d_generic_core_ss1p08v125c
INVDHD             4    15.360    fsc0h_d_generic_core_ss1p08v125c
INVHHD             1     6.400    fsc0h_d_generic_core_ss1p08v125c
MAO222CHD          4    46.080    fsc0h_d_generic_core_ss1p08v125c
MAO222EHD          1    11.520    fsc0h_d_generic_core_ss1p08v125c
MAOI1CHD           1    11.520    fsc0h_d_generic_core_ss1p08v125c
MOAI1CHD          13   149.760    fsc0h_d_generic_core_ss1p08v125c
MUX2CHD            1    11.520    fsc0h_d_generic_core_ss1p08v125c
MUX2EHD            6    69.120    fsc0h_d_generic_core_ss1p08v125c
MUXB2CHD           1    14.080    fsc0h_d_generic_core_ss1p08v125c
ND2CHD            12    61.440    fsc0h_d_generic_core_ss1p08v125c
ND2DHD             4    20.480    fsc0h_d_generic_core_ss1p08v125c
ND3CHD             8    61.440    fsc0h_d_generic_core_ss1p08v125c
NR2BHD             2    10.240    fsc0h_d_generic_core_ss1p08v125c
NR2CHD             1     5.120    fsc0h_d_generic_core_ss1p08v125c
NR3BHD             3    23.040    fsc0h_d_generic_core_ss1p08v125c
OA112EHD           1    11.520    fsc0h_d_generic_core_ss1p08v125c
OA12EHD            1    10.240    fsc0h_d_generic_core_ss1p08v125c
OAI112BHD         17   152.320    fsc0h_d_generic_core_ss1p08v125c
OAI12CHD           4    30.720    fsc0h_d_generic_core_ss1p08v125c
OAI13BHD           2    17.920    fsc0h_d_generic_core_ss1p08v125c
OR2B1CHD          18   115.200    fsc0h_d_generic_core_ss1p08v125c
OR2CHD             2    12.800    fsc0h_d_generic_core_ss1p08v125c
OR2EHD             4    25.600    fsc0h_d_generic_core_ss1p08v125c
OR3B1CHD           1    10.240    fsc0h_d_generic_core_ss1p08v125c
OR3B1EHD           2    20.480    fsc0h_d_generic_core_ss1p08v125c
OR3B2CHD           5    44.800    fsc0h_d_generic_core_ss1p08v125c
OR3EHD             1     8.960    fsc0h_d_generic_core_ss1p08v125c
QDFZHHD            3    96.000    fsc0h_d_generic_core_ss1p08v125c
XOR2EHD            1    14.080    fsc0h_d_generic_core_ss1p08v125c
-------------------------------------------------------------------
total            191  2016.000


     Type      Instances   Area   Area %
-----------------------------------------
sequential           18  633.600   31.4
inverter             25   98.560    4.9
logic               148 1283.840   63.7
physical_cells        0    0.000    0.0
-----------------------------------------
total               191 2016.000  100.0
```

Figure 5: File `gates_report`, which includes all the gates used to define the charger controller at gate level and the respective occupied area, as well as a total estimated area for the circuit implementation.

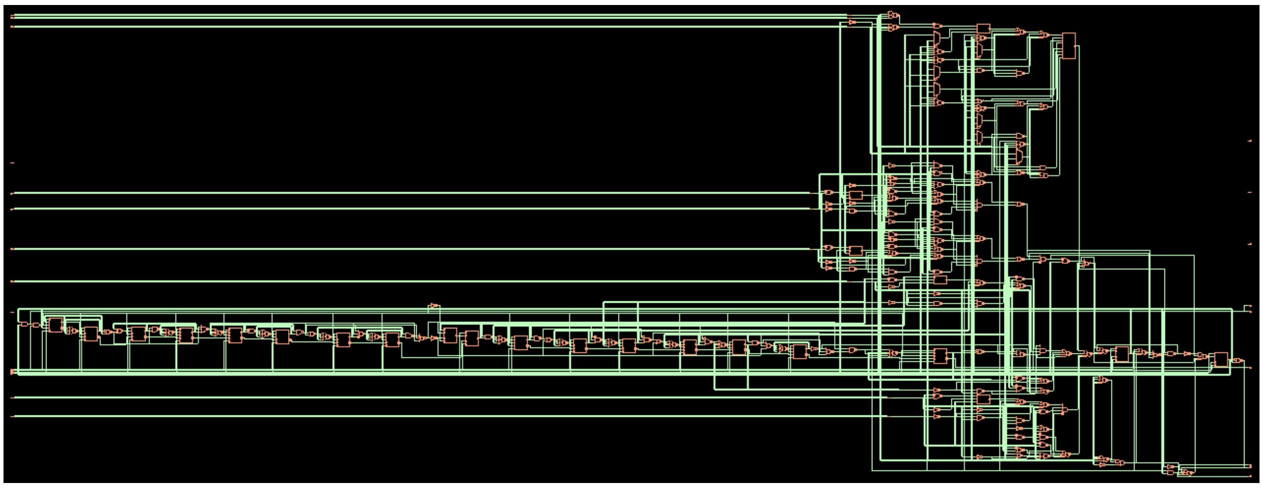


Figure 6: Schematic view of the controller after logic synthesis, using the graphical interface of genus.

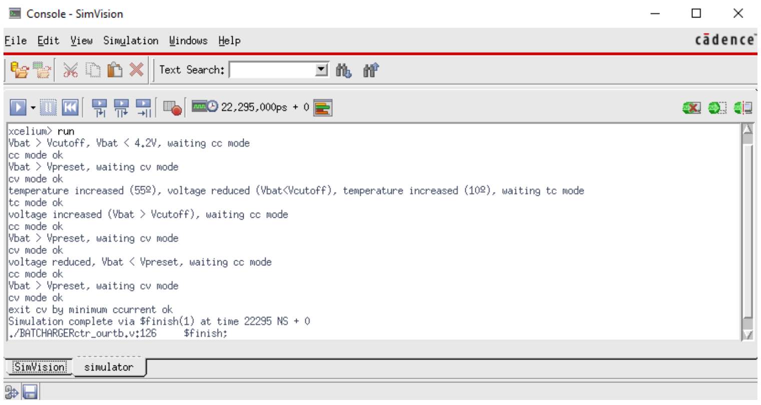
```
*************************************************************************************
                          Testmode Statistics: FULLSCAN

                        #Faults   #Tested  #Possibly    #Redund  #Untested  %TCov %ATCov
Total Static               1707      1699          0          0          8  99.53  99.53

                           Global Statistics

                        #Faults   #Tested  #Possibly    #Redund  #Untested  %TCov %ATCov
Total Static               1707      1699          0          0          8  99.53  99.53
*************************************************************************************
```

Figure 7: Portion of the ATPG result file `log_create_logic_tests_FULLSCAN_BATCHARGERctr_scan_atpg`, which includes the value of the fault coverage.

Finally, having performed logic synthesis and added scan to the controller, the testbench `BATCHARGERctr_ourtb.v` - created in a previous laboratory assignment - was used to simulate the controller defined at gate level (after logic synthesis performed by genus), instead of using the controller at RTL (as done in the first laboratory assignment). For this purpose, the script `sim_synth` (shown in Figure 8) was created in `DIGITAL/SIMULATION/BATCHARGER`. As seen below, this script calls `BATCHARGERctr_synth.v`, originated from the logic synthesis (as well as the library initially called in `sim_synth` of the 4-bit counter logic simulation tutorial). As seen in Figures 10 and 9, the exact same results from Lab 1 were obtained (thus, the same analysis applies here), which validates the performed logic synthesis of the charger controller.

```
rm -Rf ../worklib/*
##
xmvlog BATCHARGERctr_ourtb.v ../../SYNTHESIS_SCAN/BATCHARGER/BATCHARGERctr_synth.v ../verilog_libs/fsc0h_d_generic_core_30.lib
xmelab -access +rwc BATCHARGERctr_ourtb
xmsim -gui BATCHARGERctr_ourtb &
```

Figure 8: Script `sim_synth` used to obtain the results in Figures 9 and 10.



```
xcelium> run
Vbat > Vcutoff, Vbat < 4.2V, waiting cc mode
cc mode ok
Vbat > Vpreset, waiting cv mode
cv mode ok
temperature increased (55º), voltage reduced (Vbat<Vcutoff), temperature increased (10º), waiting tc mode
tc mode ok
voltage increased (Vbat > Vcutoff), waiting cc mode
cc mode ok
Vbat > Vpreset, waiting cv mode
cv mode ok
voltage reduced, Vbat < Vpreset, waiting cc mode
cc mode ok
Vbat > Vpreset, waiting cv mode
cv mode ok
exit cv by minimum ccurrent ok
Simulation complete via $finish(1) at time 22295 NS + 0
./BATCHARGERctr_ourtb.v:126     $finish;
```

Figure 9: Picture of the messages in the SimVision console obtained by simulating the controller design defined at gate level.

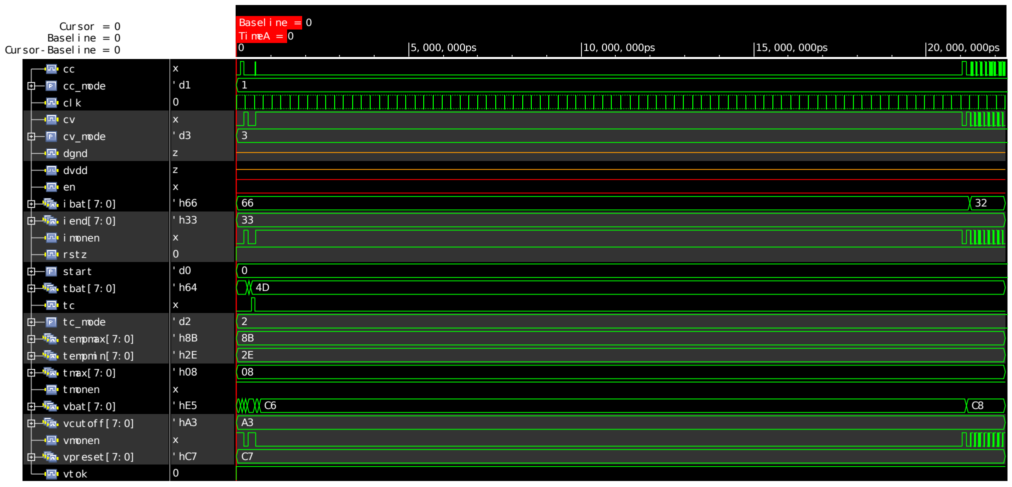


Figure 10: Waveform window obtained by simulating the controller design defined at gate level with the testbench `BATCHARGERctr_ourtb.v`, for a total simulation time of 22 295 000ps.

3 Wire bond test and test mux

Having validated the controller at gate level, the files necessary to simulate the complete battery charger were created. For this purpose, the file `mux_test.v` (shown in Figure 11) was created. In it, the dedicated 1-bit testpin `out` was added; using a multiplexer, described by the code line `assign out` in this file, depending on the value of the 3-bit word `sel_mux`, different input pins can be connected to this output (in fabrication, the wire bond test is intended to verify the process of interconnection between the semiconductor and its package). The inputs in this case are only `en` and `sel[3:0]`, as shown in `BATCHARGER_test_with_mux.v` (in Figure 12). The other `input` variables (with 64 bits each) will be converted to real values, thus should not be tested in this way. However, as described in the scan test, the inputs `si` and `se` and the output `so` were also created for this laboratory assignment. For this purpose, `so` can also be connected to the pin `out`, as indicated in Figure 11. The value of the parameter `sel_mux` is then transferred to the parameter named `testmode[2:0]` in `BATCHARGER_test_with_mux.v`. Doing this whole procedure, the number of new pins required for test purposes is reduced. Finally, it is important to mention that, in the code shown in Figure 12, a permanent connection is established between `sel[1]` and `si` (for scan purposes); additionally, **se** turns to '1' (Scan Mode) when `testmode=001` (i.e., when `testpin=so`). To sum up, the following connections are performed in `mux_test.v`:

- `testmode=000` $\rightarrow$ `testpin=1'b1`;
- `testmode=001` $\rightarrow$ `testpin=so`;
- `testmode=010` $\rightarrow$ `testpin=sel[0]`;
- `testmode=011` $\rightarrow$ `testpin=sel[1]`;
- `testmode=100` $\rightarrow$ `testpin=sel[2]`;
- `testmode=101` $\rightarrow$ `testpin=sel[3]`;
- `testmode=110` $\rightarrow$ `testpin=en`;
- `testmode=111` $\rightarrow$ `testpin=1'b0`.

```
`timescale 1ns / 1ps

module mux_test(
        // inputs and output of multiplexer
        input in0,
        input in1,
        input in2,
        input in3,
        input in4,
        input in5,
        input in6,
        input in7,
        input [2:0] sel_mux, // test_mode in full charger files
        output out
);

 // out gets assigned to inX depending on sel_mux
 assign out = sel_mux[2] ? (sel_mux[1] ? (sel_mux[0] ? in7 : in6) : (sel_mux[0] ? in5 : in4)) : (sel_mux[1] ? (sel_mux[0] ? in3 : in2) : (sel_mux[0]? in1 : in0));

endmodule // mux_test
```

Figure 11: File `mux_test.v` with the description of the test multiplexer for wire bonding.

```
`timescale 1ns / 1ps

module BATCHARGER_test_with_mux (
                       output [63:0] iforcedbat,
                       input [63:0]  vsensbat,
                       input [63:0]  vin,
                       input [63:0]  vbattemp,
                       input         en,
                       input [3:0]   sel,
                       inout [63:0]  dvdd,
                       inout [63:0]  dgnd,
                       inout [63:0]  pgnd,
                       input [2:0]  testmode, // added for test mux and wire bonding
                       output       testpin // added for test mux and wire bonding
                       );

   wire [63:0] vbatcurr;
   wire [7:0]  vbat;
   wire [7:0]  ibat;
   wire [7:0]  tbat;
   wire        imeasen;
   wire        vmeasen;
   wire        tmeasen;
   wire        cc;
   wire        tc;
   wire        cv;
   wire        vtok;
   wire [63:0] avdd;
   wire [63:0] agnd;
   wire [63:0] ibias1ua;
   wire [63:0] ibias1ub;
   wire [63:0] vrefa;
   wire [63:0] vrefb;
   real rl_vbatcurr;
   real rl_vrefa;
   real rl_vrefb;
   real rl_iforcedbat;
   real rl_vsensbat;
   real rl_vin;
   real rl_ibias1ua;
   real rl_ibias1ub;
   wire [7:0] vcutoff;
   wire [7:0] vpreset;
   wire [7:0] iend;
   wire [7:0] icc;
   wire [7:0] itc;
   wire [7:0] vcv;
   wire [7:0] tempmin;
   wire [7:0] tempmax;
   wire [7:0] tmax;
   wire       tiedown;
   parameter vcutoffpar = 8'b1001_1001;
   parameter vpresetpar = 8'b1100_0001;
   parameter iendpar = 8'b0000_0010;
   parameter itcpar = 8'b0001_1001;
   parameter vcvpar = 8'b1101_0110;
   parameter iccpar = 8'b0111_1111;
   parameter tempminpar = 8'b0011_1101;
   parameter tempmaxpar = 8'b1000_0011;
   parameter tmaxpar = 8'b1111_1111;

   // scan variables, also in BATCHARGERctr_scan.v
   wire se;
   wire si;
   wire so;

   assign vcutoff = vcutoffpar;
   assign vpreset = vpresetpar;
   assign iend = iendpar;
   assign icc = iccpar;
   assign itc = itcpar;
   assign vcv = vcvpar;
   assign tempmin = tempminpar;
   assign tempmax = tempmaxpar;
   assign tmax = tmaxpar;
   assign tiedown = 1'b0;

// New test mux block
mux_test  mux(
        .in0(1'b1), // testpin fixed at '1'
        .in1(so), // scan mode
        .in2(sel[0]),
        .in3(sel[1]),
        .in4(sel[2]),
        .in5(sel[3]),
        .in6(en),
        .in7(1'b0), // testpin fixed at '0'
        .sel_mux(testmode[2:0]),
        .out(testpin) // added 1-bit output pin
);

  assign se = testmode[0] & ~testmode[1] & ~testmode[2]; // scan mode
  assign si = sel[1]; // permanent connection
```

```
BATCHARGERbg_64b BATCHbg (
                       .vin(vin),
                       .ibias1ua(ibias1ua),
                       .ibias1ub(ibias1ub),
                       .vrefa(vrefa),
                       .vrefb(vrefb),
                       .en(en),
                       .endvdd(endvdd),
                       .clk(clk),
                       .rstz(rstz),
                       .avdd(avdd),
                       .dvdd(dvdd),
                       .dgnd(dgnd),
                       .agnd(agnd)
                       );

BATCHARGERpower_64b BATCHpower (
                                .iforcedbat(iforcedbat),
                                .vbatcurr(vbatcurr),
                                .vsensbat(vsensbat),
                                .vref(vrefa),
                                .vin(vin),
                                .ibias1u(ibias1ua),
                                .icc(icc),
                                .itc(itc),
                                .vcv(vcv),
                                .cc(cc),
                                .tc(tc),
                                .cv(cv),
                                .en(endvdd),
                                .sel(sel),
                                .avdd(avdd),
                                .dvdd(dvdd),
                                .dgnd(dgnd),
                                .agnd(agnd),
                                .pgnd(pgnd)
                                );

BATCHARGERsaradc_64b  BATCHsaradc (
                                   .vbat(vbat),
                                   .ibat(ibat),
                                   .tbat(tbat),
                                   .vref(vrefb),
                                   .vtok(vtok),
                                   .vbattemp(vbattemp),
                                   .vbatvolt(vsensbat),
                                   .vbatcurr(vbatcurr),
                                   .imeasen(imeasen),
                                   .vmeasen(vmeasen),
                                   .tmeasen(tmeasen),
                                   .en(endvdd),
                                   .clk(clk),
                                   .rstz(rstz),
                                   .ibias1u(ibias1ub),
                                   .avdd(avdd),
                                   .dvdd(dvdd),
                                   .dgnd(dgnd),
                                   .agnd(agnd)
                                   );

BATCHARGERctr_scan BATCHctr_scan(
                          .cc(cc),
                          .tc(tc),
                          .cv(cv),
                          .vtok(vtok),
                          .en(endvdd),
                          .imonen(imeasen),
                          .vmonen(vmeasen),
                          .tmonen(tmeasen),
                          .vbat(vbat),
                          .ibat(ibat),
                          .tbat(tbat),
                          .vcutoff(vcutoff),
                          .vpreset(vpreset),
                          .tempmin(tempmin),
                          .tempmax(tempmax),
                          .tmax(tmax),
                          .iend(iend),
                          .clk(clk),
                          .rstz(rstz),
                          .se(se),
                          .si(si),
                          .so(so),
                          .dvdd(endvdd),
                          .dgnd(tiedown)
                          );

   initial assign rl_iforcedbat = $bitstoreal (iforcedbat);
   initial assign rl_vbatcurr = $bitstoreal (vbatcurr);
   initial assign rl_vin = $bitstoreal (vin);
   initial assign rl_ibias1ub = $bitstoreal (ibias1ub);
   initial assign rl_ibias1ua = $bitstoreal (ibias1ua);
   initial assign rl_vrefa = $bitstoreal (vrefa);
   initial assign rl_vrefb = $bitstoreal (vrefb);
   initial assign rl_vsensbat = $bitstoreal (vsensbat);

endmodule
```

Figure 12: Complete charger file `BATCHARGER_test_with_mux.v`, in which the newly added functionalities are indicated with comments.

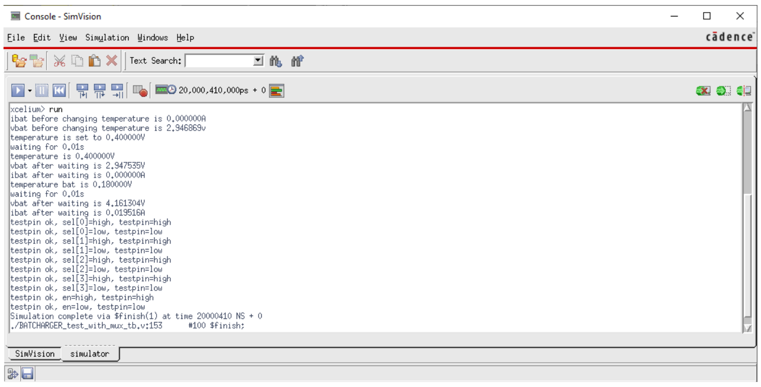
To simulate the full charger with the controller design in `BATCHARGERctr_synth.v`, the script shown in Figure 13 and the testbench shown in Figure 14 were created. In this new testbench, the necessary variables were added and various tests were performed in order to check if the connections in the test mux are properly done. This can be confirmed with the results shown in Figure 15. It is worth noting, **however**, that a small change in the script had to be made - instead of using `BATCHARGERctr_synth.v`, the charging process was only initiated by using `BATCHARGERctr_scan.v` - this unexpected situation would require more time to understand. Even then, the results shown in Figure 15 and 16 indicate that the scan, wire bond and test mux tests work successfully within the full charger (even though the synthesized controller did not).

```
rm -Rf ../worklib/* ../worklib/*
##
xmvlog BATCHARGER_test_with_mux_tb.v BATCHARGER_test_with_mux.v BATCHARGERbg_64b.v ../../SYNTHESIS_SCAN/BATCHARGER/BATCHARGERctr_synth.v BATCHARGERlipo_64b.v BATCHARGERpower_64b.v
BATCHARGERsaradc_64b.v mux_test.v ../verilog_libs/fsc0h_d_generic_core_30.lib
xmelab -access +rwc BATCHARGER_test_with_mux_tb
xmsim -gui BATCHARGER_test_with_mux_tb &
```

Figure 13: Script `sim_rtl_scan` which could be used to simulate the full charger with the controller defined at gate level.

```verilog
`timescale 1 ns/10 ps

module BATCHARGER_test_with_mux_tb;

   wire [63:0] vin;
   wire [63:0] vbat;
   wire [63:0] ibat;
   wire [63:0] vtbat;
   wire [63:0] dvdd;
   wire [63:0] dgnd;
   wire [63:0] pgnd;
   wire [63:0] vtbat_tb;
   reg         en;
   reg [3:0]   sel;
   reg [2:0]   testmode; // added for test mux and wire bonding
   wire        testpin; // added for test mux and wire bonding
   real        rl_dvdd, rl_dgnd, rl_pgnd;
   real        rl_ibat, rl_vbat, rl_vtbat;
   real        rl_vin;
   real        rl_vtbat_tb;

// New full charger
BATCHARGER_test_with_mux uut(
                   .iforcedbat(ibat),
                   .vsensbat(vbat),
                   .vin(vin),
                   .vbattemp(vtbat_tb),
                   .en(en),
                   .sel(sel),
                   .dvdd(dvdd),
                   .dgnd(dgnd),
                   .pgnd(pgnd),
                   .testmode(testmode), // added
                   .testpin(testpin) // added
);

BATCHARGERlipo lipobattery(
                   .vbat(vbat),
                   .ibat(ibat),
                   .vtbat(vtbat)
                   );

// Tests
initial
  begin
     rl_vin = 4.5;
     rl_pgnd = 0.0;
     sel[3:0] = 4'b1000;  // 450mAh selection
     en = 1'b1;

     #100;

     $display("ibat before changing temperature is %fA", rl_ibat);
     $display("vbat before changing temperature is %fv", rl_vbat);

     rl_vtbat_tb = 0.4; // temperature above normal range
     $display("temperature is set to %fV",rl_vtbat_tb);

     $display("waiting for 0.01s");
     #10000000; // wait some time

     $display("temperature is %fV", rl_vtbat_tb);
     $display("vbat after waiting is %fV", rl_vbat);
     $display("ibat after waiting is %fA", rl_ibat);

     rl_vtbat_tb = 0.18; // now set to around 20ºC
     $display("temperature bat is %fV", rl_vtbat_tb);

     $display("waiting for 0.01s");
     #10000000; // wait some time

     $display("vbat after waiting is %fV", rl_vbat);
     $display("ibat after waiting is %fA", rl_ibat);

     #100

     testmode = 3'b010; // sel[0] should connect to output testpin

     sel = 4'b0001;
     #10;
     if(testpin == sel[0]) begin
        $display("testpin ok, sel[0]=high, testpin=high");
     end
     else $display("testpin not connected to sel[0]");
     sel = 4'b0000;
     #10;
     if(testpin == sel[0]) begin
        $display("testpin ok, sel[0]=low, testpin=low");
     end
     else $display("testpin not connected to sel[0]");
     #10;

     testmode = 3'b011; // sel[1] should connect to output testpin

     sel = 4'b0010;
     #10;
     if(testpin == sel[1]) begin
        $display("testpin ok, sel[1]=high, testpin=high");
     end
     else $display("testpin not connected to sel[1]");
     sel = 4'b0000;
     #10;
     if(testpin == sel[1]) begin
        $display("testpin ok, sel[1]=low, testpin=low");
     end
     else $display("testpin not connected to sel[1]");

     testmode = 3'b100; // sel[2] should connect to output testpin

     sel = 4'b0100;
     #10;
     if(testpin == sel[2]) begin
        $display("testpin ok, sel[2]=high, testpin=high");
     end
     else $display("testpin not connected to sel[2]");
     sel = 4'b0000;
     #10;
     if(testpin == sel[2]) begin
        $display("testpin ok, sel[2]=low, testpin=low");
     end
     else $display("testpin not connected to sel[2]");

     testmode = 3'b101; // sel[3] should connect to output testpin

     sel = 4'b1000;
     #10;
     if(testpin == sel[3]) begin
        $display("testpin ok, sel[3]=high, testpin=high");
     end
     else $display("testpin not connected to sel[3]");
     sel = 4'b0000;
     #10;
     if(testpin == sel[3]) begin
        $display("testpin ok, sel[3]=low, testpin=low");
     end
     else $display("testpin not connected to sel[3]");

     testmode = 3'b110; // en should connect to output

     en = 1;
     #10;
     if(testpin == en) begin
        $display("testpin ok, en=high, testpin=high");
     end
     else $display("testpin not connected to en");
     en = 0;
     #10;
     if(testpin == en) begin
        $display("testpin ok, en=low, testpin=low");
     end
     else $display("testpin not connected to en");

     #100 $finish;
  end

   initial assign rl_vbat = $bitstoreal(vbat);
   initial assign rl_vtbat = $bitstoreal(vtbat);
   initial assign rl_ibat = $bitstoreal(ibat);
   initial assign rl_dvdd = $bitstoreal(dvdd);
   initial assign rl_dgnd = $bitstoreal(dgnd);
   assign vin = $realtobits(rl_vin);
   assign pgnd = $realtobits(rl_pgnd);
   assign vtbat_tb = $realtobits(rl_vtbat_tb);

endmodule
```

Figure 14: Testbench `BATCHARGER_test_with_mux_tb.v` created to simulate the full battery charger, in which the tests regarding `testpin` were added to the respective testbench from the previous laboratory assignment.



```
xcelium> run
ibat before changing temperature is 0.000000A
vbat before changing temperature is 2.946869v
temperature is set to 0.400000V
waiting for 0.01s
temperature is 0.400000V
vbat after waiting is 2.947535V
ibat after waiting is 0.000000A
temperature bat is 0.180000V
waiting for 0.01s
vbat after waiting is 4.161304V
ibat after waiting is 0.019516A
testpin ok, sel[0]=high, testpin=high
testpin ok, sel[0]=low, testpin=low
testpin ok, sel[1]=high, testpin=high
testpin ok, sel[1]=low, testpin=low
testpin ok, sel[2]=high, testpin=high
testpin ok, sel[2]=low, testpin=low
testpin ok, sel[3]=high, testpin=high
testpin ok, sel[3]=low, testpin=low
testpin ok, en=high, testpin=high
testpin ok, en=low, testpin=low
Simulation complete via $finish(1) at time 20000410 NS + 0
./BATCHARGER_test_with_mux_tb.v:153      #100 $finish;
```

Figure 15: Picture of the messages in the SimVision console obtained by simulating the full charger with the command `./sim_rtl_scan`.

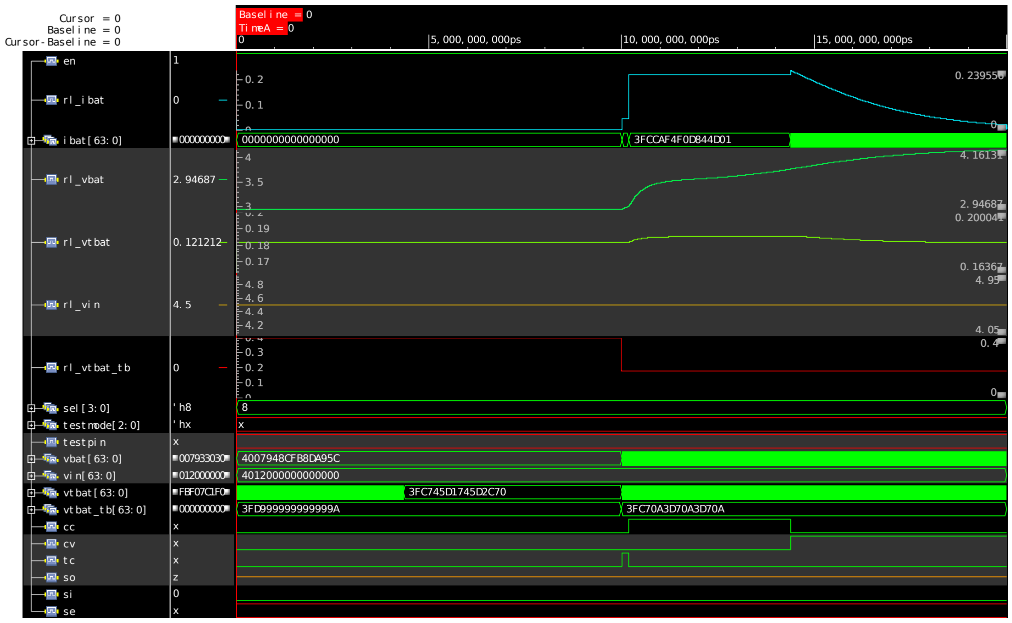


Figure 16: Waveform window obtained by simulating the complete charger with the testbench `BATCHARGER_test_with_mux_tb.v` (with test mux and wire bond), for a total simulation time of 20 00 410 000ps.

4 Conclusion

In this laboratory assignment, a **scan test** was added to the previously implemented controller design, in order to improve testability and communication. For this purpose, logic synthesis was performed using the software genus and an ATPG (Automatic Test Pattern Generator) was used with modus to obtain a very satisfactory fault coverage of 99.53% for the controller design at gate level, as well an estimated area occupied by the logic gates necessary to implement it. The synthesized controller was also validated using the same testbench from the first lab. Secondly, a **test mux** was developed in order to connect input pins of the full charger design to a newly added output pin, for **wire bond test** purposes. To do this, the 3-bit `sel_mux` was created to select different test modes. In order to further reduce the number of pins added for test purposes, some inputs from the battery charger could have been defined as `inout` (instead of `input`) - a more challenging procedure. The complete charger was then simulated and validated using a testbench in which the connections between inputs and output pin were verified successfully.