TÉCNICO LISBOA

Systems On-Chip

MEEC

Lab 3 - Synthesis and DFT

Authors:

99531 – Matilde Sardinha
102546 – Francisco Rosa
112504 – Thibaut Nivelet

Group 4

2024/2025 – 1º Semester, P2

Contents

1 Post-Synthesis Controller Changes

Due to the limitations caused after synthesis, alterations were implemented to prevent some unusual behaviors. This included an increase in the number of states to prevent unwanted behavior regarding current monitoring. When the current signal monitoring *imonen* is transits between $H \rightarrow L$, there is a small delay until the current *ibat* has a valid value, being not defined (`0'xXX`) until then. This is expected since the ADC cannot make the current value available instantaneously. However, the undefined state will cause an unexpected state transition, not going to `CV MODE` as it should, producing undesired outputs. This problem was solved creating state `TRANSIT`. This state has the same outputs as `CV MODE`, and the state machine is kept here for 64 clock cycles, enough time for the ADC to give a valid current value for *ibat*. This is done using the counter that increases the *charge_time* signal that indicates the end of the charging process.

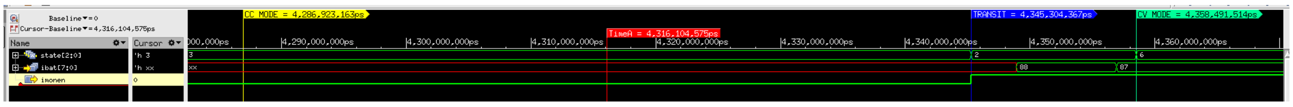
CC MODE = 4,286,923,163ps
TimeA = 4,316,104,575ps
TRANSIT = 4,345,304,367ps
CV MODE = 4,358,491,514ps
state[2:0]
ibat[7:0]
imonen

Figure 1: Valid current value delay. As seen in the image there is a small delay until the current *ibat* has a valid value. This problem was surpassed using an intermediate state `TRANSIT`

Another problem that arises after synthesis is the state change speed of the flip-flops. As the PMOS and NMOS transistors that make up the logic gates have different charge mobilities, the speed of the transitions $HIGH \rightarrow LOW$ and $LOW \rightarrow HIGH$ will be different. This will cause two main problems. The first is that if the registers are synchronous with the clock (as it is the case), the outputs, for example, tc, cc, and cv could be subject to small undesired effects. For example, since the transistors do not change at the same time, if the transitions $tc : H \rightarrow L$ and $cc : L \rightarrow H$, there could be a short period of time where tc and cc are 0, with the power block impose a $0A$ current to the battery, causing a short glitch.

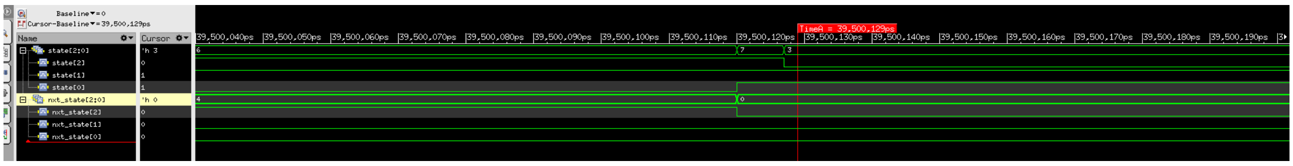

Figure 2: State change transition problem. As mentioned, if the state transition requires 2 or more bits to change, it could cause undesired intermediate state, sometimes not declared properly.

The second problem also affects the state changes. If a state change requires a transition of 2 or more bits, there could be an unwanted intermediate state. As in Figure 2, the desired transition is between the states with codes 6 (`110`) and 3 (`011`). This transition required two bits to change, $state[2]$ ($H \rightarrow L$) and $state[1]$ ($L \rightarrow H$). However, since the transition of $state[0]$ happens faster, there is a small period of time where *state* has code 7 (`111`), which is undefined in the state machine (Table 1). Even if this problem is impossible to solve for this state machine, there are some possible partial solutions. One of them is to use Gray encoding to define the states. This way, only one bit is changed between consecutive state transitions, with the exceptions being the interruptions caused by the control signals (*vtok*, *rstz*, *en*) or a temperature out-of-bounds (condition C_0), and the transition between `IDLE` and *CC MODE*.

State	Digit	Code	Outputs					
			TC	CC	CV	imonen	vmonen	tmonen
START	0	000	0	0	0	0	1	1
TC MODE	1	001	1	0	0	0	1	1
CC MODE	3	011	0	1	0	0	1	1
TRANSIT	2	010	0	0	1	1	0	1
CV MODE	6	110	0	0	1	1	0	1
END	4	100	0	0	0	0	0	0

Table 1: Output assignment of the updated controller's state machine

This also disables any kind of association between inputs and outputs (there is not a common pattern between them), that could lead the synthesis tool to perform unwanted optimization that generates unconnected pins, and drive some registers to high impedance (Z). Another change performed was the decrease of the number of registers (that were being used to store the state change conditions) and more importantly, the dependence on the *negedge* of the *en*. This was a feature added to ensure that, if $en = 0$, the state should be `IDLE`. Since the state registers are tied to the positive edge, and the clock is turned out when $en = 0$ (since the band-gap reference block is turned off), the outputs would be kept the same, which could be harmful if the power block wasn't also turned off, as there would not be any state updates, so the dependency of the negative edge of *en* was added.

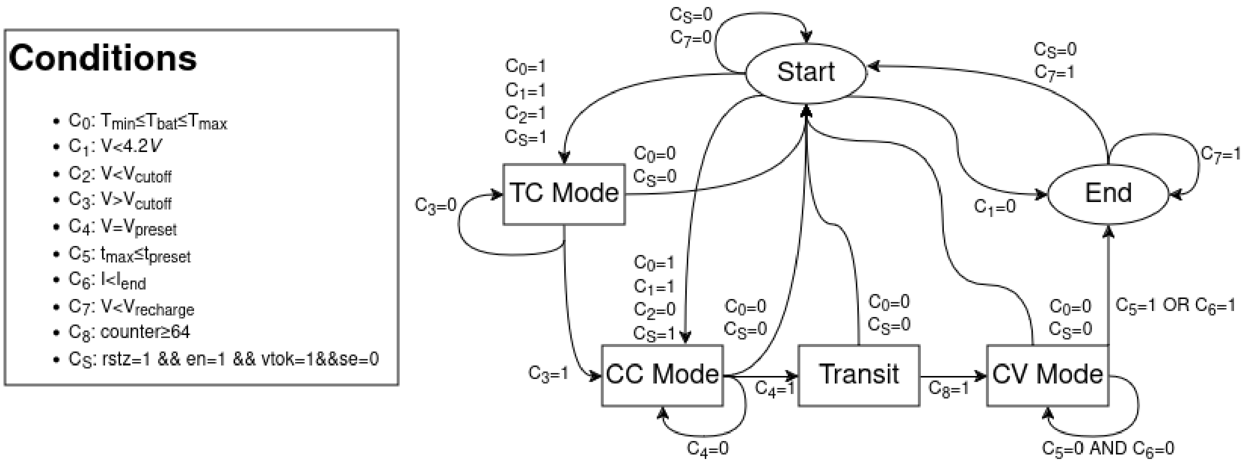

Figure 3: Updated state machine after the inclusion of a new state, state coding correction, and scan enable

While performing scan this is undesirable, since it would require adding a second scan chain[1],

[1]This method was tested, yielding a fault coverage of around 97.5%, a little bit lower than what could be

which would increase the number of untested and redundant faults. This detail was solved by changing the structure of the output logic, placing a multiplexer (in the form of a *if* clause), forcing all outputs (with the exception of *vmonen* and *tmonen*) to 0 if $en = 0$, or if $se = 1$, since a similar problem arises when performing scan, with the state and next state register having unpredictable values during scan, generating unwanted variations in the outputs. As such, it is necessary to take this precaution. The final state machine can be found Figure 3.

Regarding the addition of scan, the procedure consisted of adding the pins corresponding to scan enable (se), scan input (si), and scan output (se) to the controller, while the synthesis tool Genus performed the scan insertion on all flip-flops, also creating the scan chain between all registers. Since the number of flip-flops in the circuit is relatively small (3 from registers *state* and *nxt-state* each, 8 from registers *charge-time* and *counter* each, plus one for each of the outputs), it was decided to perform full scan, that is, to test all flip-flops.

2 Gate level simulation of the charger controller

In order to assure that after synthesis, the controller was still working properly, the self-testable testbench for the controller produced on Lab 1 was reused for this laboratory, with the addition of a new test for the additional scan feature.

Every test was run after initializing all variables with initial values and assuring that the battery was in an `IDLE` state. After this initialization, each test forces *vbat*, *ibat*, *tbat*, *en*, *rstz* or *vtok* to induce the desired state changes. If the state change expected does not happen, an `ERROR` message will be displayed on the console, and the simulation will finish.

The first test tested the exit of `CVMODE` to `END` due to the current ,*ibat*, exceeding a certain threshold, *iend*. This simulation starts in `TCMODE` (blue marker), progresses to `CCMODE`(orange marker) and than to `CVMODE`(pink marker). Even though `CVMODE` is activated at the pink marker, we can observe a state change to `TRANSIT` and only after, to `CVMODE`, at the dark blue marker. After a while, the current drops bellow *iend*, at the yellow marker, but only at the last marker (cyan) the controller exits `CVMODE` to `EXIT` (figure 4). On Lab 1, there wasn't this delay, since before synthesis, propagation delays inherit to logic gates were not considered.

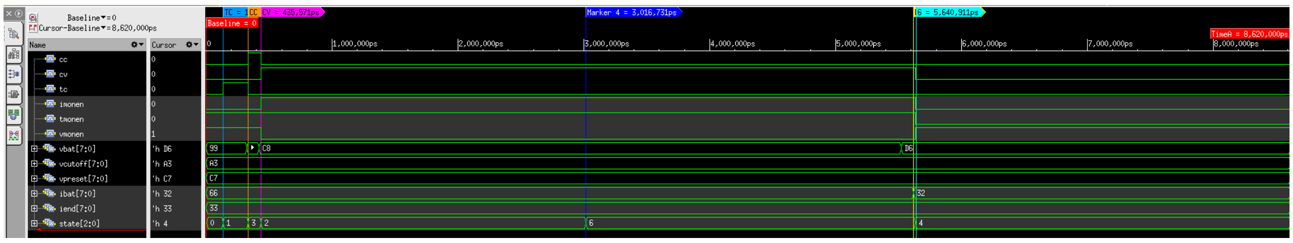

Figure 4: Current exit test

The test shown in figure 5, tests exiting `CVMODE` by *charge_time* being greater than *tmax*. The

achieved.

battery starts to go through the two states that precede `CVMODE`, `TCMODE`(blue) and `CCMODE`(orange). The last marker(cyan) on the figure shows the exit to `END` when the time runs out, being almost instantaneous in this case.

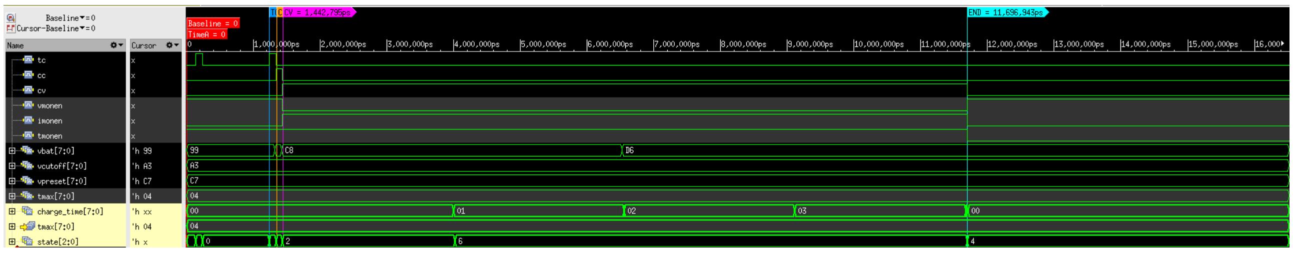

Figure 5: Timeout exit

In all states the temperature condition is constantly checked and if any of the boundaries (*tempmax* and *tempmin*) is exceeded, the battery charger proceeds to the `START` mode. In this case (figure 6), the temperature exits the upper limit at the third marker (in blue) and only exists the current mode, `CCMODE`, at the fourth marker (in pink), being visible a small delay on the charger response.

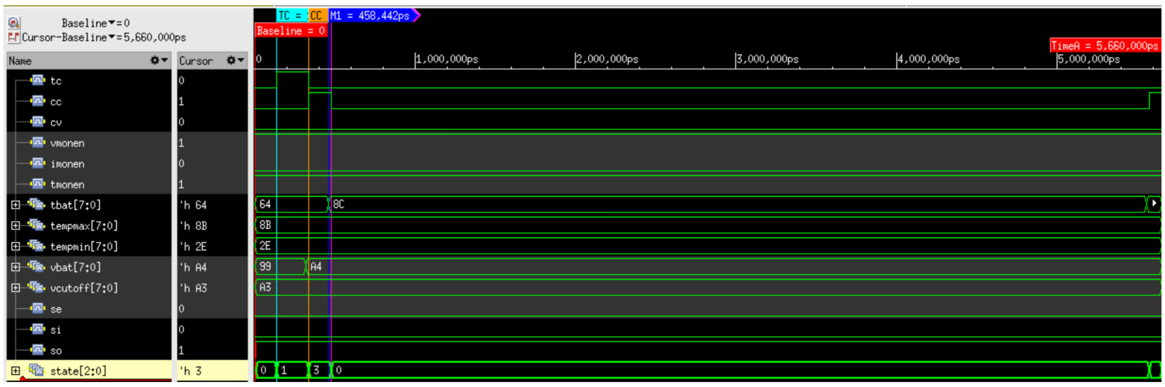

Figure 6: Temperature exit

If the battery is fully charged, it should remain in an idle state until the voltage drops requiring a recharge. In the full charge simulation, shown in figure 7, the battery is fully charged at marker one (blue), but only at marker two (purple) `TCMODE` is deactivated, fact explained, once again, by the propagation delay on the logic gates.

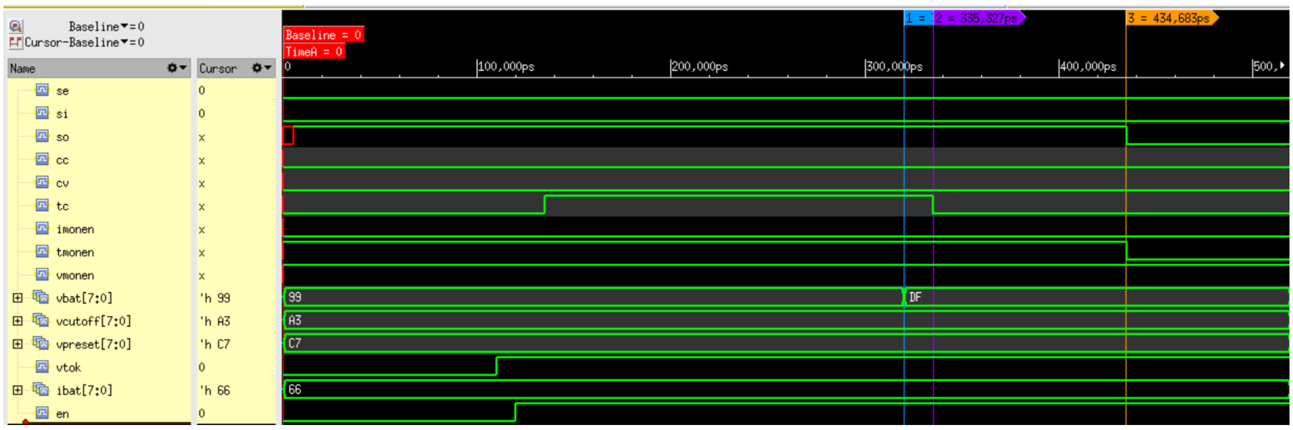

Figure 7: Full charge exit test

In the enable test (figure 8), it is observable the delay in the state change from `TCMODE` to `IDLE`. The variable *en* is activated at the first marker (in blue) and at marker 2 (in orange), the state transitions to `TCMODE`. At marker 3, in pink, *en* is deactivated and the state becomes `IDLE` only at the last marker, in yellow, presenting a delay that before synthesis was not present.

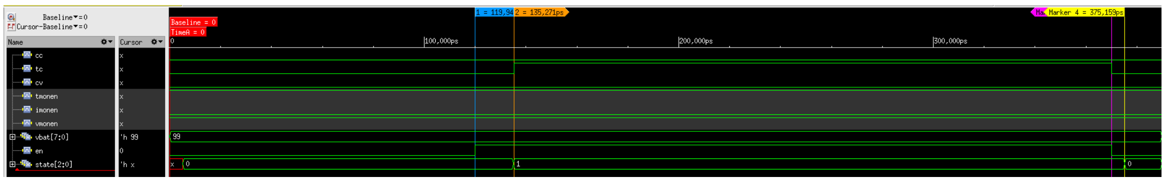

Figure 8: Enable test

The recharge test, evaluates if the battery is able to recharge after usage. At marker 4, in figure 9, *vbat* drops to a value small enough to trigger recharge, however, recharge only starts at marker 5, accounting with a delay just like in the other tests.

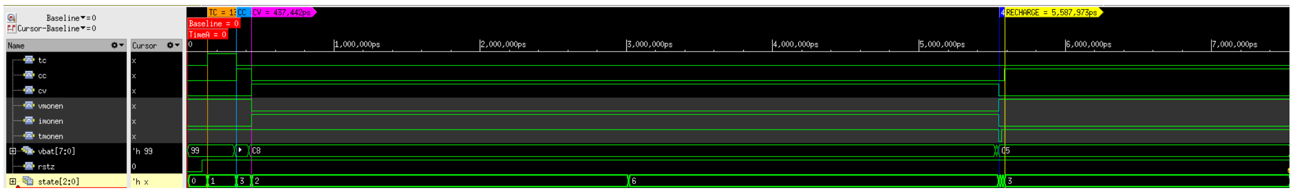

Figure 9: Recharge test

This detailed description and analysis of the controller simulation allowed to spot the differences between the synthesized controller and the previous one. Even though a certain delay was observable in all the tests, the battery still passed all of them, as was checked on the Cadence terminal where messages of all the passed tests were printed.

Additionally, on this testbench, a test to analyze the behavior of the scan function was designed. This test activates *se* (scan enable) and after, in a cycle, forces alternating values on *si* in order to induce some possible combinations on the flip flops. This allowed to observe some transitions between all the states in figure 10, a prove that the flip-flops are being tested. However, even after *se* is disabled at the last marker (in yellow), *so* keeps changing values. This happens because the flip flops are still processing the last values of *si*, given to the scan chain before deactivating scan. All the logs of this test are presented in listing 2.

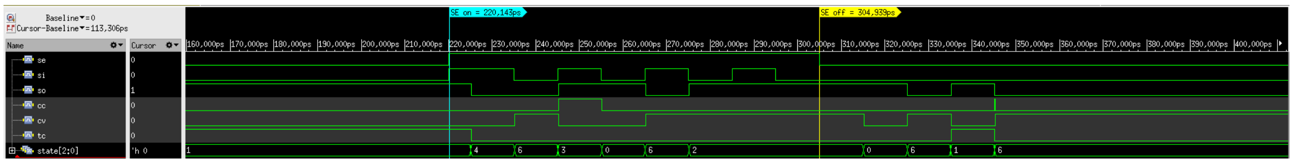

Figure 10: Scan test

```
===== STARTING SCAN TEST =====
Initialization end
Scan mode enabled. Testing scan chain.
Cycle 0: si = 1, so = 0
Cycle 1: si = 0, so = 0
Cycle 2: si = 1, so = 1
Cycle 3: si = 0, so = 1
Cycle 4: si = 1, so = 0
Cycle 5: si = 0, so = 1
Cycle 6: si = 1, so = 1
Cycle 7: si = 0, so = 1
Scan mode disabled. Checking normal operation.
Normal operation successfully resumed after scan.
===== SCAN TEST COMPLETED =====
```

Listing 1: Output logs of scan test

The log of the full tests can also be found below.

```
===== TESTING CHARGING PROCESS WITH FULL CHARGE EXIT =====
Initialization end
Exit by battery fully charged
SUCCESS: Exit by full charge
===== TESTING CHARGING PROCESS WITH CURRENT EXIT =====
Initialization end
Testing with Vbat < Vcutoff, checking for tc mode
Entered tc mode
Vbat > Vcutoff, checking for cc mode
Entered cc mode
Vbat > Vpreset, checking for cv mode
Entered cv mode
Exited cv mode by minimum current
SUCCESS: Exit by minimum current condition
===== TESTING CHARGING PROCESS WITH TEMPERATURE EXIT =====
Initialization end
Testing with Vbat < Vcutoff, checking for tc mode
Entered tc mode
Vbat > Vcutoff, checking for cc mode
Entered cc mode
Temperature condition met
Cooled out, temperature down
Tmin <= Tbat <= Tmax, checking for charge beginning
Entered tc mode, cc mode for recharge or idle for voltage drop
SUCCESS: Temperature control test
===== TESTING CHARGING PROCESS WITH TIMEOUT EXIT =====
Initialization end
```

```
Testing with Vbat < Vcutoff , checking for tc mode
Entered tc mode
Vbat > Vcutoff , checking for cc mode
Entered cc mode
Vbat > Vpreset , checking for cv mode
Entered cv mode
Exited cv mode by timeout
SUCCESS: Exit by timeout condition
===== TESTING RECHARGING PROCESS WITH CURRENT EXIT =====
Initialization end
Testing with Vbat < Vcutoff , checking for tc mode
Entered tc mode
Vbat > Vcutoff , checking for cc mode
Entered cc mode
Vbat > Vpreset , checking for cv mode
Entered cv mode
Voltage decrease to lead to discharge
Testing if it entered one of the charging modes (either TC or CC)
SUCCESS: Recharge process test passed
===== TESTING RESET EXIT =====
Initialization end
Testing with Vbat < Vcutoff , checking for tc mode
Entered tc mode
SUCCESS: Reset confirmation passed
===== STARTING SCAN TEST =====
Initialization end
Scan mode enabled. Testing scan chain.
Cycle 0: si = 1, so = 1
Cycle 1: si = 0, so = 0
Cycle 2: si = 1, so = 0
Cycle 3: si = 0, so = 1
Cycle 4: si = 1, so = 1
Cycle 5: si = 0, so = 0
Cycle 6: si = 1, so = 1
Cycle 7: si = 0, so = 1
Scan mode disabled. Checking normal operation.
Normal operation successfully resumed after scan.
===== SCAN TEST COMPLETED =====
===== ALL TESTS COMPLETED =====
End of simulation
Simulation complete via $finish(1) at time 39750 NS + 0
```

Listing 2: Output logs of the full controller test

3 Charger simulation with gate level controller

After confirming the functionality of the controller at the gate level, this section will test the complete charger with all modules, following the same procedure as in Lab 1, section 1.2. These modules include the power block designed in Verilog in Lab 2, as well as the controller synthesized after the scan generated previously by Cadence Genus.

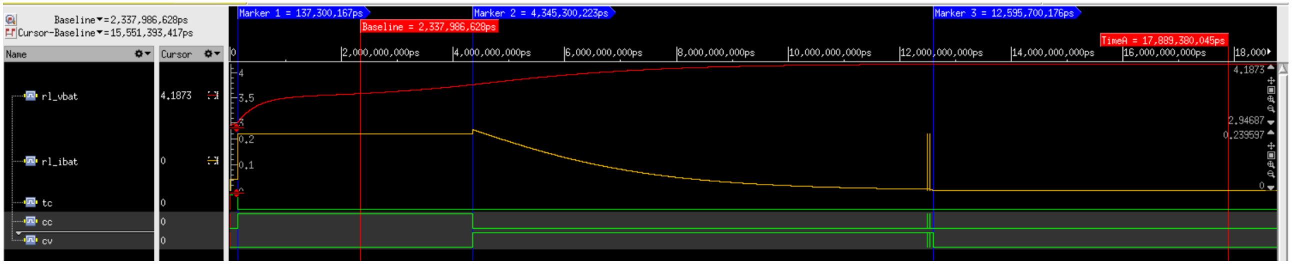

Figure 11: Waveform output of the real signal `ibat` and `vbat`

First, we study the charging behavior using the curves obtained from the simulation, as shown in the Figure 11 above. This figure illustrates that the charging results in an increase in the value of the battery voltage (`rl_vbat`), at constant current (`rl_ibat`). This corresponds to the state of `CC` mode, as expected.

Additionally, at the beginning, the charger is in start-up mode, and switches to `TC` mode, battery voltage is below `Vcutoff`. Once again, we can confirm that the battery is being charged with reduced current as intended. After reaching `Vcutoff`, at **Marker 1**, the state switches to `CC` mode, charging the battery with a constant current as assumed, while the voltage increases, until `vbat` = `Vpreset`, entering `CV` mode (**Marker 2**).

We can also notice that, contrary, to the expected behavior shown in the scientific paper studied in the lab 1, the battery does not have a constant voltage, since monitoring is performed in the battery and not in the control system. This behaviour is totally identical to that encountered in the lab 1. The imperfection found near **Marker 3** is due to a behavior that is highlighted after synthesis. Since in `CV MODE` there is no voltage monitoring ($vmonen = 0$), when the signal is activated again in `END`, there is a small delay until the updated value of $vbat$ is made available to the controller, causing these unwanted transitions. This problem could be solved by introducing an intermediate state (much like `TRANSIT`). Since the glitch just happens very briefly, this solution was not applied.

Also, to carry out this simulation at gate level, we reused the same self-testable test bench created for the Lab 1. Furthermore, this allowed us to verify, once again, the correct functionality of the entire charger via the SimVision console, the output log of which is shown in the listing 3 below.

```
===== Starting the testbench and charging the battery =====
Correct state - TC MODE at time 30300000
In TC MODE, current within boundaries: 0.044100
Correct state - TC MODE at time 60300000
In TC MODE, current within boundaries: 0.044100
Correct state - CC MODE at time 180300000
In CC MODE, current within boundaries: 0.224100
In CC MODE, current within boundaries: 0.224100
In CC MODE, current within boundaries: 0.224100
In CC MODE, current within boundaries: 0.224100
In CC MODE, current within boundaries: 0.224100
In CC MODE, current within boundaries: 0.224100
In CC MODE, current within boundaries: 0.224100
In CC MODE, current within boundaries: 0.224100
Correct state - CV MODE at time 4380300000
In CV MODE, current within boundaries: 0.237014
In CV MODE, current within boundaries: 0.206403
In CV MODE, current within boundaries: 0.165499
In CV MODE, current within boundaries: 0.129751
In CV MODE, current within boundaries: 0.099944
In CV MODE, current within boundaries: 0.075944
In CV MODE, current within boundaries: 0.057112
In CV MODE, current within boundaries: 0.042617
In CV MODE, current within boundaries: 0.031617
In CV MODE, current within boundaries: 0.023354
In CV MODE, current within boundaries: 0.017196
In CV MODE, current within boundaries: 0.012632
In CV MODE, current within boundaries: 0.009264
In CV MODE, current within boundaries: 0.006785
Correct state - END MODE at time 12630300000
END, current is 0A
In END, current within boundaries: 0.000000
In END, current within boundaries: 0.000000
In END, current within boundaries: 0.000000
In END, current within boundaries: 0.000000
In END, current within boundaries: 0.000000
In END, current within boundaries: 0.000000
Verification completed or timed out after 5000 time units
Test 1 : completed with success, Vbat=4.187301
==== Test concluded with success ====
Simulation complete via $finish(1) at time 150000300 NS + 0
```

Listing 3: Output logs after the simulation of the whole charger with the synthesized controller

4 Performance after synthesis with scan

4.1 Power performance

After performing synthesis and scan, some extra information regarding the power consumption was given by the synthesis tool. Besides indicating an estimation for the power consumption (including leakage, internal consumption, and switching power), it also indicates the category of components used, or not in the design. These components are divided into registers, internal logic and clock. The final estimation for the power consumption is $72.03 \mu W$.

```
Instance: /BATCHARGERctr
Power Unit: W
PDB Frames: /stim#0/frame#0
  -------------------------------------------------------------------------
    Category         Leakage      Internal     Switching         Total    Row%
  -------------------------------------------------------------------------
      memory     0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00%
    register     6.74431e-07   3.11708e-05   1.73952e-06   3.35848e-05  46.62%
       latch     0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00%
       logic     1.10340e-06   1.17682e-05   2.12135e-05   3.40851e-05  47.32%
        bbox     0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00%
       clock     0.00000e+00   0.00000e+00   4.36234e-06   4.36234e-06   6.06%
         pad     0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00%
          pm     0.00000e+00   0.00000e+00   0.00000e+00   0.00000e+00   0.00%
  -------------------------------------------------------------------------
    Subtotal     1.77783e-06   4.29390e-05   2.73154e-05   7.20322e-05 100.00%
  Percentage           2.47%      59.61% 37.92%      100.00% 100.00%
  -------------------------------------------------------------------------
```

Listing 4: Power report after performing synthesis

4.2 Estimated area

After synthesizing the battery charger controller block, we use the *report_area* command in Genus to generate an estimate of the surface area of this block.

```
@genus:root: 10> report_area
============================================================
  Generated by:           Genus(TM) Synthesis Solution 19.14-s108_1
  Generated on:           Dec 10 2024  10:18:19 am
  Module:                 BATCHARGERctr
  Operating conditions:   _nominal_ (balanced_tree)
  Wireload mode:          enclosed
  Area mode:              timing library
============================================================

   Instance   Module  Cell Count  Cell Area  Net Area   Total Area  Wireload
--------------------------------------------------------------------------------
BATCHARGERctr                264   2684.160     0.000     2684.160     enG5K (S)

 (S) = wireload was automatically selected
```

Listing 5: Report Area of the controller block

The synthesis report, as showed above in the listing 5, obtained from Genus for the `BATCHARGERctr` module provides us an overview of the design's area utilization in terms of silicon footprint. The total cell count is 264, with a corresponding **cell area** of $2684.16\,\mu m^2$. This cell area represents the area consumed by the actual logic cells created after the synthesis (e.g., flip-flops, gates, and other standard cells) in the design.

Consequently, the **total area of the design** matches the cell area at $\underline{2684.16\,\mu m^2}$.

The synthesis process utilized the "enclosed" wireload model, automatically selected as the most suitable for the design under the operating conditions nominal. The information regarding the estimated area will be relevant for the next report.

4.3 Fault coverage

After synthesizing and generating the ATPG of the controller, we study the logs obtained following this generation. We are particulary interested in the module fault coverage shown in detail below in the listing 7.

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

                         #Faults    #Tested  #Possibly    #Redund  #Untested   %TCov %ATCov
Total Static                2092       2084          0          0          8   99.62  99.62
```

```
                         Global  Statistics

                      #Faults    #Tested  #Possibly    #Redund  #Untested   %TCov  %ATCov
Total Static             2092       2084          0          0          8   99.62   99.62
***************************************************************************************
```

Listing 6: Fault coverage statistics for the synthesized Batcharger controller using full scan testing

The fault coverage analysis of the synthesized controller was performed using a full scan test mode (testing all registers in the circuit). The test report indicated that the total number of faults analyzed was 2084. Out of these, **2092 faults were successfully** tested, achieving a **test coverage - %TCov** of 99.62 %. Additionally, no were identified as redundant, meaning they do not affect the functionality of the circuit and are not testable.

Therefore, only 8 faults remain untested out of the 2092 faults in total, contributing to an **adjusted test coverage - %ATCov of 99.62 %** when redundant faults are excluded.

One possible reason for the remaining 0.38% untested faults in the fault coverage report could be the power supply pins: `inout dvdd` (digital supply) and `inout dgnd` (digital ground). These signals are typically tied to constant values (1 for `dvdd` and 0 for `dgnd`) and are not toggled during circuit operation. As such, they are inherently untestable by scan-based DFT methods. This hypothesis was tested, by performing synthesis without these pins, yielding the following log.

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

                      #Faults    #Tested  #Possibly    #Redund  #Untested   %TCov  %ATCov
Total Static             1858       1858          0          0          0  100.00  100.00

                         Global  Statistics

                      #Faults    #Tested  #Possibly    #Redund  #Untested   %TCov  %ATCov
Total Static             1858       1858          0          0          0  100.00  100.00
***************************************************************************************
```

Listing 7: Fault coverage statistics for the synthesized Batcharger controller using full scan testing without dvdd and dgnd

On the other hand, ATPG excludes these constant power and ground connections from the fault list, as faults on these signals cannot exhibit observable behavior in standard testing. Consequently, these pins contribute to the small percentage of untested faults in the coverage report, despite their critical role in ensuring proper circuit operation.

5 Test multiplexer and wire bond test

Starting with the wire bond test, our goal is to test all possible inputs. Since *vsensbat*, *vin*, and *vbattemp* are actually analog inputs, but represented with 64 bits, it is only possible to test the input *sel* and *en* using a wire bond test. Here, the goal is to test the input connections, connecting them directly to the test output. We also assume that if one of the connections fail, the circuit will go to waste, eliminating the need for debug, that is, the need to know which connection is failing. At the same time, it is necessary to test the scan mode of the controller, so the test multiplexer has to be designed to be able to address both tests.

Regarding the usage of controller scan in the full charger, it was possible to adapt the input pin $sel[0]$ to serve as scan input (si), saving an extra pin in the charger. However, since the only output of the charger block is analog (represented in 64 bits), a new output pin *test* has to be added, that will represent either the scan output of the controller (so), or the results of the wire bond test. As a result, a total of two pins (*test* and *se*) were added for test purposes at the full controller level.

Since we assume that lack of need to debug, only the final value of the wire bond is relevant, which means if one part of the test fails, it is possible to assume the device is faulty, discarding it. To be able to control the wire bond, and to have undesired outputs during a normal functioning, a wire bond test enable wb was defined as $NOT(en)\, AND\, sel[1]$. This way, en and $sel[1]$ can be checked for wire bond test, while assuming the remaining pins can be properly tested.

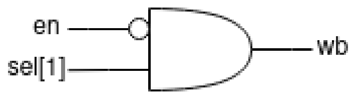

Figure 12: Wire bond test enable, used gate. Since the goal is to no perturb the normal functioning of the device, the wire bond test is only enabled when the general enable is $en = 0$.

The remaining *sel* bits can be checked using $sel[0]$ for inference in the multiplexer. This is, if $wb = 1$, $sel[0]$ can be used to toggled to check $sel[2]$ if $sel[0] = 0$, and $sel[3]$ if $sel[0] = 1$. This way, if any of the results fail, it is possible to consider as faulty and discard it. For scan, the selection in the multiplexer is done using the scan enable *se*, and it takes priority above the wire bond test. The truth table representing the multiplexer can be found in Table 2. The clock supplied by the band-gap module is also used for the scan.

se	wb			sel[0]	testOutput
	en	sel[1]	!en AND sel[1]		
0	1	X	0	X	0
0	0	1	1	0	sel[2]
0	0	1	1	1	sel[3]
1	X	X	X	X	so

Table 2: Test Multiplexer Output Assignment. X represent "don't care".

This truth table can be represented using a $MUX5:1$, with 3 selecting bits (*se*, *wb*, and $sel[0]$). If neither *se* or *wb* are activated, as it would during the controller's regular functioning, the output is forced to 0. A general schematic of the multiplexer implemented in the charger top module can be found in Figure 13, with the corresponding code below.

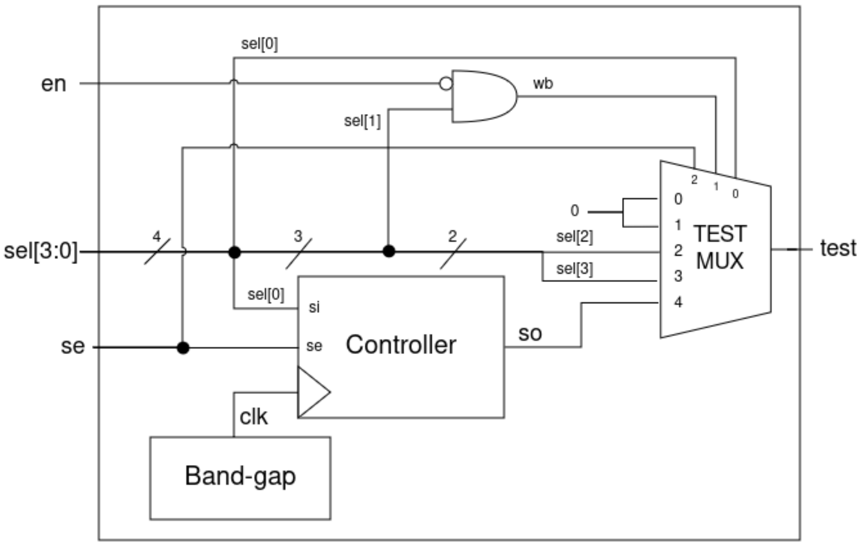

Figure 13: General schematic of the test multiplexer connections. If $se = 1$, the scan mode is enabled, and the output of the controller will be the same as `IDLE`. If $en = 1$, the output will be $0'b0$. The wire bond test will only work if $en = 0$ and $sel[1] = 1$.

```
always @(*) begin
    if (se) begin
        test <= so;
    end else if ((!en && sel[1]) && !sel[0]) begin
        test <= sel[2];
    end else if ((!en && sel[1]) && sel[0]) begin
        test <= sel[3];
    end else begin
        test <= 1'b0;
    end
end
```

Listing 8: Verilog Logic for Test Output