Systems On-Chip

Lab 1

Verilog design and simulation

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 5th 2022

1 Introduction [1, 2]

In this laboratory assignment, the Cadence SimVision tool will be used for the Verilog design and simulation of a **battery charger**. Initially, the simulations will be run with an already existing design, which includes an encrypted file for the **controller block**. A new controller is therefore designed - according to a requested functionality - and validated by simulation, along with the complete charger with this new controller block. The files from the original design (available in a given `tar` file) and files created for the full charger and new design simulations are shown in Table 1. The proposed battery charger topology includes four blocks:

Files	Source
BATCHARGER_64b.v	Initially available in `tar` file
BATCHARGER_64b_tb.v	
BATCHARGERbg_64b.v	
BATCHARGERpower_64b.vp	
BATCHARGERpower_64b_tb.v	
BATCHARGERsaradc_64b.v	
BATCHARGERsaradc_64b_tb.v	
BATCHARGERctr.vp	
BATCHARGERctr_tb.v	
BATCHARGERlipo_64b.v	
BATCHARGERlipo_64b_tb.v	
BATCHARGER_tb.v	
BATCHARGERctr.v	New controller design and scripts for running simulations
BATCHARGERctr_ourtb.v	
sim_rtl	
sim_rtl_all	
sim_rtl_our_ctr_tb	
sim_rtl_our_all	

Table 1: Files included in directory `/DIGITAL/SIMULATION/BATTERYCHARGER`.

- `BATCHARGERbg` - used to generate the reference voltage, reference current, clock and power-on reset, for which the file `BATCHARGERbg_64b.v` is used;
- `BATCHARGERpower` - for which `BATCHARGERpower_64b_tb.v` (testbench) and the encrypted file `BATCHARGERpower_64b.vp` are used; this block controls the charging current according to a digital control signal from the control block - `tc=1` for the trickle current mode (current value configured by 8-bit word `itc`), `cc=1` for the constant current mode (with the 8-bit word `icc`) and `cv=1` for the constant voltage mode (with the 8-bit word `icv`);
- `BATCHARGERsaradc` - the outputs `tbat[7:0]`, `vbat[7:0]` and `ibat[7:0]`, resulting from the analog-to-digital conversion (in an ideally linear scale in which `vref=0.5V` corresponds to the maximum digital value `8'b1111_1111`) of the battery temperature, voltage and current (respectively), are refreshed periodically if enabled by the respective `tmeasen`, `vmeasen` and `imeasen`. These procedures are defined in `BATCHARGERsaradc_64b.v` and can be tested with `BATCHARGERsaradc_64b_tb.v`;
- `BATCHARGERctrl` - coordinates the current mode of the power block (TC, CC or CV) based on the battery temperature, current and voltage provided as digital words from the ADC. The corresponding testbench `BATTERYCHARGERctr_tb.v` is initially used alongside the encrypted file `BATCHARHERctr.vp`, after which a new controller design is made in `BATCHARHERctr.v` to run new simulations.

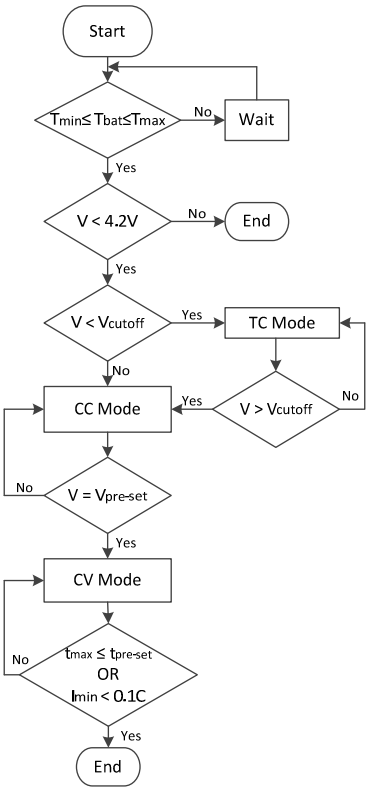


Figure 1: Flow chart of CC/CV charger [2].

Once inside the proper directory containing the files mentioned above, the search path and other environment variables can be configured with the command `source/opt/ic_tools/init/init-xcelium20-09-hf001`. After that, the RTL code is compiled and the initial battery charger controller design can be simulated with the respective script by running `./sim_rtl`. For the purpose of simulating the complete battery charger, the script `sim_rtl_all` can otherwise be used, whereas `sim_rtl_our_ctr_tb` and `sim_rtl_our_all` were created to simulate the new controller block and the complete charger with this new design, respectively.

The design of the new controller block of the battery charger was made according to the functionality described in Figure 1. The corresponding article presents an overview of charging

algorithms for lithium-ion batteries, which include the well developed, simple, easy to implement and widely adopted constant current-constant voltage (CC/CV), used as a benchmark to compare with other charging algorithms in terms of charging time and efficiency, influences on battery life and other aspects.

The charging process of the CC/CV consists of three steps. As shown in the flow chart (Figure 1), safety and protection checks are initially performed by verifying if the battery initial conditions (temperature and open circuit voltage, OCV) are in the respective normal ranges. **Trickle charge (TC)** mode is used to charge the battery (with a small current) in case the voltage is lower than $\mathrm{V_{cutoff}}$, until it increases above it. Once $\mathrm{V} \geq \mathrm{V_{cutoff}}$, the **constant current (CC)** mode is used to further charge the battery. Finally, when the battery voltage reaches $\mathrm{V_{pre-set}}$ (preset maximum charging voltage), the process switches to **constant voltage (CV)** mode, in which charging voltage is held constant at $\mathrm{V_{pre-set}}$ and the charging current decreases exponentially - the charging process stops when the charging current reaches a preset small current, as shown in Figure 2.

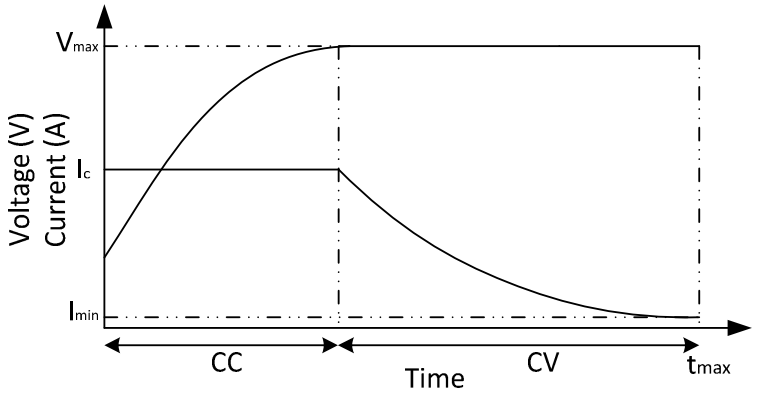


Figure 2: Charging profile of CC/CV (constant current-constant voltage) [2].

2 Original controller and full charger simulations

2.1 Controller

Initially, using the script `sim_rtl` shown in Figure 3, the original controller design described in the encrypted file `BATCHARGERctr.vp` was simulated with the testbench `BATCHARGERctr_tb.v`, previously included in the `tar` file. The respective results are shown in Figure 4.

```
rm -Rf ../worklib/* ../worklib/.*
##
xmvlog BATCHARGERctr_tb.v BATCHARGERctr.vp
xmelab -access +rwc BATCHARGERctr_tb
xmsim -gui BATCHARGERctr_tb &
```

Figure 3: Script created to separately simulate the original controller design, using `./sim_rtl` on the command line.

As shown in Figure 4, the variables `tc`, `cc` and `cv` all start with a value of '0'. The first variable to change is `tc`; its change to '1' indicates that the condition `Vbat<Vcutoff` is verified - as indicated in the SimVision console. During the whole process (ran for 22 070 000ps), none of the bits of the variable `tbat[7:0]` change, thus the temperature value remains in its normal range (i.e., $\mathrm{T_{min}} \leq \mathrm{T_{bat}} \leq \mathrm{T_{max}}$). On the other hand, the variable `vbat[7:0]` changes (due to changes in each individual bit) and TC mode is only reached after some initial time has passed.

Shortly after, the value of `tc` changes to '0' once again and this comes with a change in `cc` from '0' to '1', which indicates that CC mode is entered - due to changes in the battery voltage. After this, the variable `cc` returns to zero, as `cv` mode changes to '1', due to an increase in $\mathrm{V_{bat}}$, whose value surpasses $\mathrm{V_{pre-set}}$. Apart from some changes in `vbat[7:0]` shortly after this, the charger remains

in CV mode for most of the simulation run time (from about 252 500ps to about 20 740 000ps) and all parameters shown in Figure 4 (apart from `clk`, of course) remain constant.

Towards the end of the run time, the charger ends up leaving CV mode and almost immediately afterwards enters CC mode, since `cv` and `cc` change to '0' and '1', respectively. This happened due to changes in `imonen`, `vmonen` and `tmonen` (`wire` variables which enable the current, voltage and temperature monitors, respectively) - which might have occurred because the condition $t_{max} \leq t_{pre-set}$ was checked (thus, CV mode was exited and the system goes to CC mode due to the voltage value), since `vmonen=0` when in CV mode (voltage is not monitored). When the battery voltage changes once again, the system re-enters CV mode. However, this only happens during a short period of time, since `ibat[7:0]` changes and the system leaves CV mode. However, in this case, the system does not return to CC mode. Now, all parameters `cc`, `cv` and `tc` remain at zero. Due to a delay of 1000ns present at the end of `BATCHARGERctr_tb.v`, the simulation eventually ends 1000ns after `cv` changes permanently to '0'.

As a side note, the other parameters shown in the simulation (`ccmode`, `cvmode`, `dgnd`, `dvdd`, `en`, `end1`, `iend[7:0]`, `rstz`, `start`, `tbat[7:0]`, `tcmode`, `tempmax[7:0]`, `tempmin[7:0]`, `tmax[7:0]`, `vcutoff[7:0]`, `vpreset[7:0]` and `vtok`) remain constant throughout the simulation run time (excluding right after 0ps, when the simulation begins, and when 22 070 000ps is reached). In fact, some of these - such as `tempmax[7:0]` or `vcutoff[7:0]` - are defined as constants in order to be used as comparative terms to control changes in other simulation parameters.

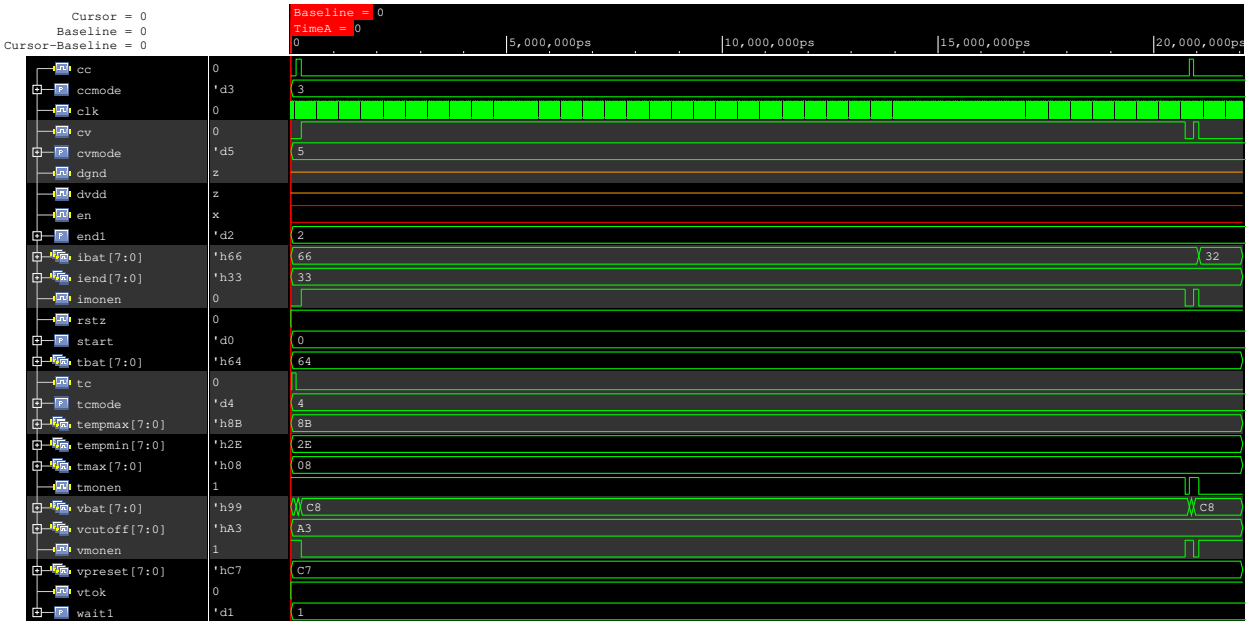


Figure 4: Waveform window obtained by simulating the original controller design described in the encrypted file `BATCHARGERctr.vp` with the testbench `BATCHARGERctr_tb.v`, for a total simulation time of 22 070 000ps.

2.2 Full charger

After this, the complete battery charger was simulated by using the script shown in Figure 5, in which the original controller design (described in the respective encrypted file) has been included and having no change been made in any of the files obtained from the initial `tar`. In this case, the waveform windows shown in Figure 6 were obtained.

```
rm -Rf ../worklib/* ../worklib/.*
##
xmvlog BATCHARGER_64b_tb.v BATCHARGER_64b.v BATCHARGERbg_64b.v BATCHARGERctr.vp BATCHARGERlipo_64b.v BATCHARGERpower_64b.vp BATCHARGERsaradc_64b.v
xmelab -access +rwc BATCHARGER_64b_tb
xmsim -gui BATCHARGER_64b_tb &
```

Figure 5: Script created to simulate the full charger with the original controller design, using `./sim_rtl_all` on the command line.

As shown below, almost immediately after the simulation starts, the system enters TC mode, since the parameter `tc` changes to '1´. Until that time, the variable `tbat[7:0]` does not change, thus the condition $T_{min} \leq T_{bat} \leq T_{max}$ is verified with the initial battery temperature. The variable `vbat[7:0]` also does not change before that, thus $V_{bat} < 4.2V$ and $V_{bat} < V_{cutoff}$ are initially verified. Around 137 000 000ps, the charger goes to CC mode (since `cc=1` and `tc=0`), due to changes in `vbat[7:0]`. The system then remains in CC mode for a long time (approximately until 4 340 000 000ps). In fact, it can be seen that, in this time interval, `rl_vbat` increases as initially represented in Figure 2.

In order to leave CC mode, `vbat[7:0]` changes and its value becomes the same as `vpreset[7:0]`, given in binary format by '11000001'. After this, as presented in the flow chart of Figure 1, the charger goes to CV mode (`cv=1`), in which no bits in `vbat[7:0]` change, while the variable `ibat[7:0]` changes significantly; in fact, it can be seen that `rl_ibat` decreases. When `ibat[7:0]` equals `iend[7:0]` (the constant used here for the "end of charging" criteria), given in binary format by '00000010', `cv` changes to '0'; CV mode was therefore exited due to the current criteria, not by $t_{max} \leq t_{pre-set}$.

Once this happens, all parameters `tc`, `cc` and `cv` remain at '0'. The only variables that change in this time are `vtbat[63:0]` (battery temperature) and `clk`. The simulation eventually ends at 20 000 000 000ps because two delays of 10 000 000ns (i.e., 10 000 000 000ps each) were introduced in `BATCHARGER_64b_tb.v`.

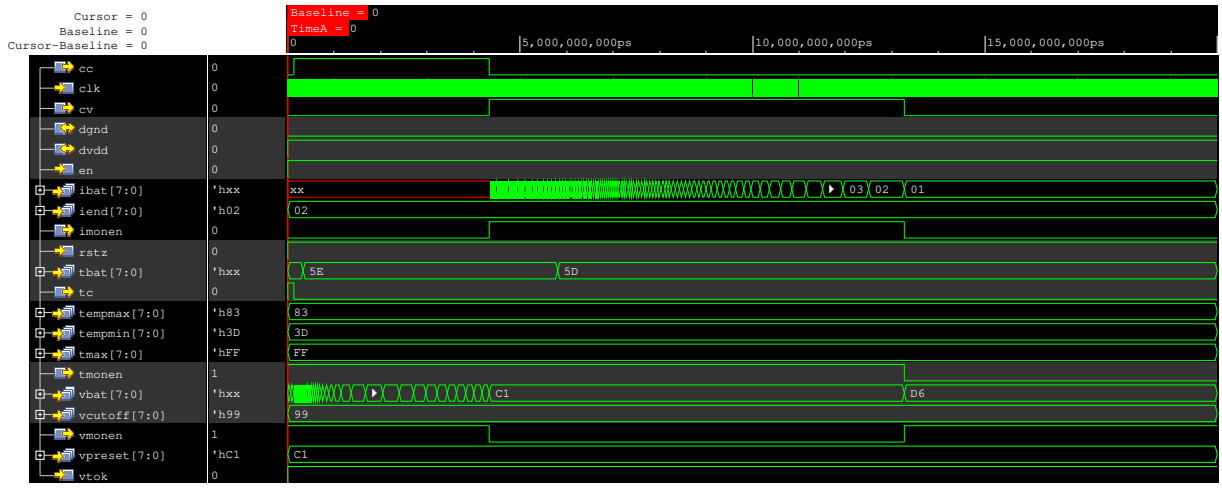


Figure 6: Waveform windows obtained by simulating the full battery charger with the original controller design, using the testbench `BATCHARGER_64b_tb.v`, for a total simulation time of 20 000 000 000ps. On the bottom, only the respective simulation results for the parameters of the controller block are shown.

3 Designed controller and full charger

3.1 Controller

Having tested the original controller and the respective full charger, a new controller design was implemented in `BATCHARGERctr.v`. For this purpose, the same header as in the encrypted file for the previous design was used here. The new controller was implemented based on the functionalities described in section 1, but with some modifications, as shown in the diagram of Figure 7. As seen here, the temperature, voltage and current conditions to leave and enter the states `start`, `cc_mode`, `cv_mode` and `tc_mode` are the same as before. However, there are no `wait` and `end` modes (previously included in `BATCHARGERctr_tb.v`). Now, when the system leaves CV mode or is not ready to enter CC nor TC modes, it goes to `start` once again, thus the charging process can be reinitiated - the charger can continue its operation after the first charging cycle ends. In practice, this will not occur infinitely due to the delays introduced in the testbenches before the `$finish` command.

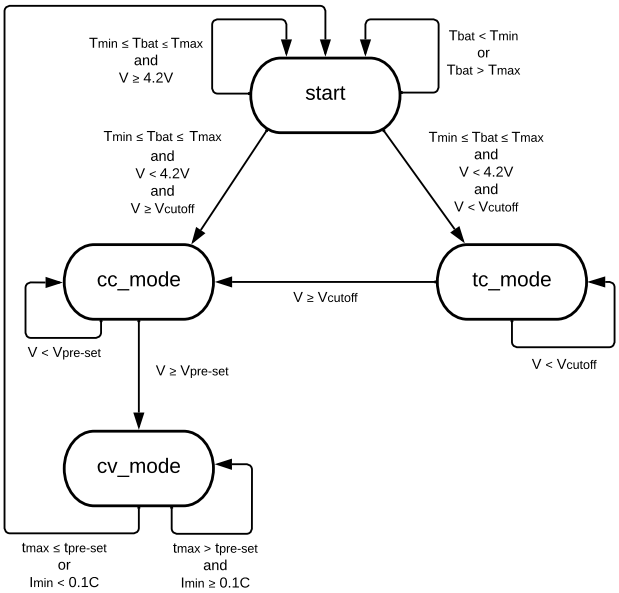


Figure 7: Alternative diagram (based on Figure 1) used to implement the new controller design.

Additionally, according to the state diagram previously shown in Figure 1, the battery temperature would only be monitored at one specific state. Nevertheless, the monitoring of the battery temperature is now carried permanently during the charging process. Moreover, in the new controller design, the condition used to leave CC mode is now $\mathrm{V} \geq \mathrm{V}_{\mathrm{pre-set}}$ (instead of $\mathrm{V} = \mathrm{V}_{\mathrm{pre-set}}$) to make sure that a jump in the voltage above $\mathrm{V}_{\mathrm{pre-set}}$ (without equalling its value) also leads to CV mode.

The Verilog code developed for this new controller design is included in `BATCHARGERctr.v` and is shown in Figure 8. Alongside the initial header contained in the encrypted file, the registers `state` and `nxt_state` were created to store the current and next states, respectively. Additionally, the 16-bit `timecv` was created to test for the condition $\mathrm{t}_{\mathrm{max}} \leq \mathrm{t}_{\mathrm{pre-set}}$ used to leave CV mode. For this purpose, every time $2^8 = 256$ clock cycles pass, the bits `reg[15:8]` change by adding `16'h0001`; once these more significant bits equal or surpass `tmax[7:0]`, CV mode is exited. In other states, this time counting is not performed.

```
module BATCHARGERctr (
                        output reg   cc, // output to analog block: constant current mode with ich current
                        output reg   tc, // output to analog block: trickle mode with 0.1 x ich current
                        output reg   cv, // output to analog block: constant voltage mode vpreset voltage
                        output reg   imonen, // enables current monitor
                        output reg   vmonen, // enables voltage monitor
                        output reg   tmonen, // enables temperature monitor
                        input        vtok, // signals that voltage and temperature values are valid
                        input [7:0]  vbat, // 8 bits data from adc with battery voltage; vbat = adc(vref=0.5V, battery_voltage /10)
                        input [7:0]  ibat, // 8 bits data from adc with battery current; ibat = adc(vref=0.5V, battery_current * Rsens);
                                           // Rsens = 0.5*vref/(0.5C) ; C=nominal capacity of battery; vadc(Ibat=0.5C) = vref/2
                        input [7:0]  tbat, // 8 bits data from adc with battery temperature;  vadc = Temp/330 + 20/165 ;
                                           // tbat = adc(vref=0.5V, vadc); vadc(-40º)=0V, vadc(125º)=0.5V
                        input [7:0]  vcutoff, // constant from OTP: voltage threshold for exiting trickle mode;
                                              // vcutoff = Vcutof_dec * 255/5 = 51 * Vcutof_dec Ex: 2.9V - 1001_0011
                        input [7:0]  vpreset, // constant from OTP: voltage for constant voltage mode;
                                              //vpreset = Vpreset_dec * 255/5 = 51 * Vpreset_dec Ex: 3.7V - 1011_1100
                        input [7:0]  tempmin, // constant from OTP: minimum temperature ; see tbat for scaling
                        input [7:0]  tempmax, // constant from OTP: maximum temperature ; see tbat for scaling
                        input [7:0]  tmax, // constant from OTP: maximum charge time; unit is 2^time_div_bits clock cycles
                        input [7:0]  iend, // charge current to be used as "end of charging" end criteria
                                           // Ex: 0.01C=0.01*3.5=0.035 0000_0010
                        input        clk, // state machine clock
                        input        en, // enables the module
                        input        rstz, // system reset
                        inout        dvdd, // digital supply
                        inout        dgnd // digital ground
                        );

reg[1:0] state, nxt_state;
reg[15:0] timecv;

parameter start=0,cc_mode=1,tc_mode=2, cv_mode=3;

always @(state or tbat or vbat or ibat or timecv)
        begin:next_state_logic
        case (state)
                start:begin
                        if(tbat < tempmin || tbat > tempmax) nxt_state=start;
                        else if (vbat > 8'b1101_0110) nxt_state = start; // V>4.2
                        else if (vbat > vcutoff) nxt_state = cc_mode;
                        else nxt_state = tc_mode;
                end

                cc_mode : begin
                        if(tbat < tempmin || tbat > tempmax) nxt_state=start;
                        else if (vbat >= vpreset) nxt_state = cv_mode;
                        else nxt_state = cc_mode;
                end

                tc_mode :begin
                        if(tbat < tempmin || tbat > tempmax) nxt_state=start;
                        else if (vbat > vcutoff) nxt_state = cc_mode;
                        else nxt_state = tc_mode;
                end

                cv_mode: begin
                        if(tbat < tempmin || tbat > tempmax) nxt_state=start;
                        else if ((tmax[7:0]<=timecv[15:8])||(ibat < iend)) nxt_state = start;
                        else nxt_state = cv_mode;
                end

        endcase
end

always @(posedge clk or posedge rstz)
        begin : register_generation
                if (rstz==0) state = start;
                else state = nxt_state;
        end

always @(posedge clk or posedge rstz)
        begin : clock_time
                if (rstz==0) timecv = 16'b0;
                else if (state==cv_mode) timecv=timecv+16'h0001; // count time in cv_mode
                else timecv = 16'b0;
        end

always @(state)begin:outputlogic
        case(state)
                cc_mode:begin
                        cc=1;
                        cv=0;
                        tc=0;
                        imonen=0;
                        vmonen=1;
                        tmonen=1; // in all states we have to monitor the temperature
                        end
                cv_mode:begin
                        cc=0;
                        cv=1;
                        tc=0;
                        imonen=1;  // only in cv_mode we have to monitor the current
                        vmonen=0;
                        tmonen=1;
                        end
                tc_mode:begin
                        cc=0;
                        cv=0;
                        tc=1;
                        imonen=0;
                        vmonen=1;
                        tmonen=1;
                        end
                start:begin
                        cc=0;
                        cv=0;
                        tc=0;
                        imonen=0;
                        vmonen=1;
                        tmonen=1;
                        end
                default:begin
                        cc=0;
                        cv=0;
                        tc=0;
                        imonen=0;
                        vmonen=1;
                        tmonen=1;
                        end
        endcase

end

endmodule
```

Figure 8: Verilog code of the new charger controller design, included in `BATCHARGERctr.v`.

Having written this code, the script shown in Figure 9 was used to test the new controller block with a new controller testbench (`BATCHARGERctr_ourtb.v`, shown in Figure 11), having the results shown in Figures 10 and 12 been obtained.

```
rm -Rf ../worklib/* ../worklib/.*
##
xmvlog BATCHARGERctr_ourtb.v BATCHARGERctr.v
xmelab -access +rwc BATCHARGERctr_ourtb
xmsim -gui BATCHARGERctr_ourtb &
```

Figure 9: Script created to separately simulate the new controller design with the new controller testbench, using `./sim_rtl_our_ctr_tb` on the command line.

When compared with the original controller testbench (`BATCHARGERctr_tb.v`), the new controller testbench (`BATCHARGERctr_ourtb.v`) includes the same code before the main block (`initial[...] end`), apart from the attribution of values to the four states (previously, the `wait1` and `end1` states also existed). However, in the new testbench, new changes in the voltage and temperature are performed, in order to test if the correct changes in modes are performed. Moreover, the code lines `$display ( "waiting timeout");\\wait(uut.timeout);\\$display ( "timeout ok");`, used with the previous controller design to test the time condition imposed in Figure 1, have been removed (now, this procedure is made with the register `timecv`).

In this new simulation, different changes in the modes are to be expected. The code included in the original testbench led to a change in modes given by `TC→CC →CV→CC→CV`. In this case, the changes in voltage, current and temperature included in the testbench shown in Figure 11 should lead to `CC→CV→TC→CC→CV→CC→CV`, as explained in the code itself.

Having run the proper script, the messages which appeared in the SimVision console (shown in Figure 10) indicate that the desired behaviour of the controller block has been achieved. This will be further verified with the respective waveform window shown in Figure 12.

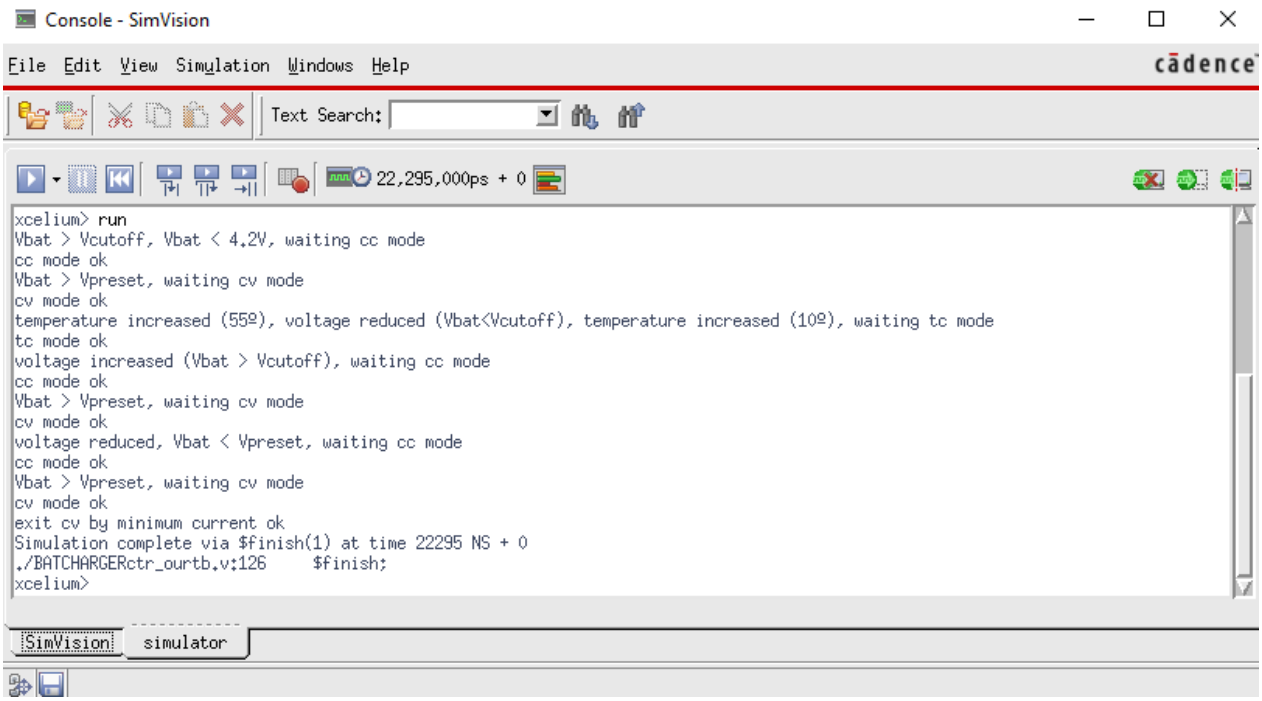


Figure 10: Picture of the messages in the SimVision console obtained by simulating the new controller design described in `BATCHARGERctr.v` with the new testbench `BATCHARGERctr_ourtb.v`.

```
`timescale 1ns / 1ps

module BATCHARGERctr_ourtb;

   wire  cc; // output to analog block: constant current mode with ich current
   wire  tc; // output to analog block: trickle mode with 0.1 x ich current
   wire  cv; // output to analog block: constant voltage mode vpreset voltage
   wire  imonen; // enables current monitor
   wire  vmonen; // enables voltage monitor
   wire  tmonen; // enables temperature monitor
   reg [7:0] vbat; // 8 bits data from adc with battery voltage; vbat = adc(vref=0.5V, battery_voltage /10)
   reg [7:0] ibat; // 8 bits data from adc with battery current; ibat = adc(vref=0.5V, battery_current * Rsens);
                   // Rsens = 1/C ; C=nominal capacity of battery; vadc(Ibat=0.5C)=0.5V
   reg [7:0] tbat; // 8 bits data from adc with battery temperature; vadc = Temp/330 + 20/165 ; tbat = adc(vref=0,5, vadc)
   reg [7:0] vcutoff; // constant from OTP: voltage threshold for exiting trickle mode
   reg [7:0] vpreset; // constant from OTP: voltage for constant voltage mode
   reg [7:0] tempmin; // constant from OTP: minimum temperature
   reg [7:0] tempmax; // constant from OTP: maximum temperature
   reg [7:0] tmax; // constant from OTP: maximum charge time
   reg [7:0] iend; // charge current to be used as "end charging" end criteria
   reg       clk; // state machine clock
   reg       en;
   reg       rstz; // system reset
   reg       vtok; // singals that voltage and temperature values are valid
   wire      dvdd;  // digital supply
   wire      dgnd;  // digital ground

   parameter start=0, cc_mode=1, tc_mode=2, cv_mode=3;

BATCHARGERctr uut(
                     .cc(cc),
                     .tc(tc),
                     .cv(cv),
                     .vtok(vtok),
                     .imonen(imonen),
                     .vmonen(vmonen),
                     .tmonen(tmonen),
                     .vbat(vbat),
                     .ibat(ibat),
                     .tbat(tbat),
                     .vcutoff(vcutoff),
                     .vpreset(vpreset),
                     .tempmin(tempmin),
                     .tempmax(tempmax),
                     .tmax(tmax),
                     .iend(iend),
                     .clk(clk),
                     .rstz(rstz),
                     .en(en),
                     .dvdd(dvdd),
                     .dgnd(dgnd)
                     );

   initial
     begin
        clk=0;
        vtok = 0;
        rstz = 0;    // active 0 reset at the begining

        vbat[7:0] = 8'b11100101 ; // Vbat=4.5V -> after resistor divider: 0.3V -> adc with Vref=0.5V:  vabt=8'b10011001
        ibat[7:0] = 8'b01100110 ; // 8 bits data from adc with battery current: 0.2*C
        tbat[7:0] = 8'b01100100 ; // 8 bits data from adc with battery temperature: 25ºC -> 0.2V -> adc with Vref=0.5V:  tbat=8'b01100100
        vcutoff[7:0] = 8'b10100011; // constant from OTP: voltage threshold for exiting trickle mode; Vcutoff=3.2V
        vpreset[7:0] = 8'b11000111; // constant from OTP: voltage for constant voltage mode; Vpreset=3.9V
        tempmin[7:0] = 8'b00101110; // constant from OTP: minimum temperature: -10ºC
        tempmax[7:0] = 8'b10001011; // constant from OTP: maximum temperature: 50ºC
        tmax[7:0] = 8'b00001000; // constant from OTP: maximum charge time (units of 255 clock periods): 255*8 = 2040 clock periods
        iend[7:0] = 8'b00110011; // charge current div by 10 to be used as charging end criteria: 0.1C

        #12 rstz = 1;   // reset end
        vtok = 1; // voltage and temperature values are valid

        /* Changes in mode: start->cc->cv->start->tc->cc->cv->start->cc-cv*/

        #100  vbat[7:0] = 8'b11000001 ; // Vbat=3.8V to exit start

        $display ( "Vbat > Vcutoff, Vbat < 4.2V, waiting cc mode");
        wait(cc);
        $display ( "cc mode ok");

        #100  vbat[7:0] = 8'b11001000 ; // Vbat=Vpreset+1 to change to cv_mode

        $display ( "Vbat > Vpreset, waiting cv mode");
        wait(cv);
        $display ( "cv mode ok");

        #100 tbat[7:0] = 8'b10010010 ; // tbat=55º to exit cv_mode
        vbat[7:0] = 8'b10011001 ; // Vbat<Vcutoff to enter tc_mode
        #100 tbat[7:0] = 8'b01001101; // tbat=10º to exit start mode

        $display ( "temperature increased (55º), voltage reduced (Vbat<Vcutoff), temperature increased (10º), waiting tc mode");
        wait(tc);
        $display ( "tc mode ok");

        #100  vbat[7:0] = 8'b11001000 ; // Vbat=Vpreset +1 to enter cc_mode and pass to cv_mode

        $display ( "voltage increased (Vbat > Vcutoff), waiting cc mode");
        wait(cc);
        $display ( "cc mode ok");

        $display ( "Vbat > Vpreset, waiting cv mode");
        wait(cv);
        $display ( "cv mode ok");

        #100  vbat[7:0] = 8'b11000110 ; // Vbat=preset -1 to return to cc_mode

        $display ( "voltage reduced, Vbat < Vpreset, waiting cc mode");
        wait(cc);
        $display ( "cc mode ok");

        #100  vbat[7:0] = 8'b11001000 ; // Vbat=preset +1 to exit cc_mode

        $display ( "Vbat > Vpreset, waiting cv mode");
        wait(cv);
        $display ( "cv mode ok");

        #100  ibat[7:0] = 8'b00110010 ; // ibat<0.1C to exit cv_mode
        #30 if (cv==1) begin
           $display( "Error: no exit of cv by minimum current");
           $finish();
        end
        else $display( "exit cv by minimum current ok");

        #1000

        $finish;
     end

   always
     #5 clk = ~clk;

endmodule
```

Figure 11: Verilog code of the new charger controller testbench, included in `BATCHARGERctr_ourtb.v`.

As seen below, having all parameters `cc`, `cv` and `tc` started at zero, `cc` is the first to change to '1', indicating the change to CC mode, which occurs due to a modification in `vbat[7:0]`. After that, new changes in the latter lead to CV mode. As seen below, the temperature `tbat[7:0]` and the voltage `vbat[7:0]` change, which then lead to an exit from CV to TC mode for a short period of time, after which it changes to CC mode and immediately to CV mode, due to changes in the battery voltage. The system then remains in CV mode for most of the run time, until it goes to CC mode. This happens a long time after entering CV mode, since `vmonen` (which enables the voltage monitor) is at '0' when `cv=1`, as described in the Verilog code of Figure 8 - a similarly large time interval was verified with the original controller design, as seen in Figure 4, which indicated that the code in the encrypted file should also consider `vmonen=0` when in CV mode. Thus, only once $t_{max} \leq t_{pre-set}$ is verified should the change in `vbat[7:0]` lead to a change to CC mode (in which `vmonen=1`).

Finally, another change in `vbat[7:0]` leads to CV mode once again, which is exited when the battery current is decreased to '00110010'. Thus, the predicted evolution `CC`→`CV`→`TC`→`CC`→`CV`→`CC`→`CV` has been achieved. A key difference between the results shown in Figures 4 (for the original controller) and 12 is that, in the latter, there is a frenetic back and forth between CC and CV modes, as the parameters `cv` and `cc` commute between '0' and '1'. This is because, as indicated in Figure 7, a continuous loop has now been implemented, with returns to `start` mode (so that the charger can continue its operation after the first charging cycle ends). Due to the values of the battery voltage and current, which do not change anymore and satisfy the conditions $V < 4.2V$, $V \geq V_{cutoff}$, $V > V_{pre-set}$ and $I_{min} < 0.1C$, the system alternates between the two states. Due to the delay `#1000` of 1 000 000ps introduced after the change in the battery current (as shown in Figure 8), the simulation eventually ends after a total run time of 22 295 000ps.

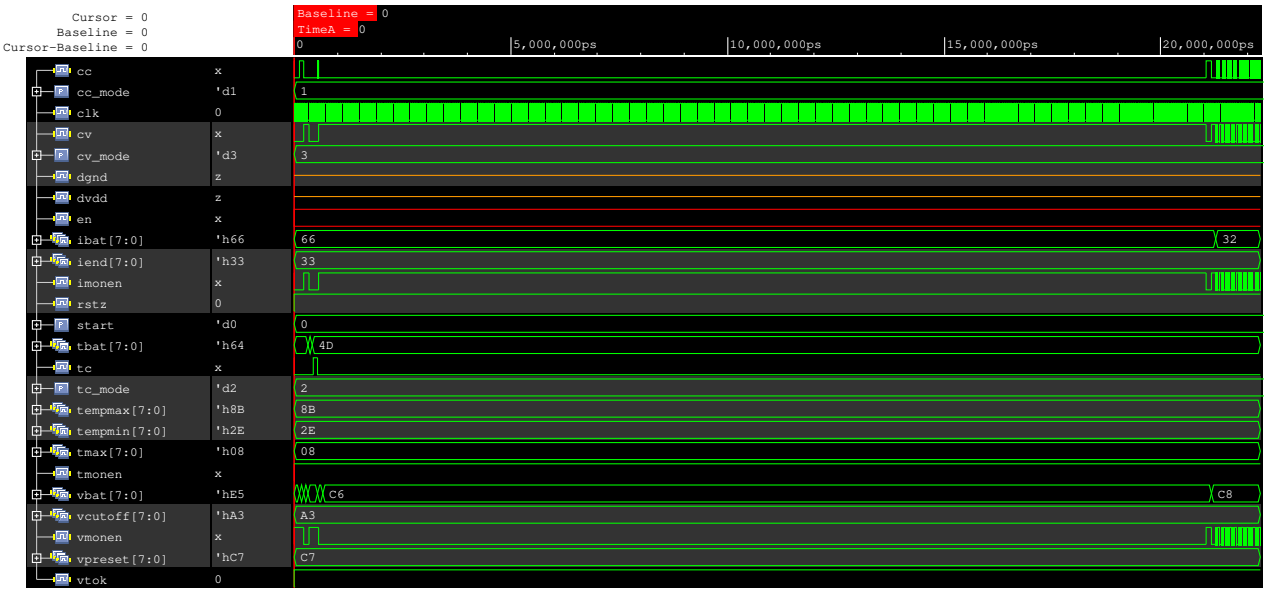


Figure 12: Waveform window obtained by simulating the new controller design described in `BATCHARGERctr.v` with the new testbench `BATCHARGERctr_ourtb.v`, for a total simulation time of 22 295 000ps.

3.2 Full charger

Finally, using the new controller design included in `BATCHARGERctr.v` (Figure 8), the complete charger was validated using the testbench shown in Figure 13. In this case, the testbench is very similar to the one initially obtained from the `tar` file, having only `$display` messages been introduced in the `initial[...]end` block to check the battery voltage value in the middle of the total simulation time (i.e., after 10 000 000 000ps). Due to only these small changes in the testbench, a new file was not created, but instead these line codes were added to the original `BATCHARGER_64b_tb.v`. In order to go further beyond in terms of self-validation of the complete charger, the other

modules of the battery charger (`BATCHARGERbg`, `BATCHARGERpower` and `BATCHARGERsaradc`) should be analysed in detail; however, this goes a bit beyond the scope of this first laboratory assignment, whose main goal is to work on the controller block. In that sense, the simulation performed in this section will prove that the new designed controller works as intended when incorporated in the complete charger architecture.

```
`timescale 1 ns/10 ps

module BATCHARGER_64b_tb;

   wire [63:0] vin; // input voltage; must be at least 200mV higher than vsensbat to allow iforcedbat > 0
   wire [63:0] vbat; // battery voltage (V)
   wire [63:0] ibat; // battery current (A)
   wire [63:0] vtbat; // Battery temperature
   wire [63:0] dvdd; // digital supply
   wire [63:0] dgnd; // digital ground
   wire [63:0] pgnd; // power ground

   reg         en; // enables the module
   reg [3:0]   sel; // battery capacity selection bits:
                    // b[3,2,1,0] weights are 400,200,100,50 mAh + offset of 50mAh covers the range from 50 up to 800 mAh

   real        rl_dvdd, rl_dgnd, rl_pgnd;
   real        rl_ibat, rl_vbat, rl_vtbat;
   real        rl_vin;         // converted value of vin to real

BATCHARGER_64b uut(
                    .iforcedbat(ibat), // output current to battery
                    .vsensbat(vbat), // voltage sensed (obtained at the battery as "voltage from iforcedbat integration" + ESR * iforcedbat)
                    .vin(vin), // input voltage; must be at least 200mV higher than vsensbat to allow iforcedbat > 0
                    .vbattemp(vtbat), // voltage that represents the battery temperature -40ºC to 125ºC -> 0 to 0.5V
                    .en(en), // block enable control
                    .sel(sel), // battery capacity selection bits:
                               // b[3,2,1,0] weights are 400,200,100,50 mAh + offset of 50mAh covers the range from 50 up to 800 mAh
                    .dvdd(dvdd), // digital supply
                    .dgnd(dgnd), // digital ground
                    .pgnd(pgnd) // power ground
);

BATCHARGERlipo lipobattery(
                    .vbat(vbat), // battery voltage (V)
                    .ibat(ibat), // battery current (A)
                    .vtbat(vtbat) // Battery temperature
                    );

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

     #10000000;

     if (rl_vbat>3.9) begin
        $display( "Vbat > Vpreset");
     end
        else $display("Vbat < Vpreset");

     if (rl_vbat>=4.2) begin
        $display( "Vbat >= 4.2V");
     end
        else $display("Vbat < 4.2V");

     #10000000 $finish;

  end

//-- Signals conversion ------------------------------------------------
   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);

endmodule
```

Figure 13: Verilog code of the full charger testbench, included in `BATCHARGERctr_64b_tb.v`.

By running the script shown in Figure 14, the results shown in Figures 15 and 16 were obtained. As seen in the former, the displayed messages indicate that, in the middle of the total simulation time, the battery voltage satisfies $\mathrm{V}_{\mathrm{pre-set}} < \mathrm{V} < 4.2\mathrm{V}$ and that the simulation was performed successfully.

```
rm -Rf ../worklib/* ../worklib/.*
##
xmvlog BATCHARGER_64b_tb.v BATCHARGER_64b.v BATCHARGERbg_64b.v BATCHARGERctr.v BATCHARGERlipo_64b.v BATCHARGERpower_64b.vp BATCHARGERsaradc_64b.v
xmelab -access +rwc BATCHARGER_64b_tb
xmsim -gui BATCHARGER_64b_tb &
```

Figure 14: Script created to simulate the full charger with the new controller design, using `./sim_rtl_our_all` on the command line.

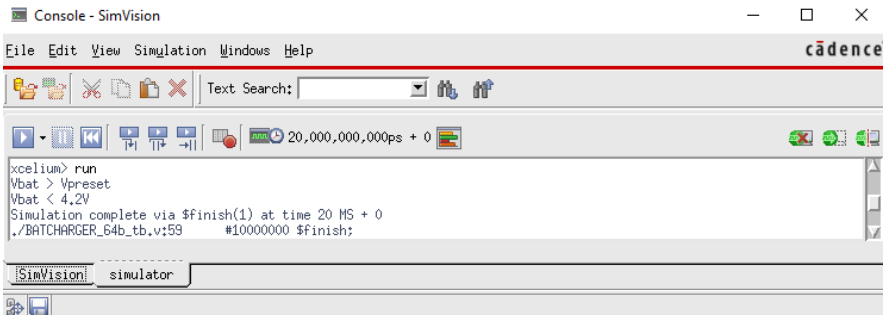
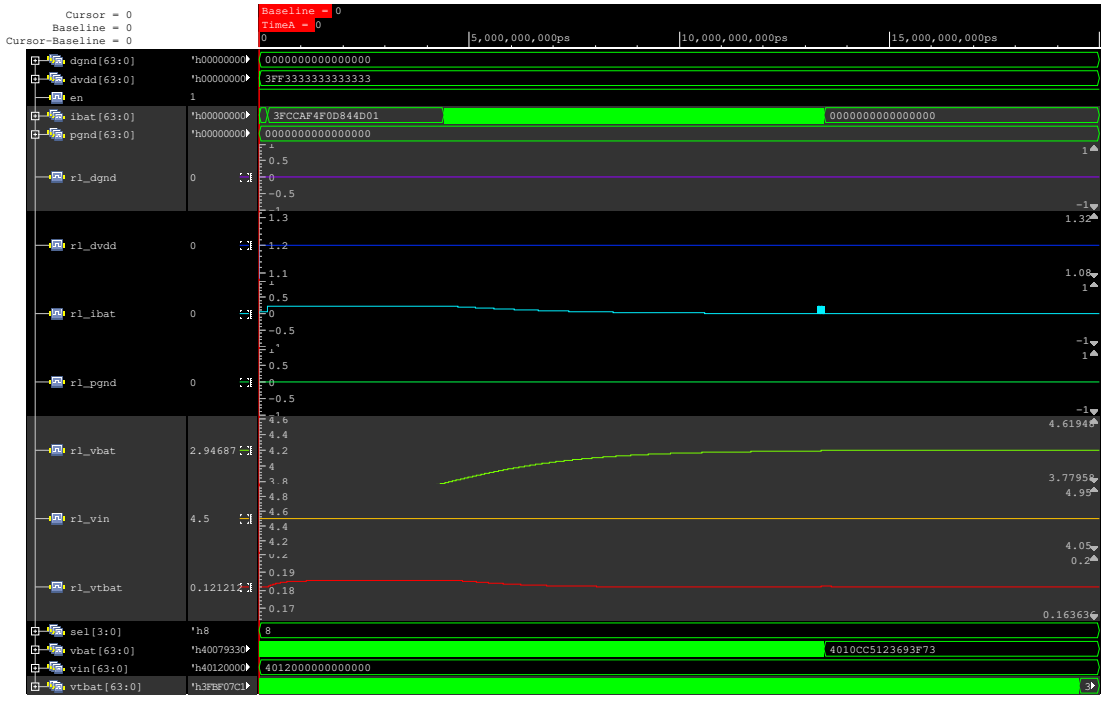


Figure 15: Picture of the messages in the SimVision console obtained by simulating the complete charger with the new controller design using a slightly altered testbench `BATCHARGERctr_ourtb.v`.

As intended, similar graphs to the ones in Figure 6 (original complete charger) have now been obtained. Initially, the battery voltage (whose real value is given by `rl_vbat`) has its minimum value, whereas the current (whose real values is given by `rl_ibat`) reaches its maximum value soon after the simulation begins (same as in Figure 6) and decreases with time. Initially, the system is in TC mode, but changes to CC mode at around 200 000 000ps, due to the increase in the battery voltage ($V > V_{cutoff}$, where `vcutoff[7:0]`=10011001≡'h99). At around 4 400 000 000ps, the charger enters CV mode, due to an increase in the battery voltage; in fact, it can be checked that this occurs when `rl_vbat`≈3.77V, which means that the real value of $V_{pre-set}$ is not actually 3.9V, but lower (in binary format, it can be checked that its value is '11000001'). As CV mode is entered, a noticeable decrease in the current `rl_ibat` occurs and the voltage continues to increase.

At a certain point in time (around 13 300 000 000ps), `rl_ibat`≈0.00350 and `rl_vbat`≈4.19 and the system leaves CV mode. A key difference seen in these simulation results (when compared with Figure 6) is that, around this point, a rapid alternation between CC and CV mode occurs, as the current value suffers rapid changes. As explained in section 3.1, this happens because of the loop implemented in the new controller design (and represented in Figure 7); since $V_{cutoff} < V < V_{pre-set}$, the controller leads to a continuous alternation between these two modes and the current suffers some modifications because of them. This situation can be avoided in the future by changing the code of other blocks of the battery charger. After these moments, the parameter `rl_vbat` reaches its maximum value of 4.19953 and all parameters `cc`, `cv` and `tc` remain at zero. The simulation ends at 200 20 000 000ns due to the respective delays introduced in the testbench shown in Figure 13.



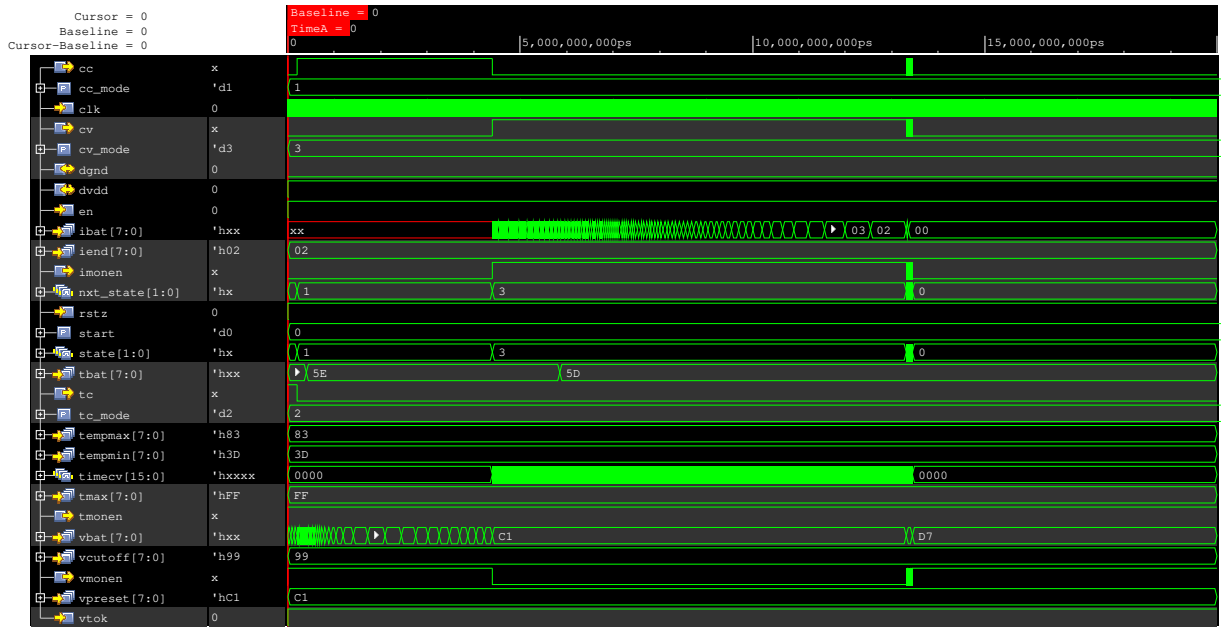


Figure 16: Waveform windows obtained by simulating the complete battery charger with the new controller design, using the testbench `BATCHARGER_64b_tb.v`, for a total simulation time of 20 000 000 000ps. On the bottom, only the respective simulation results for the parameters of the controller block are shown.

4 Conclusion

In this laboratory assignment, all of the proposed objectives have been achieved. By referring to the course documents and the paper containing the flow chart of the constant-current-constant voltage (CC/CV) charger, its proposed functionality was analysed in detail. Using the encrypted file with an original controller design, the desired behaviour was obtained through simulation. Using this same Verilog code, along with the files containing the architecture of the other blocks, the full charger simulation was also performed and the intended charging functionality was confirmed.

In compliance with the proper specifications and design practices, a new controller block Verilog code was developed, along with a new controller testbench. For this purpose, an alternative flow chart was considered, in which the battery temperature is monitored in every state and the charging process can be reinitiated. The proper simulations validated this controller and the complete charger functionality using this new design.

References

[1] Documents available in the subject's Fenix webpage.

[2] Weixiang Shen, Thanh Tu Vo and Ajay Kapoor. "Charging algorithms of lithium-ion batteries: an overview". In: *IEEE Conference on Industrial Electronics and Applications (ICIEA)* (July 2012), pp. 1567–1572.