University of Southern Denmark

Robot Electronics

Robot Systems 7th semester - Fall 2019

Smart Battery Charging and Monitoring System

Group 3

Emil Vincent Ancker	Mathias Emil Slettemark-Nielsen	Mikkel Larsen
09/06-1997	23/02-1995	08/08-1996
emanc16@student.sdu.dk	masle16@student.sdu.dk	milar16@student.sdu.dk

Supervisor: Emad Samuel Malki Ebeid & Martin Skriver

Project period: 14.11.2019 - 19.12.2019

Contents

1 Introduction

This report explores how to build an advanced electronic circuit that interacts with analogue and digital signals by creating a smart battery charging and monitoring system.

The goal is to design a circuit that can charge batteries of different voltages, e.g. 1.5 volt, 3 volt, 4.5 volt, etc., and monitor the state of charge and the remaining charging time.

It is chosen to use a combination of the Processing System, PS, and Programmable Logic, PL, for the battery management. Figure 1 shows an illustration of the desired system.

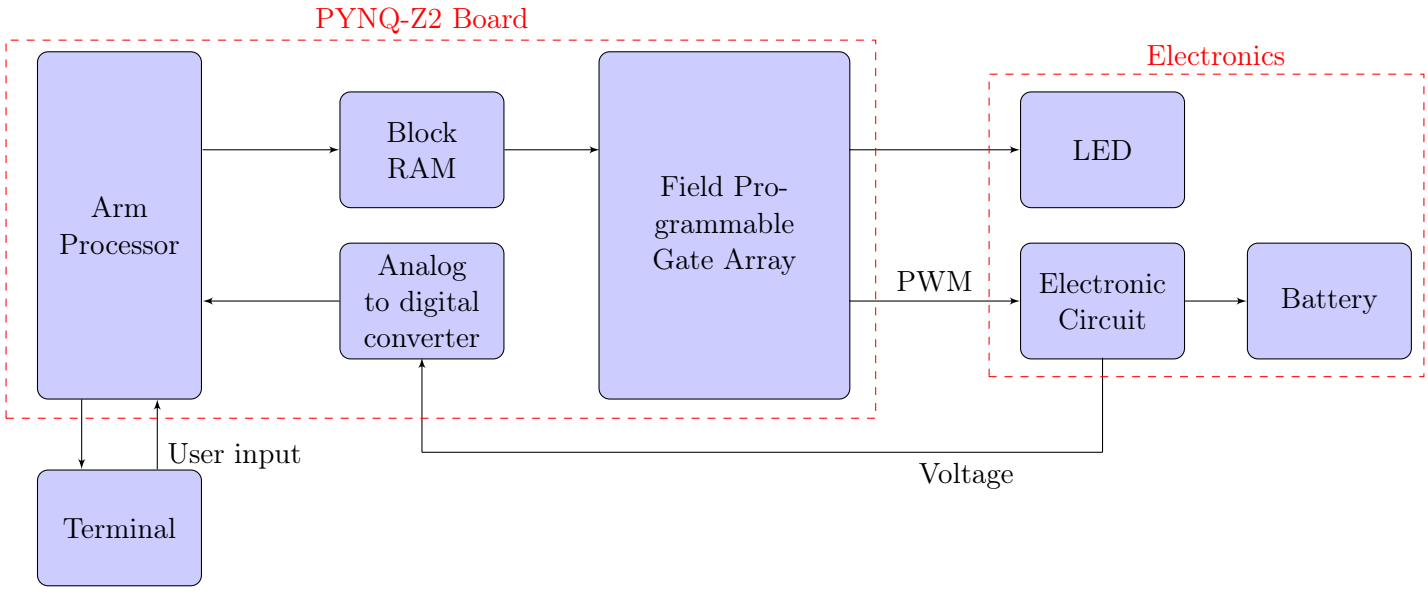


Figure 1: Diagram of the system

In Figure 1, the overall system is illustrated, and the pipeline of the system is outlined below.

1. The battery is placed in the battery holder.
2. The user enters the battery capacity, nominal voltage, rate of charge, and charging speed via terminal.
3. The Arm Processor calculates an increase or decrease in Pulse-Width Modulation, PWM, duty cycle and the current battery level. This is done based upon the user inputs and the voltage from the electronic circuit, which is measured by the Analog to Digital Converter, ADC.
4. The Field-Programmable Gate Array, FPGA, reads from Block Random Access Memory, BRAM, and acts upon the readings. The PWM signal is adjusted based on the value from the BRAM, describing if an increasing or decrease in PWM value

is needed. Furthermore, if the battery level is above 20 % a Light-Emitting Diode, LED, is turned on.

5. Step 3 and 4 is repeated until the measured battery voltage has reached the nominal voltage, which was specified by the user.

The code for the project can be found at [1].

2 Equipment & Components

The following equipments have been used in this project:

Power supply	Tenma DC Power Supply 72-8690A
Oscilloscope	RS Pro Digital Storage Oscilloscope 200MHz 2GS/s IDS-2204Axt
Probes	Agilent N2863B 10:1 Passive Probe 300MHz 10MΩ//15pF
Breadboard	From stock
FPGA board	PYNQ-Z2 Development Kit.
Multimeter	RS-14 Digital Multimeter, RS PRO.

The following components have been used in this project:

- 2x 4.7[$k\Omega$] resistors (1% tolerance).
- 2x 1.6[$k\Omega$] resistors (1% tolerance).
- 2x 10[Ω] resistors (1% tolerance).
- 1x 10[$k\Omega$] resistor (1% tolerance).
- 1x 1[$k\Omega$] resistors (1% tolerance).
- 1x 0.22[Ω] power resistor (5% tolerance) measured to 0.25[Ω].
- 1x 33[Ω] power resistor (5% tolerance).
- 1x 4.7[mH] inductor.
- 1x 1.5[μF] capacitor.
- 1x IRLZ34N N-channel MOSFET.
- 1x IRF9520 P-channel MOSFET.
- 2x 1n4002 diodes.
- Single, double & triple battery holders.
- Vishay TDSR1360 7-segment.
- LM324 Quad-Operational Amplifiers.
- NiMH 1.2V RS PRO battery.
- Solid wires.

3 Background

Theory required to understand the design and choices in the report will be described in this section.

3.1 Block Random Access Memory

One way to make a communication between the processor system, PS, and the programmable logic, PL, on the PYNQ Z2 board is through BRAM. BRAM is located on the FPGA and is used to store data and has a fixed size width and depth. The width of the BRAM dictates the length of a word and the depth dictates the number of words that can be stored. BRAM is synchronized via a clock signal. In order to get the correct data out of the BRAM, an address is used to point at the data location. Furthermore, a write enable pin is used to dictate whether to write or read. A simple model of BRAM in dual-port configuration is illustrated in Figure 2.

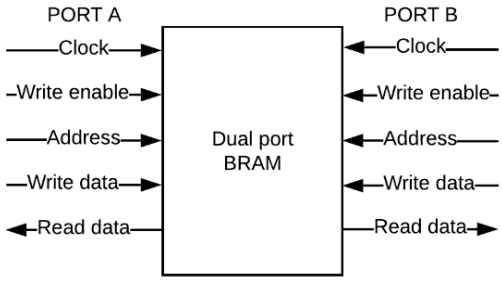


Figure 2: Illustration of the dual-port BRAN

3.2 Analog to Digital Converter

To convert an analog signal to a digital signal an ADC, is used. The ADC converts the analog input periodically, thus the digital signal is a discretization of the analog signal. At each sampling time the ADC samples the analog signal value as a discrete value. An illustration can be seen in Figure 3, in which the digital signal measures the analog signal value and hold its until a new measurement is done.

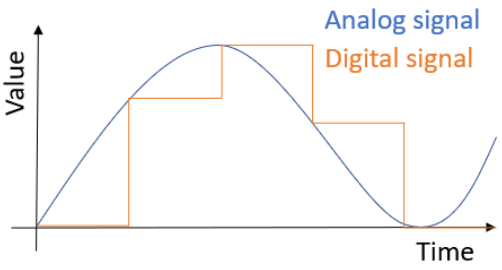


Figure 3: Illustration of analog to digital convertion.

3.3 Pulse-Width Modulation

PWM is a technique to control the average voltage applied to a component or system. This can be used to control a charging circuit which charges batteries since it would enable control of applied voltage and current to a battery.

A PWM signal is generated by usage of a timer, which is visualized in Figure 4.

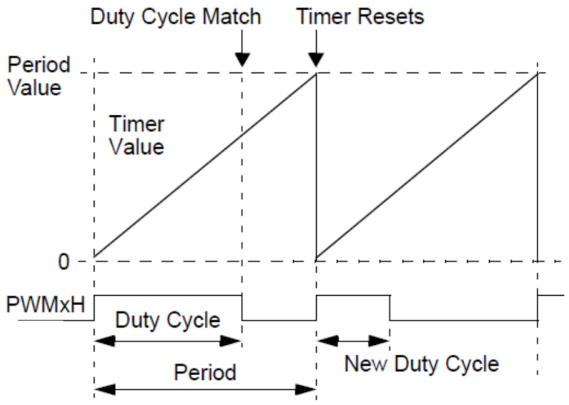


Figure 4: Visualization of a PWM signal with a timer. Figure is from lecture slides by Emad Samuel Malki Ebeid.

The duty cycle is the ratio of high output signal to low output signal. The duty cycle can be controlled by changing the threshold used to generate the PWM signal.

3.4 Buck Converter

Applying the PWM signal described above directly to a battery would probably damage the battery depending on the source voltage. Without the usage of circuits like buck converters, the charging circuit would either damage the battery during recharging or it would lose flexibility with respect to charging multiple kinds of batteries.

Figure 5: Buck converter switching principle [2].

The principle of using a buck circuit with a PWM input controlling a switch is shown in Figure 5. When the switch is closed the capacitor and inductor is charged when the switch opens the inductor will maintain the current by inducing a voltage.

The effect of the buck circuit is visualized in Figure 6.

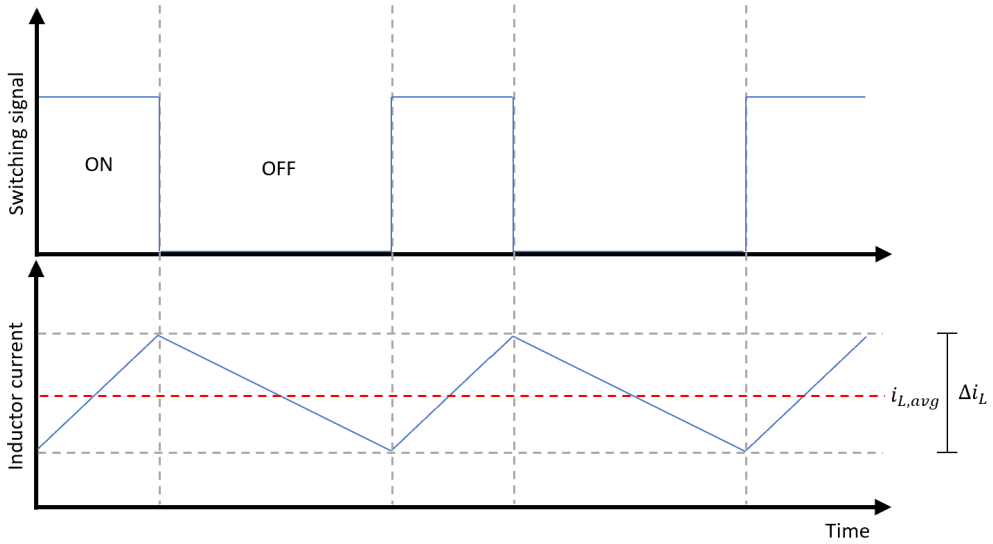


Figure 6: Visualization of the current looping in the buck circuit and the switching input.

It is seen that the Buck circuit tries to maintain the current when the PWM signal is in the off state, this is also the case for the voltage. The buck circuit will be designed based on the ripple Δi_L and the voltage ripple ΔV_C, which is not shown in Figure 6.

4 Designing the Electrical Charging Circuit

The purpose of the electrical circuit is to charge various types of battery, which results in a requirement of the circuit to be very flexible. Furthermore, there are a set of requirements for the circuit to be able to monitor and control the charging process.

1. It should be possible to control the charging voltage using a PWM signal from the FPGA.
2. It should be possible to measure the battery voltage using an ADC.
3. It should be possible to indirectly measure the charging current using two ADCs.

Since the FPGA has a maximum input voltage of $3.3[V]$ this needs to be taken into consideration since various types of batteries is to be charged by the circuit, some of these batteries might have a battery voltage of $6.0[V]$.

The full electrical circuit is presented in Figure 7.

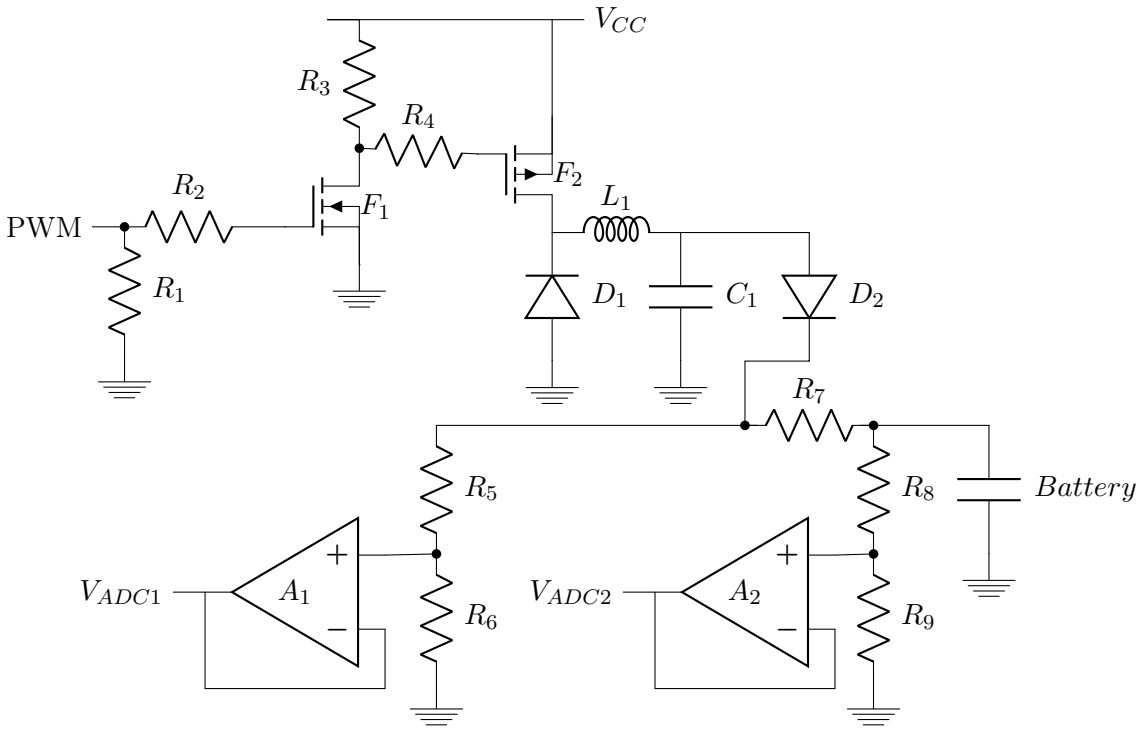


Figure 7: The charging circuit, which can handle various battery voltages, charging currents and monitoring of the charging process.

The reasons behind the components and design considerations will now be stated.

The pull-down resistor $R_1 = 10[k\Omega]$ is used to limit the current drawn from the FPGA:

$$i_1 = \frac{3.3[V]}{10[k\Omega]} = 0.33[mA] \tag{1}$$

The result in equation 1 is based on the logical high voltage value of a port on the FPGA is $3.3[V]$. The resistor $R_2 = 10[\Omega]$ is used to limit the current drawn by the N-channel MOSFET F_1 when being switched. The pull-up resistor $R_3 = 1[k\Omega]$ is used to not short circuit V_{CC} and GND, when the MOSFET F_1 is enabled. The resistor $R_4 = 10[\Omega]$ is used to limit the current drawn by the P-channel MOSFET F_2 when switched.

The buck circuit, consisting of D_1, L_1 and C_1, is used to smooth the PWM signal, since applying the PWM signal directly to the battery would result in applying $12[V]$ in short bursts at the battery, which is not tolerated by many battery types.

The resistor $R_7 = 0.25[\Omega]$ is a shunt resistor used to measure the charge current i_{charge}. The resistors R_5, R_6, R_8 and R_9 is voltage dividers used to indirectly measure the voltage above the resistor R_7, since knowing this would lead to knowledge of the charge current i_{charge} being drawn by the battery. The voltage division is needed since the ports of the FPGA can not handle V_{CC} being applied directly since this is chosen to be $V_{CC} = 12[V]$. The values of the resistors is chosen to be:

$$R_5 = 4.7\ [k\Omega],\ R_8 = 4.7\ [k\Omega],\ R_6 = 1.6\ [k\Omega],\ R_9 = 1.6\ [k\Omega] \tag{2}$$

This would lead to a maximal voltage being applied at the ports of the FPGA to be:

$$\frac{R_9}{R_8 + R_9} \cdot V_{CC} = \frac{R_6}{R_5 + R_6} \cdot V_{CC} = \frac{1.6[k\Omega]}{4.7[k\Omega] + 1.6[k\Omega]} \cdot 12[V] \approx 3.05[V] \tag{3}$$

The result in equation 3 is calculated assuming that the PWM is at a duty cycle of 100% and there is no voltage drops over other components. Based on the calculated max input voltage to the FPGA of $3.05[V]$, which is below the maximum $3.3[V]$ the FPGA can handle, it is considered safe to use this circuit with the FPGA at the $V_{CC} = 12[V]$.

The two voltage followers A_1 and A_2 is used to avoid the ADCs in the development kit affecting the electrical charging circuit since the voltage division used in the ADCs is of the approximately same size as the voltage dividers in the charging circuit [3].

4.1 Designing the Buck Circuit

The buck circuit consists of the inductor L_1, the capacitor C_1 and the diode D_1 shown in Figure 7.

The design formulas stated in equations 4 and 5 are from the paper [4].

$$L = \frac{(V_{IN} - V_{OUT})\,\frac{V_{OUT}}{V_{IN} \cdot f}}{\Delta I_L} \tag{4}$$

Where V_{IN} is the input voltage at the buck circuit which is $V_{IN} = V_{CC} = 12[V]$, V_{OUT} is the output voltages of the buck circuit, which is chosen to be in the range of $V_{OUT} = [1.5, 6.0]$, meaning that we are able to charge batteries ranging from $1.5[V]$ to $6.0[V]$. The graph in Figure 8 shows the inductance of L_1 as a function of varying current ripples at an operation frequency $f = 20[kHz]$.

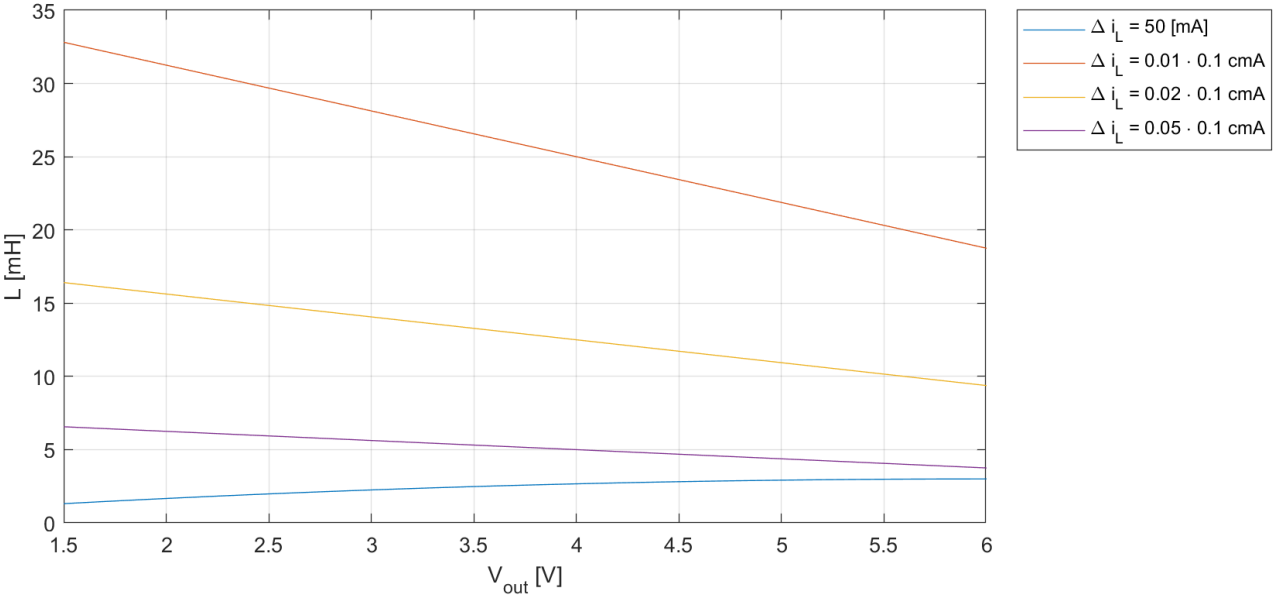


Figure 8: Inductance of L_1 as a function of output voltage.

Based on the graph in Figure 8, which is expressing the inductance of L_1 as a function of the output voltage, it is chosen to use an inductance of $L_1 = 4.7[mH]$.

$$C = \frac{\Delta I_L \left(\frac{V_{OUT}}{V_{IN} \cdot f}\right)}{\Delta V_C} \tag{5}$$

The same choices of values applies for equations 5. The capacitance of C_1 is now visualized as a function of the output voltage and different levels of ripple.

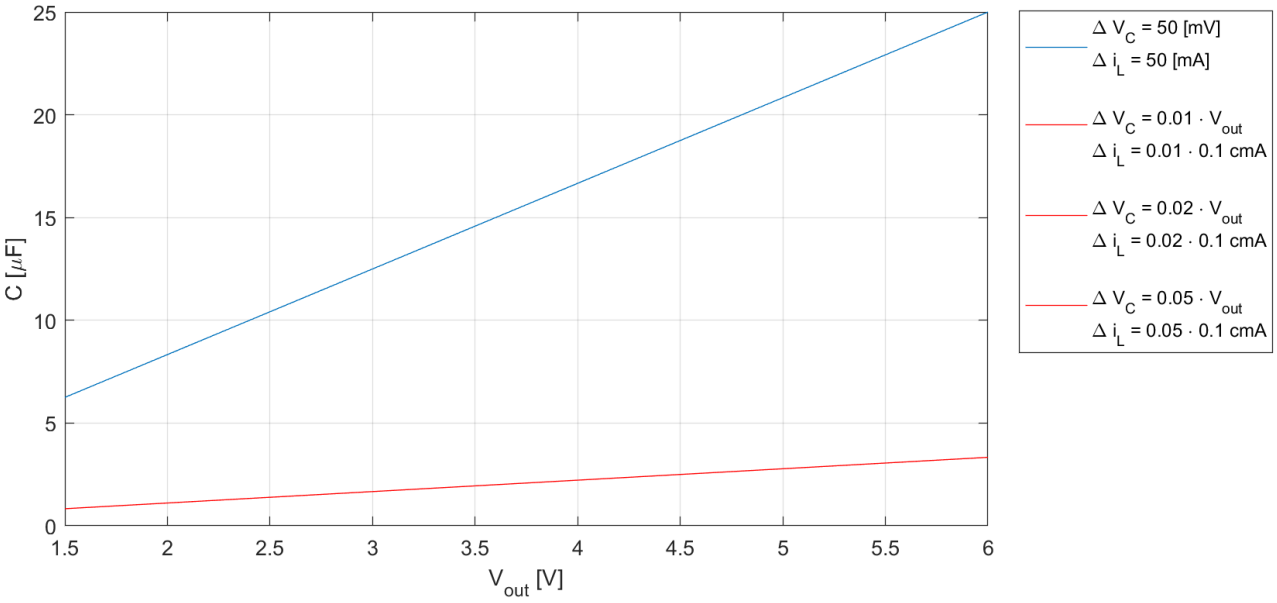


Figure 9: Capacitance of C_1 as a function of output voltage.

Based on the graph in Figure 9 it is chosen to use a capacitance of $C_1 = 1.5[\mu F]$.

4.2 Component Choices for Diodes and MOSFETs

The components D_1, D_2, F_1 and F_2 will be chosen with respect to the requirements of the circuit. It was chosen to use the component 1N4002 [5] for the diodes D_1 and D_2.

The component choice for the N-channel MOSFET F_1 was chosen to be the fast switching IRLZ34N [6], the gate threshold voltage is between $1[V]$ and $2[V]$, which is an acceptable threshold since the FPGA PWM signal has a logical low of $0[V]$ and logical high of $3.3[V]$. Furthermore, it can handle a V_{CC} voltage of $12[V]$.

The P-channel MOSFET F_2 was chosen to be the fast switching IRF9520 [7] which can handle a V_{CC} voltage of $12[V]$, and has a gate-source threshold voltage of $-2[V]$ to $-4[V]$, which means that the signal caused by F_1 can control the MOSFET F_2.
It was chosen to use LM324 [8] for the two voltage followers A_1 and A_2 since this has a low source current draw and is compatible with the V_{CC} choice of $12[V]$.

4.3 Simulating the Circuit with a Load

The circuit is now simulated with a load of $R = 30[\Omega]$ to simulate the load of the battery using CircuitLab [9].

The input PWM signal to the buck circuit is varying between $0[V]$ and $12[V]$ and the voltage at the load can be seen in Figure 10. It can be seen that a stable load voltage is

achieved.

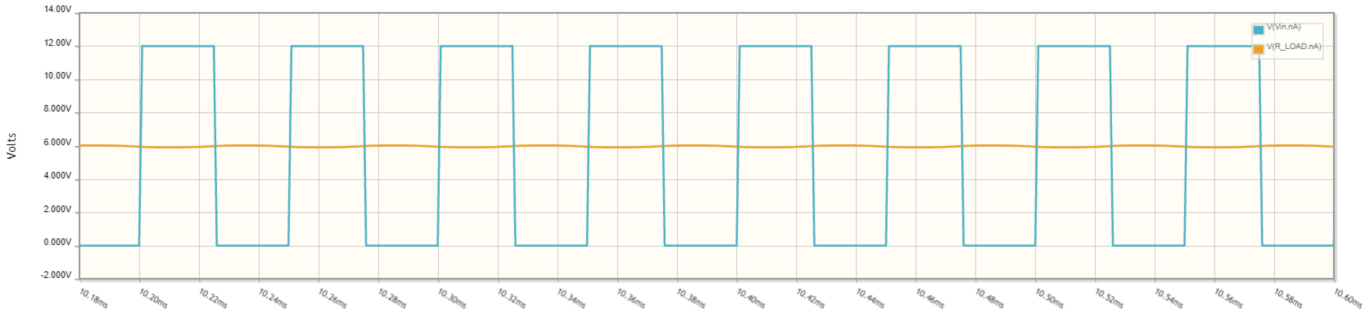


Figure 10: Simulation result of voltage. The blue line is the input voltage at the buck circuit with low voltage of $0[V]$, high voltage of $12[V]$ and a frequency of $20[kHz]$. The yellow line is the output voltage of the buck circuit.

The current drawn by the load is shown in Figure 11, where it can be seen that a stable current drawn by the load is achieved.

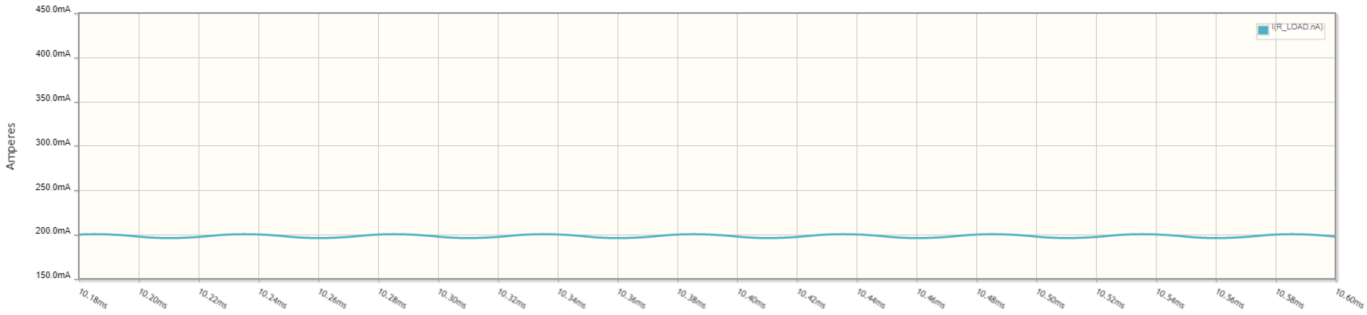


Figure 11: Simulation result of the current drawn by the load to visualize the ripple at the load.

5 Monitoring the System

Monitoring the system is necessary to provide feedback to the user and to enable control of the charging process.

5.1 Monitoring the Charge Current and Battery Voltage

The battery voltage can be calculated from the measurement provided by V_{ADC2}:

$$V_{ADC2} = \frac{R_9}{R_8 + R_9} V_{Bat} \Rightarrow V_{Bat} = \left(1 + \frac{R_8}{R_9}\right) V_{ADC1} = 3.9375 \cdot V_{ADC2} \tag{6}$$

It is seen that the battery voltage is directly proportional to the measured voltage: $V_{Bat} \propto V_{ADC2}$.

The charge current, i_{charge}, can be calculated by the measurements V_{ADC1} and V_{ADC2}. This is done by calculating the voltage over the shunt resistor R_7, after finding the voltage potentials at both sides:

$$V_{R_7} = \left(1 + \frac{R_5}{R_6}\right) V_{ADC1} - \left(1 + \frac{R_8}{R_9}\right) V_{ADC2} = 3.9375 \cdot V_{ADC1} - 3.9375 \cdot V_{ADC2} \tag{7}$$

$$\Rightarrow i_{charge} = \frac{V_{R_7}}{R_7} = \frac{3.9375 \cdot V_{ADC1} - 3.9375 \cdot V_{ADC2}}{R_7} = 15.75\,(V_{ADC1} - V_{ADC2}) \tag{8}$$

It is seen that the charge current is directly proportional to the difference in the two voltage measurements: $i_{charge} \propto (V_{ADC1} - V_{ADC2})$.

Thereby, both the battery voltage and charge current can be calculated very simply from the measured voltage potentials at the shunt resistor R_7:

$$V_{Bat} = 3.9375 V_{ADC2} \tag{9}$$

$$i_{charge} = 15.75\,(V_{ADC1} - V_{ADC2}) \tag{10}$$

5.2 Monitoring the State of Charge

The state of charge can be represented as a function of the battery voltage, as shown in [10]. The charging progress can be represented by the following equation:

$$\%_{Bat} = \frac{V_{Bat} - V_{Bat(0)}}{V_{Bat(Nominal)} - V_{Bat(0)}} \tag{11}$$

In equation 11 a linear characteristic is assumed between the battery charge in percentage and the battery voltage. During this report, the nominal voltage is $V_{Bat(Nominal)} = 1.5[V]$,

for a single battery, since this is the voltage at which the battery is fully recharged. The variable $V_{Bat(0)} = 1.1[V]$ is the voltage at which the battery is fully decharged.

5.3 Monitoring the Remaining Charge Time

The following information about the battery is needed to calculate the remaining charge time: capacity, Q, and the charging current drawn by the battery.

Assuming a linear relationship between the charge current drawn by the battery and the charging time left in hours, the time left in hours can be calculated by the following:

$$T_{LEFT} = \frac{Q}{i_{charge}} \tag{12}$$

5.4 Simulating the Monitoring of the Circuit

The voltages measured by the two ADCs is now simulated to verify the design of the monitoring.

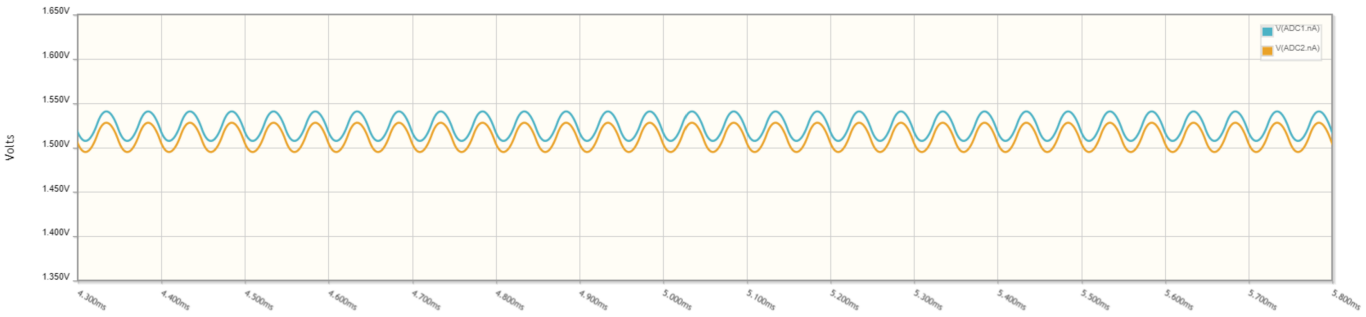


Figure 12: Simulation result of voltages read by ADC1 and ADC2. The blue line is the voltage read by ADC1, the yellow line is the voltage read by ADC2.

The verification of the design is done by two measurements. At time $t = 5.009[ms]$ the voltages is $V_{ADC1} = 1.507[V]$ and $V_{ADC2} = 1.495[V]$. By utilizing the formulas above the charge current and battery voltage can be calculated:

$$t = 5.009[ms] \tag{13}$$

$$V_{Bat}(t) = 3.9375 \cdot V_{ADC2} = 5.88656[V] \tag{14}$$

$$i_{charge}(t) = 15.75\,(V_{ADC1} - V_{ADC2}) = 0.189[A] \tag{15}$$

The actual battery voltage at $t = 5.009[ms]$ is $5.885[V]$ and the actual charge current is $0.196[A]$, which is very close to the estimates. The actual values are read from the simulation results in Figure 10 and Figure 11.

At the second timestamp $t = 5.134[ms]$ the read voltages is $V_{ADC1} = 1.54048[V]$ and

$V_{ADC2} = 1.52768[V]$.

$$t = 5.134[ms] \tag{16}$$

$$V_{Bat}(t) = 3.9375 V_{ADC2} = 6.0165[V] \tag{17}$$

$$i_{charge}(t) = 15.75\,(V_{ADC1} - V_{ADC2}) = 0.201[A] \tag{18}$$

The actual values are read in Figure 10 and Figure 11, where it can be seen that the battery voltage at $t = 5.134[ms]$ is $6.015[V]$ and the actual charge current is $0.200[A]$, which are also very close to the estimates.

6 Designing the Control for the Charging Process

The control of the charging process is made flexible to encompass batteries with varying nominal voltages and charging rates.

The first steps is to read relevant user inputs.

1. Read the entered nominal battery voltage.
2. Read the battery capacity.
3. Read the battery rate of charge.
4. Read the charging speed.

In the datasheet [10] the charging rate of the battery is standard at $0.1[cmA]$ and rapid at $0.5[cmA]$, thereby it was chosen that the user can enter a value in the interval $k = [0.1, 0.5]$ to control the charging speed. Since the battery used is a NiMH battery, it is charged using constant current [2]. Therefore, the control for the charging process will be designed for constant current battery recharging.

The charging speed entered by the user affects the target for the charge current:

$$i_{target} = k \cdot i_{MAX}, \quad i_{MAX} = C \cdot Q \tag{19}$$

The purpose of the controller is to make the current charge equal to the desired target current, $i_{charge} = i_{target}$, to achieve the charging speed set by the user.
A control loop is created to control the PWM signal sent from the FPGA to the electrical circuit. The pseudocode for the control loop is shown in Algorithm 6, and is implemented in the PS.

Algorithm 1: Control loop for controlling the charging process

Requires initialization of: $V_{Nominal}$(nominal battery voltage), i_{target} (target charge current).

while $V_{BAT} < V_{Nominal}$ **do**

1 V_{ADC1} = readVADC1()

2 V_{ADC2} = readVADC2()

3 i_{charge} = calculateChargeCurrent(V_{ADC1}, V_{ADC2})

if $i_{charge} > i_{target}$ **then**

4 DecreasePWMDutyCycle()

end

if $i_{charge} < i_{target}$ **then**

5 IncreasePWMDutyCycle()

end

6 V_{BAT} = calculateBatteryVoltage(V_{ADC2})

end

The designed control loops requires a flexible PWM module being implemented on the FPGA, which is capable of reading a variable which controls the duty cycle of the PWM.

6.1 Designing the PWM Module

The PWM module is responsible for delivering the control signal to the charging circuit. The PWM module is designed to have a variable frequency and variable duty cycle. It was chosen to use a PWM signal with a frequency of $f = 20[kHz]$. This frequency can be set in VHDL using a constant in the designed module. The duty cycle is being read as a variable, making it possible for the PS to control this.

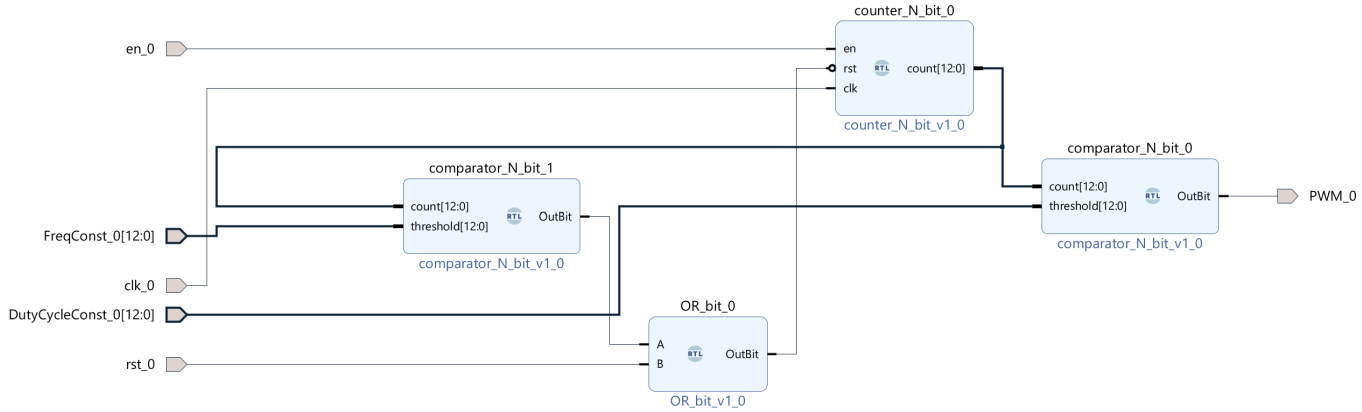


Figure 13: Block design of PWM module with variable pwm frequency and duty cycle.

The counter frequency is $f_{osc} = 100[MHz]$ since it is connected to the clock driving the ZYNQ processing system implemented in the FPGA. The threshold for when to reset the counter can be calculated to achieve a frequency of $f_{PWM} = 20[kHz]$.

$$\frac{f_{osc}}{f_{PWM}} = 5000 \tag{20}$$

The amount of bits needed for the counter and comparators can now be calculated.

$$Log_2(5000) = 12.29 \Rightarrow bits = 13 \tag{21}$$

6.1.1 Simulating the PWM Module

The PWM module is simulated with a duty cycle of 50% and the frequency is set to $20[kHz]$. This is done by applying the calculated constants to the block design and tying the enable pin to '1' and reset pin to '0' as in Figure 14.

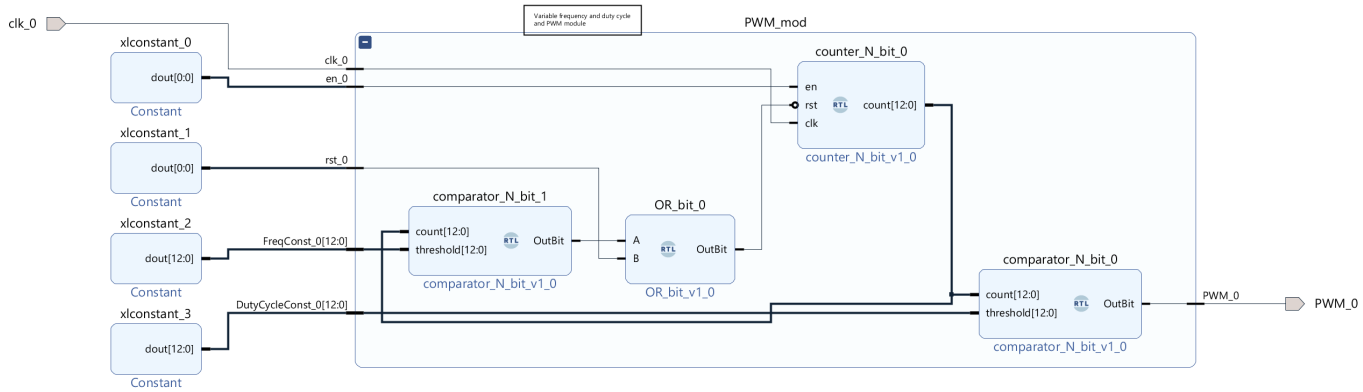


Figure 14: A hierachy containing the PWM module is created and the constants is connected using VHDL constants.

The simulation results are shown in Figure 15.

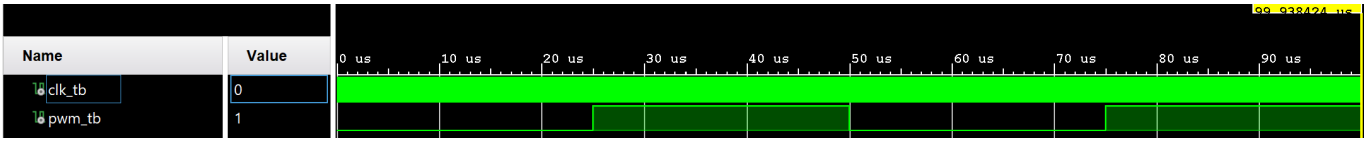


Figure 15: Test bench simulation of the designed variable PWM module.

It can be seen in Figure 15 that the PWM period is $T_{PWM} = 50[\mu s]$, which confirms that a frequency of $\frac{1}{T_{PWM}} = f_{PWM} = 20[kHz]$. Furthermore, the PWM switches between low and high at $t = 25[\mu s]$, which gives yields a duty cycle of 50%.

6.2 Implementation of the Xilinx Analog to Digital Converter

In this section, the analogue to digital converter on the Xilinx board, XADC, is set up by following the guide [11]. The XADC is connected to the PS through an module in PL using Advanced eXtensible Interface, AXI; this is shown in figure 16.

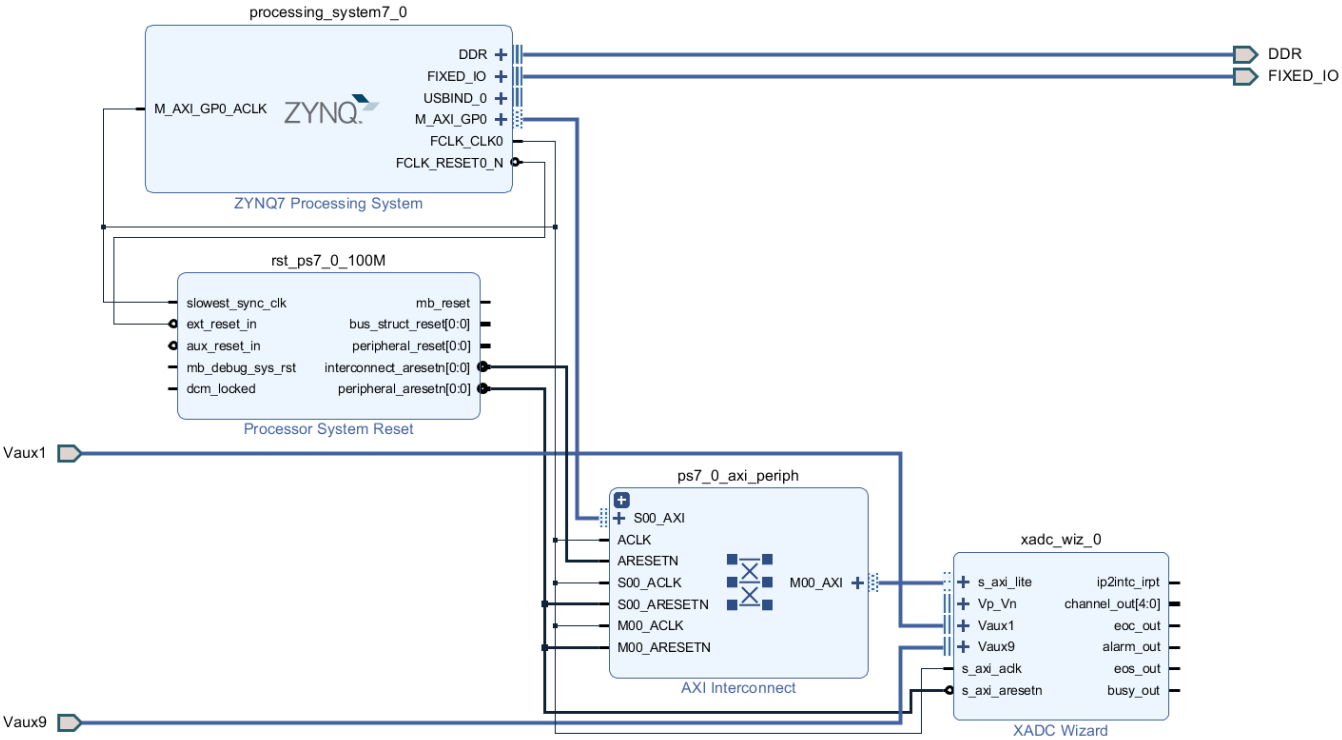
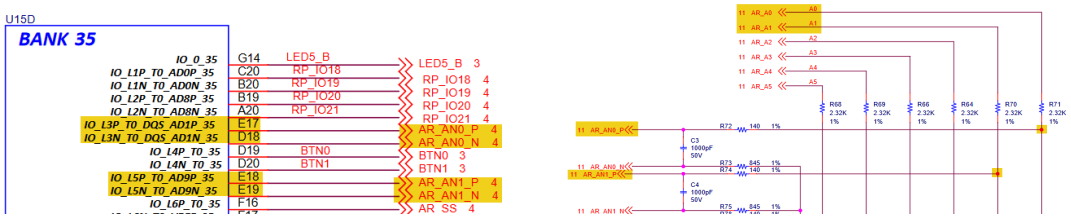


Figure 16: Block design of XADC

The XADC is used to measure a voltage ranging from $[0, 3.3]V$, but the XADC can only accept voltage ranging between $[0, 1]V$. However, the pins $A0 - A5$ on the PYNQ Z2 board scales down the input voltage from $3.3[V]$ to $1[V]$ via an internal voltage divider[3], thus the pin $A0$ and pin $A1$ are used. In order to use the pins, it is necessary to state in the XADC wizard which Vaux ports that maps to I/O ports of the PYNQ Z2 board.



(a) Schematic of I/O mapping of Arduino I/O

(b) Schematic of bank 35

Figure 17: PYNQ Z2 Schematic snip of bank 35 and I/O mapping of Arduino I/O[12]

From the schematic snippet of the PYNQ Z2 board illustrated in Figure 17, the pin $A0$'s positive and neutral maps to $D17$ and $D18$ respectively and pin $A1$'s positive and neutral maps to $E18$ and $E19$ respectively in bank 35. The bank values $D17$ and $D18$ corresponds to Vaux1, and $E18$ and $E19$ corresponds to Vaux9 on the XADC wizard.

The XADC on the PYNQ Z2 board is a 12 bit ADC, which gives a resolution of $\frac{3.3[V]}{2^{12}} \approx 0.806[mV]$ when meassuring voltages in range of $[0, 3.3]V$.

6.2.1 Transfer PWM Value and Battery Percent to Block RAM

In order to regulate the battery charger circuit, a PS to PL communication were designed. The communication is done through a true dual-port block ram. The data send from the PS has a maximum size of 3 bits, and one address of the BRAM can hold 32 bits. Thus it is only necessary to write to one address. From the bitstream illustrated in Figure 18, it can be seen that the two bits contains the PWM duty cycle increase/decrease and the last bit contains whether the battery is above 20% or not.

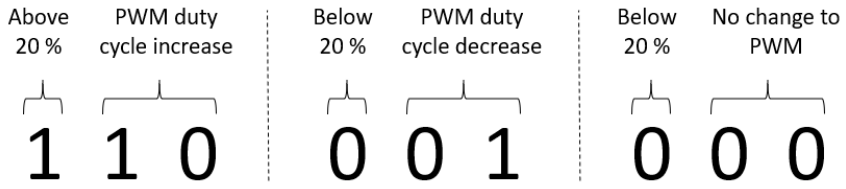


Figure 18: Examples of the bitstream send to the PL.

The overall block diagram of the XADC and PS-PL communication is shown in Figure 19.

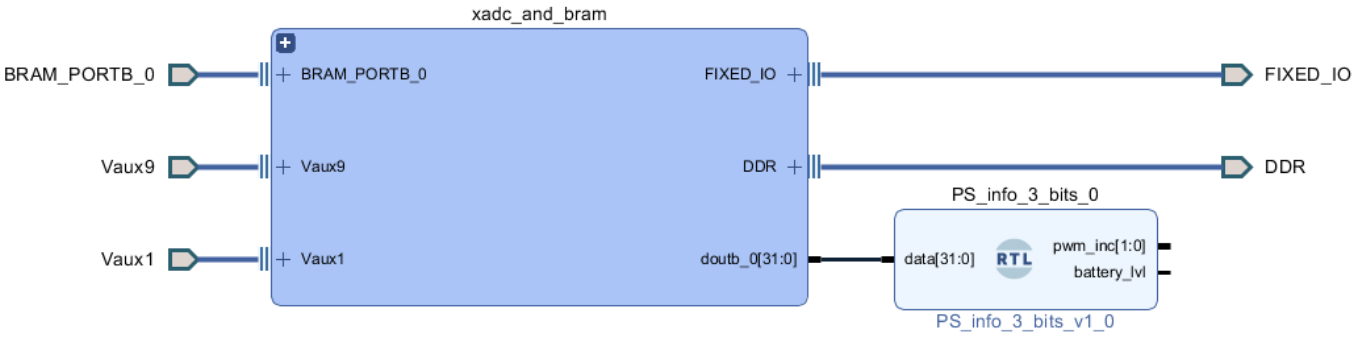


Figure 19: Overall block design of XADC and PS-PL communication

The overall block design consist of two modules. The module `PS_info_3_bits_0` splits the data read from the BRAM into the PWM-value and battery percentage value. The hierarchy module `xadc_and_bram` takes care of the XADC and PS to PL communication which is illustrated in Figure 20.

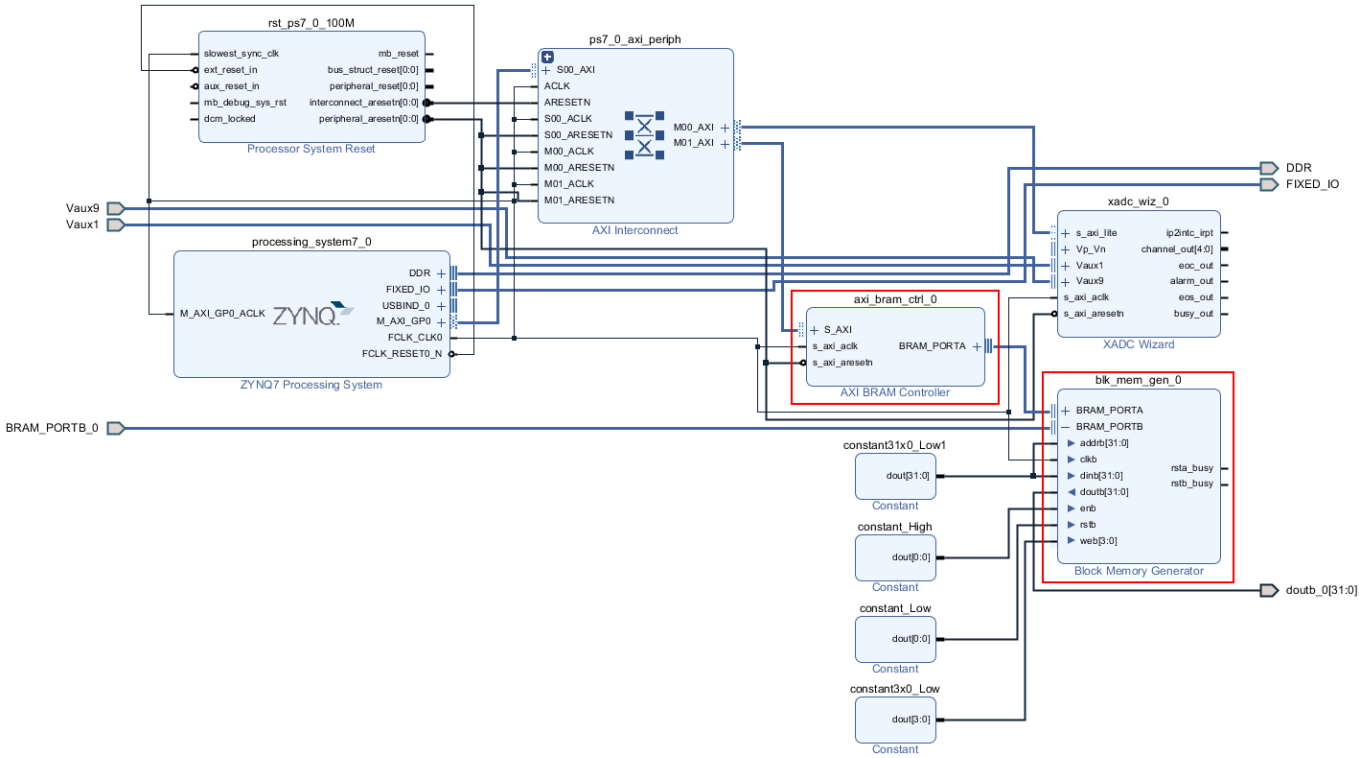


Figure 20: Block design of hierarchy module `xadc_and_bram`.

The block design in Figure 20 has the same main element as the block design of the XADC shown in Figure 16. The two major components added in contrast to the XADC block design are an AXI BRAM Controller and a Block Memory Generator which are marked with red.

Testing the XADC

A physical test was conducted to confirm both the XADC and the PS to PL communication worked as intended. This was done by measuring a voltage ranging of $[0, 3]V$ on the PYNQ™ Z2 board's $A1$ pin and writing the value to the BRAM. The FPGA then reads the value and shows it on a Vishay TDSR1360 7-segment display. The block design used to conduct the test is shown in Figure 21.

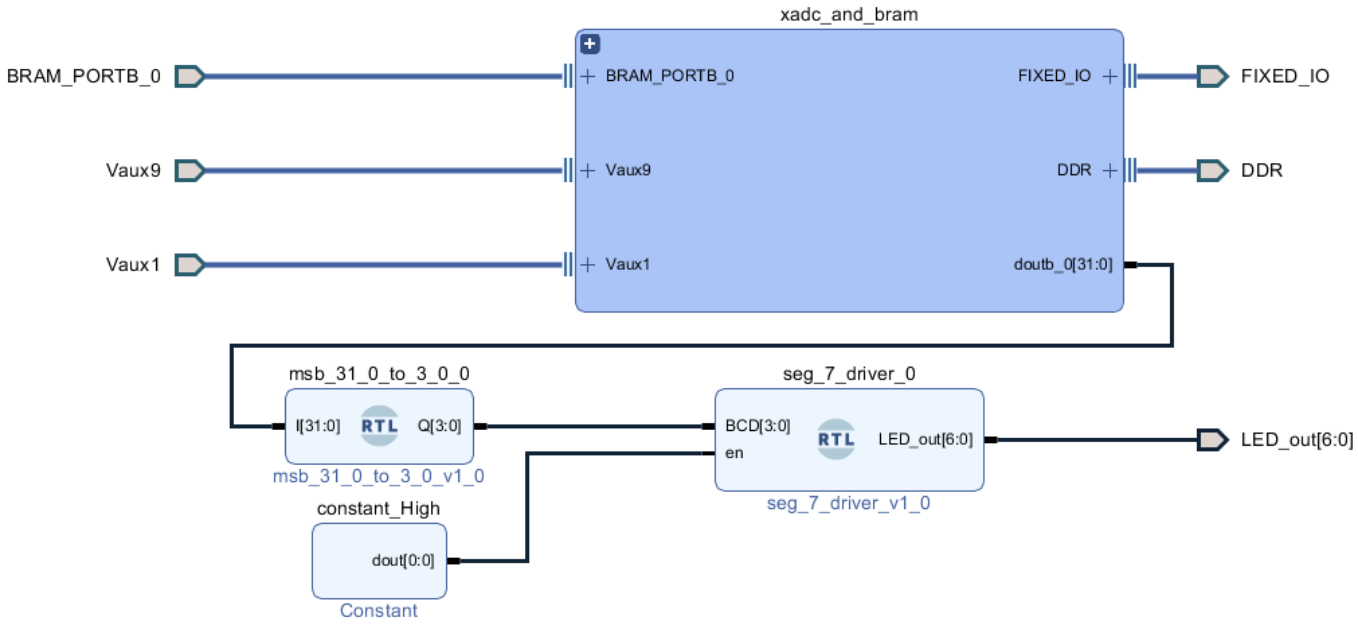
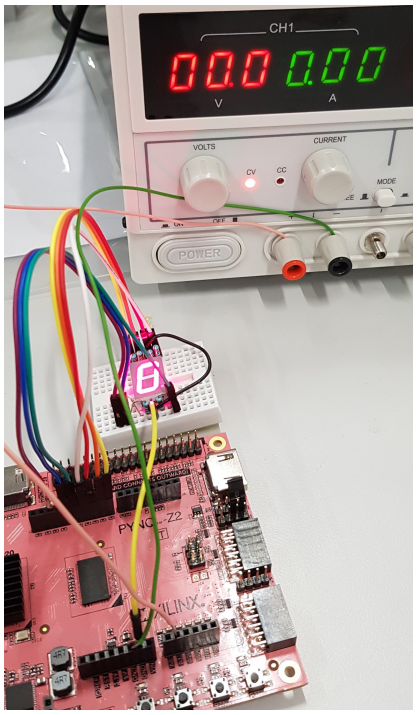
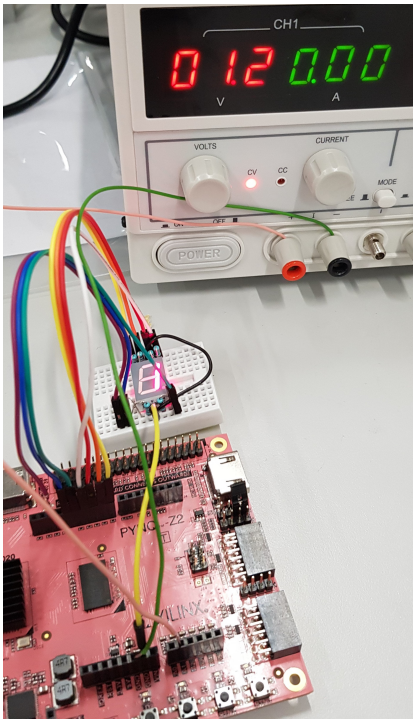


Figure 21: Block design used when testing XADC and PS to PL communication.

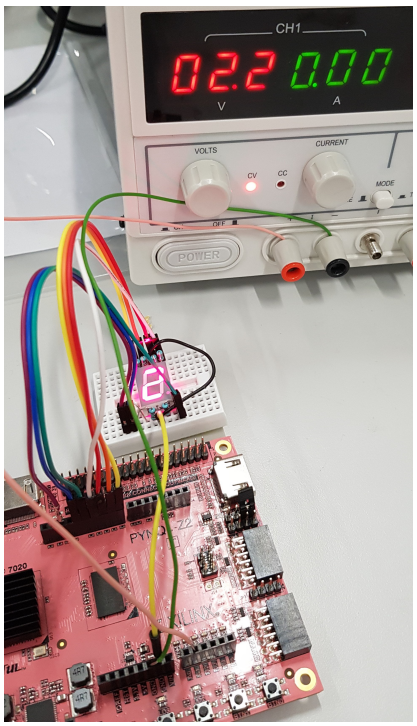
The results from the test can be seen in Figure 22.



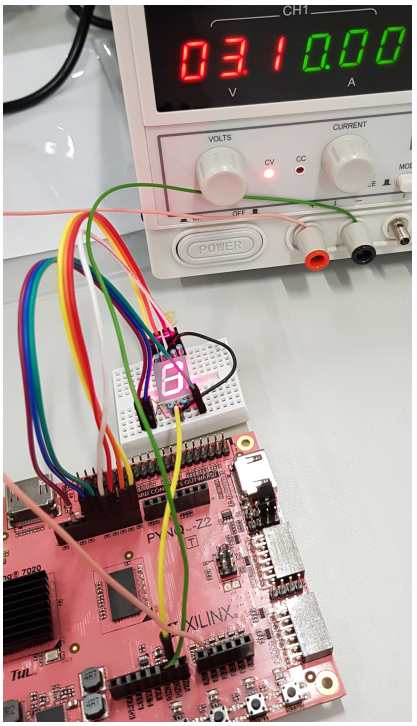
(a) 0 [V] measured by the ADC displays 0 on the 7-segment display.



(b) 1.2 [V] measured by the ADC displays 1 on the 7-segment display.



(c) 2.2 [V] measured by the ADC displays 2 on the 7-segment display.



(d) 3.1 [V] measured by the ADC displays 3 on the 7-segment display.

Figure 22: Test results of measured voltage with XADC using PS to PL communication to show value on bcd display

From the results in Figure 22 it can be concluded the integer of the voltage measured via

the XADC is being transferred to the FPGA via PS to PL communication.

7 Combining the Designed Modules

The final block diagram implemented in VHDL is shown in Figure 23.

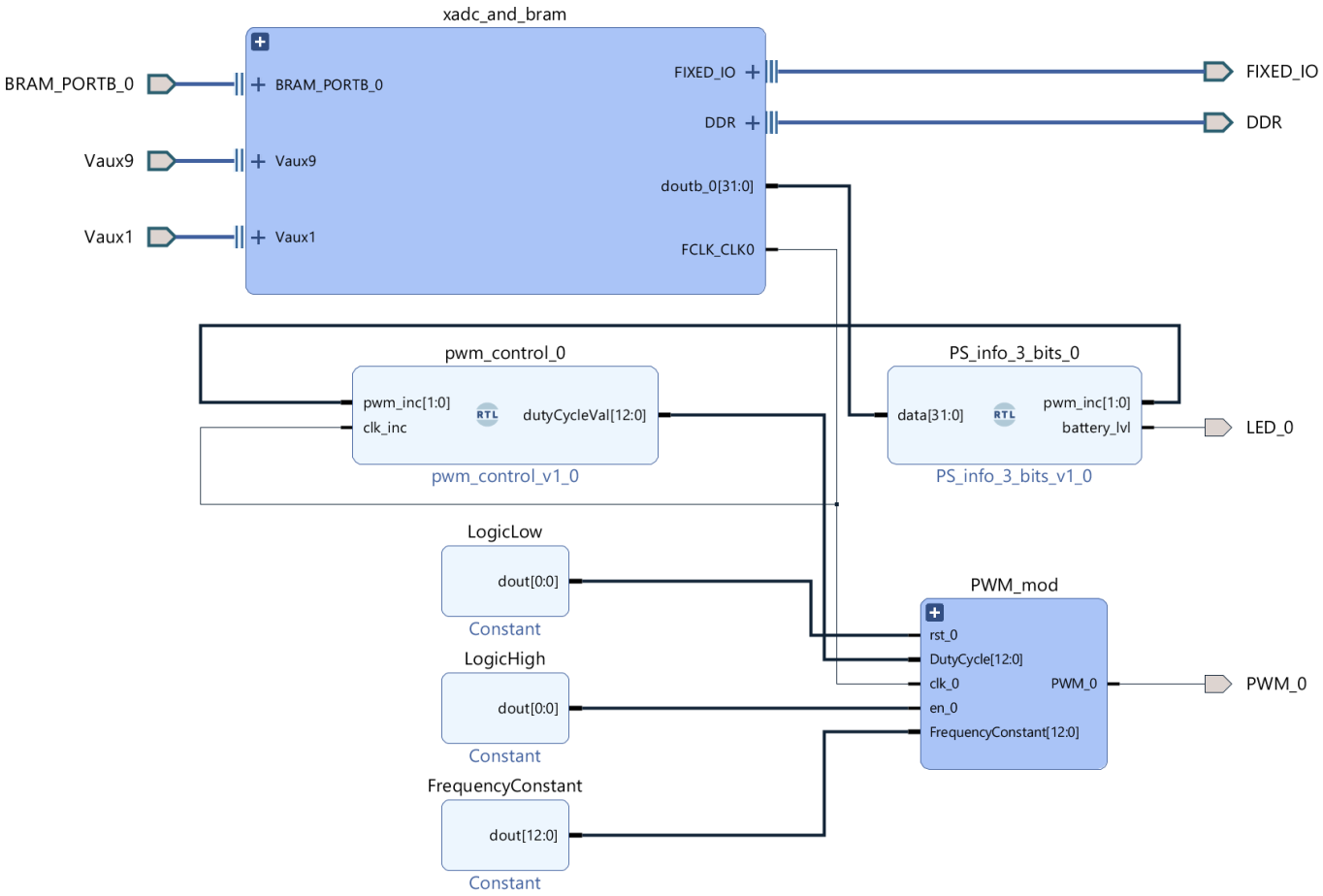


Figure 23: Block diagram of the final product implemented in VHDL.

The `PWM_0` pin is mapped to an external pin and attached to the charging circuit. The pin `LED_0` is attached to a LED through the constraint file, to show when the battery percentage is above 20 %. The circuit's V_{ADC1} and V_{ADC2} is connected to `Vaux1` and `Vaux9`, through a mapping in the constraint file.

The block `pwm_control_0` is used as a layer between the PWM module and the reading of PWM command. This module increases, decreases or keeps the duty cycle constant, which is used for the module `PWM_mod`.

8 Test and Results

This sections provides the results of the final product through measurements of the circuit.

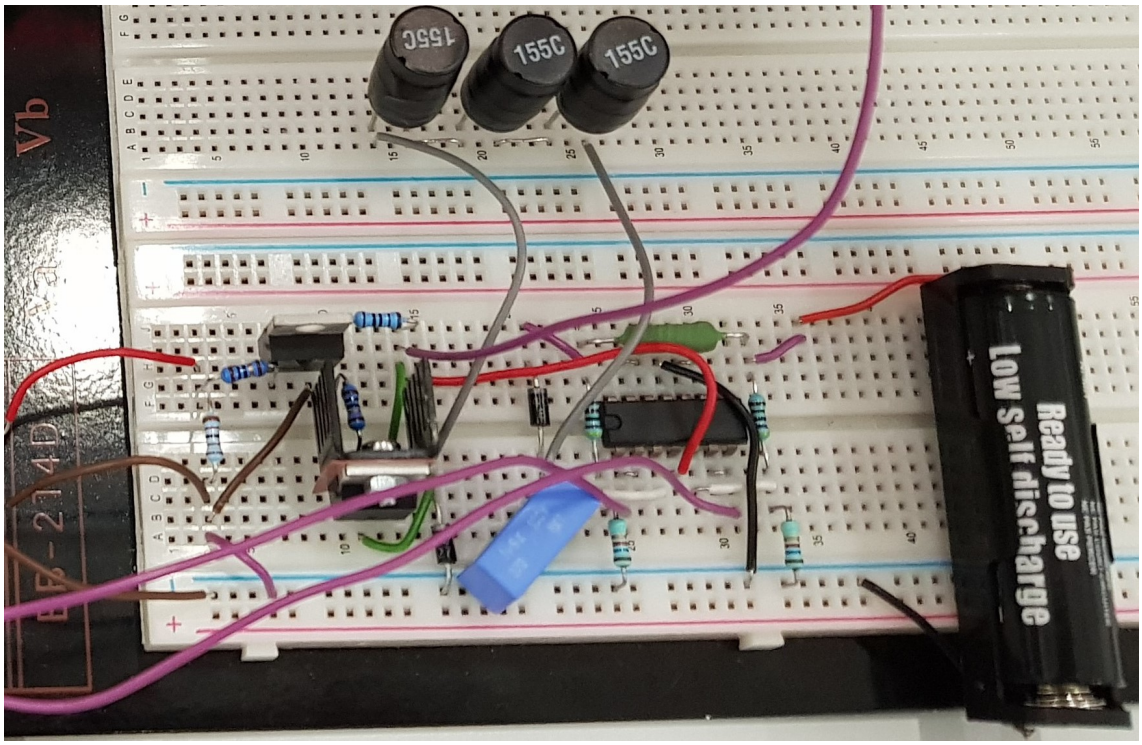


Figure 24: The designed charging circuit used for testing.

The circuit used for testing in this section is shown in Figure 24.

8.1 Measurements of the Circuit

First the PWM module designed in subsection 6.1 is evaluated. A duty cycle of 50% is sent to the FPGA:

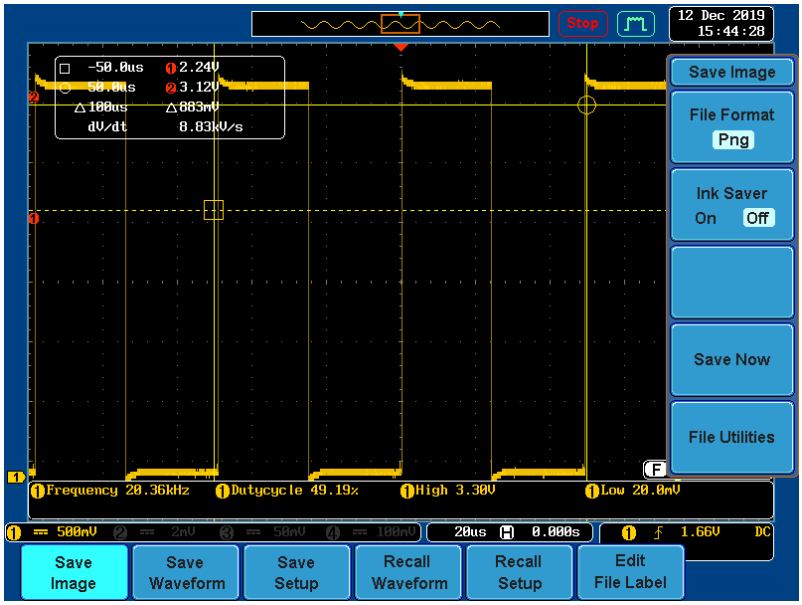


Figure 25: The PWM signal generated by the FPGA.

In Figure 25 it is seen that the achieved frequency is approximately 20$[kHz]$ and the high and low voltage levels is approximately 0$[V]$ and 3.3$[V]$.

The PWM signal at the output of MOSFET F_1 is now measured to see if it correctly switches between 0$[V]$ and 12$[V]$ with a duty cycle of approximately 50% given the input PWM signal from the FPGA of approximately 50%.

Figure 26: The output PWM signal of MOSFET F_1 switching between 0$[V]$ and 12$[V]$.

It is seen in Figure 26, that the output of F_1 reaches the desired low of approximately 0$[V]$

and high level of approximately 12[V] with the same duty cycle and frequency.
The output signal of the MOSFET F_2 is now measured to see if it correctly switch between 0[V] and 12[V].

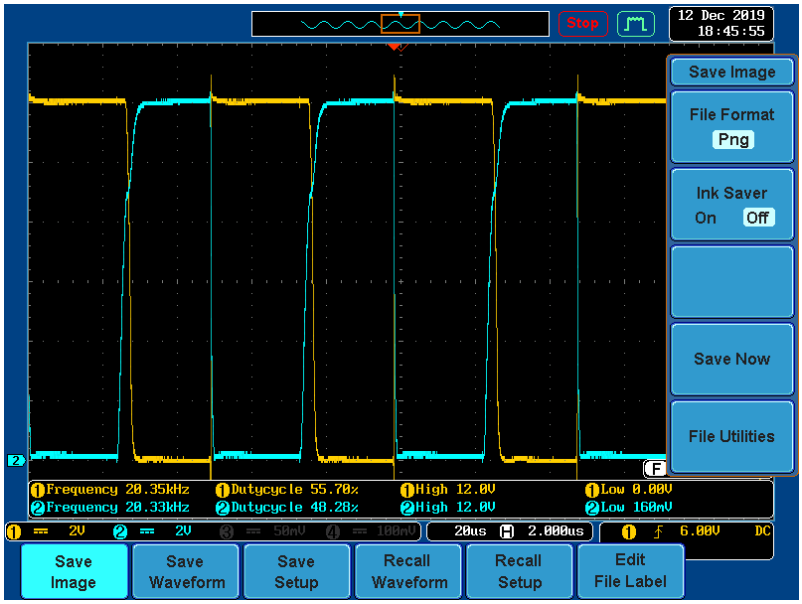


Figure 27: The output PWM signals of both MOSFETS F_1 and F_2. Channel 1 is output of F_1 and channel 2 is output of F_2.

It can be seen in Figure 27, that the MOSFET F_2 switches between 0[V] and 12[V] as wanted at the same frequency and approximately same duty cycle.
The output of the buck converter is now measured. To do this a power resistor of size 30[Ω] is placed at the output of the buck converter, which acts as a load on the buck converter. The expected output is that the buck converter smooths the PWM signal with a ripple around the designed. The measurements is performed at duty cycles which creates an output of 1.5[V], 3.0[V], 4.5[V] and 6.0[V].

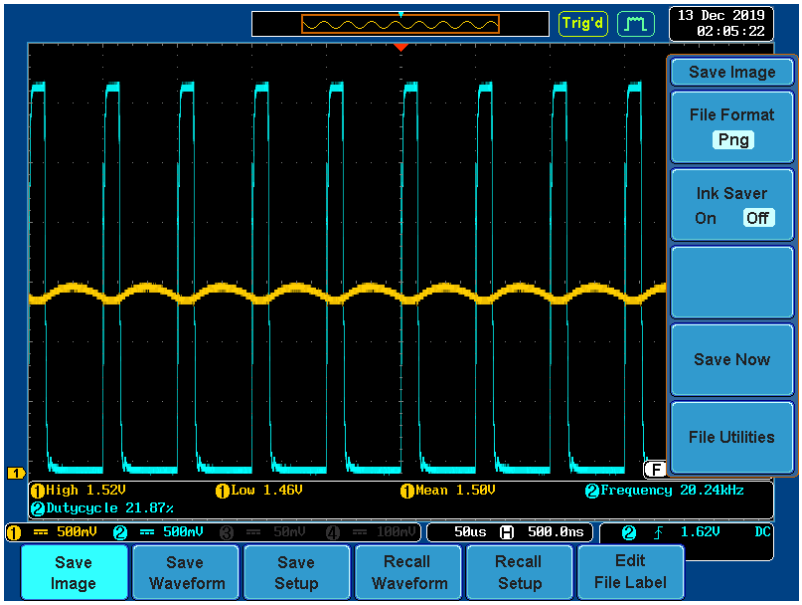


Figure 28: Channel 1 is the output of the buck converter with a load of $R = 33[\Omega]$, channel 2 is the FPGA PWM input to the circuit. Target voltage is $1.5[V]$.

It can be seen in Figure 28, that a duty cycle of 21.7% yields an output at the buck converter with an average of $1.5[V]$ and a ripple of $\Delta V_C = 0.040 V_{out}$.

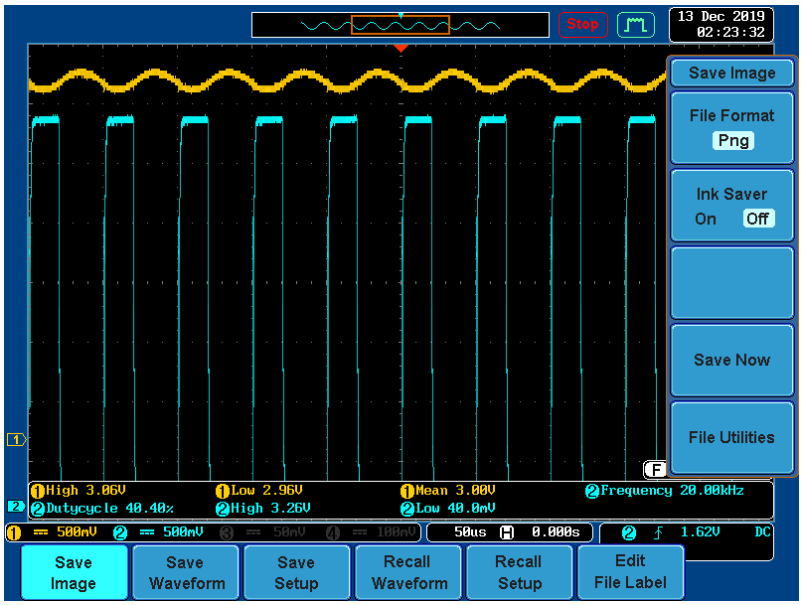


Figure 29: Channel 1 is the output of the buck converter with a load of $R = 33[\Omega]$, channel 2 is the FPGA PWM input to the circuit. Target voltage is $3.0[V]$.

In Figure 29 it can be seen that a duty cycle of 40.4% yields an output at the buck converter with an average of $3.0[V]$ with a ripple of $\Delta V_C = 0.033 V_{out}$.

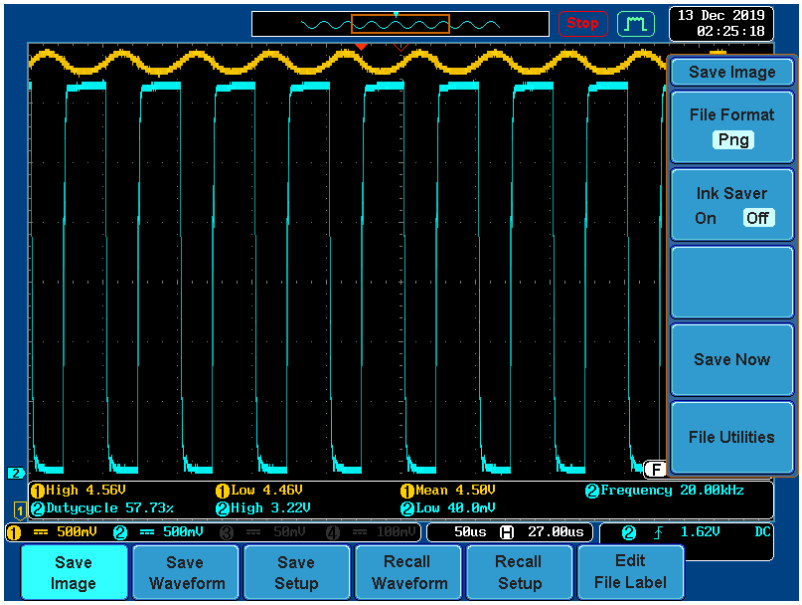


Figure 30: Channel 1 is the output of the buck converter with a load of $R = 33[\Omega]$, channel 2 is the FPGA PWM input to the circuit. Target voltage is $4.5[V]$.

In Figure 30 it can be seen that a duty cycle of 57.73% yields an output at the buck converter with an average of $4.5[V]$ with a ripple of $\Delta V_C = 0.022 V_{out}$.

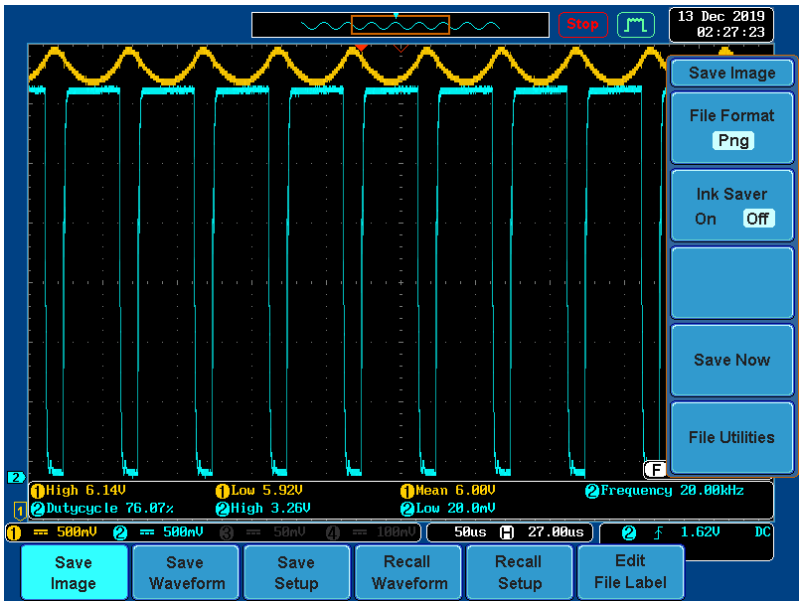


Figure 31: Channel 1 is the output of the buck converter with a load of $R = 33[\Omega]$. Channel 2 is the FPGA PWM input to the circuit. Target voltage is $6.0[V]$.

In Figure 31 it can be seen that a duty cycle of 76.07% yields an output at the buck converter with an average of $6.0[V]$ with a ripple of $\Delta V_C = 0.036 V_{out}$.

The ripple at the four voltage levels was at a maximum of 4% of the output voltage, which

is acceptable. The battery is now placed at the output of the buck converter instead of the power resistor.

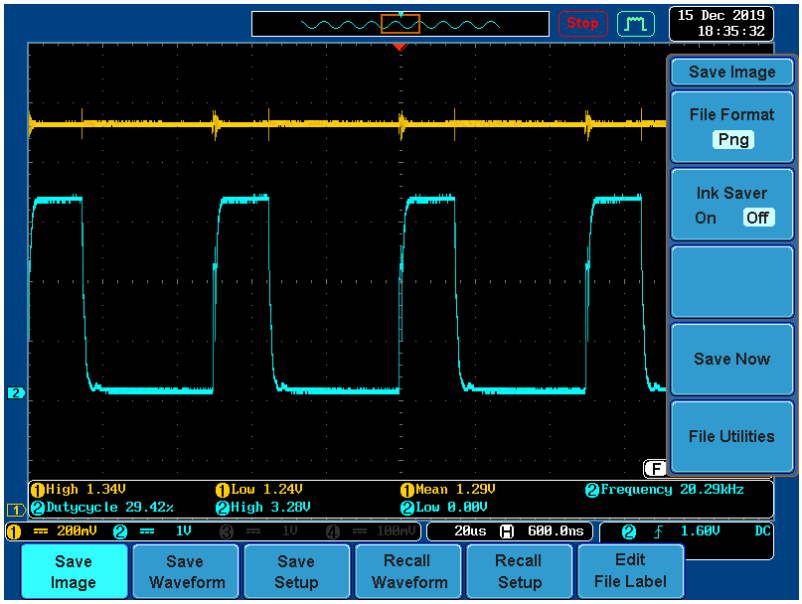


Figure 32: Channel 1 is the battery voltage, channel 2 is the FPGA PWM input to the circuit. Target current is 0.2$[mA]$.

It is seen that the ripple is generally much lower in Figure 33 than the measurements with a load resistor, but there is some high frequent ripple present.

8.2 Measurements of the Monitoring

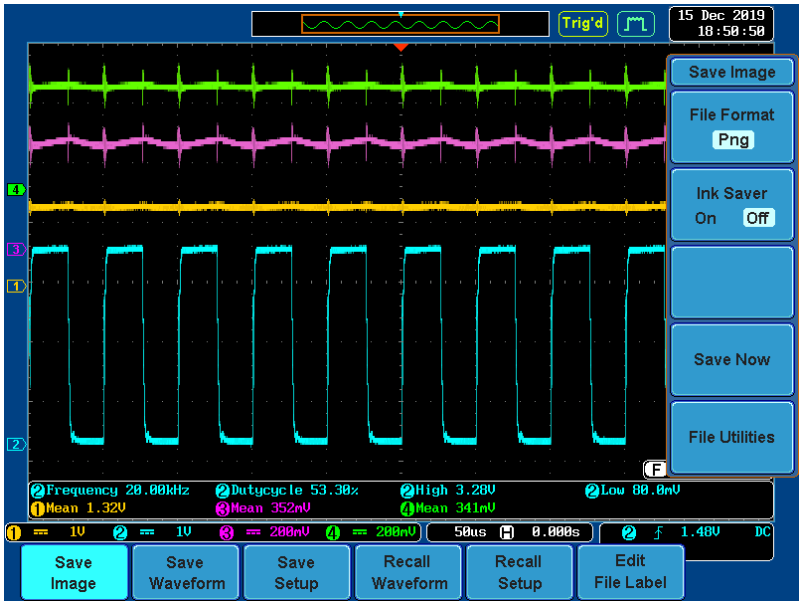


Figure 33: Channel 1 is the battery voltage, channel 2 is the FPGA PWM input to the circuit, channel 3 is the V_{ADC1} signal and channel 4 is the V_{ADC2} signal. Target current is 0.25$[mA]$.

The actual value of i_{charge}, current drawn, by the battery was measured to be $i_{charge} = 254[mA]$ using a digital multimeter. The two voltages for the ADCs were measured: $V_{ADC1} = 354[mV]$ and $V_{ADC2} = 338[mV]$, this yields the following calculation of battery voltage: $V_{Bat} = 3.9375 \cdot V_{ADC2} = 1.33[V]$, which is very close to the battery voltage (channel 1) in Figure 33. The current i_{charge} is calculated as $i_{charge} = 15.75\,(V_{ADC1} - V_{ADC2}) = 252[mA]$ which is very close to the one measured with the digital multimeter.

8.3 The Combined System

The combined system with the processing system and programmable logic combined with the designed electrical system is tested.

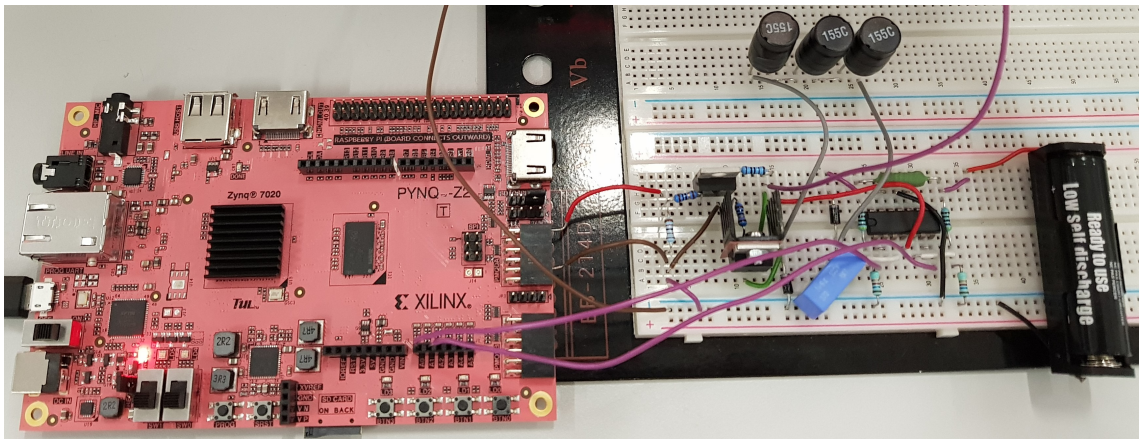


Figure 34: The combined system with the microprocessor, FPGA and charging circuit.

The combined system is shown in Figure 34. The test is performed by first entering the following values:

1. Nominal battery voltage: $V_{Bat(Nominal)} = 1.5[V]$.
2. Battery capacity: $Q = 2000[mAh]$.
3. Battery rate of charge: $C = 1$.
4. Charging speed: $k = 0.1$.

By entering the above values the target current is $i_{target} = 0.2[A]$. It is observed that the system balances the target current at around $i_{charge} = i_{target} = 0.2[A]$.
A video of the test is uploaded to Youtube: `https://youtu.be/0lCbZjbHUG8`.

The combined solution is then tested if capable of charging a $3.0[V]$ battery. The following values are entered in the terminal:

1. Nominal battery voltage: $V_{Bat(Nominal)} = 3.0[V]$.
2. Battery capacity: $Q = 4000[mAh]$.

3. Battery rate of charge: $C = 1$.
4. Charging speed: $k = 0.1$.

Using the above values the target current is $i_{target} = 0.4[A]$. It is observed that the system balances the target current at around $i_{charge} = i_{target} = 0.4[A]$.
A video of the test is uploaded to Youtube: `https://youtu.be/0lCbZjbHUG8`.

Lastly, the combined solution is tested if capable of charging a $4.5[V]$ battery. The following values are entered in the terminal:

1. Nominal battery voltage: $V_{Bat(Nominal)} = 4.5[V]$.
2. Battery capacity: $Q = 6000[mAh]$.
3. Battery rate of charge: $C = 1$.
4. Charging speed: $k = 0.1$.

Using the above values the target current is $i_{target} = 0.6[A]$. It is observed that the system balances the target current at around $i_{charge} = i_{target} = 0.6[A]$.
A video of the test is uploaded to Youtube: `https://youtu.be/0lCbZjbHUG8`.

9 Discussion

Reading from Multiple Addresses

The PWM-value and battery threshold send from the ARM processor to the BRAM described in subsection 6.2 only uses one address to write the values. If more data was to be send a finites state machine could be designed to switch between addresses of the BRAM on the FPGA. An example of how the state machine could be designed is illustrated in Figure 35.

The suggested state machine waits one clock cycle before reading the data from the BRAM since the BRAM reads the address at rising edges and afterwards outputs the data.

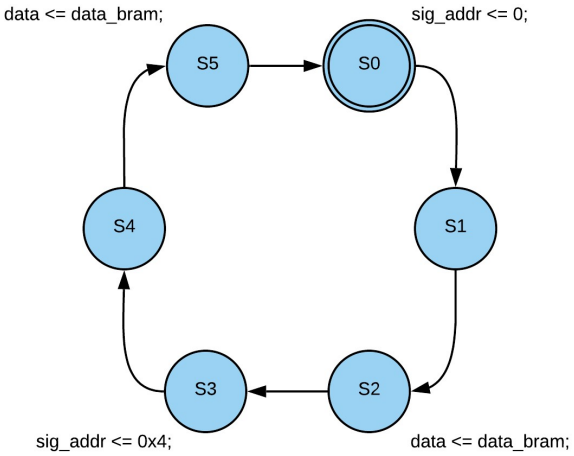
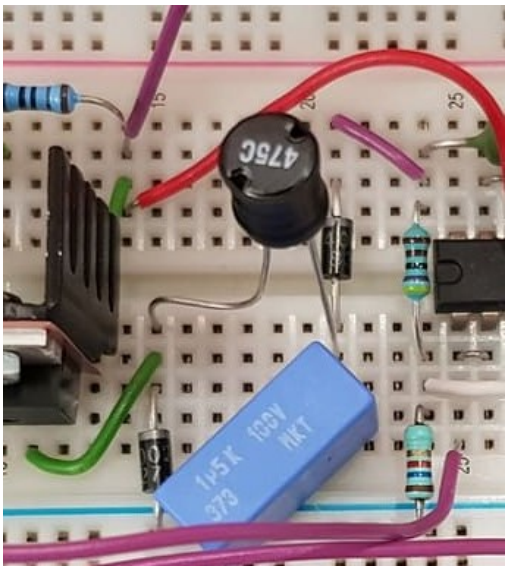


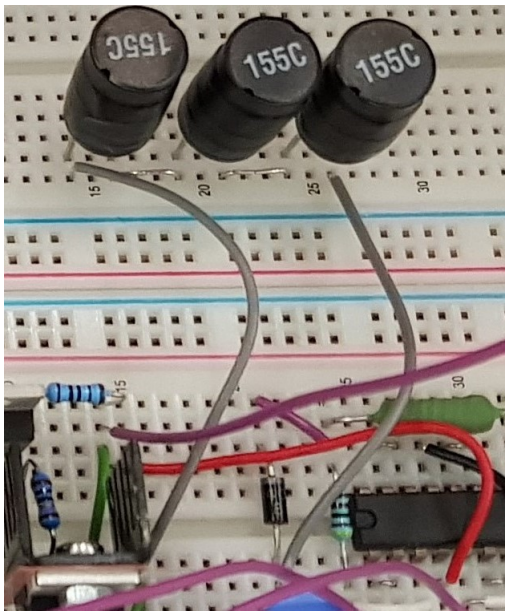
Figure 35: A suggested design of a state machine to switch between ram addresses.

Heating of the Inductor

Initially, the charging circuit was constructed with a single inductor, as shown in 36a, but this showed to cause problems since it heated drastically.



(a) The single inductor used in the initial circuit.



(b) The three inductors used in the final circuit.

Figure 36: The inductors used through the development of the circuit.

Therefore, it was chosen to use three inductors in series instead, which summed up to the same size. This distributed the heating between the three components and therefore allowed the circuit to be run for an extended period of time.

Missing Evaluation of 6V Battery

The circuit was designed to charge batteries up to $6.0[V]$, but it was not tested in section 8, since a $6.0[V]$ battery was not available at the time of testing. Therefore, it is not known if the circuit in practice actually is able to charge a $6.0[V]$ battery.

10 Conclusion

Throughout the report, a system that can charge batteries of different voltages was developed. The designed system can monitor the charging progress every 10'th milliseconds and display the remaining charging time of the battery.

An electrical charging circuit was build which made it possible to charge batteries ranging from 1.5 [V] to 6 [V], the charging process was controlled with a PWM signal, while the battery voltage and charging current was monitored.

A controller was designed on the PYNQ Z2 board that adjusts the charge current to be equal to the desired target current set by the user. The controller receives feedback from the electrical circuit via two XADCs and sends input to the electrical circuit via a PWM module. The PWM module was designed to have a frequency of 20 [kHz]. The designed XADC can measure a voltage from 0 [V] to 3.3 [V] with a resolution of 0.806 [mV]. A protocol was designed to transfer the PWM value and whether the battery level is above 20 %, trough PS to PL communication.

The final product[1] was capable of reliably charging batteries of various types and sizes.

[1] A demonstration of the final product can be seen in `https://youtu.be/0lCbZjbHUG8`

References

[1] Emil Vincent Ancker, Mikkel Larsen, and Mathias Emil Slettemark-Nielsen. *Mini Project Robot Electronics*. 2019. URL: https://github.com/Masle16/mini_project_robot_electronics (visited on 12/16/2019).

[2] AVR. *AVR450: Battery Charger for SLA, NiCd, NiMH and Li-Ion Batteries, Application Note*. Atmel.

[3] Technology Un-Limited. *PYNQ-Z2 Reference Manual*. URL: http://www.tul.com.tw/download/PYNQ_Z2_User_Manual_v1.1.pdf (visited on 12/15/2019).

[4] Jose Formenti and Robert Martinez. *Design Trade-offs for Switch-Mode Battery Chargers*. Texas Instruments, 2004.

[5] ON Semiconductor. *Axial-Lead Glass Passivated Standard Recovery Rectifiers*. Datasheet, www.onsemi.com.

[6] International IR Rectifier. *IRLZ34N MOSFET Datasheet*. Infenion, www.irf.com.

[7] Vishay Siliconix. *Power MOSFET Datasheet*. Vishay, www.vishay.com, 2019.

[8] Texas Instruments. *LMx24-N, LM2902-N Low Power, Quad-Operational Amplifiers*. Texas Instruments, www.ti.com.

[9] Circuitlab. *Circuit simulation and schematics*. 2019. URL: https://www.circuitlab.com/circuit/79v456zdcv6a/buck-circuit/ (visited on 12/16/2019).

[10] RS Article No.: 9053781. *NI-MH LOW-SELF DISCHARGE, Product Datasheet*. RS PRO.

[11] Jørgen Christian Larsen. *Using the XADC on the Zybo board*. URL: http://realisenow.sdu.dk/using-the-xadc-on-the-zybo-board/?fbclid=IwAR3h6SrpQ2-a8G0b_H6IPLb-3_U6KnJRWyYUuvmo5beOxH5hrqeu_4r7CJg (visited on 12/13/2019).

[12] *PYNQ-Z2 Board Schematic*. URL: http://www.tul.com.tw/download/TUL_PYNQ%20Schematic_R12.pdf (visited on 12/16/2019).

Appendices

The project Github repository is publicly available at

A Source Code

Relevant parts of the source code is shown in these appendices to enable the reader to better understand the functionality of the modules.

A.1 PWM control Module

```
1
2  architecture Behavioral of pwm_control is
3  signal count_sig: unsigned(30 downto 0) := (others => '0');
4  signal dutyCycleVal_sig : unsigned(12 downto 0) := "0101110001000";
5  begin
6
7  process(clk_inc)
8  begin
9
10     if ( rising_edge(clk_inc) ) then
11         count_sig <= count_sig + 1;
12     end if;
13
14     -- if 10 [ms] have passed evaluate
15     if ( count_sig >= 100000 ) then
16         if ( pwm_inc = "01" ) then
17             dutyCycleVal_sig <= dutyCycleVal_sig + 1; -- DECREASE PWM
18         elsif ( pwm_inc = "10" ) then -- and dutyCycleVal_sig > 0
19             dutyCycleVal_sig <= dutyCycleVal_sig - 1; -- INCREASE PWM
20         end if;
21         count_sig <= (others => '0');
22     end if;
23
24 end process;
25
26 dutyCycleVal <= std_logic_vector(dutyCycleVal_sig);
```

```
27
28
29 end Behavioral;
```

A.2 PS Info Bits Module

```
1
2 architecture Behavioral of PS_info_3_bits is
3
4 begin
5 pwm_inc <= data(1 downto 0);
6 battery_lvl <= data(2);
7
8 end Behavioral;
```

A.3 Control Loop

The code for implementation of BRAM communication in C is inspired by the code supplied during lecture 6 held by Beck Steckhahn-Strohmer.

```
1
2 /* Include Files */
3 #include <stdio.h>
4 #include "platform.h"
5 #include "xil_printf.h"
6 #include "xil_types.h"
7 #include "xparameters.h"
8 #include "bram.h"
9 #include "xadcps.h"
10 #include "xstatus.h"
11 #include <unistd.h>
12 #include "core_portme.h"
13 /* Constant Definitions */
14 #define XADC_DEVICE_ID   XPAR_XADCPS_0_DEVICE_ID
15 /* Function Prototypes */
16 static int XAdcGetVoltageFromRegisters(u16 XAdcDeviceId, int *voltage, int
    ↪ register_offset);
17 static int XAdcFractionToInt(float FloatNum);
```

```
18 static int float2Int(float f);
19 /* Macros (Inline Functions) Definitions */
20 #define printf xil_printf /* Small foot-print printf function */
21 // converting RAW data from external sourse to voltage
22 #define XAdcPs_ExternalRawToVoltage(AdcData)\
23   ((((float)(AdcData))* (1.0f))/65536.0f)
24 #define V_BAT_EQ 3.9375
25 #define I_CHARGE_EQ 15.75
26 #define V_BAT_0 1.1
27 /* Variable Definitions */
28 static XAdcPs XAdcInst; /* XADC driver instance */
29 /*************************/
30 int main(void)
31 {
32   init_platform();
33   initMemory(); // BRAM
34   //XADC
35   int voltage_reading_A0, voltage_reading_A1 = 0; // Result is in mV going
       ↪ from 0V = 0 to 1000 = 3.3V
36   int pwm = 0b00;
37   print("Hello XADC!\n");
38   int Status;
39   // Get user input
40
41   float v_nominal;
42   xil_printf("Enter the nominal battery voltage [V]:\r\n");
43   scanf("%f", &v_nominal);
44   xil_printf("V_nominal = %d.%d [V]\r\n",(int)v_nominal, XAdcFractionToInt(
       ↪ v_nominal));
45
46   int capacity;
47   xil_printf("Enter the battery capacity [mAh]:\r\n");
48   scanf("%d", &capacity);
49   xil_printf("Capacity = %d [mAh]:\r\n", capacity);
50
```

```
51  float k;
52  xil_printf("Enter the charge speed( k = [0.1, 0.5] ):\r\n");
53  scanf("%f", &k);
54  xil_printf("k = %d.%d \r\n",(int)k, XAdcFractionToInt(k));
55
56  float c;
57  xil_printf("Enter the rate of charge (C):\r\n");
58  scanf("%f", &c);
59  xil_printf("Rate of charge = %d.%d\r\n",(int)c, XAdcFractionToInt(c));
60
61  Status = XAdcGetVoltageFromRegisters(XADC_DEVICE_ID, &voltage_reading_A0,
        ↪  1); // read from A0
62  XAdcGetVoltageFromRegisters(XADC_DEVICE_ID, &voltage_reading_A1, 9); //
        ↪ read from A1
63  if (Status != XST_SUCCESS) {
64    printf("Failed reading from ADC");
65    return XST_FAILURE;
66  }
67
68  // Converts to V
69  float v_adc_1 = (voltage_reading_A0*3.3)/1000.0;
70  float v_adc_2 = (voltage_reading_A1*3.3)/1000.0;
71
72  float i_charge = I_CHARGE_EQ * (v_adc_1 - v_adc_2) * 1000;
73  float v_bat = V_BAT_EQ * v_adc_2;
74  float i_target = capacity * c * k;
75  float t_left = capacity / i_charge;
76
77  xil_printf("Target current = %d.%d [mA]", (int)i_target,
        ↪ XAdcFractionToInt(i_target));
78
79  int led, v_bat_lvl;
80
81  while (v_bat < v_nominal)
82  {
```

```
83    XAdcGetVoltageFromRegisters(XADC_DEVICE_ID, &voltage_reading_A0, 1); //
          ↪ read from A0
84    XAdcGetVoltageFromRegisters(XADC_DEVICE_ID, &voltage_reading_A1, 9); //
          ↪ read from A1
85
86    v_adc_1 = (voltage_reading_A0 * 3.3) / 1000.0;
87    v_adc_2 = (voltage_reading_A1 * 3.3) / 1000.0;
88
89    i_charge = I_CHARGE_EQ * (v_adc_1 - v_adc_2) * 1000.0;
90    t_left = capacity / i_charge;
91
92    pwm = 0b00;
93    if (i_charge > i_target)
94    {
95      pwm = 0b01; //decrease
96    }
97
98    if (i_charge < i_target)
99    {
100     pwm = 0b10; //increase
101   }
102
103   v_bat = V_BAT_EQ * v_adc_2;
104
105   v_bat_lvl = (v_bat - V_BAT_0) *100 / (v_nominal - V_BAT_0);
106   if (v_bat_lvl > 20)
107     led = 0b1;
108   else
109     led = 0b0;
110
111   MYMEM_u(0)=CombineForBram(led, pwm); // using memory addr 0
112
113   xil_printf("Vbat: %d.%d, Time_left: %d.%d, itarget: %d.%d, icharge: %d
          ↪ .%d\n\r", (int)v_bat, float2Int(v_bat),(int)t_left, float2Int(
          ↪ t_left), (int)i_target, float2Int(i_target), (int)i_charge,
```

```
        ↪ float2Int(i_charge));
  }
  xil_printf("Done charging");
  cleanup_platform();
  return XST_SUCCESS;
}

/****************************************/
/* Reading voltage */
int XAdcGetVoltageFromRegisters(u16 XAdcDeviceId, int *voltage, int
    ↪ register_offset)
{
  int Status;
  XAdcPs_Config *ConfigPtr;
  u32 VccPdroRawData;
  float VccPintData;
  XAdcPs *XAdcInstPtr = &XAdcInst;

  /*
   * Initialize the XAdc driver.
   */
  ConfigPtr = XAdcPs_LookupConfig(XAdcDeviceId);
  if (ConfigPtr == NULL) {
    return XST_FAILURE;
  }
  XAdcPs_CfgInitialize(XAdcInstPtr, ConfigPtr, ConfigPtr->BaseAddress);

  /*
   * Self Test the XADC/ADC device
   */
  Status = XAdcPs_SelfTest(XAdcInstPtr);
  if (Status != XST_SUCCESS) {
    return XST_FAILURE;
  }

```

```
147   /*
148    * Read the A1 input Voltage Data from the
149    * ADC data registers.
150    */
151   VccPdroRawData = XAdcPs_GetAdcData(XAdcInstPtr, XADCPS_CH_AUX_MIN+
          ↪ register_offset);
152   VccPintData = XAdcPs_ExternalRawToVoltage(VccPdroRawData);
153
154
155   (*voltage) = XAdcFractionToInt(VccPintData);
156
157   return XST_SUCCESS;
158 }
159
160 /************************************************************/
161 /*
162 *
163 * This function converts the fraction part of the given floating point
        ↪ number
164 * (after the decimal point)to an integer.
165 *
166 * @param FloatNum is the floating point number.
167 *
168 * @return Integer number to a precision of 3 digits.
169 *
170 * @note
171 * This function is used in the printing of floating point data to a STDIO
        ↪ device
172 * using the xil_printf function. The xil_printf is a very small foot-print
173 * printf function and does not support the printing of floating point
        ↪ numbers.
174 *
175 */
176 int float2Int(float f)
177 {
```

```
178   return XAdcFractionToInt(f);
179 }
180 int XAdcFractionToInt(float FloatNum)
181 {
182   float Temp;
183
184   Temp = FloatNum;
185   if (FloatNum < 0)
186   {
187     Temp = -(FloatNum);
188   }
189
190   return( ((int)((Temp -(float)((int)Temp)) * (1000.0f))));
191 }
192
193 /*****************************************************/
194 /* Combining data for bram */
195 int CombineForBram(int battery_thresh, int pwm)
196 {
197   int result_pwm = 0b011;
198   int result_bat = 0b001;
199   int result;
200   result_pwm = result_pwm & pwm;
201
202   result_bat = result_bat & battery_thresh;
203   result_bat = result_bat << 2;
204
205   result = result_pwm | result_bat;
206
207   return result;
208 }
```