Technical University of Denmark DTU

Hands on Microcontroller programming 31070

EVSE Project - Group 10

Lynge Aske Ulbæk Moustgaard — Studienumber: s190467

Rohit Imandi — Studienumber: s202369

21/06/2021

Indholdsfortegnelse

1 Introduction

Author: Lynge

The frequency of an electric power system is a non-linear function of the electricity power consumption and production. Whenever a load is connected to or disconnected from the grid, the frequency of the grid is affected. Due to physical constraints at the generation side, the grid frequecy decays during an energy deficit and increases during energy surplus. Thus an electric power system requires a continuous balance between production and consumption of electricity to keep the grid frequency constant. Electric vehicles that are plugged into the socket affect the grid frequency. This disturbs the amount of current being drawn to charge the vehicle thereby leading to various unwanted consequences.

This report discusses a possible solution that monitors the power system's frequency using an embedded microcontroller IoT application. The application allows the electric vehicle owner (the user) to either opt for automatic charging or manually select the charing current remotely from a web application. The current charging the vehicle is controlled by the duty cycle of PWM signal generated by the microcontroller based on the *IEC61851* standard.

The author of each paragraph and/or section is stated at the start of it. We were both involved with the process of writing alle the sections and discussions about them.

2 Requirements

Author: Lynge

In this project an arduino MKR1000 microcontroller is used to process the frequency and amplitude of the elctrical system. An oscilloscope is used to provide the microcontroller with a sinusoidal wave of frequency 50 Hz and peak to peak voltage less than 1.65 V. This sinusoidal wave has been used to emulate the electrical system's frequency and amplitude. An LCD display is used to show the values of the frequency and RMS voltage ($Vrms$). Two LEDs, namely, yellow and red indicate when the frequency falls below 49.9 Hz or above 50.1 Hz respectively based on the *IEC61851* standard. MKR1000's built-in WiFi module was used to connect to the Arduino IoT cloud as well as enable remote control from the user.

And so the specification is as listed below:

- Realized on arduino MKR1000 with appropriate analog connections to LCD and LEDs.
- Ardiuno Timer5 library for the interrupts.
- An LCD to display the frequency and V_{rms} locally.
- The IoT application to display the frequency, PWM duty cycle and V_{rms} as well as accept inputs remotely from the user.
- LEDs to indicate the *IEC61851* standard frequency range.

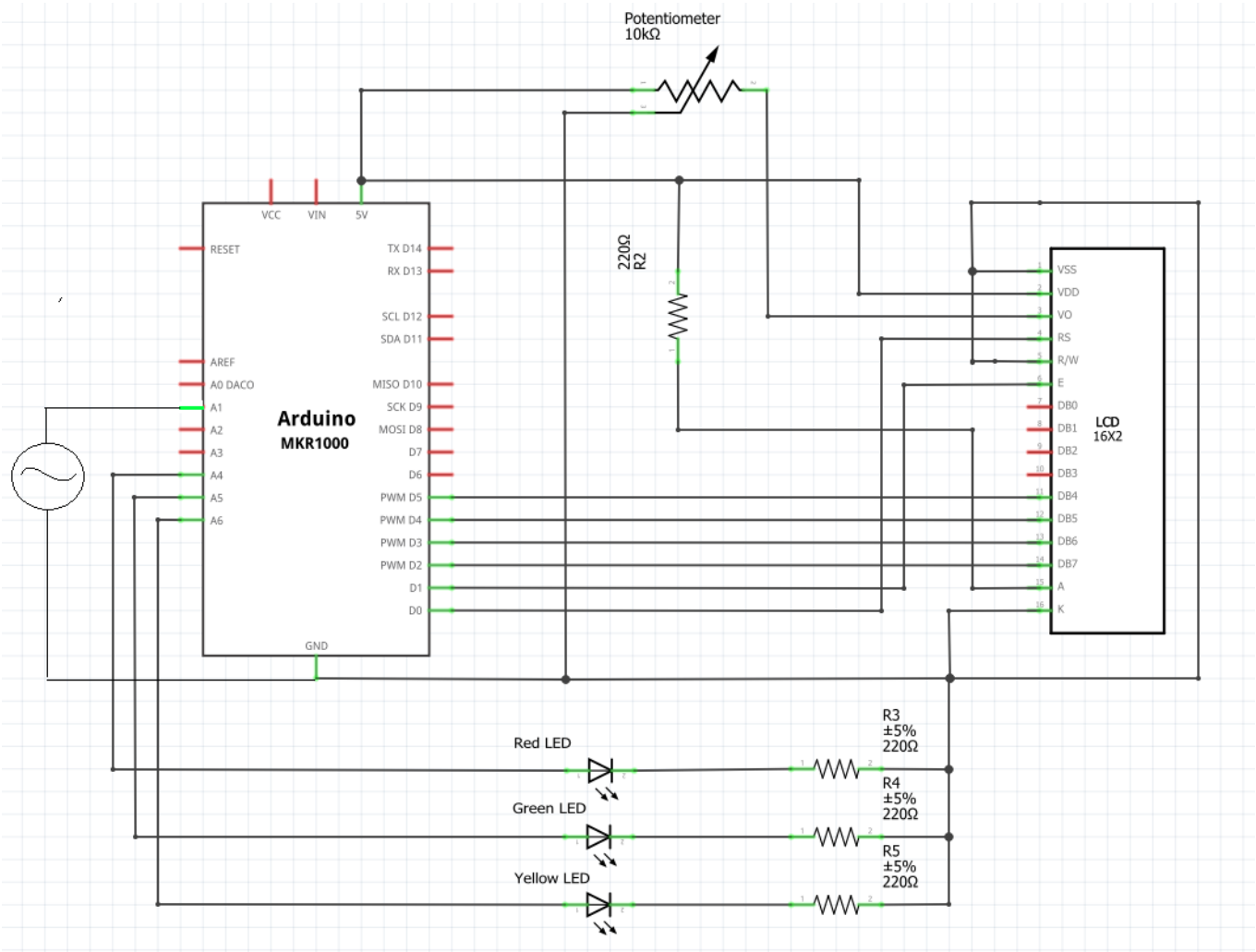
3 Description

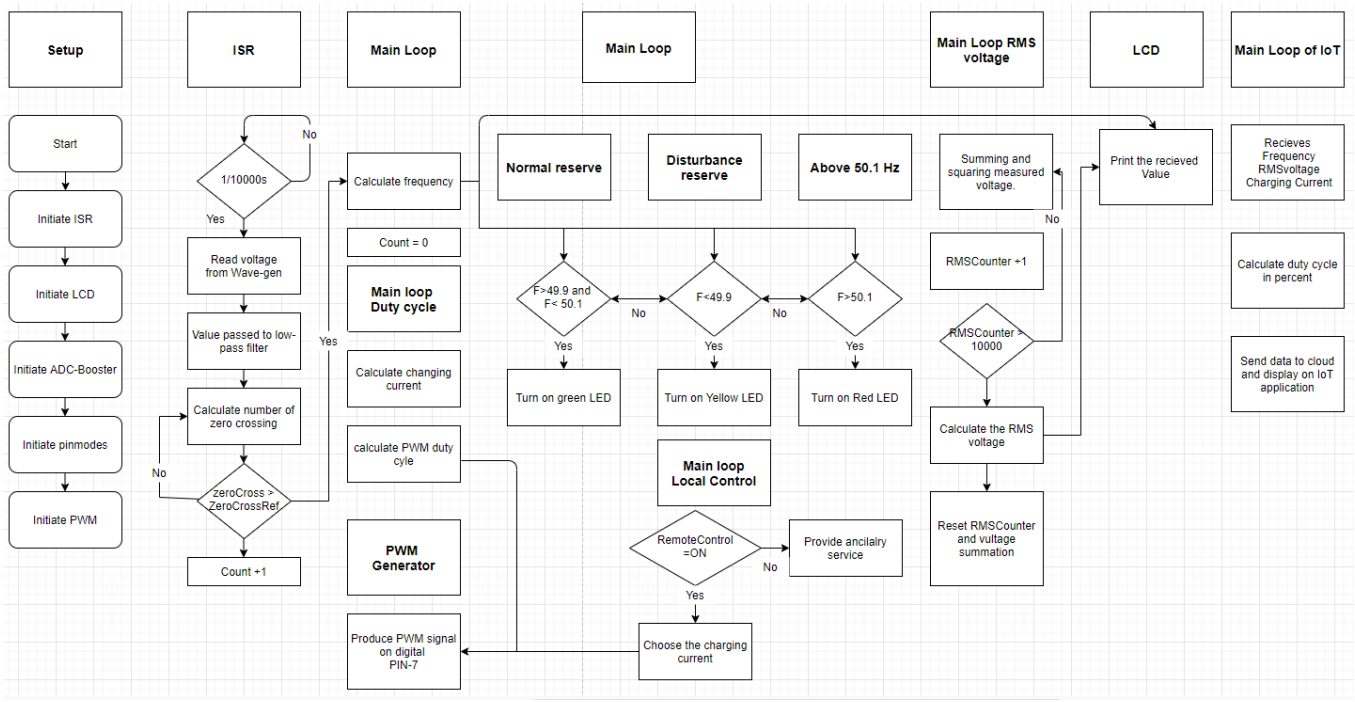
3.1 Overview

Author: Lynge

In this section the description of the implementation of the system is shown. This will mostly be with various figures and illustrations of the system. By the end of each section dedicated to a part of the implementation a flowchart of the given section is shown. The main visable output is the LCD and the IoT interface and the way in which it responds to a frequency inputs provided by the oscilloscope. Furthermore the IoT application makes it possible for the user to vary the current in order to change the PWM and Vrms.

Here is an overview of the systems in- and outputs, analogue components and the connection to the IoT:




Setup
ISR
Main Loop
Main Loop
Main Loop RMS voltage
LCD
Main Loop of IoT
Start
Initiate ISR
Initiate LCD
Initiate ADC-Booster
Initiate pinmodes
Initiate PWM
1/10000s
No
Yes
Read voltage from Wave-gen
Value passed to low-pass filter
Calculate number of zero crossing
zeroCross > ZeroCrossRef
No
Yes
Count +1
Calculate frequency
Count = 0
Main loop Duty cycle
Calculate changing current
calculate PWM duty cyle
PWM Generator
Produce PWM signal on digital PIN-7
Normal reserve
Disturbance reserve
Above 50.1 Hz
F>49.9 and F< 50.1
F<49.9
F>50.1
No
No
Yes
Yes
Yes
Turn on green LED
Turn on Yellow LED
Turn on Red LED
Main loop Local Control
RemoteControl =ON
No
Yes
Provide ancilalry service
Choose the charging current
Summing and squaring measured voltage.
No
RMSCounter +1
RMSCounter > 10000
Calculate the RMS voltage
Reset RMSCounter and vultage summation
Print the recieved Value
Recieves Frequency RMSvoltage Charging Current
Calculate duty cycle in percent
Send data to cloud and display on IoT application

3.2 Measurements and control

Author: Rohit

As mentioned eariler, the sinusoidal wave of frequency 50 Hz and peak to peak voltage below 1.65 V was generated by the oscilloscopr's wave generator. The signal was provided to the analog-to-digital converter (ADC) pin A1 of MKR1000.

3.2.1 Timers, Analog-to-Digital Converter (ADC) and Interrupts

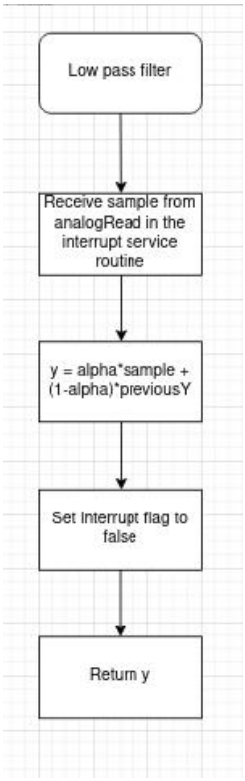
Author: Rohit

Initially, we implemented the timer and interrupt using the timer counter for control application (TCC) registers from the datasheet which involved selecting and setting the generic clock to the right clock source, feeding the generic clock to the chosen TCC register, choosing the type of waveform generation and finally enabling the desired interrupt registers. However, we realized that we could use the Timer5 library for the same. The interrupt was set to trigger at every 100 µs thus invoking the interrupt service routine (ISR) every 100 µs. The sinusoudal signal was sampled using analogRead(). However, the analogRead() function is an abstraction to control the registers and hence is not fast enough to sample the signal every 100 µs. The issue was resolved by manually changing the ADC clock rate to a sufficiently fast rate inside the AdcBooster() subroutine. The sampled signal was subsequently filtered by a first order low pass digital filter with a cut-off frequency at 50 Hz invoked within the interrupt service routine. The interrupt service routine was also used to calculate the frequency based on the number of zero crossings described in section 3.2.3.

3.2.2 Low Pass Filter Implemented as a subroutine

Author: Rohit

The low pass filter was implemented as follows:
$y_i = \alpha x_i + (1 - \alpha) y_{i-1}$ where y_i indicates the i^{th} filtered value, x_i indicates the i^{th} sampled value from ADC and $\alpha = \frac{\Delta\ T}{RC + \Delta\ T}$. ΔT is the sampling interval which in this case is 100 µs and RC is 3.18 ms calculated from the signal frequency (50 Hz) using $f = \frac{1}{2\pi RC}$. Interestingly, when the sampled values were stored in a buffer array, the output waveform of the filtered signal was distorted. The issue was resolved by using two global variables to store the currently sampled value and the previously sampled value for filter processing.



3.2.3 Frequency Calculation Using Zero Crossing Implemented as a subroutine

Author: Rohit

The frequency of the sampled wave is calculated as follows:

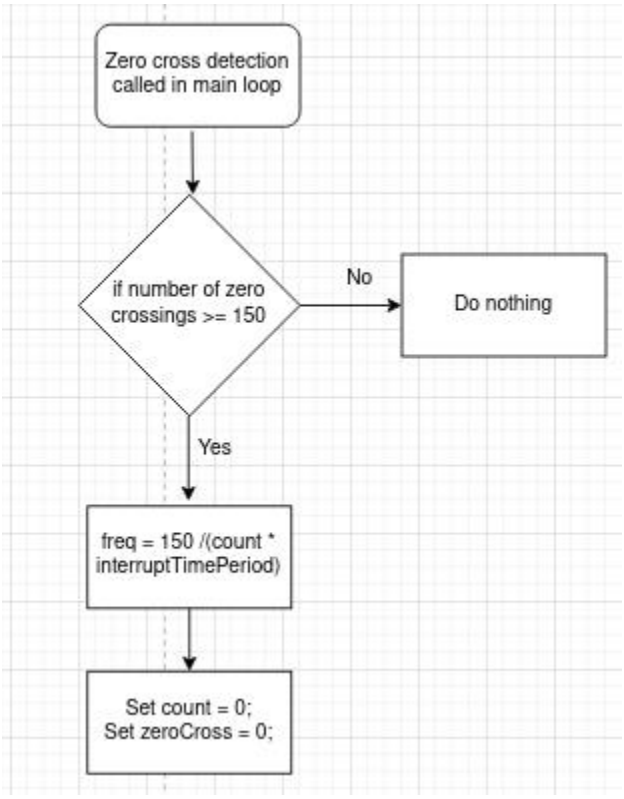
1. As the ADC outputs values between 0 and 4095, there are no negative values. Thus the zero level was determined by halving 90% of the maximum value. While 418 was the maximum of all the sampled values, the frequency of 418 was very low. Most of the values indicating the maximum votlage of the sinusoidal wave were lying in the range of 360-380. Thus it was multiplied by 90%. Any value below this level was considered to be negative and vice-versa. The zero level was set to 187.65.

2. A zero crossing occurs when the previously sampled value is less than and the currently sampled value is greater than the defined zero level. Additionally, for a standard 50 Hz sinusoudal wave, the number of zero crossings in 3 s is $\frac{3s}{200ms} = 150$

3. Thus the frequency is $\frac{number\ of\ zero\ crossings >= 150}{number\ of\ interrupts\ occured\ during\ the\ time \times\ time\ period\ of\ an\ interrupt}$

4. The time period of an interrupt is calculated by counting the number of interrupts in one second. The number of interrupts in one second was 10917 resulting in an interrupt time

period of $9.151642719 \times 10^{-5}$. Interestingly, rounding off the interrupt time period to the second decimal decreases the accurace of frequency calculation. Thus we decided against rounding off.



3.2.4 PWM Duty Cycle

Author: Rohit

The PWM duty cycle was determined in two steps. The first step involved using droop control to calculate the charing current based on the calculated frequency. The second step was about mapping the charing current to the corresponding duty cycle based on the *IEC61851* standard.
To reduce the frequency of the PWM signal to 1 kHz, a subroutine titled initializePWM() was invoked within void setup(). Understanding and manually adjusting the timer counter for control applications (TCC) was easier here because we had previously implemented the timer and interrupt in a similar way without using the Timer5 library. The subroutine performed the following steps:

1. Select the 8 MHz OSC8M clock source, generic clock 4, divide it by 8 to produce 1 MHz, set the duty cycle to 50/50 and enable the generic clock 4.

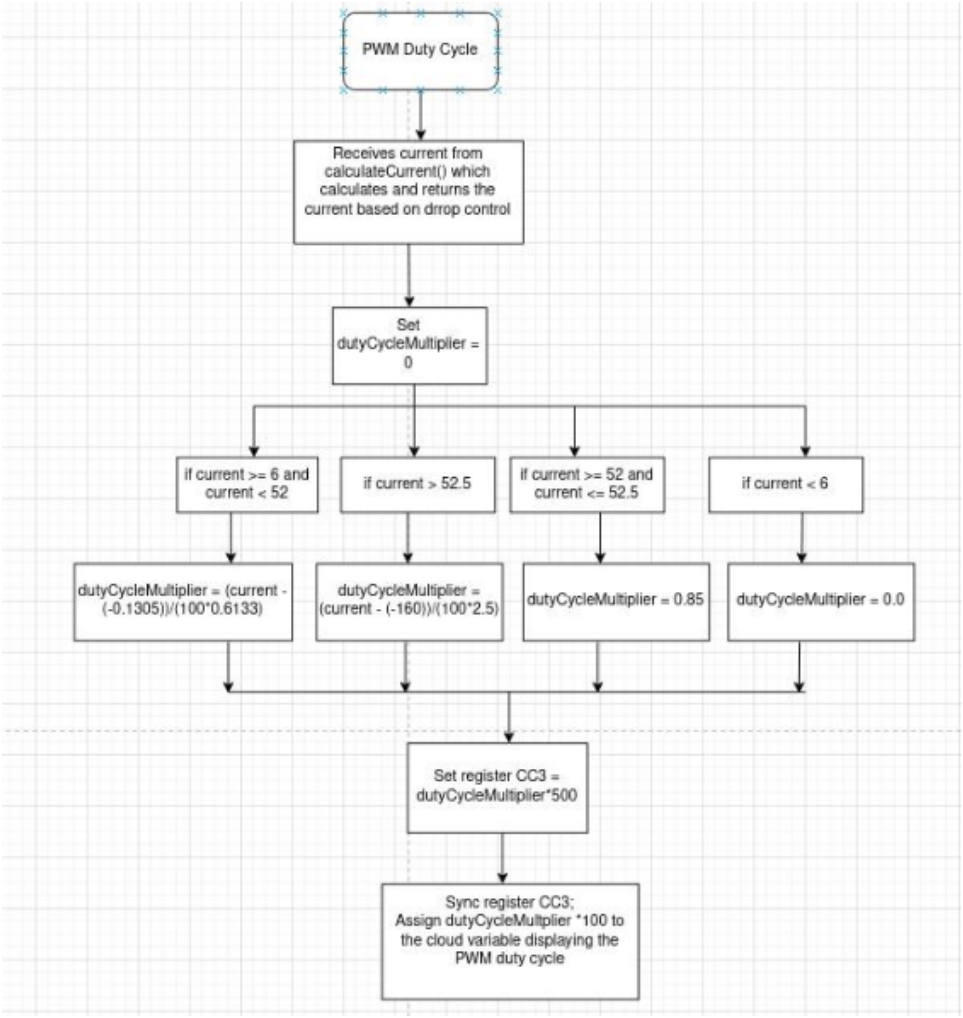
2. Enable pin D7 (pin 7 on MKR1000) and connect it to TCC0 which is the chosen timer counter

for control application

3. As we chose the dual slop PWM generation, the frequency was set to 1 kHz by calculating PER value using $frequency = \frac{1MHz}{2*N*PER}$ where N is 1, the chosen prescalar. Thus the computed PER value - 500, was assigned to the PER register.

4. The initial duty cycle was set to 50% by multiplying the PER value with 0.5 and assigning this to the CC3 register

After intializing the PWM wave, the duty cycle of the wave was set by the subroutine titled setPWMDutyCycle() which accepts the calculated current value to compute the duty cycle multiplier. The multiplier was computed based on the *IEC61851* standard using a piecewise function. The multiplier was then assigned to the CC3 register to control the PWM duty cycle. Additionally, the duty cycle multiplier as well as the computed current were assigned to the cloud variable (described in section 3.4) to display on the dashboard.



3.3 LCD

Author: Lynge

The LCD VSS and VDD on the LCD are connected to ground and 5v on the MKR1000 respectively, R/W pin is to ground. The V0 of the LCD which is the background voltage input, is connected through a potentiometer before going to ground and 5 V. The potentiometer is basically a voltage divider, that serves the purpose of scaling down the 5 V signal so that the background and foreground of the LCD are different in brightness. Since its implemented with a potentiometer, it is variable, but will most of the time simply be at an optimal setting.

The E and RS pins are connected to the MKR1000 in the digital pin 4 and 5 respectively, and together with the DB4-7 are they the "connection" between the program and the LCD, and so simply are connectied in this way in accordance with the standard found in various warm up projects and exercises.

As an addition to the LCD, there are implemented 3 LEDs, a yellow, green and red one, in order to give another visual indication as to whether or not the system frequency is within the desired interval of 49.9Hz-50.1 Hz. In this interval the green LED is on, while the red is on for values higher than 50.1 Hz and yellow on for values lower than 49.9 Hz. In the code, this is done by an if- else if statement, setting their respective pins HIGH or LOW based on the conditions described above. Furthermore, the flickering of LCD was corrected by placing the displayOutputs() in the if-statement calculating the RMS voltage.

3.4 Communication

Author: Rohit

The MKR1000 was connected to Arduino IoT Cloud platform via its built-in WiFi module. The basic network configuration involving the SSID and WiFi password as well as the skeleton sketch were automatically generated by the platform after creating a "Thing" and the "variables". The variables defined were:

1. displayFreq - a read only variable to display the frequency of the sinusoidal wave
2. displayPWMDutyCycle - a read only variable to display the duty cycle of the PWM wave
3. displayVrms - a read only variable to display the V_{rms} of the sinousoidal wave
4. remoteControl - a read/write variable to enable remote control. When this is on the user can select the desired charging current and the corresponding duty cycle will be displayed on the LCD as well as the dashboard

5. selectChargingCurrent - a read/write variable to select the charging current when the remoteControl is on

When interfacing the IoT cloud with the local application, the code had to be restructured to account for remote control via the dashboard. For instance, prior to interfacing, the current calculation was implemented within the setsetPWMDutyCycle() function. Subsequently, the current computation was placed in another function titled calculateCurrent().
The microcontroller was connected to a mobile hotspot as DTU's WiFi had multiple layers of security preventing the device from connecting.
The skeleton sketch places the function "ArduinoCloud.update()" inside void loop() thereby significantly increasing the processing and execution time. This is because "ArduinoCloud.update()" performs multiple background operations to connect and update the cloud variables. Thus, constantly invoking it during every execution of void loop() results in increased processing time. The issue was resolved by placing "ArduinoCloud.update()" inside the if-statement used to calculate V_{rms}.

4 Test

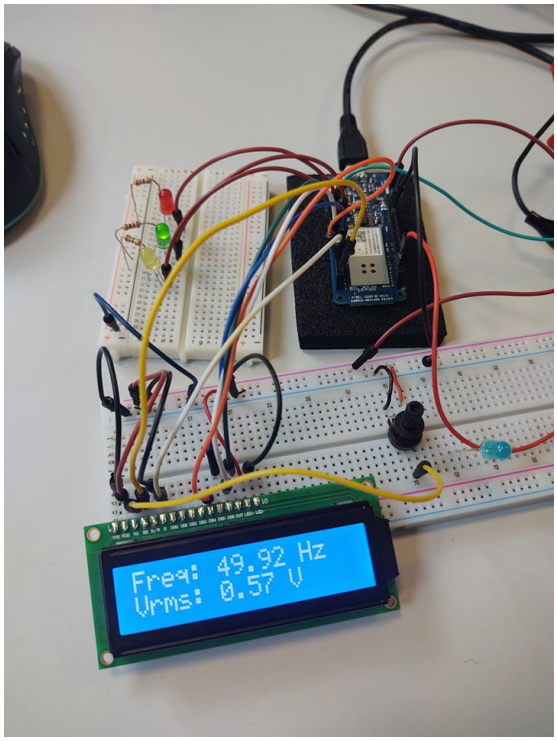
This section contains illustrations of the output of the setup, screendumps from the IoT project flow showing the functionality of the system.

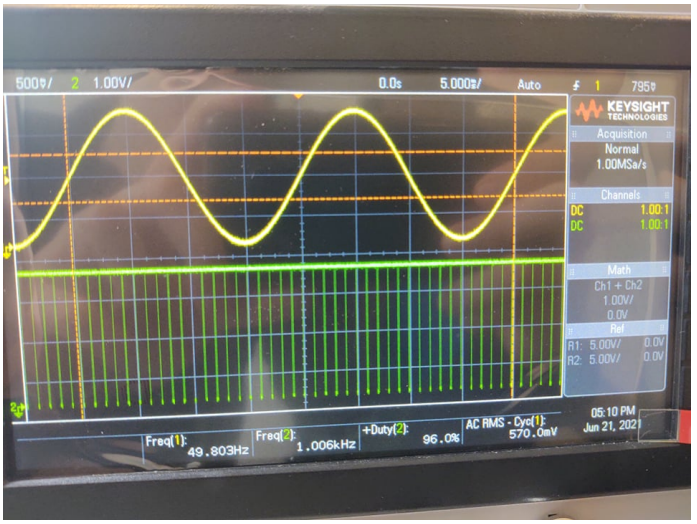
4.1 LCD test

Author: Lynge

First a test was done to check if the LCD displayed the correct values for the frequency and V_{rms}. The values were controlled manually from the oscilloscope, and so the application on the IoT was not used.

The following is a foto of the system as well as the output of the oscilloscope. There is little to no discrepancy between the scope of the frequency and Vrms shown on the LCD. It is also possible, if a bit dim, that the green LED is on, as desired. The refreshing rate was a bit slow, however the functionality was as desired.



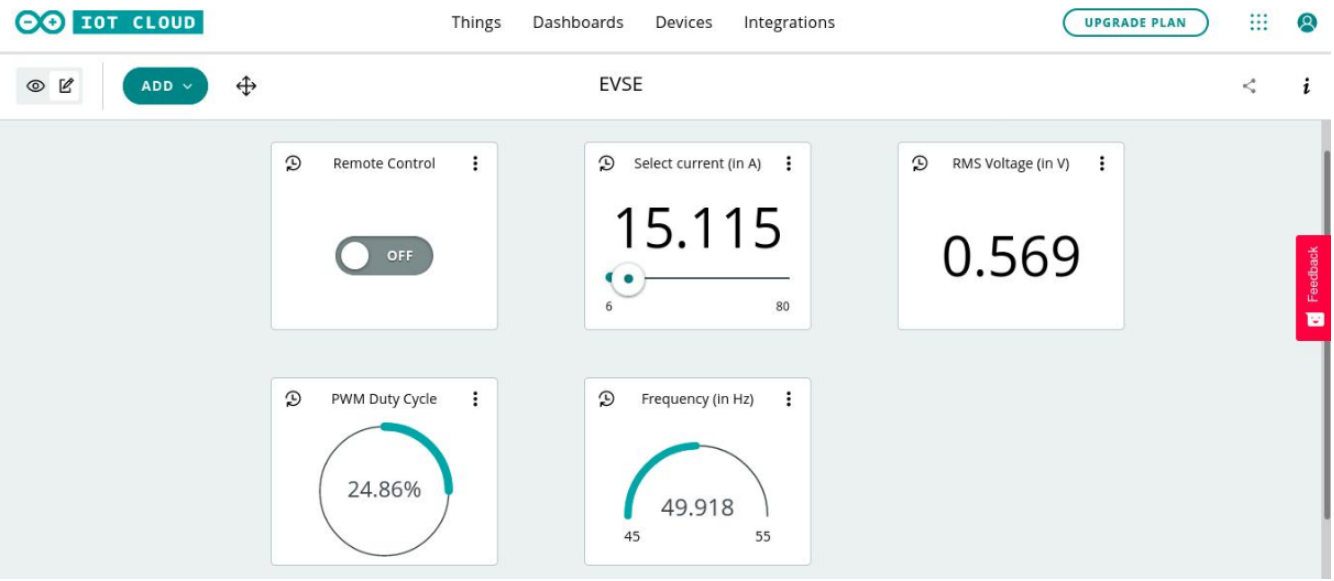


4.2 Test with IoT connection

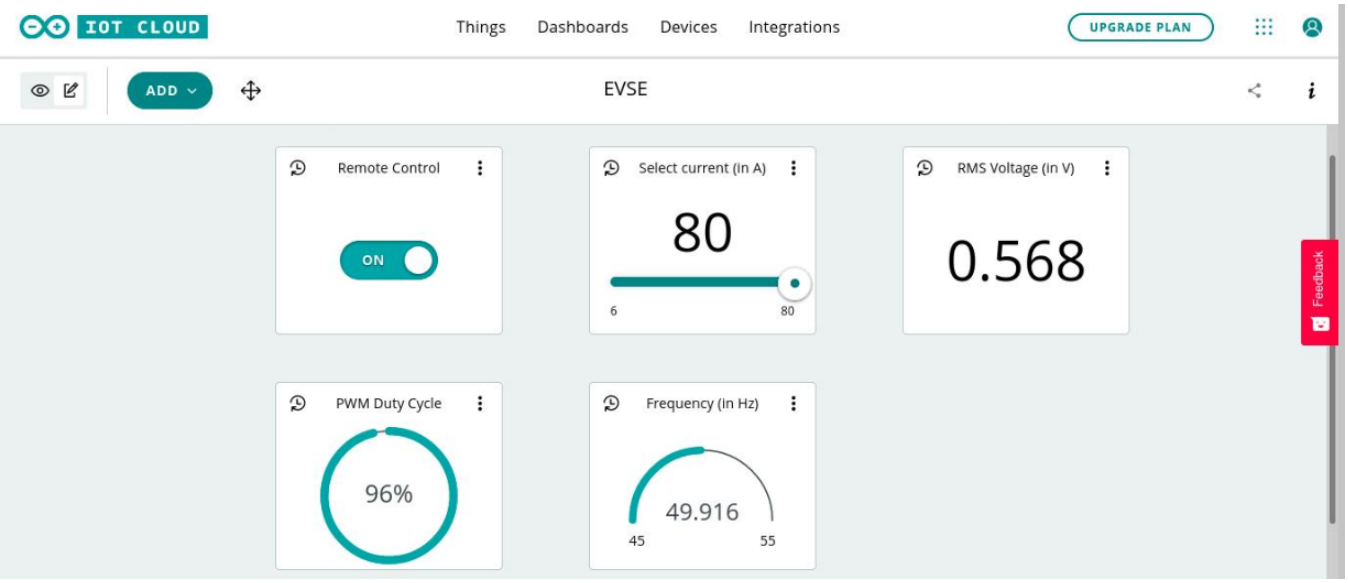
Author: Lynge

In order to verify the connection to the Iot two tests were conducted. First a test of the variables shown was conducted. As stated in section 3.4 the application was capable of showing variables and

has a remote control.The control was switched off in order to verify that the application displayed the correct values, and was as follows:



Here the remote is off as stated above. The application should in this case simply show what the oscilloscope was set to, and did so, since the dutycyle was set quite low here.
When the application is turned on it is possible to change the duty cycle by varying the current (in mA). Here is a test where the remote control is turned on, and the current turned to be at its maximum value of 80 mA.



Here the duty cycle is now at 96% which is exactly the expected value at max current. Note that the RMS voltage and Frequency are indifferent in the two tests. These two values are not changed by changing the current, and will continously show the RMS voltage and frequency of these values, until the frequency is changed on the oscilloscope.

By these three tests the system is shown to behave as specified, and that both the LCD and IoT application function as intended.

5 Conclusion

Author: Both

As seen from the tests and the presentation, the system meets the specified requirements. An application with remote control access is presented through IoT and the desired quantities are displayed both on the LCD and the IoT application. Thus the system behaves in such a way that it can addresses the problem statement outlined in the Introduction section, and is therefore successful. Furthermore, the entire system can be ported onto a PCB to ensure reusability.