Contents

List of Figures

Nomenclature

f_c	Cut off frequency.
I_{RMS}	RMS current
P_{avg}	Average power
T	Period time
x_a	X coordinate of sample below offset.
x_b	X coordinate of sample above offset
y_a	Y coordinate of sample below offset
y_b	Y coordinate of sample above offset
ΔT	Sampling interval
P	Power in watts.

·⁚A brief description to frequency controlled demand. What it means, why it is smart, and how it can be employed.⁚·

1 Introduction

The electrical power grid must be able to provided electricity in correspondence to the consumption. Since the consumption varies the power grid must constantly be adjusted. The generators can accommodate these variations to some extend due to kinetic energy storage. However the frequency of the system will drop if there is an energy deficit and will increase if there is an energy surplus. If the energy deficit becomes too great it could cause a blackout. Whenever a supplying company cannot provide the demanded power, the company must buy electricity on the market. At sudden energy deficit the system must respond within 15 minutes by buying electricity in the regulating market. Prices in this market are often much greater than day-to-day markets, however, the end-user never experiences these changing market prices. The end-user will always be priced with a fixed amount relative to the kWh used.

A possible alternative to the solution described above, is to switch off household appliances in case of energy deficit. Since the frequency of the grid can be measured anywhere it is possible to implement intelligent energy control of appliances. The energy control must thus be able to switch off individual appliances minimizing discomfort for the end-user. This approach is referred to as frequency-controlled demand.

In this project a microcontroller will be programmed to monitor the frequency of the grid and act accordingly. The microcontroller used in this project is on a development board from IAR. It has an ARM7 core with build in peripherals such as ADC, DAC and Ethernet. It has an internal clock of 72 MHz and has 64 kB on-chip SRAM used for working memory and 512 kB on-chip flash memory where code and constants are stored. Furthermore it has 64MB external SDRAM. The board has an LCD screen with a resistive touch screen mounted on top.

Besides the development board, an analog board will be used to provide low-voltage signals to the microcontroller. The analog board has two mounted relays connected to a light bulb and a standard wall socket. Furthermore a measurement transformer circuit is used to scale voltage and current to an interval of 0-3,3V. The two boards are shown on figure 1.1.

As shown on the figure, the two boards have a number of connectors making it possible to connect the board with coaxial cables. During development a oscilloscope can be connected as well. The ARM7 is programmed through a JTAG usb cable.

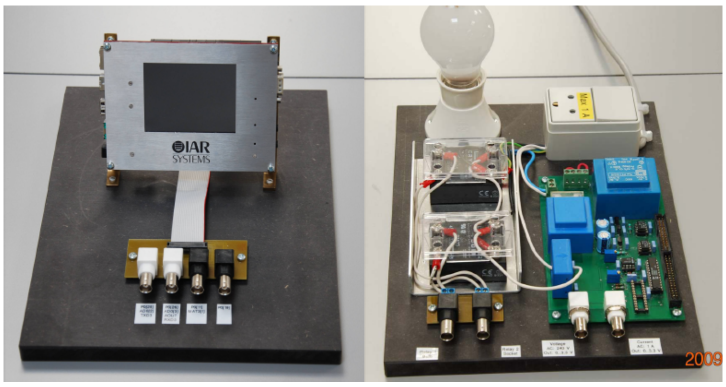


Figure 1.1: Development and analog board.

The project consists of three parts:

- Measurements and control - measure frequency, voltage and current and set relay signals accordingly. All measurements will be calculated using the ADC of the microcontroller. This will be handled through interrupt routines.
- User interface - the LCD screen and touch screen will be used to implement a graphical user interface. The LCD is updated with measurement values and various information and the user will be able to input parameters to the system using the touch screen. The touch screen uses the ADC to determine the location of the touch.
- Communication - a uIP[1] stack is implemented and communication between the microcontroller (server) and a PC (client) is established with a cross-over cable connected to the Ethernet ports. The measurement data is stored in an XML file and displayed client-side in a web browser. A web page is implemented using JavaScript and jquery to retrieve the data in the XML file and display it in a readable fashion.

For code development the IAR's Embedded Workbench for ARM - kick start edition is used, which is limited by code size. For web page development a standard text editor is used. Graphical content is implemented using the jquery JavaScript library from {P}flot, Ole Laursen, IOLA.

[1]Micro IP

·: This chapter is dedicated to the analysis that formed the base for further development. :·

2 Analysis

As described in the introduction the project consist of three part. The analysis of all three parts are described on the following sections.

2.1 Measurements and control

The main functionality of the system is to monitor the frequency of the input signal and act accordingly. As mentioned in chapter 1 the electrical power system has a normal operation interval of [49.9Hz ; 50.1Hz]. Under controlled disturbance the frequency will be in the interval [49.5Hz ; 49.9Hz]. The frequency must not drop below 49,5Hz. Hence the implemented frequency-responsive system must react to frequency drop below 49,9Hz to be compatible with the standard. This will require high resolution and precision of frequency measurements and timely response.

The development board will be connected to the analog board when the system is fully functional. The analog board has two mounted relays connected to a light bulb and a standard wall socket. The light bulb must act as disturbance reserve and the device connected to the wall socket as normal operation reserve. The connected device can be e.g. a laptop. This will insure the least amount of discomfort for the user, since the light should not be turned on and off frequently. A laptop charger being toggled between on and off will not cause noticeable disturbance for the user. However, if the toggling occurs too frequently it might cause damage to the laptop charge.

The development board has a number of connectors where the input signal must be connected. The first step of the implementation will be to convert the analog input to digital values using an ADC. The sample rate must be appropriate with respect to the input frequency, which is in the area of 50Hz. 10.000 samples per second will give a relatively good resolution and still leave time for the processor to perform other tasks, since the ADC has a operation clock of 4,5MHz and uses 11 cycles to perform a conversion.

Noise can potentially ruin measurements thus must be handled. On the development board the display in particular causes disturbance. To remove noise a filter will have to be implemented. A fist-order low-pass filter will ensure that the high frequency noise is filtered. In discrete time, the filter output y is computed from the input x as follows:

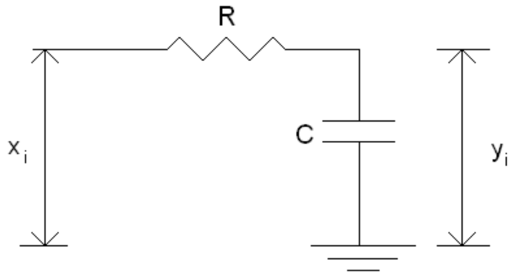


Figure 2.1: Analog RC filter

$$y_i = \alpha x_i + (1 - \alpha) y_{i-1} \tag{2.1}$$

$$\alpha = \frac{\Delta T}{RC + \Delta T} \tag{2.2}$$

ΔT denotes the sampling interval and RC a time constant with the following relation to the cutoff frequency f_c:

$$RC = \frac{1}{2\pi f_c} \tag{2.3}$$

Figure 2.2 shows the behavior of the filter. A natural choice of the cutoff frequency would be 50Hz. However, the signal amplitude will be damped 3dB at the cutoff frequency and the signal will be phase shifted. The phase shift will not influence the measurements but the damping will give incorrect results when measuring e.g. voltage and power. This must be taken into account.

To calculate the frequency of the input signal, zero-cross detection can be applied. In this case, however, the signal should be between 0V and 3,3V as the ARM7 cannot operate with negative voltage levels. Hence a virtual zero will be applied at 1,65V, which is the offset of the input signal. When the input signal crosses the offset voltage level, two consecutive samples outputted from the ADC will be lower than the offset and higher than the offset, respectfully as seen on figure 2.2.

Within a period of the input signal there exist three zero-crossings: one on an increasing slope, one on a decreasing slope and finally one on an increasing slope. To calculate the period, only the two crossings on the increasing slopes will have to be detected. The ADC output values are in the range 0x000 - 0x3FF (0 - 1023). Where 0x000 corresponds to 0V and 0x3FF corresponds to 3,3V. Thus the offset will have a value of 1023/2 = 512 (0x200).

To improve accuracy, interpolation can be applied. The simplest method is to use linear interpolation between the two consecutive samples at a crossing using the following formula:

$$y = y_a + (y_b - y_a)\frac{(x - x_a)}{(x_b - x_a)} \tag{2.4}$$

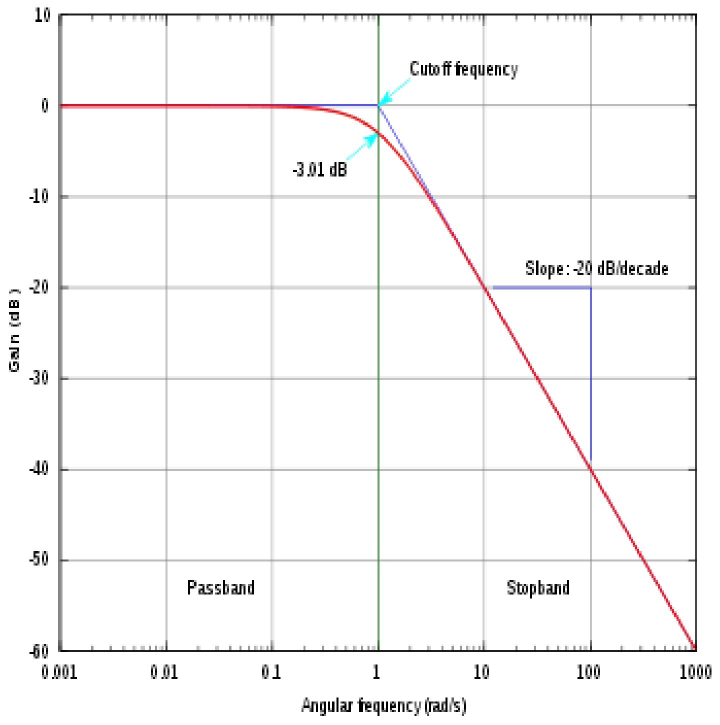


Figure 2.2: Low pass filter

Here, (x_a,y_a) denotes the sample below the offset and (x_b,y_b) the sample above the offset and y the offset itself. To calculate the crossing, the formula is rearranged:

$$x = \frac{(y - y_a)(x_b - x_a)}{(y_b - y_a)} + x_a \tag{2.5}$$

Besides the frequency, the voltage and current should be sampled as well. The results should be expressed with RMS values, which is described as follows in discrete time:

$$X_{RMS} = \sqrt{\frac{\sum_{n=0}^{N} x_n^2}{N}} \tag{2.6}$$

From the voltage and current measurements the power can be calculated as well. The power should be expressed as an average, hence can be expressed as follows in discrete time:

$$P_{avg} = \frac{\sum_{n=0}^{N} u_n i_n}{N} \tag{2.7}$$

The analog board has measurement transformers which will scale the voltage and current in the interval 0-3,3. However, a voltage of 240V or a current of 1A

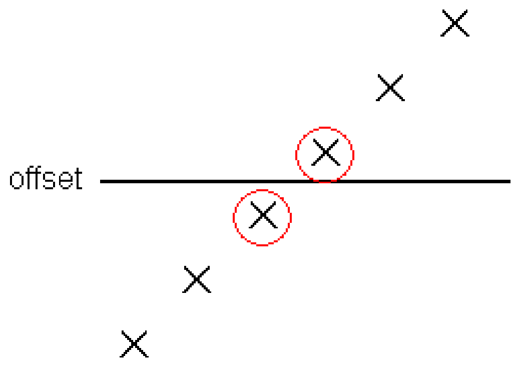


Figure 2.3: Samples at sample offset

(maximum values) is not guaranteed to output 3,3V. This will have to be calibrated in order to get correct results.

2.2 User interface

The development board has a LCD screen and a touch screen mounted on top. This provides the ability to make a graphical user interface (GUI) and have the user interacting with the system via the touch screen. The main functionality of the GUI is to display immediate measurements such as frequency, voltage, current and power. Furthermore the user should be able to set a number of parameters to be used by the system. To avoid that displayed values oscillate it might be necessary to output average values e.g. an average over one second. To implement the GUI, an image (or several) must be stored on the microcontroller. Due to limited space the images must be appropriately compressed e.g. by using a color palette with 256 colors.

The touch screen is of the type resistive. This means that a voltage across the screen will be altered when touched. This can be used to determine the position of the touch on one axis. Thus it is necessary to sample the voltage across the screen on both axes. Furthermore several samples on both axes will have to be used since this type of touch screen is not very accurate. A mean of sampled voltage levels should make it possible to determine the position fairly accurate. The voltage samples from the touch screen must be digitalized using the ADC. This poses a potential problem, since the ADC will already be used for measurements. This issue must be handled carefully in order to avoid conflicts.

2.3 Communication

The development board has an Ethernet port making communication via TCP/IP possible. The third part of this project is to implement communication between the board and a PC. The communication will be a closed network between the board and the PC by using a cross-over cable. This means that the microcontroller will act as server and the PC as client. The board supports the μIP TCP/IP stack and a compact implemented version is provided from IAR System, hence will be the main inspiration for this part of the project. The web pages on the board are compiled into the project and can be modified using the utility makefsdata, provided in this course.

The file transfer of data must be implemented using XML files. The provided μIP stack does not support the file extension .xml thus the μIP must be updated. Furthermore the μIP has a scripting engine supporting shtml. This must be extended as well to support scripted XML (sxml). The XML file should contain all measurements. Besides immediate values, some sort of history should be included in the XML file as well.

The XML file can be displayed in a web browser client-side however, this is not very readable in general. To remedy this, a web page must be implemented. To retrieve the data in the XML file JavaScript can be included in the page or by using style sheets. The best way to display historical data is properly by using some sort of graphics. This can be achieved by using jquery, hence using JavaScript to retrieve data from the XML file would be an obvious choice. When implementing the web page it is possible to make a fancy design and graphical representation of historical data. However, care must be taken not to make the web page to extensive and thereby occupying to much space on the board.

∵This chapter describes all parts of the implementation process. From ADC sampling, to web serving, to the graphical user interface∴

3 Implementation

This chapter contains a description of actual implementation. As described in chapter 2, the implementation can be divided into four different main parts, of which several subdivisions will be covered later in this chapter.

- The ADC, responsible for the actual measurements and the control of the output signals.
- The GUI, containing the graphical user interface along with the touch input.
- The web server, hosting a human readable presentation of the systems measurements along with a machine to machine xml interface,
- The web page, the actual presentation of data, in a human readable format.

The following table displays the allocated resources throughout the project.

Resource	**Usage**
ADC - channel 0	Touch screen measurements on x-axis
ADC - channel 1	Touch screen measurements on y-axis
ADC - channel 2	Current measurements
ADC - channel 3	Voltage and frequency measurements
Timer0	μIP timer
Timer1	Measurement timer
External SDRAM	Background image of main window: 0xA1000000 to 0xA1257000 Background image of setup window: 0xA1258000 to 0xA14A0000 Background image of network setup window: 0xA14B0000 to 0xA1707000 Bypass button image: 0xA1708000 to A1710810

3.1 ADC

This section covers the implementation of the actual measuring system. As previous described, the core of this project is to make precise and continues frequency measurements of the power signal. Besides measuring the frequency the current, voltage, and power is also relevant. The on-chip 10-bit ADC is used to make the actual measurements. From the analog board two indication signals are given. One for voltage and one for current. These signal are in the range of 0v to 3.3v. From these two sources the program must be able to deduct the frequency, actual power usage, the RMS voltage, and RMS current. From these measurements the program must control a couple of relays on the analog board. To implement these requirements the following elements must be implemented.

- Initialization the ADC to make voltage measurements and the necessary input ports.
- A timer function that will enable continues sampling with the same intervals.
- Make both voltage and current measurements.
- Processing of internal ADC output data and physical unit values such as Voltage and Frequency.
- Make the mathematical calculations in order to convert the samples into useful values.
- A algorithm that controls the analog boards relays based on the frequency measurements.

3.1.1 ADC Initialization

This part is primarily done in the adc_init function, found in the adc.c file. Firstly the ADC is powered up and enabled. Next the ADC clock frequency is set. A frequency of 4.5MHz is chosen. This is to insure that the ADC has enough time to get a stable output. Since the ADC clock per default is 18MHz, the clock divider is set to the value of 4. For input the ports AD0[2] and AD0[3] are enabled with their pull-up transistors disabled. The ADC interrupt is set to run the ADCInterrupt function in the main.c file whenever the ADC has a result ready.

3.1.2 Timing Function

To get a stable sample rate the Timer1 is used, because the μIP web server per default uses the timer0. In order to insure precise measurements a fast sampling frequency of 10’000 samples per second is chosen. This sample rate can easily be increased or decreased depending on the workload of the rest of the system, the more samples the better the precision. If the sampling rate is set to high the system may not have sufficient time to run the non essential functionality between the interrupts. A possible way to maximize the utilization of the system resources, is to implement a scheduling algorithm to handle the execution of the different tasks. When ever the timer interrupt flag is set, the system jumps to the Timer1IntrHandler function. Here the ADC is set to start making a sampling and consequently make a ADC interrupt.

3.1.3 Measurements

All measured data are passed through a digital lowpass filter, as described in chapter 2, this is done in order to remove the high frequency noise generated by the board it self. The actual measurements are done in the voltage_measure and current_measure functions. These functions are called from the ADC interrupt, by listening to which channel the result of the ADC is stored. Channel 2 indicates a current measurement is done and channel 3 used for voltage measurements. The current measurement is used for both the rms current calculation and the power calculations. Because the rms current calculations uses a summation of the current squared, as seen in formula 3.1, and the power calculation uses the current times voltage, as seen in formula 3.2, two separate values are stored.

$$I_{RMS} = \sqrt{\frac{1}{T}\int_0^T i(t)^2\,dt} \tag{3.1}$$

$$P = \frac{1}{T}\int_0^T u(t)i(t)\,dt \tag{3.2}$$

After a current measurement is done, the ADC the channel is changed to channel 2 and a new sample cycle is started. A voltage measurement is collected the next time the ADC interrupt runs so, each timer interrupt is followed by a ADC interrupt with current measurement and then a new ADC interrupt with voltage measurement. Besides using the voltage measurement to calculate rms voltage and power, it is also used for frequency calculation. After each voltage measurement a zero crossing detection is done. Because the voltage signal is DC offset the actual zero crossing is defined as 1.65V, which corresponds to a ADC output value of 512. Since the signal make two zero crossings per period it is necessary to distinguish between the two. The new voltage measurement is therefore compared with the previous measurement. If the old measurement is below the offset value and the new measurement is above the offset value it means that one period has passed. After a period is verified to have passed all the power, voltage, current, and frequency measurements are moved into separate calculations variables and a ready flag is set to indicate that a new set data is ready to be processed. With this structure, a significant part for the calculations have de spread out over all the sample periods and thus reducing time needed to finalize the differen calculations.

3.1.4 Data Processing

When the data ready flag is set, the actual data processing begins in the freq_calc function. In order to improve the precision of the measurement, linear function interpolation can be used to find the time for the exact zero crossing as previously described in chapter 2. In order to improve the performance of the linear interpolation calculations a faster approximation is used. By finding the ratio between the voltage measurements before and after the zero crossing, the same ration can be used to approximate the time of the zero crossing. By using this approximation the speed of the computations greatly improve, freeing up vital system resources for other system elements. The degradation of precision by using the approximation is greatly weight out by the possible increase in sampling frequency.

3.1.5 Signal Values and Calibration

A total of 50 sets of frequency, voltage, current and power measurements are accumulated and averaged over to get the final measurement values. This gives reliable and precise measurements free from glitches. Along side these values the maximal and minimal variations of the frequency measurements are also stored. This is to give a better view of the stability of the signal. In order to convert the ADC measurements into SI units, it is necessary to calibrate the system. With the voltage measurements this is no problem since voltage of the running system easily can be measured with a voltage-meter. However current measurements are more difficult, with the given configuration of the analog board. But since the load of the system is know ie. a 60W light bulb the expected current can to some extent be calculated. With respects to the frequency measurements a small offset also need to be adjusted, this was achieved with simple oscilloscope frequency measurements. With correct conventions factors derived from the calibration, the measurements are ready to be store. 60 sets of processed data are stored in the memory for later use in the web server, see section 3.2, resulting in around a minute worth of data stored. Having somewhat larger data storage is not in it self a problem. However when the system needs to process the data ie. send them to the web server, there will be a problem generating the data files and the sending them without using up all the system resources.

3.1.6 Control

With precise frequency measurements in the memory, the core functionality of this system can be implemented. The control of the load of the electrical system. Since this is a prof of concept implementation, with a relatively limited analog testing board, the actual functionality of this intelligent control will be somewhat simple. In this testing setup a laptop will be connected to the electrical socket controlled by one relay and an other relay controlling the light bulb. The main thought here is, that the laptop does not need to be connected to the electrical grid all the time, since it is equipped with an internal battery. This makes it possible for a intelligent system to help keep the frequency of the power signal stable, by disconnecting this no essential appliance with little or no discomfort to the user. With basis in NORDEL, the system must disconnect the non essential appliance when the frequency varies more that $0.1Hz$. Giving the power grid time to correct it self. This correction should be done within 2-3 minutes Therefore it shouldn't be a problem for a system like a laptop to be disconnected from time to time. In case of a larger disturbance on the power grid resulting in a frequency change of more than $0.5Hz$, the essential appliance ie. the light bulb, will also be turned off for a short period of time. The actual implementation of the control of the appliances is somewhat simple, since it in this test case only has two control signal that must be turn on or off depending on the frequency measurements. In a real world example this control implementation would be quite more complex.

3.2 Web Server

This section covers the implementation of the web server. The main job of the web server is to provide a machine-to- human interface (html) and a machine-to-machine (xml) interface. More precisely the job of the web server is to provide a remote access to the measurement data collected by the system. The construction of the actual web page is covered in section 3.3. From the course material the web server implementation is suggested to be based upon a example of a μIP server. Because of this, most of the web server implementation is actually pre made and will therefore not be covered in this report. However to use the μIP server several changes has to be made to the functionality. These changes will be covered in this section.

- Intergrade the μIP server into existing implementation.
- Add support for additional file types, to support the web page.
- Change the scripting functionality to publicize the measurement data.

A short note on the structure of the web server. In order to reduce the amount of data processing done server side, the measurement data will the publicized in a xml format. The web page will then use java scripting to analyze the measurement data, all this is covered in section 3.3.

3.2.1 Integration

For the TCP/IP stack to function, it needs to be able to meet different timing criteria. The web server therefore need to utilize a timer and its interrupt functionality. This gives rice to possible conflict with the existing timer- controlled ADC measuring structure, since a interrupt in this system will not break into the runtime of an other interrupt. However with the structure in the μIP example most of the heavy non-time-critical functionality is run in the main loop, thus not interfering with the measuring interrupts. This structure can thus with benefit be transferred into this system. In the ADC implementation timer1 is used, the web server can therefore continue to use timer0.

3.2.2 File Support

Since the μIP web server uses pre complied web pages, the file support is for internal use. The following file support is added to server.

- `.xml`, the base for machine-to-machine communication.
- `.sxml`, scripted xml format. This format is used to add server scripts to xml files.
- `.js`, java script support is need for the web page.

To add these formats changes are made to the http−strings.c file and corresponding header file. Here the file extension and content type is defined as constants. These constants are then used in the httpd.c file, where support for the files and the scripting format .sxml is added.

3.2.3 Scripting

For this system one script is added to the httpd−cgi.c file. The script needs to make a xml structured data file for the .sxml file format. The script takes the array of processed data and publicizes them in the data.sxml file with the following structure:

```
<measurements>
    <measurement id="0">
        <power>xx.xx</power>
        <voltage>xxx.xx</voltage>
        <current>0.xxx</current>
        <frequency>50.xxx</frequency>
        <frequency_min>49.xxx</frequency_min>
        <frequency_max>50.xxx</frequency_max>
        <sample_cnt>xxxx</sample_cnt>
    </measurement>
    <measurement id="1">
    ...
    </measurement>
</measurements>
```

Most of the tags are quite self explanatory. Except the `sample_cnt`, this counter is a way to chronologicalize the samples. The counter increments one for every new sample. This way it is possible to a external machine to accumulate samples from the μIP server by reading the data.sxml file once per minutes and still not lose the chronological order of the measurements.

3.3 Web Page

The values stored in the XML file are to be displayed on a web page. This is implemented in a JavaScript in a HTML file. The script consists of the following functions:

- loadXML(xmlFile)
- verify()
- update()

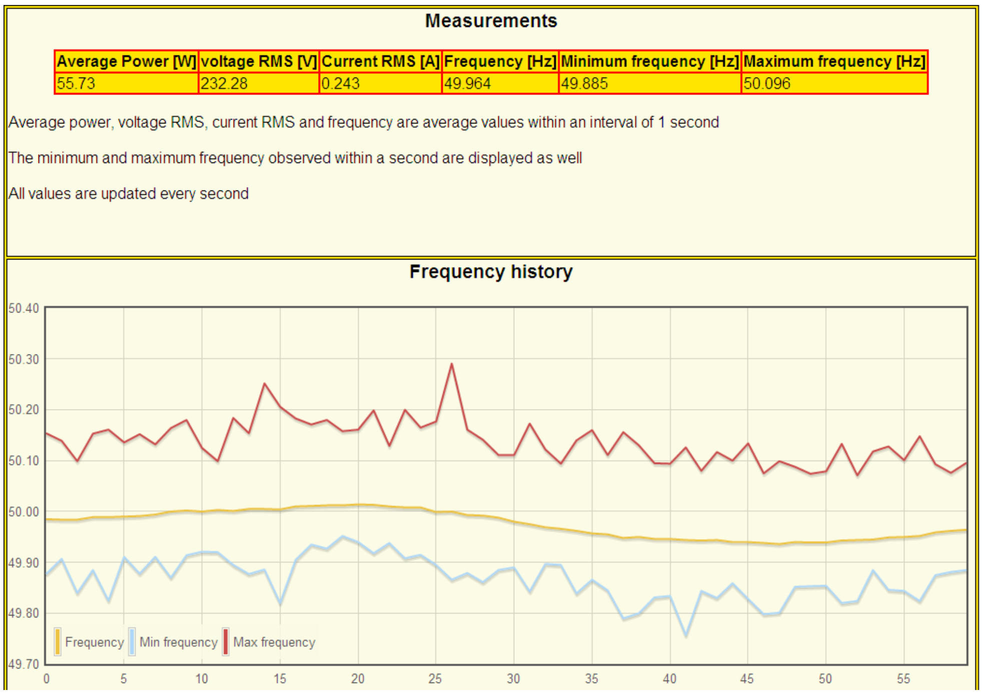
Before loadXML is called the variable xmlFile is created with respect to the browser in use. An element object is created in the function which will be used to load an XML file. The function verify is called from within loadXML to check the status of the object.

When the XML object is created the main function is called, namely update. The first operation of this function is to load the XML file from the server called data.sxml. The XML file contains 60 measurement tags hence the one containing the most resent samples must be located. Each measurement tag has an ID attribute going from 0 to 59. The ID will always remain the same however, the tag containing the latest samples varies over time. Besides the sample values each measurement tag contains a sample count.

The tag containing the latest samples has the highest sample count. A for-loop searches through the sample count values updating a variable with the largest value and the corresponding ID attribute on-the-fly. When the latest measurement values are located in the XML file there are to be displayed. A variable is created and a string is appended with html table tags and the measurement values. A HTML element is given the ID ”output” and the content of the element is set equal to the string variable.

The next step is to produce plots of historical measurements. The jquery script flot[1] is used for his. All 60 measurements in the XML file are measurements over 1 minute combined, hence one measurement per second. The values of the entire minute will be displayed on the web page. Only the most resent values are displayed in a table as described above. In the XML file the oldest measurement tag is discarded every second and the newest measurements are added in a tag. Even tough the newest values can potentially be located in any measurement tag, all other values are ordered according to this tag. This means that if the measurement tag with ID = 27 holds the newest values, the tag with ID = 28 will contain the oldest values, the tag with ID = 29 the second to oldest values etc. Going beyond the last tag with

[1]http://code.google.com/p/flot/



Measurements

Average Power [W]	voltage RMS [V]	Current RMS [A]	Frequency [Hz]	Minimum frequency [Hz]	Maximum frequency [Hz]
55.73	232.28	0.243	49.964	49.885	50.096

Average power, voltage RMS, current RMS and frequency are average values within an interval of 1 second

The minimum and maximum frequency observed within a second are displayed as well

All values are updated every second

Frequency history

Figure 3.1: An overview of the first part of the webpage

ID = 59, the tags wrap-around and continues from the tag with ID = 0 to the tag with ID = 27 (in this example).

The tag with the oldest value is located from the previous calculated ID of the newest value by adding one to that value. The value is stored in a variable denoted start_cnt (start count). In the case where the newest value has ID = 59 the start count is set equal to 0. Six variables are created to collect the following data from the measurement tags:

power, voltage, current, frequency, minimum frequency and maximum frequency

A for-loop runs through tags starting from the start count value to 59 and the variables are extended on-the-fly with measurement values. A second for-loop runs through the remaining tags going from 0 up to the start count. Finally three plots are created using the jquery library. The first plot shows the frequency, the minimum frequency and the maximum frequency. The second plot shows the voltage and the current (RMS values) using two separate y-axes. The third plot shows the power.

The script will run the entire update() function once every second. This is achieved as follows:

```
window.setTimeout(update,1000);
```

This will make the values in the table update every second and the plots will move right-to-left, where the oldest values disappear from the plot to the left and new values appear to the right. More specific; the plots will move one sample to the left every second.

In the body of the HTML file the table and the plots are called along with some heading and paragraphs. The body, headings and tables are formatted using a CSS file. A part of the implemented web page is shown below on figure 3.1 where one out of the three plots are visible.

3.4 GUI functional description

To accommodate user-system interaction a graphical user interface promoting the on-board graphical LCD display and touch screen module has been developed. Much work has been put into the development of the GUI in order to explore the possibilities as well as the limitations of the LPC-2478-STK board, and to break down this frontier. The result is a GUI including a numeric keypad, an analog rotary dial gauge indicator, transparency and an 8 bit image encoding to supply each GUI window with a unique full-screen background.

3.4.1 The main GUI window.

The centerpiece of the main GUI window is a full-height rotary dial gauge indicator that displays the deviation from the expected net frequency of 50 Hz in percent. The pixels describing the face of the dial gauge has been defined as a part of the 320x240 pixels background image whereas the dial gauge needle is animated, and for eye-candy changing color relative to its position on the scale. At the foot of the gauge face are three text labels displaying the net voltage, the amperage and wattage consumed by the attached appliance. Delineating the right hand border of the LCD

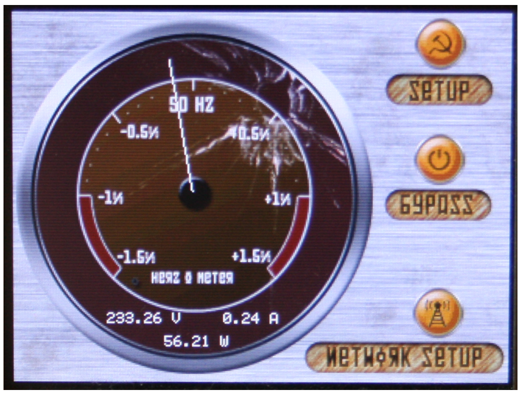


Figure 3.2: The main GUI window.

display are three buttons. The topmost button is a push-button that is used to gain access the setup GUI window. The equidistant button is a toggle button that is employed to either pass or bypass the regulation system. Bypassing the regulation system will leave the appliance in an always-on state. The subjacent button is a push-button that is used to gain access to the network configuration window.

3.4.2 The setup GUI window.

The setup GUI window is employed to give the operator access to configurable systems parameters. These are low pass filter cut-off frequency, sample rate, and operational limits. Delineating the left hand border of the LCD display are text labels with a transparent background indicating the currently active system parameters. These can be edited by tapping on a distinct parameter to set it active. The active parameter will visually be represented by a green background border. The active pa-

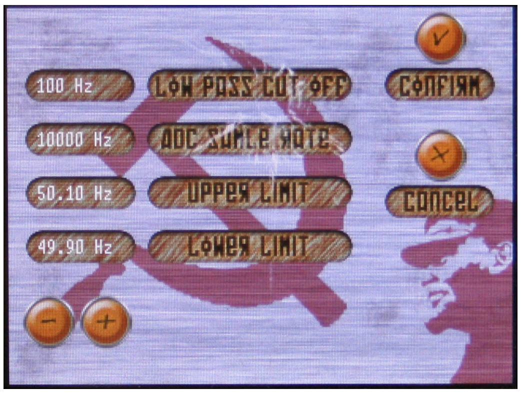


Figure 3.3: The setup GUI window.

rameter can subsequently be adjusted by tapping the subjacent push-buttons. Any edited parameter will be promoted active when the operator confirms the changes by means of the topmost push-button delineating the right hand border of the LCD display. The edited parameters can be discarded by using the by using the subjacent Cancel button.

3.4.3 The network GUI window.

The network GUI window is employed to give the operator access to configurable network parameters of the embedded web server. These are TCP/IP address and Subnet. In order to allow the operator to easily input numeric values, a 10-digit keypad is drawn on-screen. The configurable IP address and subnet are described using standard dot-decimal notation. Mimicking the behavior of the user input from the setup GUI[2], the operator can tap on a distinct 1-byte part of the 4-bytes that

[2]Section 3.4.2

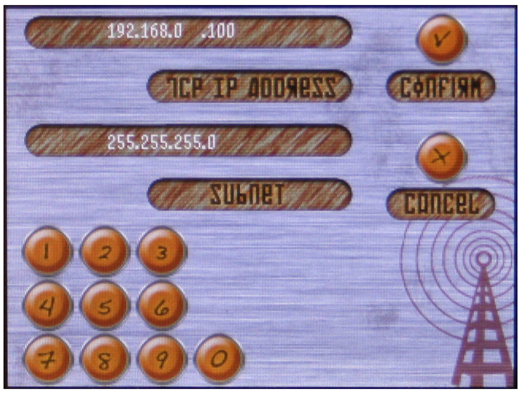


Figure 3.4: The setup GUI window.

comprise the full IP address, and assign a number between 0-255 using the numeric keypad. If an illegal number is tapped, the byte is cleared. As with the setup GUI any edited parameter will be promoted active when the operator confirms the changes by means of the topmost push-button delineating the right hand border of the LCD display. The edited parameters can be discarded by using the by using the subjacent Cancel button.

3.4.4 Development of the GUI

The GUI of the project is based on the compiler-provide demonstation project, TouchDemo.eww[3] . This demonstation project has been tailored to suit its new application as a graphical frontend capable of handling user input.

3.5 Expansion of the drv_glcd.c display driver

In order to display a unique full-screen background on each of the three main windows of the user interface, the compiler-provided display driver[4] has be expanded to handle 8-bit image decompression using custom generated 256 color palettes. The compressed images are expanded to the SD ram of the development board starting at address 0xA1000000. Likewise a c++ terminal application has been written in order to compress and format data from the utility by the supervisor to handle 32 bit hex encoding of images for insertion as constant arrays in the source code. This is done in order to lower the projects build size below the 512 kB barrier defined by the flash memory size of the Arm processor. The image decompression functions, and a documentation of their input arguments and return parameters can be found

[3]Revision 34711 for the IAR LPC2478-STK

[4]drv_glcd.c from the demonstration project.

in the project source code in drv_glcd.c in the project. The source code of the image compression utility is available in appendix A.1.

Animation of the analog dial gauge needle has been done by a pixel-by-pixel method. The dial gauge needle is drawn and redrawn pixel-by-pixel by first reconstructing the pixels at the current needle position. The pixels at the new needle position are subsequently accumulated in an array for consecutive restoration and postliminary the new needle can be drawn. Two functions has been made for this purpose. One that accepts a gauge needle position as an angle, and one that is calibrated with the dial gauge face that accepts a needle position in milHz. Helper functions has been created to read and write individual pixel RGB values to the display. All these functions can be found in the drv_glcd.c display driver.

In order to prevent including a bigger version of the printf family in the compiler options, a ftoa()[5] has been written. This function is emplyed by the need animation functions and hence also located in the drv_glcd.c display driver.

Text labels with transparent background has been introduced to the drv_glcd.c display driver. This has been done by expanding existing functions as well as defining new helper functions.

A list of added and modified functions can be seen below. A list including comments, input arguments and return values can be seen in appendix A.2

- void GLCD_WritePixel (Int32U x,Int32U y,Int32U PixelColor)
- Int32U GLCD_ReadPixel (Int32U x,Int32U y)
- void GLCD_DrawGauge(Int32U Angle ,Int32U GagueColor,Boolean ForceUpdate)
- void GLCD_SetGaugeValue(Int32U milHz, Boolean ForceUpdate)
- void GLCD_ExpandPictures()
- static void LCD_SKIP_PIXEL()
- void GLCD_print_transparent (const char *fmt, ...)
- int GLCD_putchar (int c, Boolean Transparent)

With the above expansions of the drv_glcd.c display driver, all wanted graphical display functionality can be implemented.

[5] float to ASCII function

3.6 Touch screen driver

All operator input is received by means of the touch screen hardware. Input is detected by means of the ADC module of the MCU. This must be synchronized with other utilization of the ADC module. As a result of this a touch screen driver has been developed that can be run in series with other ADC routines and be initiated from an ADC interrupt handler. The touch screen driver named Touch_Scr_Driver.c is comprised of an initialization part, a touch detection part, and a coordinate calculation part. From the operator point of view, a touch event is triggered the moment the finger is released from the touch screen. Hence a de-bounce handling has been build into the touch screen driver. The list of functions in Touch_Scr_Driver.c can be seen below. A list including comments, input arguments and return values can be seen in appendix A.2

- Int32U Touch_Measure(void)
- void Update_TouchScr(void)
- void Init_TouchScr(void)
- Boolean TouchGet (ToushRes_t * pData)

Touch and GUI calculations are started from the main program loop in main.c as follows:

```
    Update_TouchScr(); // Opdater Touchskærm og undersøg om den berøres
    Touch = TouchGet(&XY_Touch); // Hent koordinaterne for berøringspunktet
    Update_GUI(XY_Touch.X,XY_Touch.Y,Touch);
```

Initially Update_TouchScr() is called. If the screen is touched this will start a measurement series. Touch coordinates are pulled from the touch driver using TouchGet(). TouchGet will return true if a touch event has occurred and update the X and Y coordinates where the tap took place. These parameters are finally passed to the Update_GUI() function that handles the GUI.

3.6.1 The GUI event handler UserInterface.c

Handling of most of the elements of the GUI is done by a GUI event handler function. One distinct out of the three GUI windows are shown dependent on a flag that defines the GUI state in the event handler.

```

typedef enum
{
    MainScreen = 0,      // Main GUI window
    ConfigScreen,        // Configuration window
    NetScreen,           // Network setup window
} GUIState_t;
```

The GUI event handler function is called each time a tap is registered on the touch screen. Using a function called...

```
Boolean Button_Hit(Int32U x, Int32U y, ButtonArea_t button)
```

...the event handler performs a check to see if any interactive area of the touch screen in its active state has been tapped. If this is the case, relevant code sections are executed.

Interactive areas are defined as follows and are passed to the Button_Hit function as input argument.

```
ButtonArea_t TestButton =
{
    iLeft,iTop,iRight,iBottom, false
};
```

The list of functions in UserInterface.c can be seen below. A list including comments, input arguments and return values can be seen in appendix A.4

- Ivoid Init_GUI()
- Boolean Button_Hit(Int32U x, Int32U y, ButtonArea_t button)
- void Update_GUI(Int32U cursor_x, Int32U cursor_y, Boolean Touch)
- void DrawNetWindow()
- void DrawSetupWindow()
- void DrawMainWindowLabels(float Voltage,float Current, float Power)
- char *ftoa(float f, char* buffer, int bufSize, int decimals)
- void IncrementAddress(Int32U KeyValue)

This chapter contains verification of the developed system, testing functionality, performance and concluding on the results.

4 Test and Results

4.1 Test

Before the implementation is tested with the application, it is verified that no interrupts are delayed and that computations are finished on time. This is done by toggling the output pins P0[11] and P0[19] in the interrupts and calculation routines and displaying the signals on an oscilloscope. This test was not initially a success due to long calculation periods. This was solved by simplifying the interpolation as described in section 3.1.4. After this adjustment the test proved successful.

The implementation was first tested with a frequency generator. The output was set to 3,3V_pp with an offset of 1,65V. Furthermore the output was set to high impedance. During testing one of the LED's on the board was set to switch on and off, to indicate the logical value of the output (input to the relay). As the frequency was altered the LED switched on and off within the normal operation frequency range with correspondence to the NORDEL standard. This indicated a good and precise resolution of the measured frequency and a correct response. Hence the test was a success. Furthermore this test setup was uses to verify the functionality of the GUI, in particular the dial gauge needle indicating the frequency.

Finally the implementation was tested using the analog board. During the entire test the light bulb was on as the analog board is operated by the power grid. However the connected laptop charger did switch on and off within the normal operation. This test provided verification of the measured voltage, current and power as well. The test was also used to verify the functionality of the web page.

Unfortunately there wasn't enough time to test the implementation with the Omicron test setup.

4.2 Conclusion

In this project a Frequency-Controlled Demand system has been implemented. The main functionality of the system is to measure the frequency, voltage, current and power of input signals and set output signals accordingly. The output signals control relays making it possible to switch connected appliances on and off. As described in the introduction, appliances are switches of as the frequency of the input signals decreases. By method is applied to stabilize the frequency of the power grid. In addition to the frequency control, a GUI and TCP/IP communication between the system and a PC has been implemented.

The development and implementation of the described application was split into three major part:

- Measurements and control
- User interface
- Communication

Regarding measurements, much effort was invested in producing as accurate results as possible. Interpolation was implemented to detect zero-crossing more accurately. However, this approach posed a problem when the GUI and measurement code was merged. The calculations simple took to long and could not finish before the ADC outputted a new set of measurements. Thus an approximation had to be implemented, sacrificing precision. The cutoff frequency of the low-pass filter was set to 100Hz to minimize damping of the input signals. However, this parameter is changeable in the GUI. As a standard value, 100Hz results in good measurements but if the user should be able to set the cutoff frequency manually, the damping of the signal would have to be taken into account.

The user interface has been one of the focus areas in the implementation. The LCD driver provided from IAR (LCD Demo) has been extended greatly, making it possible to use 8-bit compression of images. This has made several backgrounds in various menus possible. The GUI is implemented as an ”on-demand” event, only updating the GUI when its necessary. The implementation of the touch screen driver has resulted in good precision. This is achieved by storing sample history and detecting, when a finger (or stylus) is removed from the screen. When this occurs, samples are unreliable thus the most recent samples at release are discarded and the sample history is used to detect the location of the touch. The user is able to enter parameters to the system by selected the given parameter and increase/decrease the value using buttons. Furthermore a keypad of buttons makes it possible to enter IP address in the network setup menu.

The communication was implemented using the μIP stack. The module provided by IAR (Web Server Demo) was extended with the support of XML files and scripts. All measurements are stored in an XML file and is the basis of the entire communication. This approach was chosen because a further development of the application would be to enable the microcontroller to communicate with other appliances, making XML an obvious choice. The implemented web page reads the data from the XML file in a JavaScript. This script is also responsible for updating the web page with measurements and measurement history. A graphical representation of history was chosen since this is an intuitive way of reading it. All displayed values are averages over one second except the minimum and maximum frequency observed within a second. The average frequency history is plotted with the minimum and maximum frequency values, which gives a good visualization of the stability of the frequency.

A shortage of the implementation is that the normal operation reserve is switched on and off as the frequency varies. This is not ideal since the connected appliance is likely to be damaged. In future development of the implementation it would be necessary to implement some timing, switching off the normal operation reserve for a given time, e.g. 2 minutes, and afterwards keeping it switched on for a minimum of e.g. a few minutes, depending on the specification of the connected appliance.

A
GUI related appendices

A.1 8-bit encoding utility

```
/*
 ******************************************************
 *                                                    *
 *  8bit encoder for 32bit image data in hex format   *
 *                                                    *
 *                                                    *
 *  Usage:                                            *
 *   Insert your image data where indicated below.    *
 *   and compile.                                     *
 *                                                    *
 *   From your terminal do a:                         *
 *                                                    *
 *      ./converter >> outpufile                      *
 *                                                    *
 *   Author: David Bue Pedersen                       *
 *                                                    *
 ******************************************************
 */
#include <iostream>

static int palette[256];
static int nextPaletteIndex = 0;

unsigned int mainguiStream[] =
{
     //Insert 32 bit hex data as follows here: 0x00000000
};



/*
 ******************************************************
 *  Function: findPaletteIndex                        *
 *  Parameters: int rgb                               *
 *                                                    *
 *  Returns: int                                      *
 *  Usage:                                            *
 *   Builds palette and returns palette index         *
 *                                                    *
 ******************************************************
 */

```

```c
int findPaletteIndex(int rgb)
{

    for (int index = 0; index < nextPaletteIndex; index++)
    {
        if (palette[index] == rgb) return index;
    }

    if (nextPaletteIndex >= 256)
    {
        printf("Your source hex array contains too many colors.\n");
        exit(1);
    }

    palette[nextPaletteIndex] = rgb;
    return nextPaletteIndex++;
}

/*
 *****************************************************
 *  Function: main                                   *
 *  Parameters: None used                            *
 *                                                   *
 *  Returns: None used                               *
 *  Usage:                                           *
 *   Reads source hex data, 8 bit image date and     *
 *   generates palette. Then outputs image data and  *
 *   palette to terminal                             *
 *                                                   *
 *****************************************************
 */

int main (int argc, char * const argv[]) {



    for (Counter = 0; Counter < sizeof(mainguiStream) / sizeof(mainguiStream[0]); Counter++)
    {
        Color = mainguiStream[Counter];
        int index = findPaletteIndex(Color);
        printf("0x%02X,", index);

        if (!( (Counter+1) % 8)){ // Modulus giver hver 8'nde linje
            printf("\n");
        }
    }

    printf("\nPrinting Palette:\n\n");

    for (Counter = 0; Counter < sizeof(palette) / sizeof(palette[0]); Counter++)
    {

        printf("0x%06X,", palette[Counter]);

        if (!( (Counter+1) % 8)){ // Modulus giver hver 8'nde linje
            printf("\n");
        }
    }
    return 0;
}
```

A.2 List of functions added or expanded in the drv_glcd.c display driver

Added functions

```
void GLCD_WritePixel (Int32U x,Int32U y,Int32U PixelColor)
```

Sets the pixel at coordinate x,y on the screen to the color defined by PixelColor.

```
Int32U GLCD_ReadPixel (Int32U x,Int32U y)
```

Get the pixel color at coordinate x,y on the screen.

```
void GLCD_DrawGauge(Int32U Angle ,Int32U GagueColor,Boolean ForceUpdate)
```

Draws the gauge needle at the angle and with the color specified by the input arguments. The Angle argument is in degrees.

```
void GLCD_SetGaugeValue(Int32U milHz, Boolean ForceUpdate)
```

Updates the gauge needle from a deviation of the input parameter in respect to 50,000 mHz

```
void GLCD_ExpandPictures()
```

Expands the 3 GUI pictures from 8 bit arrays to 32 bit arrays stored in SD ram.

```
 static void LCD_SKIP_PIXEL()
```

Used for implementing transparent text labels. This function is used within GLCD_putchar and should not be called directly.

```
void GLCD_print_transparent (const char *fmt, ...)
```

Same as the native GLCD_print but with transparent background.

Modified functions

```
int GLCD_putchar (int c, Boolean Transparent)
```

- Added a Transparent option as input parameter as well as appertaining code.

A.3 List of functions contained within the TouchScr_Driver.c touch driver

```
Int32U Touch_Measure(void)
```

State swapping in-between ADC readings.

```
void Update_TouchScr(void)
```

Testes if the screen is touched. If this is the case, a reading series is started.

```
void Init_TouchScr(void)
```

Initialization routines for the touch screen. These include pin configuration and setting initial states.

```
Boolean TouchGet (ToushRes_t * pData)
```

Initialization routines for the touch screen. These include pin configuration and setting initial states.

A.4 List of functions contained within the UserInterface.c event handler

```
void Init_GUI()
```

Initialization routines for the GUI

```
Boolean Button_Hit(Int32U x, Int32U y, ButtonArea_t button)
```

Comparison of the touch coordinates with button areas as defined in the header op this module.

```
void DrawNetWindow()
```

Draws the elements of the net screen

```
void DrawSetupWindow()
```

Draws the elements of the setup screen

```
void DrawSetupWindow()
```

Draws labels on the main screen

```
char *ftoa(float f, char* buffer, int bufSize, int decimals)
```

Float to arithmatic convertion. This in order to use the tiny version of the printf() variants NB: Only handles positive floating point values.

```
void IncrementAddress(Int32U KeyValue)
```

Increment the active numeric field of either the IP address or subnet on the Net window.