Technical University of Denmark

Anders Bahnsen
Aleksander Gosk
Kim Rostgaard Christensen

31070

Hands-on mikrocontroller programmering

3 week project report, January 2010

DTU Electrical Engineering
Department of Electrical Engineering

Contents

Chapter 1

Introduction

This course in hands-on micro controller programming concerns the use of micro controllers applied in an intelligent energy system.

Chapter 2

Requirements analysis and specification

The ELSAM agreement states that the grid frequency at all times must be 50Hz. If the values goes lower, our device should turn off unneeded devices. ELSAM defines two types of loads; Normal operation reserve and Disturbance reserve.
The Normal operation reserve should off-load when the frequency drops 49.9Hz, and shut off equipment able to do so in 2-3 minutes or less, not crucial to production (e.g. heater or lighting).
Disturbance reserve mode is used when the grid frequency does not return to normal state. In our case we turn off a battery powered device (a laptop) and try to make sure we dont empty their batteries completely before getting the power back.

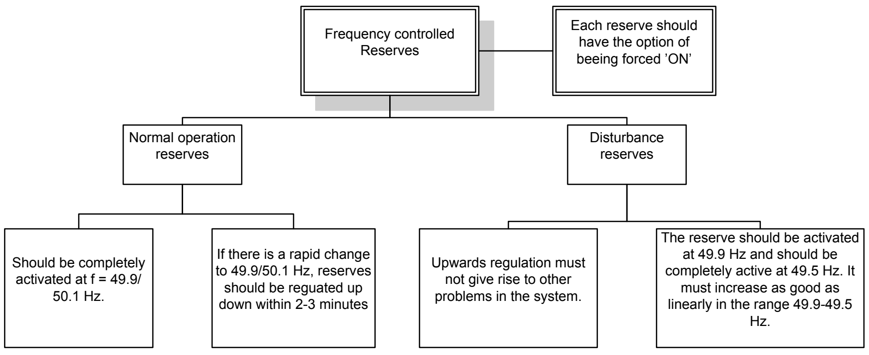
Frequency controlled Reserves
Each reserve should have the option of beeing forced 'ON'
Normal operation reserves
Disturbance reserves
Should be completely activated at f = 49.9/ 50.1 Hz.
If there is a rapid change to 49.9/50.1 Hz, reserves should be reguated up down within 2-3 minutes
Upwards regulation must not give rise to other problems in the system.
The reserve should be activated at 49.9 Hz and should be completely active at 49.5 Hz. It must increase as good as linearly in the range 49.9-49.5 Hz.

Figure 2.1: ANDERS WRITES THIS ONE

To be able to do more “intelligent” grid-off loading, we must be able to communicate with the outside world. As there is no protocol specified, the logical choice would be tcp/ip for transport, as there is is already a well-established global infrastructure using this. The data should be transferred in an easy parsable format, such as XML.

Users of the device should also be able to view the status, current configuration and make changes to the configuration as well. The status and configuration should be available both from a physical interface and via a remote interface. As out device has a touchscreen it will serve the purpose as a physical interface, and the remote interface should be done in html.

This leaves us with the following requirements:

- Detect grid frequency, and respond to changes
- Toggle relays based on frequency algorithm
- Implement a TCP/IP stack
- Implement a HTTP server
- Design human interfaces for local access
- Design human interfaces for remote access
- Design machine interfaces for remote access

2.1 Tools used

An application development process’ success or failure can depend largely on the tools used in the process. Good tools for documenting, debugging and versioning should be considered bare minimums.

2.1.1 Debugging tools and methods

When developing the communications interface we use the program “wireshark” [1] to verify http requests and responses.
The board we will be using contains a JTAG port, for interactive debugging. This means we are able to stop the processor, read and modify registers or memory. This feature is extemly useful when debugging an application.

[1] www.wireshark.org

2.1.2 Information scraping

For doing quick notes on random information concerning code, documentation, requirements or specifications. We will be using a wiki for this purpose

2.1.3 Versioning

When developing an application, a versioning system is very useful both for experimenting with new features, because you have the ability to quickly roll-back to a working version. It is also a great way to do backup on your project.

Chapter 3

Description of program

3.1 Program structure

The developed application is devided into three modules as seen in figure 3.1

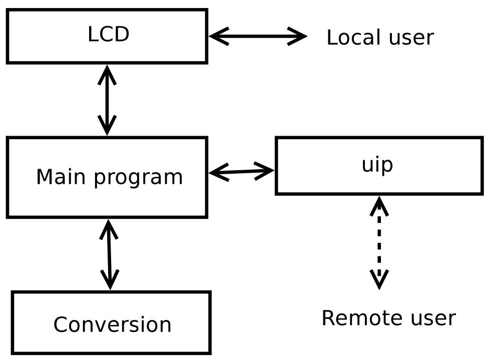

Figure 3.1: The overall program structure

The conversion part is responsible for most of the ADC, DAC, GPIO and interrupt handling parts of the program. The LCD is taking care of the physical user interface, both touchscreen and displaying values on the screen. The uip is accepting remote connections and serving data to remote clients.

The program runs roughly this sequence:

The first thing done is the initialization. This obviously only run once. It

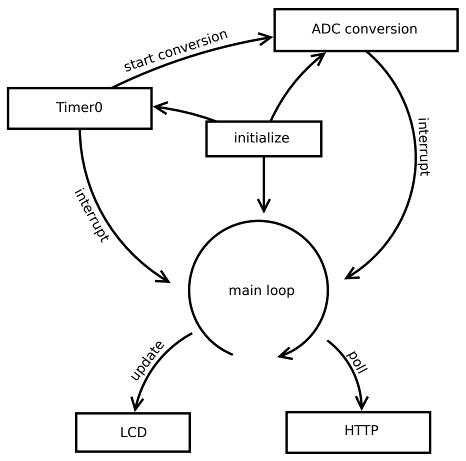

Figure 3.2: The overall program sequence

gets the interrupts up and running and configures the appropriate GPIO pins. When done, it enters the main loop, where the application remains for remainder of run-time, while not processing interrupts. The Timer0 runs at a predefined frequency, and takes care of starting the ADC conversion process

[INSERT TEXT]

The ADC interrupt takes care of [INSERT TEXT].

When not doing interrupts the main loop updates the screen, checks if a user has touched the screen and moves the cursor accordingly.
It also does periodic checks on the IP stack, sending and recieving packets at a resonably stable frequency. This is done via the HTTP server.

3.2 Measurements and control

3.2.1 Hardware resource allocation

3.2.1.1 ADC

To get the maximum number of samples per second, we use burst mode.

Burst is started by entering Timer0 interrupt

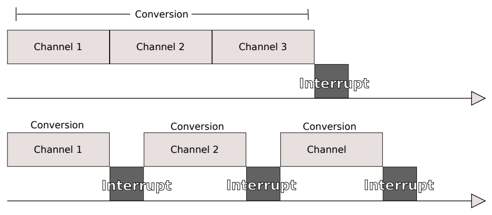

Figure 3.3: Illustration of the priciple behind burst mode

3.2.2 Measurement algorithms and their implementation

The values measured with the microcontroller (ADC) are:

- Grid frequency
- Grid voltage
- Grid current
- LCD voltege (checking if, and if yes then where, LCD screen has been touched)

3.2.2.1 Voltage and Current measurements

The voltage from the grid is passed through a circuit that lowers it's value to 0...3.3V range, before it can be processed by the microcontroller. The same thing is done with the current - it is converted to a voltage signal of the same scope. Both signals are scaled afterwards so the output of calculations is in grid values. Two different approaches were used for each of them.

Voltage scaling Firs the RMS and offset values of the grid voltage has been measured. The results were:

$$V_{RMS} = 236V$$

$$V_{Offset} \approx 0V$$

We can conver them into peak-to-peak values:

$$V_{p-p} = 2(V_{RMS} - V_{Offset})\sqrt{2} = 2(236V - 0)\sqrt{2} = 667.5V$$

Since the offset is approximately zero we can wrtie that maximum an minimum values of volatege are $\pm 333.75V$. The highest and lowest quantization levels accuired by ADC conversions were 1002 and 20. Hence the zero Volts was represented by $(1002 - 20)/2 = 491_{level}$ and $1_{level} = 333.75_{volts}/491_{level} = 0.679735234_{volts}$.

Current scaling A bit different approach have been used while calculating the scaling ratio in the case of the grid's current. Here we've assumed that the current converting circuit is very precise and that the output values are exactly 0....3.3V. Thus we can suspect that the quantization level corresponding to 0A is equal to 512. After aquring the raw samples of the measured current from ADC, substracting 512, calculating their RMS value and multiplying the result by $\sqrt{2}$ we got the current signal's digital representation's amplitude, which represents the real signals amplitude. Sclaing is done as in the case of the voltege from this point onward.

3.2.2.2 Frequency measurement

The frequency measurement is done by calculating the length of the period of a given signal. It is done by calculating the number of samples between the points at wich the signal is crossing an arbitrairly defined zero line with an assumed direction (rising edge or falling edge) and comparing it (number of samples) with the sampling frequency.
In order to detect zero-crossing two succeding samples have to be remembered by the system. If the previous sample was below the zero-level and the current one is above, than the system marks this time as a rising edge zero-crossing and starts to count the samples until the next one.
Due to significant disturbances a low pass filter has to be used, in order to smoothen out the data. Without it detecting false zero-crossings would be very common, which in turn would render the method completly useless.
The lowpass filter is used only for this particular purouse, though. All the other calculations, like voltage RMS value, power etc. are performed on raw data. That is due to the fact that the filters output differ significnatly from its input (the amplitude is attenuated and the signal is shifted in phase). On the other hand filters, by definition, never change frequency, which make them usefull in this particular case. This is shown on the Figure 3.4

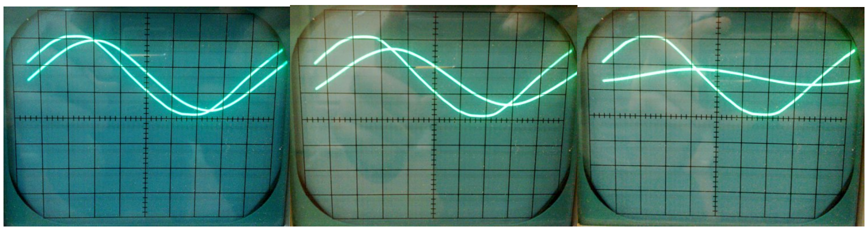
(a) Cut-off at 100Hz **(b)** Cut-off at 50Hz **(c)** Cut-off at 10Hz

Figure 3.4: Input and output signals of a low-pass filter at a sampling frequency of 10kHz for three different cut-off frequencies.

3.2.2.3 LCD measurements

The LCD measurements are conducted to check if the touch screen has been touched and if yes, where exactly did it happen. The possition of the touch is determined by measuring the voltage on the LCD's outputs. Measurement of the X and Y coordinates aren't done simultanously. If we want to measure the X coordinate the ADC and the pins have to be set to a certain state. In order to meassure the Y coordinate those set up options have to be different. Hence, every coordinate is meassured on every second ADC action.

3.3 User interface

The user interface's layout is presened below:
====[`PASTE A SCREEN LAYOUT`]==== It is build out of three different screens:

- Main screen
- Config screen
- Test screen

There is a panel's name lable at the top of each screen. There are also two buttons on it's both sides. They allow the user to switch between screens in a ring fashion (if one button is pressed continously, then the screens repeat their selves).

3.3.1 Main screen

The main screen is the default panel presented on the LCD. It shows the values of:

- $Voltage_{RMS}$
- Frequency
- Power
- State of relay 1
- State of relay 2

Apart from the above there are two bistable buttons, next to the information on the realy status. If they are on the system forces the relays to be switched on at all times. If not, the relays will be automaticly switched off if the grid frequency will drop below a certain level.

3.3.2 Config screen

This panel may depict some of the information on the configuration of the system [*IMPLEMENT IN THE uC*]:

- The cut-off frequency of the low-pass filter
- Sampling frequency
- ADC convertion mode
- Usages of the linearization function
- Relay switch off frequency
- Relay switch on frequency

3.3.3 Test screen

This panel may be used during the updating/debuging of the programme for displaying certain variables. Thanks to a special function (*enable_line_of_text()*) only two simple lines of code are required to display a given text line in a specified position on the screen, wich makes checking how certain variables change in the real time, fairly easy.

3.4 Communication

The communication part of the program is done by reusing the uip network stack. The built-in http server has the ability to both serve files and do some basic server-side scripting.
We plan on refactoring the server, making able to serve xml and xsl[1] (with the right content-type). Xsl will enable us to generate a single xml file and then transform it for the user to view via a browser. The xml document will be treated as a regulear html page, making the transformation transparent for the user.
The really nice thing about xsl is that we only need one source of information

[1]XML Stylesheet

for both the userinterface (webpage) and for automatically downloading done by programs interested in only the data. The programs will disreguard the xml stylesheet and only fetch the data, minimizing overhead.

3.4.1 Implementation

The first step in getting the webserver working as we intended was getting the stand-alone server accepting xml files. This was relatively easy, and so was getting the server to generate content dynamically. This was mostly done by copy-pasting, so we wont bore you with the details[OR PERHAPS WE WILL? MWUAHAHAHA!!] - the code is included.
One thing that did baffle us though, was the fact that the uIP http server strings was written as arrays of bytes. Very unreadable and pointless, as a string or a char array would been just as good.
Our next step was getting the generic server merged into our main project. The big challenge was that the uip server used a global timer, and not just any timer - the same timer we already used i our main project.
The migration was again done in two steps. First step was moving the httpd timer from timer0 to timer1, and check if that worked. Next step was merging the code of the two timers into one, and get the timing right. The change from timer0 to timer1 resulted in a very slow, but working, webserver.

The problem in merging the two timers was due to the fact that they ran at two different frequencies. One at 20MHz and another at 100Hz. As these was both defined in our config.h as respectively TIMER0_TICK_PER_SEC and HTTPD_TICK_PER_SEC we figured we needed to add a counter and do the following:

$$counter\% \frac{TIMER0\_TICK\_PER\_SEC}{HTTPD\_TICK\_PER\_SEC} = 0 \tag{3.1}$$

The % is the modulus operation. Upon true, the counter should reset and the http tick should increment. This implementation should make the http server insensible to changes in the timer0 frequency.

The webpage itself refreshes every 5 seconds by asking the browser to do a delayed redirect to the same page.

Upon studying our code more closely, we discovered our slowdown was caused by the LCD doing refresh every free cputime it could get its dirty hands on. After getting the refresh down to 5Hz the server performance improved significally. Later on in the coding process the webserver suffered another slowdown, which we have not been able to fix yet.

3.4.1.1 Historical values

Although not implemented, due to different focus, we have spent some time considering a possible implementation.
At first glance the buffer could be implemented using a linked list - but due to the fact that we do not have at memory manager we would at some point run out of free memory to store these these historical values. So, we could use a fixed size array of int pointers, and a pointer to the last element inserted. Then it should be possible to run through the array “array-size” number of times, starting from “last_inserted” and break if we encounter a null pointer.
The historical values should be displayed on a graph generated as an svg image, using JQuery to reload the values without reloading the page.

3.5 Chapter Summary

3.5.1 Communication

The finished result can be viewed by accessing the ip (typically 192.168.0.100) and requesting the data.xml document. When doing this from a browser the transformation works like a charma and gives us a graphical representation of our values. The server is still slow when we enable the LCD and touch-screen, but this should just a matter of opimizing the code that handles these.
The browser refresh is, agreed, a bit quick n’ dirty, but also works as intended. The xsl and xml files is with the other files in the uip/http-fs folder.
For communication via TCP/IP on an intelligent powergrid, we do not think that xml+http should be the way to go. The overhead would be too big in comparison to a binary protocol.

Chapter 4

Conclusion

In the beginning of the project we spent a great deal of time cleaning and refactoring code, making it more modulary and easy to navigate. But as the application evolved and grew in size and complexity, it became increasingly difficult to maintain a consistency in the code. Espeacially when thing were not doing as planned. New code arose in places where it did not belong.
From this we have learned about the importance of defining a clear program structure and functionality from the start of the project, and enforcing it thoughout the development process.

The webserver is responding slowly, but is very functional. The xsl transformations works like a charm in modern browsers and should definitly be considered in the case of an official http+xml implementation.

The LCD screen, acting as both in- an output, gives a quick way of getting an overview of the current status of the system. As well as modifying the system paramters realtime.
The values displayed are a bit off, due to programming errors. But it gives you an idea on how a finished system should look and behave.