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

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.

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

2.1.1 Debugging tools

When developing the communications interface we used the program “wireshark” [1] to verify http requests and responses.

2.1.2 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.

[1] www.wireshark.org

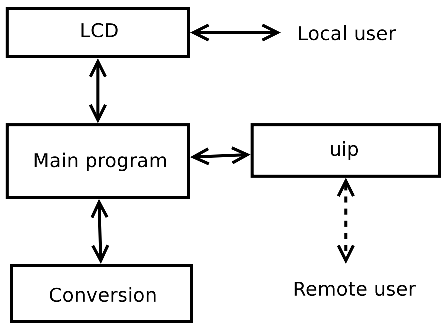
Chapter 3

Description of program

3.1 Program structure

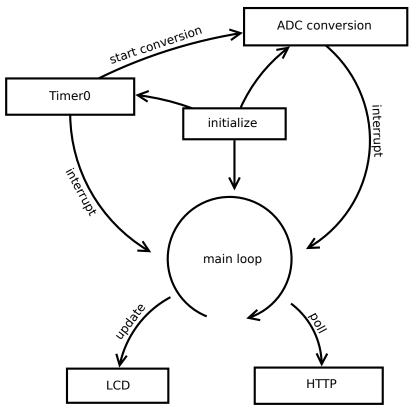
The developed application is devided into three modules.

- Conversion
- LCD
- uip



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 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 [INSERT TEXT]

The ADC interrupt takes care of [INSERT TEXT].

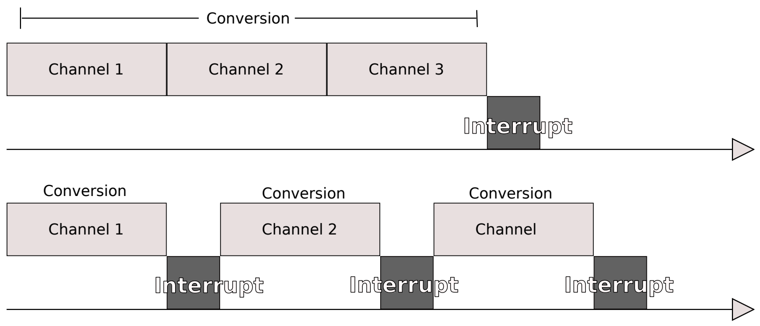
When not doing interrupts the main loop updates the screen, checks if a user has touched the screen and moves the cursor accordingly.
It also does periodic checks on the IP stack, sending and recieving packets at a resonably stable frequency. This is done via the HTTP server.

3.2 Measurements and control

3.2.1 Hardware resource allocation

3.2.1.1 ADC

We are using the ADC for two purposes: Detecting input from the touch-screen and measuring grid frequency. This takes up three channels (one for each dimension on the screen, and one dedicated to the frequency). To get the maximum number of samples per second, we use burst mode.



Burst is started by entering Timer0 interrupt

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 had to be passed through a circuit that would lower it's value to 0...3.3V range, before it could be processed by the microcontroller. The same thing had to be done with the current - it had to be converted to a voltage signal of the same scope. Both signals were scaled afterwards so the output of calculations would be 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 maximum and minimum number of bits after ADC conversions were 1002 and 20. Hence the zero Volts was represented by $(1002_{bits} - 20_{bits})/2 = 491_{bits}$ and $1_{bit} = 333.75_{volts}/491_{bits} = 0.679735234_{volts/bits}$.

Current scaling some text

3.3 User interface

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 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 do was getting the server to generate content dynamically. This was mostly done by copy-pasting, so we wont bore you with the details - the code is included.
The 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.

[1]XML Stylesheet

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.

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.

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 xsl file is with the other files in the uip/http-fs folder.

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.

CHAPTER 5

Appendix A