31070

Hands-on Microcontroller Programming

Introductory notes and Frequency-Controlled

Electric Vehicle Supply Equipment

Project Assignment

Edited by Yi Zong for Arduino platform
Section of Power-to-X and Storage
Division for Power and Energy Systems
Department of Wind and Energy Systems
Technical University of Denmark

Table of contents

Part A. Introduction

The First Rule of Program Optimization: Don't do it.

The Second Rule of Program Optimization (for experts only!): Don't do it yet.

— *Michael A. Jackson*

In the electric power system, there must be balance between production and consumption of electricity at all times. To establish this balance, generators are continuously adjusted to match variations in the load, or to handle contingencies. However, due to the kinetic energy of the rotating mass in the system's generators, an energy unbalance will not immediately result in a blackout. Due to the laws of motion, the system frequency will decay if there is an energy deficit, and vice versa. It gives the generators in the system a few seconds to adjust their production.

On the other hand, since the system frequency can be measured anywhere in the system, it is also possible to adjust the load in order to maintain the energy balance, based on the changes in frequency. This is often referred to as frequency-responsive demand, and has a currently unused potential to increase the reliability of the system.

The difference between frequency-responsive demand and traditional load-shedding that takes place in emergency situations, is that the frequency-responsive demand responds to much smaller frequency variations, and that it typically only switches off individual appliances, or even parts of appliances, to minimize the loss of comfort for the end-user.

Examples are the tumble dryer's heat element, but not the motor, or the refrigerator compressor, but not the light. Evidently such applications can only remain switched off for a very limited time (minutes) but that is often enough to allow slower secondary resources to take over. Another example of interrupt tolerant loads is all kinds of battery chargers (including UPS systems) that can potentially be switched off for longer periods.

In this course, you will develop an embedded microcontroller application that will monitor the power system's frequency, and take actions based on those measurements. In the following sections you can read about the hardware and software used in the course, and how the project assignment is divided into subtasks.

Hardware

The embedded system setup consists of two parts, one is the microcontroller Arduino MKR 1000, and another containing some sensors (temperature e.g.TMP36, phototransistor and tilt), actuators (DC/servo motor) and the input/output devices (push button/ LED or LCD), etc. The detailed information can be found here (https://store.arduino.cc/arduino-mkr-iot-bundle). To monitor/ measure signals an oscilloscope is needed.

ARDUINO MKR IOT BUNDLE

The microcontroller is an SAMD21 Cortex-M0+ 32-bit low power ARM MCU. It is with many built-in peripherals, such as **8**/10/12-bit ADC, 10-bit DAC, and a variety of serial channels, including Wifi. It runs at an internal operating frequency of 48 MHz. It has 32 kB on-chip SRAM (used for working memory during program execution) and 256 kB on-chip flash memory (stores program code and initialized data variables). The detailed information can be found here: http://docs.arduino.cc/hardware/mkr-1000-wifi

To download/upload and debug software to the microcontroller, a micro USB cable is used, and is controlled from the development environment (ARDUINO IDE).

Software (tools) (to download the latest Arduino IDE here :https://www.arduino.cc/en/software)

To program the microcontroller application, you can use Arduino Development Environment from local (your own laptop) or from Arduino Web Editor (online). To install the Arduino desktop IDE on different platforms (Windows, Linux, or Mac OS X), please find the guideline here: https://www.arduino.cc/en/Guide/HomePage

Software (microcontroller) structure

In this course it is the intention that you should program an application without an operating system on the microcontroller platform. The structure of such a program is first ***initialization*** of hardware (e.g. void setup ( ) function in Arduino) and variables, and after that ***an infinite loop*** that repeats different tasks such as polling a user interface. This loop is often referred to as the main loop (e.g. void loop ( ) function in Arduino) of the program. Tasks that are repeatedly carried out in the main loop will not follow any strict time schedule as they are simply carried out in a sequential order. Some tasks, like sampling an analog input signal, must be done with precise intervals. For this purpose, an ***interrupt*** is used, that interrupts the main loop, does whatever is needed here and now, and then resumes execution at the point of interruption.

Course structure

To get familiar with the software and hardware environment and the lab facilities in general, you start out with some simple tasks on the microcontroller. Then you continue with the project assignment that is divided into 3 major parts. At first measurements and control are combined in the basic frequency responsive application, and after that, the application is expanded with a graphical user interface (LCD), and a remote communication/control interface based on protocols commonly used on the internet.

It is recommended that you spend some time on all 3 parts, but of course there is a little freedom for you to emphasize the parts you find most interesting. In this way, you can twist the course contents to fit your personal interests for e.g. user interfaces, measurement techniques and signal conditioning, or internet technology.

Where to find help

A number of resources are used when programming the microcontroller system:

1. The microcontroller's user manual (on DTU learn). In this you will find important information about the built-in peripherals and how to use them. You can also read about the interrupt controller, memory maps and much more. Be aware that the manual covers several variants of the microcontroller. Be sure to use the information relevant for the SAM D21G.

2. There are many tutorials on YouTube that give a very good overview of the entire process, from programming your code until it runs in the microcontroller (**Note**: *most of them are using Arduino Uno, in our course, Arduino MKR 1000 is used*) . Please find relevant references on DTU learn!

3. Your favorite C handbook. Depending on your C programming skills, you will most likely need to use a handbook for reference. If you do not have a favorite already, consider Brian W. Kernighan and Dennis M. Ritchie's "The C Programming Language".

4. The Internet. Learn C - Free Interactive C Tutorial (https://www.learn-c.org/)

5. Schematics of the microcontroller board is provided for your reference on DTU Learn.

How to pass the course

In the end, you must submit a concluding report, and prepare a demonstration of your final project in the lab. The outline of the report should contain:

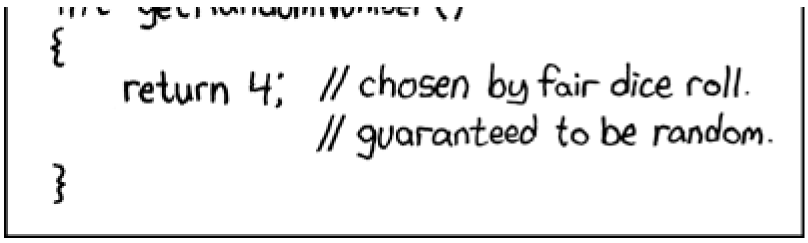
1. Introduction to the problem and its solution;
2. Requirements analysis and specification of the application, including all three parts (e.g. Measurement; Local user interface(LCD); Web User interface (data monitoring and remote control);
3. Description of program
   a. Overview of hardware resource allocation (timers, ADC and I/O etc.) including a hardware schematic diagram for your project setup.
   b. Overview/chart of program structure including the data flow between interrupt functions and main function;
   c. Overview/chart of program modules (e.g. initialization, register configuration, interrupt ……);
   d. Short description of the most important subroutines.
4. Test and conclusion

Mark each section or paragraph clearly with the author's name, and make sure to distribute the writing on all team members in a fair way (you will receive individual grades). One (and only one) team member must submit the report on DTU Learn under Assignments, along with a ZIP-file with the contents of your working directory (source codes, images and video files, etc.). The report can be in PDF (preferred) or Microsoft Word or Open Document formats. Its length must not exceed 20 pages. The report **MUST** be written in English.

On the last day of the course, a demonstration is required, the teachers will examine each group's project, testing the functionality based on the requirements that you yourself define in the report, including a Q&A section.

Tips

- Write useful comments while coding – do not wait until you are done.
- Utilize a version control system to manage changes to your project and aid collaboration. Please make sure to save all versions of your project!
- Learning by doing is main objective of this course. Please be patient when there are errors by double checking both your hardware and software!



```
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

source: xkcd.com

Getting started

In this section you will get started with the lab setup using an example application, and after that, program a very basic application of your own. This section is not part of the assignment and must not be documented in the report – consider it a good starting point for the actual assignment:

1. Complete the some simple projects to warm up. You can find them in the Document subfolder on DTU Learn.

2. You should now have a feeling of how the Embedded Workbench-Arduino IDE is used. Proceed with your own projects. To get a good starting point, start with the *Basics/Blink* example project to get an overview.

HINT: Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO it is attached to digital pin 13, on MKR1000 on pin 6. Variable_LED_BUILTIN is set to the correct LED pin independent of which board is used. If you want to know what pin the on-board LED is connected to on your Arduino model, check the Technical Specs of your board at: https://www.arduino.cc/en/Main/Products

Please follow the Arduino project book (on DTU learn/Document), to run at least 8 small projects. Among them, projects 1, 2, 3, 5, 8 and 11must be run in the first week!

3. After that, use a hardware/software timer to implement a 1 ms timetick (a variable that is incremented once every 1 ms). Make the timetick globally accessible (timeticks are commonly used to repeat tasks on a regular basis).

4. Use the timer to flash a LED. A flashing LED is the visible heartbeat of your system and will always let you know that your software is running.

5. Connect the DAC (AOUT) to the oscilloscope, and use the timer to output a 50 Hz sinewave on the DAC.
HINT: Read about DAC configuration in the microcontroller's User Manual (chapter 34)

Part B. Measurements and control

power

In this part of the assignment you will make a frequency measurement of the system, and control the some output (LED/ motor) accordingly.

The application must – to the largest possible extent – be compliant with Nordel's requirements for frequency-controlled reserves, as specified in the System Operation Agreement (on DTU Learn). The LED must act as a disturbance reserve, and a suitable device, e.g. a motor/other LED, can be connected to the switchable power outlet and act as a normal operation reserve. You must specify the requirements for both applications in terms of well-defined reactions to frequency steps and/or ramps of certain magnitudes and durations.

1. First, the hardware must be connected correctly before you start. Connect ADC channel with the oscilloscope which is used to generate 50 Hz sinewave (setting **Oscilloscope (0-3V), offset= 700-900 mv**).

2. The ADC can reach up to 350,000 samples per second (350ksps). Later on you can experiment with the sampling rate, but a rate of 10,000 samples per second is recommended to start out with. Set up the ADC to sample the voltage (channel 2) at this sample rate.
   HINT: Check if any of the example projects uses the ADC and get inspired. Read about ADC configuration in the microcontroller's User Manual (chapter 32).
   HINT II: Note the difference between the ADC clock and the sample rate (determined by a timer); these are completely independent.

*Note: Arduino **AnalogRead( ) function is too slow** and you need to find a solution to speed up the A/D Conversion to reach a high accuracy of frequency measurement.*

3. Output the sampled values on the DAC. Compare with the input signal. Are they alike?

4. To remove noise from the measured values, a first-order low-pass filter must be applied. In discrete time, the filtered value y is computed from the input x with:

   $$y_i = \alpha x_i + (1 - \alpha) y_{i-1} \qquad \alpha = \frac{\Delta T}{RC + \Delta T}$$

   where
   In this equation, ΔT is the sampling interval, and the time constant RC is originating from the analog implementation using a resistor and a capacitor. The relation between RC and the cut-off frequency f_c is given by:

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

Based on the above equations, implement a low-pass filter on the voltage measurement with a cut-off frequency at 50 Hz. Output the filtered value on the DAC instead of the raw samples. Compare input, output and comment.

5. Try to modify the cut-off frequency to 10 Hz and 100 Hz respectively. What happens? Set the cut-off frequency back to 50 Hz.

6. Change the sample rate to 20,000 Hz and verify that your results are still valid or not? Set the sample rate back to 10,000 Hz.

7. Based on the filtered voltage measurement, calculate the frequency based on a zero-crossing detection. Note that the ADC outputs values in the range from 0x0 to 0x0FFF.

8. To improve the accuracy, interpolate the signal around the zero-crossings. Verify the result using a frequency generator (**remember** to setup offset and amplitude **before** connecting it - **ask the teacher, *Vpp*** setting should be identical with a multi-meter in the lab).

9. Verify that no single interrupt takes so long time to execute that the next interrupt is delayed. One way to do this is to change the state of an IO pin upon entering and exiting the interrupt service routine, and then monitor this pin on the oscilloscope.

10. Use the frequency measurement to switch off one LED if the frequency drops below 49.975 Hz. Switch them back on if the frequency is above 50.025 Hz. It is a quite narrow gap, but it will let you see the light turn on and off during normal frequency variations of the power system. If the narrow gap causes problems, choose a larger interval e.g. 45.0 Hz and 55.0 HZ, and test it with an oscilloscope (wave generator). The final project should follow the grid code to develop the control strategies.

11. At this point you will have a basic implementation of a frequency-responsive device that helps stabilizing the system frequency by switching off load at low frequencies. You can choose to proceed with a user interface and/or communication interface before you move on with the last tasks of this part.

12 The sampled values represent the sinusoidal curves of voltage and current. In most applications of AC voltages and currents, the magnitude of a signal is described by its RMS value given by:

$$X_{RMS} = \sqrt{\frac{1}{T}\int_0^T (x(t))^2\,dt}$$

where *x(t)* is the signal and *T* is the period time of the signal. Similarly, the power calculated as voltage, *u(t)*, times current, *i(t)*, will also be a sinusoidal signal, but is most often described using the average, i.e.

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

Calculate real power, RMS current, and RMS voltage using the above equations, and implement scaling factors to convert the values to physical units (V). Supposing the hardware is designed so that 240 V, will utilize the full range of the ADCs, which is 3.3 V.

14. Verify the results. If necessary, calibrate the scaling factors of voltage and frequency.

15. Once again, verify that no single interrupt takes so long time to execute that the next interrupt is delayed.

16. The frequency-response implemented so far is quite simple, and is mostly suitable for switching off loads in case of disturbances. Look at the sections PWM signaling and Droop control on the following pages and implement proportional (droop) control for primary frequency regulation according to requirement for Frequency-controlled normal operation reserve, DK2.

17. The power system's requirements (from Energinet.DK) can be found in Document of DTU learn.

PWM signaling

PWM or Pulse Width Modulation is a modulation technique commonly used to encode a message into a pulsing square wave signal. A duty cycle is the percentage of one period in which a signal is active.

PWM signal on the Control Pilot (CP) line from EVSE to the EV has the following properties:

- Frequency 1kHz
- Duty cycle 10% to 96%
- Duty cycle to charging current limit relation is calculated from the graph shown in Figure 3

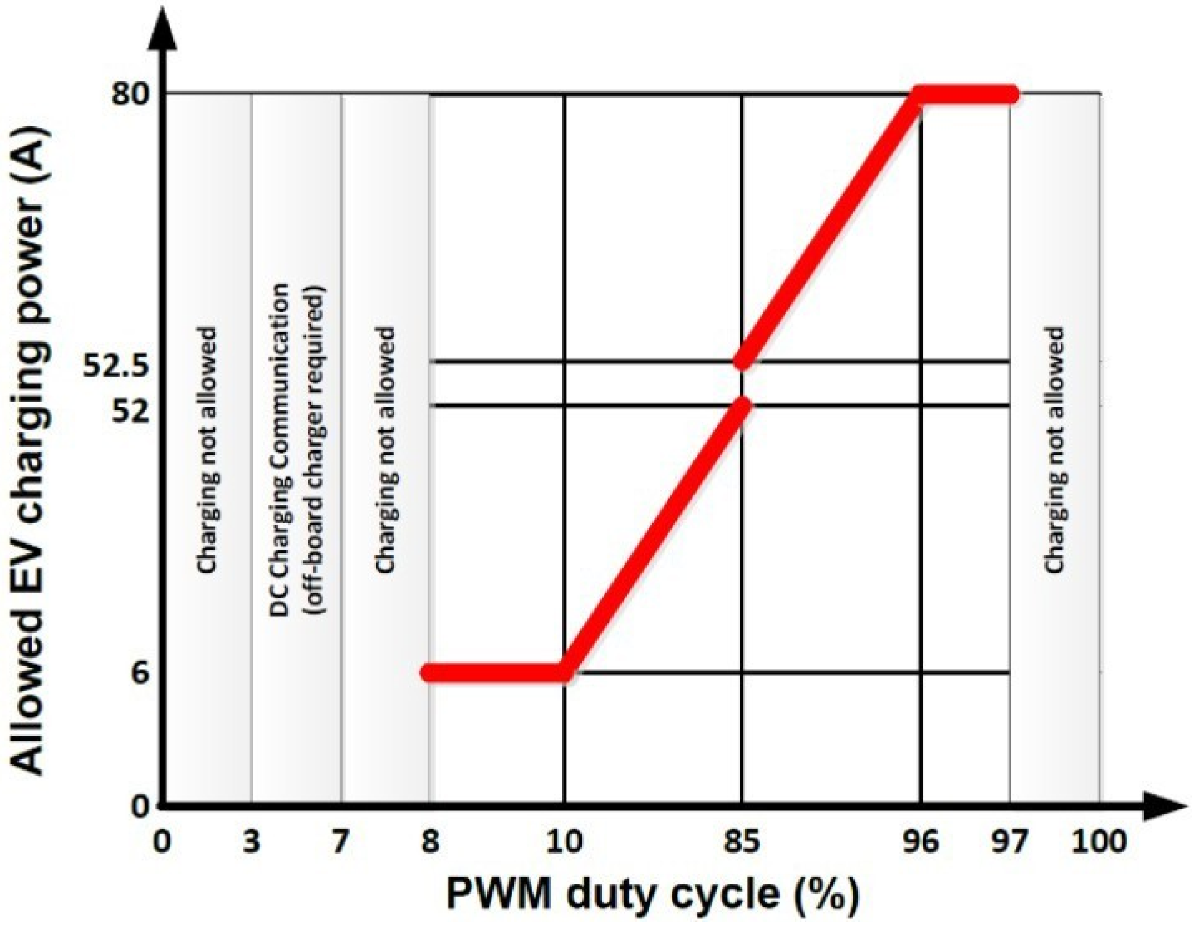


Figure 3: Relation between Control Pilot duty cycle and maximal charging rate as per IEC61851.

Reference:

https://en.wikipedia.org/wiki/Duty_cycle

Droop control

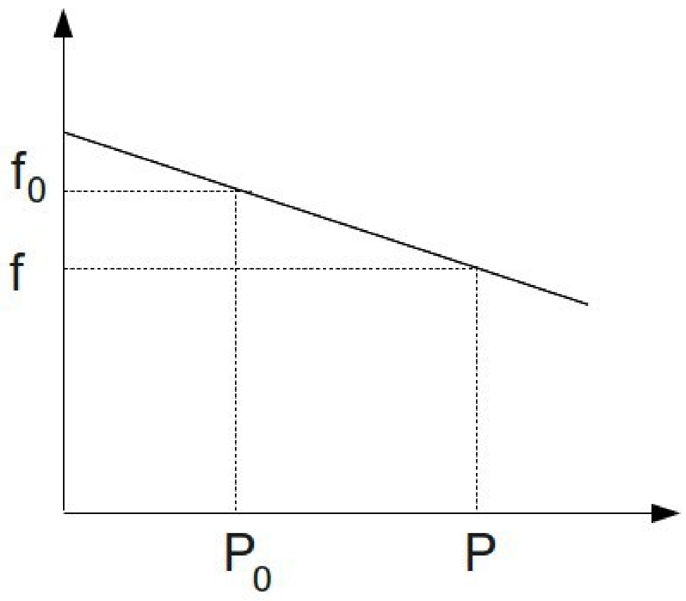
Droop control is a control strategy commonly used to control generators for primary frequency control (and occasionally voltage control). It resembles a simple proportional controller.

$$f = f_0 - k_p(P - P_0)$$

Where f is the system frequency

- f_0 is the base frequency
- k_p is the frequency droop control setting
- P is the active power of the unit
- P_0 is the base active power of the unit

Droop settings are normally quoted in % droop. The setting indicates the percentage amount the measured quantity must change to cause a 100% change in the controlled quantity. For example, a 5% frequency droop setting means that for a 5% change in frequency, the unit's power output changes by 100%. This means that if the frequency falls by 1%, the unit with a 5% droop setting will increase its power output by 20%.



Reference:

https://wiki.openelectrical.org/index.php?title=Droop_Control

Useful references can be found here:
Advanced PWM for Arduino Zero or any Atmel SAMD21 Based Arduino Board
https://www.youtube.com/watch?v=W6HSWKlGeVg

https://forcetronic.blogspot.com/search?q=Advanced+PWM+for+Arduino+Zero+or+any+Atmel+SAMD21+Based+Arduino+Board

Part C. User interface

In this part of the assignment you will extend your frequency-responsive device with a user interface. The user interface should provide relevant information to the user of the device, and possibly receive information about the user's intentions. Here, you only have a LCD to show your measurement of frequency and voltage.

Reference to ***Project 11*** (DTU learn – Document) or a tutorial here:
http://toptechboy.com/lesson-19-arduino-lcd-display/

Note: Be careful with Pins conflict for the final project integration (e.g. Pins for LCD & IoT)

Part D. Communication

In this part of the assignment, you will extend your application with a communication interface. It can provide remote users (humans) or remote control systems (machines) with information about the state of your device.

The reference project can be found here: https://create.arduino.cc/projecthub/133030/iot-cloud-getting-started-c93255?ref=search&ref_id=iot%20cloud&offset=0