

Motor Controller

June 14th, 2016

For this exercise, the SES boards will be equipped with a DC motor. Your task is to control and measure the speed of the motor.



This is a graded exercise. For further information regarding the evaluation criteria read the document *gradedExercises.pdf* carefully. The submission deadline is 2016-06-26 23:59 (CEST). The interviews will be held on Tuesday after submission, on 2016-06-28.

Your solution has to be committed to your team folder in the SVN repository by the specified time. We expect the following directories/files to be present in your root directory at <https://svn.ti5.tu-harburg.de/courses/ses/2016/teamX/> where X is your team's number.

- `ses` (contains your current ses library source/header files, including the new drivers)
- `task_6` (containing source/header files for the main project and optionally the challenge task)
- `readme.txt` (file; contains `<Name1> <Matr. number 1>, <Name2> <Matr. number 2>`)

Ignoring this directory structure will lead to point deductions. Make sure that all **header** and **source** files necessary to build and link your solution are present (you do NOT need to check in the compiled libraries!).




Keep the structure of those directories flat, i.e., do not use subdirectories there!

Task 6.1 : PWM Speed Control


The current delivered to the motor can be controlled by a transistor whose basis is connected to `PORTG5`. This pin is also the `OC0B` pin, which can be changed by the timer 0 hardware, e.g. on compare match. Thus, a PWM signal with a fixed frequency, but adjustable duty cycle can be used to control the motor speed. In the SES library create a file `ses_pwm.h` with the following interface

```
void pwm_init(void);
void pwm_setDutyCycle(uint8_t dutyCycle);
```

In the file `ses_pwm.c` implement the following

- Write a 0 to bit `PRTIM0` in `PRR0` to enable timer 0.
- Read Sect. 17.7.3 of the datasheet to understand the functionality of the fast PWM mode and select the respective mode in `TCCR0A` and `TCCR0B`.
- Disable the prescaler so that the timer is directly driven from the processor clock. **Think about why a high prescaler is unsuitable.** 
- Configure the timer registers to set the `OC0B` pin when the counter reaches the value of `OCR0B`.
- Implement the `pwm_setDutyCycle` to set the value of `OCR0B`.

Test your code by creating a new project `task_6`.

- At initialization, the motor shall be stopped.
- After pressing the button, start the motor by configuring a `OCR0B` value of 170. 
- A further press shall stop the motor again.

Task 6.2 : Frequency Measurement

Due to the three motor windings and the commutator, the motor current drops sharply 6 times per revolution¹. This current signal is filtered in the circuitry and for every current drop, a spike is generated at the `PORTD0` pin that can be used as external interrupt `INT0`.

In the SES library create a file `ses_motorFrequency.h` to provide an interface to measure the revolutions of the motor in Hertz with the following functions.

```
void motorFrequency_init();
uint16_t motorFrequency_getRecent();
uint16_t motorFrequency_getMedian();
```

- Similar to the button interrupt pins, a signal edge at `INT0` can generate an interrupt. Read Chapter 16 and configure an interrupt for every rising edge and toggle the yellow LED.
- For calculating the frequency, timer 5 shall be used.
 - ◆ Implement an appropriate timer configuration that fulfills the requirements for the following functionality. Choose a correct prescaler to **measure frequencies** down to 10 Hz.
 - ◆ Use the interrupt service routine of `INT0` to measure the time required for one revolution.
 - ◆ Implement a functionality that allows to recognize a stopped motor, e.g. by using the CTC mode and the timer interrupt. While a stopped motor is recognized the green LED shall light up and the get functions shall return a frequency of 0 Hz.
- Implement `motorFrequency_getRecent()` to return the most recent measurement in Hertz.
- In the `task_6` project, use the scheduler to display the motor frequency in revolutions per minute (rpm) on the LCD every second.



The current value of the timer can be read out from `TCNT5`. The timer can be reset by `TCNT5 = 0`.

Task 6.3 : Median Calculation

As you might notice, the results of `motorFrequency_getRecent` are quite unstable and many erroneous measurements occur. Therefore, the measurements should be filtered with a median filter. Implement the function `motorFrequency_getMedian` that shall calculate the median of the last `N` interval measurements.



The interrupt service routine shall be used to store the last `N` interval measurements in a suitable data structure. When `motorFrequency_getMedian` is called, calculate the median over these measurements and return the inverse in Hertz. Show the result in rpm on the LCD together with the result of `motorFrequency_getRecent`.



Find a good `N` to allow for a stable, but responsive measurement.



Pay attention to prevent data inconsistencies if the external interrupt fires during the calculation. Though, avoid long atomic blocks, so wrapping the whole median calculation in a single atomic block has to be avoided.



¹For details see <https://www.precisionmicrodrives.com/tech-blog/2011/06/08/using-dc-motor-commutation-spikes-to-measure-motor-speed-rpm>




Task 6.4 : Challenge

In task 6.2 you have used a fixed duty cycle to control the motor speed. However, due to production variances and different motor loads, this does not lead to a constant, predefined motor speed. In this challenge you shall implement a PID controller to **maintain a given motor frequency** f_{target} once the button is pressed, while measuring the current motor frequency $f_{current}$. As a first step, plot the trace of the frequency on the LCD to better see variations and oscillations. You can find a modified LCD library at StudIP that allows you to set single pixels of the LCD.

On a microcontroller a PID controller with anti-windup can be implemented by executing the following algorithm given as pseudo code in regular intervals.

$$\begin{aligned} e &:= f_{target} - f_{current} \\ e_{\Sigma} &:= \max(\min(e_{\Sigma} + e, A_w), -A_w) \\ y &:= K_p \cdot e + K_I \cdot e_{\Sigma} + K_D \cdot (e_{last} - e) \\ e_{last} &:= e \end{aligned}$$

The result y can be used to determine the duty cycle. Find appropriate values for K_p , K_I , K_D and A_w and initialize the controller properly.

-  The motor is responding very fast, so it is advised to start with a pure I controller ($K_p = K_D = 0$) until a good, but slow control is maintained. Then adapt the values to get a faster response.
-  Do not use **floating point operations**. Thus, you have to scale the calculations.
-  Use the UART for debugging the course of the variables.

