ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for Dynamic Systems and Control
IDSC
Institut für Dynamische Systeme und Regelungstechnik

151-0593-00 **Embedded Control Systems** (Fall 2017) **Lab 5**

Topic: **Interrupts, Timing, and Frequency Analysis of PWM Signals**

Pre-Lab due: 18. 09. 17 at 1 p.m. Post-Lab due: 18. 09. 17 at 5 p.m.

Name: ..... Forename: ..... Initials: .....

marianne.schmid@idsc.mavt.ethz.ch, 18. September 2017

1 Overview

In the first four labs, you have not dealt with time in the design of your code. For many applications (in fact, for almost all embedded control applications), time is an essential element. For example, sampling an input signal or generating a periodic output requires that your code runs periodically. To solve this problem, the MPC5553 provides several timers, all of which can be configured to produce interrupt requests (IRQs) at timed intervals.

In this lab you will learn how to set up the MPC5553's decrementer interrupt to periodically call your own C function. A function that is called in response to an IRQ is referred to as an interrupt service routine (ISR). You will write three ISRs. One of these will periodically sample a sinusoidal input and the other two will generate samples of a sine wave numerically. The samples of a sine wave, whether from an external analog input or internally generated in software, will be used to modulate the duty cycle of a PWM signal. You will then reconstruct the sine wave from the PWM signal using a low-pass filter to attenuate the high frequency content of the PWM, leaving only the (sinusoidal) modulation frequency.

2 Design Specifications

2.1 Hardware

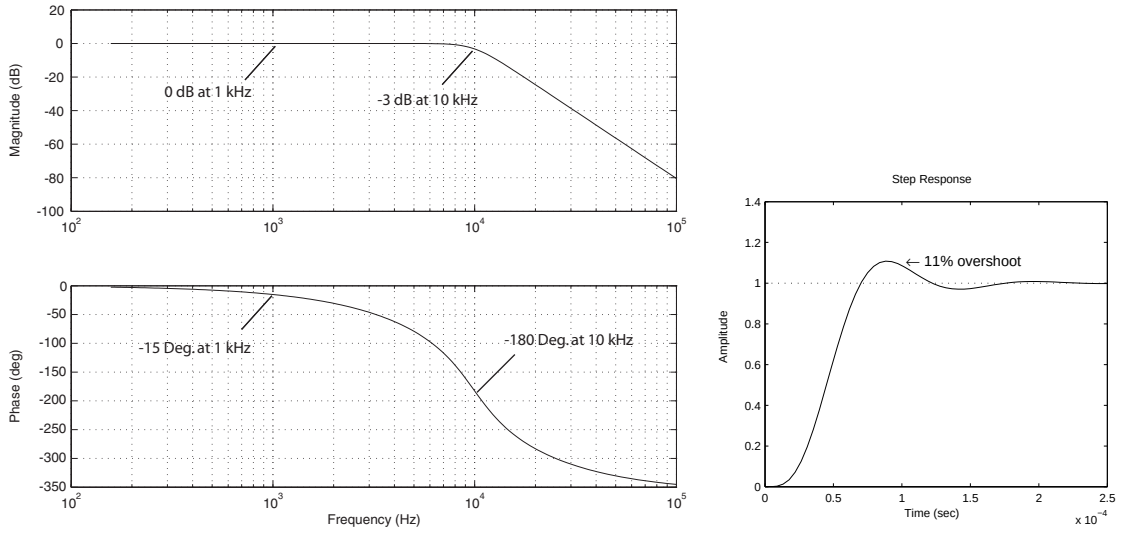
To generate an IRQ at a specific frequency, the decrementer (DEC) uses a 32-bit down-counter, generating an IRQ each time the counter reaches zero. A 32-bit register, called DECAR, contains the value that is automatically reloaded into the DEC counter after reaching zero. In this way, DECAR determines the period of the DEC counter. Because the DEC resets itself automatically, the IRQs are guaranteed to occur with a constant period and the ISR that is called in response to the IRQ is called at a fixed frequency. By adjusting DECAR, the DEC can generate IRQs over a wide range of frequencies. For more details about the configuration and operation of these timing registers, see Section 2.9 of the e200z6 PowerPC Core Reference Manual (see website).

In this lab, you will write three ISRs: IsrA, IsrB and IsrC, all of which will generate PWM signals where the duty cycle of the PWM encodes the values of a sine wave. IsrA will use the first channel of the eQADC module to sample a sine wave produced by a function generator. IsrB and IsrC will generate samples of a sine wave numerically, where IsrB will call the C library function `sin()` and IsrC will use a look-up table. The PWM output generated by each ISR will be sent through a low-pass filter, with frequency response shown in Figure 1a) and a step response shown in Figure 1b). The filter in the lab is a Butterworth fourth-order switched capacitor (sc) filter.

2.2 Software

qadc.h and qadc.c

The *qadc.h* file was given to you in Lab 3 and you completed the *qadc.c* file. The functions defined in *qadc.c* initialize and read analog-to-digital conversions.



(a) Frequency Response (b) Step Response

Figure 1: Frequency and step response of a 4-pole Butterworth low-pass filter with a bandwidth of 10 kHz.

mios.h and mios.c

The *mios.h* file was given to you in Lab 4 and you completed the *mios.c* file. The functions defined in *mios.c* initialize the eMIOS PWM function and set the frequency and duty cycle of the output. **For this lab, initialize PWM output on eMIOS channel 1, instead of channel 0.**

isr.h and isr.c

The *isr.c* code contains functions that configure the decrementer IRQ and initialize and enable the ISRs. The function `init_interrupts()` initializes the ISR frequency and allows you to specify your own ISR to handle DEC IRQs. The function `enable_interrupts()` simply enables the interrupts. Once the calls to these two functions are made, the infinite while loop in the main function will be periodically interrupted by calls to your ISRs.

isr.h and *isr.c* are already placed in their respective directories and were already used in Lab 3. You don't need to do any changes in these files.

lab5.c

The file *lab5_template.c* has been provided as a starting point for your *lab5.c*.

Makefile

Use the provided makefile to compile your code.

3 Pre-Lab Assignment

Pre-lab questions must be done **individually** and handed in at the start of the in-lab section. You must also, **with your partners**, design an initial version of your software before the in-lab section.

1. Carefully study the PWM generation section of the lecture, particularly the section on PWM frequency response. Write implementations of the ISRs A, B and C described below and insert them into the existing *lab5_template.c*.

   IsrA
   ISR frequency: 20 kHz
   Sine wave: 1 kHz, 1 to 4 volts, external input
   PWM: varies (see below)
   Procedure:

   (a) Turn on LED 29.
   (b) Read ANin0 analog input
   (c) Calculate duty cycle
   (d) Set the PWM duty cycle
   (e) Turn off LED 29.

   This ISR samples a 1 kHz sine wave produced by a function generator and regulates the duty cycle of a PWM signal to correspond to the measured voltage of the sine wave. Further, it will illustrate the significance of the PWM duty cycle and frequency range.

   You should set up your ISR such that the duty cycle is proportional to the measured voltage. The measured voltage of 1 volt should correspond to 10% duty cycle while 4 volts should correspond to 90% duty cycle. Further, when DIP switch 122 is high, the PWM frequency is 60 kHz and when it is low, the PWM frequency is 20 kHz. Similarly, DIP switch 123 should control the duty cycle limits. When the DIP switch is high, the PWM duty cycle should be limited between 40% and 60% and when the DIP switch is low, the duty cycle should be limited between 10% and 90%. You may need to change the software limits imposed on the duty cycle with the function `set_PWMDutyCycleLimits`.

   Also, include a line of code to turn on a GPIO LED at the start of the routine, and turn that bit off at the end of the routine. By monitoring that LED on an oscilloscope, you will be able to measure the frequency and run-time of your function by measuring the frequency and length of the pulse.

   IsrB
   ISR frequency: 1 kHz
   Sine wave: 100 Hz, numerically calculated by `sin()` function
   PWM: 60 kHz, 10% to 90% duty cycle
   Procedure:

   (a) Turn on LED 29.
   (b) Calculate `sin`(2*pi * i / 10), where i is incremented by 1 each invocation.
   (c) Set the PWM duty cycle
   (d) Turn off LED 29.

   Instead of sampling an externally generated sine wave, this ISR creates the sample points for a 100 Hz sine wave by calling the `sin()` function. Use `double` variables as argument for the `sin()` function. Because the ISR is running ten times faster than the sine wave that it is constructing, there will be 10 calls to your ISR for one period of the sine wave. In other words, each time the `sin()` function is calculated, the argument to the function should be incremented by 2*pi / 10.

   As before, the duty cycle should be set proportional to the sine wave, which is now between -1 and 1, so for a value of -1, the duty cycle should be 10% and for a value of 1 the duty cycle should be 90%. Initially, the ISR frequency should be set to 1 kHz, but later you will increase the ISR frequency. The 10:1 ratio between the ISR and sine wave frequency should still be maintained by your implementation.

IsrC
ISR frequency: 1 kHz
Sine wave: 100 Hz, *pre-computed* by `sin()` function
PWM: 60 kHz, 10% to 90% duty cycle
Procedure:

(a) Turn on LED 29.
(b) Read global look-up table to determine the PWM duty cycle based on a sine wave.
(c) Set the PWM duty cycle
(d) Turn off LED 29.

This ISR is nearly the same as IsrB except you will stream-line the code to produce a sine wave with the maximum possible frequency. To generate a sine wave at the highest possible frequency requires the shortest possible run-time for the ISR, but calculating `sin()` is computationally intensive. Instead of calling `sin()` every time your ISR runs, your program will pre-compute the values it needs once, store them in a global array and the ISR will look-up the appropriate value from the array. Be creative and try to find a solution that minimizes the run-time of the ISR. You will begin by setting the ISR frequency to 1 kHz and then you will increase the ISR frequency. As you increase the frequency of your ISR, the frequency of the sine wave should increase to maintain the 10:1 frequency ratio such that there are always 10 sample points per sine wave.

2. Use the frequency response (both magnitude and phase) in Figure 1 to plot or sketch by-hand the steady-state time response of the filter to an input sine wave at 1 kHz. Indicate both the input sine wave and the output sine wave and specify how the amplitude and phase change.

3. Make another time response of the filter but this time for 10 kHz. Indicate both the input and output in your plot and specify how the amplitude and phase change.

4. Suppose you have a low-pass filter with pass-band gain of 1, and to its input, you apply a 0-5 volt PWM with a 40% duty cycle and a frequency much greater than the cut-off frequency of the filter. What is the output of the filter?

5. Complete *lab5.c*.

4 In-Lab Assignment

1. Look at the main function within *lab5.c*. Edit the call to `init_interrupts()` so it uses IsrA. Make sure that the ISR frequency and the PWM frequencies are set correctly for IsrA.

2. Compile your lab5 code using the provided makefile.

3. Disconnect the motor encoder and the amplifier cable from the interface board. Power up the two boards.

4. Download your code to the processor and start it. Using the oscilloscope, measure the frequency of IsrA by measuring the frequency of LED 29. **Make sure that your function does not take longer than 50** μs.

5. Use a t-splitter at the function generator. Connect one end to the oscilloscope to set the appropriate voltage range and frequency (1 - 4 V and 1 kHz). By inspecting the oscilloscope plot check, that the voltage limits are satisfied. Then connect the other end to the ANin0 pin on the board.

   Connect the eMIOS1 and the ANout1 output of the board to the oscilloscope, so that you can display the input sine wave, the output PWM and the analog output signal simultaneously like it is shown in Figure 2.

6. Change DIP switches 122 and 123 and make sure that the PWM output matches what is specified in the IsrA specification. Check the output of eMIOS1 (PWM signal) and ANout1 (filtered signal) with the oscilloscope. Observe what affect the duty cycle range and PWM frequency have on the analog output. **Show the running code to the TAs and demonstrate the influences of the DIP switches on the output signals of the board.**

7. Make a sketch of what you see on the oscilloscope when the PWM frequency is 60 kHz and the duty cycle range is 40% to 60%.

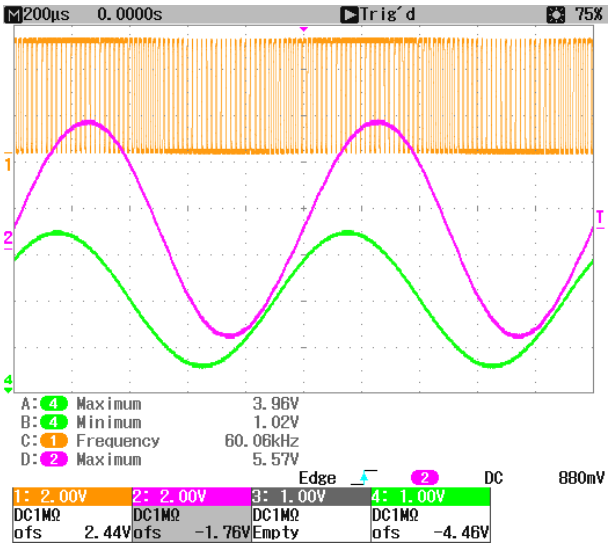


Figure 2: Input Sinusoid (Channel 4), Output PWM Signal (Channel 1), and low-pass filtered demodulated PWM Signals (Channel 2)

8. Modify *lab5.c* so that the DEC interrupts are handled by IsrB. Make sure that the ISR frequency and the PWM frequency are set correctly for IsrB.

9. Make another sketch of the filtered output.

10. Using the measurement functions of the oscilloscope, measure the minimal and maximal run-time of IsrB by measuring the time that LED 29 is set. What is the highest possible frequency of IsrB based on this measurement?

11. Find the highest frequency sine wave that can be generated by IsrB. Remember that the sine wave should always be 1/10 the frequency of the ISR, so modify the ISR frequency and be sure your ISR outputs 10 sample points per sine wave.

12. Display one period of the output sine wave and the output of the LED on the oscilloscope. You should notice that not all executions of the ISR take the same amount of time. **Show the oscilloscope output to the TAs.**

13. Try using `floats` instead of `doubles` in IsrB and repeat Steps 4, 9, 10 and 11. Note that the function name of the sine function is different for a double or a float argument: `sin()` for double or `sinf()` for float.

    4. Measure the frequency of IsrB by measuring the frequency of LED 29.
    9. Make sketch of the filtered output.
    10. Measure the minimal and maximal run-time of IsrB with doubles.
    11. Find the highest frequency sine wave that can be generated by IsrB with floats.

14. Modify *lab5.c* so that the DEC interrupts are handled by IsrC.

15. Repeat Steps 4, 9, 10 and 11 for IsrC.

    4. Measure the frequency of IsrC by measuring the frequency of LED 29.
    9. Make sketch of the filtered output.
    10. Measure the minimal and maximal run-time of IsrC.
    11. Find the highest frequency sine wave that can be generated by IsrC.

5 Post-Lab Assignment

1. Briefly explain your results from Step 6 of the In-Lab. Include a careful qualitative description of the frequency characteristics of the various signals you displayed on the scope, and how they matched (or didn't match!) what you expected to see. Based on what you saw in Step 6 why do we use a PWM frequency of 60 kHz instead of 20 kHz and why do we use a duty cycle range of 10% to 90% instead of 40% to 60%?

2. Explain any changes you observed when you changed IsrB to use `floats` instead of `doubles`. Under what circumstances would you want to use a `double` instead of a `float`, or vice versa, in an embedded systems application?

3. Explain your observations in In-Lab Step 12. Why would different executions of the `sin()` function take different amounts of time to execute? What problems would this cause, and how would you overcome these problems?

4. Explain the different range in maximum frequencies you observed between the three measurements you took. (IsrB with `doubles`, IsrB with `floats` and IsrC)

5. Include your sketches from In-Lab 7 & 9.

6. Document your code well and hand in your lab5.c.

If you have comments that you feel would help improve the clarity or direction of this assignment, please include them following your post-lab solutions.