Project A: Digital Data Acquisition System

Objective

This work aims at developing a digital data acquisition system, using the chipKIT Max32 platform, from Digilent. This system should acquire (read) an analogue signal produced by a bench top signal generator and produce an output, corresponding to some processing on the input signal.

Non-functional specifications

- The input signal will be acquired by using the PIC32 ADC peripheral.
- The analogue inputs will be protected against over-voltage.
- The analogue output will be generated by means of a PWM signal and a low-pass filter.
- The filter that implements the relationship from input to output should be defined as a C function that can be adapted as needed.
- The system will allow the definition of a set of parameters. These parameters are: i) sampling frequency; ii) ADC channel; iii) PWM frequency; and iv) Output Compare module used to generate PWM. These parameters are defined in the file `sys_config.h` that is provided. Your project must read the configuration from the provided `sys_config.h` file and strictly respect the syntax of the parameters indicated there. See Figure 1 for an example of a `sys_config.h` that configures the system for the parameter values defined in Table 1.

```
/* Analog to Digital section: */

/* Sampling frequency */
#define SAMPL_FREQ_HZ       50

/* ADC channel (0 to 15) */
#define ADC_CHAN             0


/* Analog Output section: */

/* PWM Frequency */
#define PWM_FREQ_HZ       2000

/* PWM channel  (1 to 5) */
#define PWM_OC_CHAN          1
```

Figure 1: Example of `sys_config.h`

Parameter	Value
Sampling freq.	50 Hz
ADC Channel	0
PWM freq.	2000 Hz
PWM output	OC1

Table 1: Example system parameters

Functional specifications

- The input signal voltage will be in the range [0 V, 3.3 V] (maximal range).
- The system sampling frequency will be at least in the range [20 Hz, 200 Hz].
- The system should generate a square wave with a frequency half of the sampling frequency (output changes state at every sample instant) in pin RA3.
- The base frequency of the PWM signal will be 2 000 Hz and the resolution will be no less that 100 steps.

Methodology proposal

The following methodology may help you when developing the system. It is taken for granted that the basic issues related to programming the chipKIT platform (installing the IDE and compiler, downloading and running code in the platform, ...) are solved.

1. Choose one of the available timers. Start by programming the timer to count at a slow rate, that you can check by watching one of the LEDs LD4 or LD5 going on and off. The timer should be working by polling, not interrupt (check the status of the interrupt flag in the main program and reset it when the counter reaches max count). Modify the counting rate and check that results are as expected (if you double the count, the frequency should reduce to half, and so on). This will give some grip on how to program the timer frequency.

2. Create a function that programs the required register values that control the timer. This will encapsulate all the register programming in a code block, and you will only have to provide the necessary input.
3. Program the ADC converter to read the input channel and produce the corresponding conversion value. Do it firstly by polling (no interrupts), at a slow rate (e.g. once per second) and print the ADC conversion value using the USART. Verify if it operates as desired by using diverse input voltage levels (e.g. through a potentiometer). The successful completion of this step assures a correct signal acquisition.
4. Join the timer and ADC input functions. Verify the correct operation. The successful completion of this step assures a correct signal acquisition at the required rate;
5. Refactor the code in the previous step to encapsulate it in a function to configure the ADC. Include the ADC channel as one of the function parameters, so you can choose which channel the system will use.
6. Create a function that sets the PWM module for generating a PWM signal with a frequency of 2 000 Hz. Create another function that accepts as input a Duty-cycle value. The PWM configuration should allow at least 100 steps for the Duty-Cycle.
7. Test the functions of Step 6 with several Duty-Cycle values. For that, examine the PWM signal with the oscilloscope. The successful completion of this step assures that a correct output signal is generated.
8. Refactor the code in steps 6 and 7 to create a function that configures one of the Output Compare modules. Include the identification of the OC module as an argument, so you can control where the PWM value will be generated.
9. Associate the ADC value (Step 4) with the Duty-cycle function (Step 6). The system should be working properly.

Demo project

A MPLab project with a complete solution is provided as a working example. This code contains the source code for the main file and all the modules are distributed as object files, together with the corresponding headers. Your job will be to create the equivalent to these modules.

The modules interfaces that are proposed in the example are just that: an example. You are not obliged to use the same interfaces and you can choose other options, whenever you find them more adequate. The only mandatory requirement concerning interfaces is that the system must read the configuration from the provided `sys_config.h` file and strictly respect the syntax of the parameters indicated there.

This demo project contains a few extras:

- The transfer function is defined by using function pointers. Changing the transfer function merely requires changing the value of a function pointer. This is a strategy used in several instances, such as callback functions.
- The user interface uses VT-100 codes to clear the screen at program start. VT-100 codes are used for cursor control and text formatting in text-based interfaces. Feel free to use them!
- The code is documented using Doxygen. You can find the Doxygen commands that generate the documentation in the header files. Have a look at Doxygen documentation in https://www.doxygen.nl/