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 1**

Topic: **Familiarization and Digital I/O**

Pre-Lab due: 11. 09. 17 at 3 p.m. Post-Lab due: 12. 09. 17 at 5 p.m.

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

marianne.schmid@idsc.mavt.ethz.ch, 11. September 2017

Before you start, please carefully read the **How to use VMware Player** document in the red folder.

1 Overview

The purpose of this lab is to familiarize you with the hardware and software used in Embedded Control Systems (ECS). For this class we will be using a 32-bit floating-point Freescale MPC5553 microcontroller running on an Axiom/Freescale MPC5553EVB development board. The goals of this lab are to:

1. Familiarize yourself with the lab hardware (MPC5553EVB board, the MPC5553 microcontroller, and the interface board) and its documentation. Specifically, you will use the documentation to program the MPC5553 for digital I/O. The MPC5553 Reference Manual is provided on the course website.

2. Learn how to write C code that performs low-level bit manipulation and writes to memory mapped registers, and compile the code for the MPC5553 using the CodeWarrior compiler.

3. Write a simple program that uses the digital I/O features of the MPC5553 and interface board, compile the program, and debug it using the P&E debugger.

4. Familiarize yourself with the Freescale MPC5553 header files available from the course website and use them to simplify programming the registers.

A good reference for C programming can be found at Dave Marshall's C programming page *http://www.cs.cf.ac.uk/Dave/C/CE.html* or in the book "C, a Software Engineering Approach" by Peter A. Darnell and Philip E. Margolis.

2 Design Specifications

2.1 Hardware

The interface board contains two banks of digital input DIP switches, LED displays, and digital output pins. In this lab we will be using the DIP switches to input binary values that will be added and output to the LEDs, and later we will be using the serial interface instead of the DIP switches to input the numbers.

2.2 Software

You will first run a simple test to make sure that you are able to read and write digital I/O values using the P&E debugger without running any code. You will then develop a C header file for the digital I/O registers and C code to add two 4-bit numbers specified on the DIP switches and echo the results onto the LED bank. Then you will use the Freescale header files to simplify your C code and eliminate the need to write your own header file.

mpc5553.h, typedefs.h

These files include all of the Freescale submodules, and declarations for the CodeWarrior compiler. **Do not modify** files in the Freescale directory.

serial.h and serial.c

These files provide support for serial communication.

ecs.h and ecs.c

These files provide initialization functions to set the processor speed, initialize the floating point unit and enable the motor in Lab 4 and Lab 6. You must call the function `init_ECS(labnumber)` with the current lab number to initialize the processor to a state suitable for the current lab. **This function call must be implemented in all the upcoming labs inside the corresponding c-file.**

makefile

This file contains configuration information for the CodeWarrior compiler and defines compile commands for the lab. **Do not modify** this file.

2.3 Organizing Your Files

Subsequent labs will require you to include many files in your compiler commands, so we have written makefiles to simplify your job. In order for these makefiles to work, you must arrange your files using a standard method.

You MUST organize your files in your working directory using the following subdirectories (You can download the lab files from the course website, they are already organized as below):

`WORK/freescale/` contains all Freescale-provided files. You **should not modify** any file in this folder.

`WORK/include/` contains all header files (*.h) for the ECS labs. Throughout the labs you will add header files to this directory for the libraries you write and edit.

`WORK/lib/` contains the library C files (*.c) you create in the labs and the output files (*.o) made by the compiler. Files in this directory will be used in multiple labs. For Lab 1, the *serial.c* and *ecs.c* files should be located here.

`WORK/lab1/` contains the .c files you write for your lab and the makefile. After you compile your .c files, this directory will contain .elf, .mot, .bin, .map and .o files.

You should create a new directory for each subsequent lab (lab2, lab3, ...) as needed.

3 Pre-Lab Assignment

1. Turn to Chapter 6 of the MPC5553 Reference Manual and read section 6.1.3 to familiarize yourself with the System Integration Unit (SIU). We shall use the SIU for General Purpose Input and Output (GPIO), although it has many additional features. The table in Section 6.2 describes the various pins that may be used to interface with external signals. Section 6.3 describes the memory registers associated with the SIU. Because the SIU configures the pins on the processor, it will be a fundamental tool we will need to set up interaction with our devices throughout this course.
   (a) Use section 6.3 of the manual to identify which SIU register we need to configure the use of a GPIO on the MPC5553. Find an expression which transforms an I/O pin number into the address of the corresponding pad configuration register (PCR). Each memory address refers to an 8-bit (Byte) of memory. Hence two bytes of memory are required to store each 16 bit pad configuration register.
   (b) Once a pin is configured for general-purpose digital output (GPDO), which group of registers will we use to set the pin state (1 or 0)? Find an expression that transforms an I/O pin number into the address of the appropriate GPDO register.

(c) Once a pin is configured for general-purpose digital input (GPDI), which group of registers will we use to retrieve the pin state (1 or 0)? Find an expression that transforms an I/O pin number into the address of the appropriate GPDI register.

2. Suppose you wanted to set up four consecutive output pins (pins 28-31) and four consecutive input pins (pins 122-125). Which 8 registers would you use to do so? Note that there are only a few bitfields that need to be set in each register: the pin assignment field, the appropriate input or output buffer field, and the weak pullup/down enable field. The latter should be disabled, because input from our interface board will not work properly if a pull up or pull down is present. What settings should these bitfields take in each case?

3. Suppose you wish to output the hex value 0xC to the 4 LEDs you initialized. What are the appropriate GPIO pin data output (GPDO) registers you would write to and what values would you write to them? By referring to Figure 1, we see that LED 31 should display the most significant, and LED 28 should display the least significant bits of the binary representation of 0xC.

4. Review the union C declaration discussed in class. Write a C header file using union to describe the register fields needed to configure the SIU pads. Name the file `lab1.h` and place it in the Lab 1 folder. Here is an example union, to remind you of the syntax. **Note**, the volatile specifier is needed, because the union is used for device registers:

```
typedef union {

   /* This allows access to all 16-bits in the register */
   volatile unsigned short REG;

   /* This structure allows access to the individual bytes of the register */
   struct {
      volatile unsigned short UPPER:8;   /* access to the top 8 bits */
      volatile unsigned short LOWER:8;   /* access to the bottom 8 bits */
   } BYTE;

   /* This structure splits apart the different fields of the register */
   struct {
      volatile unsigned short :2;        /* indicates 2 unused bits in the register */
      volatile unsigned short FIELD1:8;  /* access to the 8-bit field named FIELD1 */
      volatile unsigned short FIELD2:6;  /* access to the next 6-bit field */
   } FIELDS;

} EXAMPLE_REGISTER;
```

   You need to provide one union **generic enough** to suit any SIU pad.

   (a) Set up the bit field to allow for easy access to individual fields of a configuration register for a pad. In lab we will need to set up multiple pads.

   (b) Does each SIU pad configuration register have the same bit fields and bit field sizes?

5. In the lab we will need the 8 input pads 122-129 and 5 output pads 28-32. **Write a main function in C that declares a pointer of the type of the union you wrote for the configuration registers, and uses it to set up each of these pads for input or output.** Because there are (over 200) individual configuration registers, you must address each configuration register correctly by declaring your union as a pointer and setting the address, see 6.3 Memory Map/Register Definition in the Reference Manual. You may declare many pointers to set each I/O pad, or you may find it easier to review pointer indexing (ie: `padptr[122]`) and declare a single instance of your union to more easily access a specific configuration register in memory. Here is an example of declaring a pointer to a union and setting its address:

```
void main()
{
  volatile EXAMPLE_REGISTER* myconfig;       /* Pointer to a EXAMPLE_REGISTER union,
                                              * which we defined earlier */
  myconfig = (EXAMPLE_REGISTER*)0x0AF8C134;  /* Sets the memory location of the
                                              * target pad register */
  (*myconfig).FIELDS.FIELD1 = 0x5;           /* Sets FIELD1 of the register to 0x5 */

  /* Note: the previous line only sets the register at address 0x0AF8C134
   * How would we set the fields of another EXAMPLE_REGISTER register?   */

  /* The rest of the program... */
```

6. Freescale has provided header files that access the registers on the MPC5553. Now use the declarations in the file *mpc5553.h* to set the configuration registers needed for digital I/O. **Write a few lines of code to show how to set up I/O pads now using the union structures from *mpc5553.h*.** Use the same pads from question 5.

7. Suppose that you are given a 16 bit register (not necessarily one of the SIU registers) whose bits were numbered left to right as in the Freescale MPC5553 manual (0 msb, 15 lsb). Without the use of a union structure, how would you read bits 2-6 of this register and express the result as an 8-bit binary number whose least significant bit (lsb) is equal to bit 6? **Write a few lines of code in C to show how this is done.** See the section on "Low Level Operators and Bit Fields" in Marshall's C reference page for details. Note the bitwise operators in C for AND, OR, NOT, and shifting.

You will need to use some of your pre-lab solutions as the basis of your in-lab work. Because your pre-lab solutions will be collected at the beginning of the lab meeting, please make a copy for yourself. You may find it useful to type and retain an electronic copy to save time.

4 In-Lab Assignment

Throughout the laboratory we will be using an interface board with the MPC5553EVB. The purpose of the interface board is to provide you easy access to the signal channels you will need in the laboratories, and provide a safe buffer between your input signals and the microprocessor. The interface board includes 16 DIP switches and 16 LED lights, connected to the GPIOs on the microprocessor as shown in Figure 1.

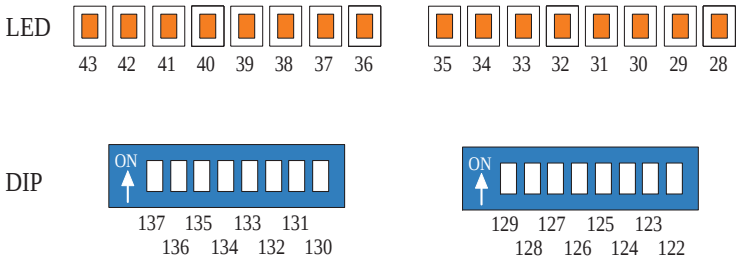

Figure 1: Enumeration of pad connections to hardware input and output devices.

4.1 Debugger

1. Power up the evaluation and the interface board with the on/off switch at the MPC5553EVB. Start ICDPPCNexus In-Circuit Debugger on the computer. Initialization code will scroll by, and memory addresses with assembly instructions next to them will appear in the Code Window.

2. The debugger provides the ability to view and set the MPC5553's many configuration registers. System registers can be viewed and modified by clicking the button with the Books icon at the top right of the debugger window, or by typing the command "R" in the status window of the debugger. Open this registers tool and navigate to the MPC5553 SIU. Within the SIU you should find many registers, including the configuration registers you encountered in the pre-lab. Double-click on any register to read or modify their values. Set the appropriate configuration registers to accept inputs from 2 bits using pads 122 and 123, and output to another two bits using pads 29 and 30.

   In the same SIU window, check the GPDO registers for the output pads to see if the data on the screen matches the LED display. Change the values in the GPDO29 and GPDO30 registers and see if your LED display changes. You will have to close the GPDOxx register window and click Yes when prompted to actually update the register on the processor before you will see changes on the LEDs. Now change the values on your DIP switches. To see if they changed the register values in the debugger, open the GPDI registers for those pads one at a time (GPDI122 and GPDI123) and note the data field. Before you will see a change in the input, you will need to close and open the window for the GPDI register to refresh the debugger with the current values of the input pads.

4.2 4-bit Adder: Your Header

3. Write a simple main program in C (*lab1.c*) based on your pre-lab code that uses your header file union to add two 4-bit numbers specified on the DIP switches 122-125 and 126-129 and echo the results onto the LED display. Place a loop in your program so you do not have to keep re-executing the program from start whenever you change the DIP switches. Remember to map the unions to the memory addresses that you looked up in the MPC5553 Reference Manual.

4. To compile your program, open a command shell (cmd.exe), navigate to the `lab1` directory containing your source code, and type `gmake`. Any compile errors will display at the prompt. Fix any errors and recompile before proceeding.

5. In the debugger, go to "File... Load Object/Debug File" and select the *lab1.elf* file generated by the compiler in your lab1 directory. You can run your code in the debugger program by either clicking the button with a solid green arrow labeled HL SOURCE, or by running the command "HGO" in the status window. Once your code is running the source window will go blank. Run the program and toggle the DIP switches and see if your program does what it is supposed to do.

4.3 4-bit Adder: Freescale Header

6. Write a program *lab1m.c* that performs the same operation as *lab1.c*, but uses the Freescale header files instead of the union you created in *lab1.c*. Then compile the program using `gmake lab1m` and confirm that it works in the same way as *lab1.c*.

4.4 Communicating with the Computer

7. Make a copy of *lab1m.c* and call it *lab1h.c* (you will be re-using most of its functionality in this question). Here you will be using the serial interface program *TeraTerm* in Windows to communicate to your program with the keyboard. Your program should continuously check for keyboard input and, if the key is a single-digit number, add it to the last number entered. Display the sum of the last two inputs on the interface board LEDs.

   Here is some code to get you started using serial communications in your program: (include it in *lab1h.c*)

```
char byte_in;
int finished=0;
init_ECS(1);
/* In all labs, we use ESCI_A that corresponds to port 1
 * and a baud rate of 115200 */
init_COM(1,115200);
serial_puts(1,"\n\n\n\n\n\n\n\n\rSerial Output Enabled.");
while (!finished) {
   if (serial_readyToReceive(1)) {
      byte_in = serial_getchar(1);
      switch (byte_in) {
         case '0':
            /* insert your code here */
         break;

         case '1':
            /* insert your code here */
         break;

         /* case 2 - 9 ... */
      }
   }
}
```

The functions above are defined in *serial.c* and *serial.h*, which must be compiled with *lab1h.c*. Be sure to include the serial header file (`#include <serial.h>`) from your *lab1h.c*.

Command to compile your code: `gmake lab1h`

Tips on using *TeraTerm*: Before running your code, open *TeraTerm* from the desktop. Check that the port and the baud rate correspond to the COM interface of your computer and your code. When debugging the code *TeraTerm* must be in focus.

5 Post-Lab Assignment

1. Suppose we have a 16 bit register with bits numbered left to right as in the Freescale MPC5553 manual (0 msb, 15 lsb). Set bits 4-8 to the pattern 0b11101, without changing the other bits in the register, by using:
   (a) A union
   (b) Bit masking and shifting
2. The Freescale headers make extensive use of the `volatile` storage-class modifier, defining many variables as `volatile unsigned int`s (`vuint32_t`, defined in typedefs.h), for example. Explain the purpose of the volatile keyword and the effect its absence in the header files would have on your program.
3. Include well-documented printouts of your header file, your *lab1.c* file, your *lab1m.c*, and your *lab1h.c* file.

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.