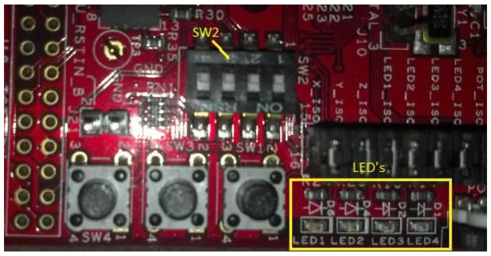
Spring 2013 - CSE 325: Embedded Microprocessor Systems

Project 3: Programming the GPIO port and the DMA timer

Points: 10, Due Date: Feb 18, 2013

On the TWR-MCF52259 microcontroller board (hereafter referred to as the "board") locate four LED's labeled LED1, LED2, LED3, and LED4. Also locate a 4-switch DIP switch labeled SW2.



With the board oriented as shown in the picture, the DIP switches are 1 (on the far right), 2, 3, and 4 (on the far left). The LED's are LED1 on the far left and LED4 on the far right. If the switch is moved down, the switch is on (closed), and if the switch is moved up, the switch is off (open). Here are the requirements for the project.

Your program shall initialize the LED's on the microcontroller board so at the beginning of the program, all four LED's are off. It should then drop into an infinite loop where the LED's are flashed in either a right-to-left sequence or a left-to-right sequence (the default starting sequence is left-to-right),

Right-to-Left Sequence	**Left-to-Right Sequence**
LED4 is turned on	LED1 is turned on
Delay 250 ms	Delay 250 ms
LED4 is turned off	LED1 is turned off
Delay 250 MS	Delay 250 MS
LED3 is turned on	LED2 is turned on
Delay 250 ms	Delay 250 ms
LED3 is turned off	LED2 is turned off
Delay 250 MS	Delay 250 MS
LED2 is turned on	LED3 is turned on
Delay 250 ms	Delay 250 ms
LED2 is turned off	LED3 is turned off
Delay 250 MS	Delay 250 MS
LED1 is turned on	LED4 is turned on
Delay 250 ms	Delay 250 ms
LED1 is turned off	LED4 is turned off
Delay 250 MS	Delay 250 MS

Sequence repeats *Sequence repeats*

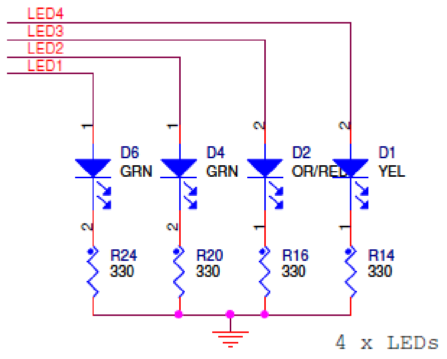
As the led sequence is being displayed, DIP switch 2 shall be periodically polled. If the program detects that DIP switch 2 has been moved to the off position, then the program shall reverse direction. When DIP switch 2 is moved back to the on position, the program shall resume the original direction.

DIP switches 3 and 4, shall alter the delay between turning on and turning off the LED's. These switches shall be periodically polled while the lights are being displayed.

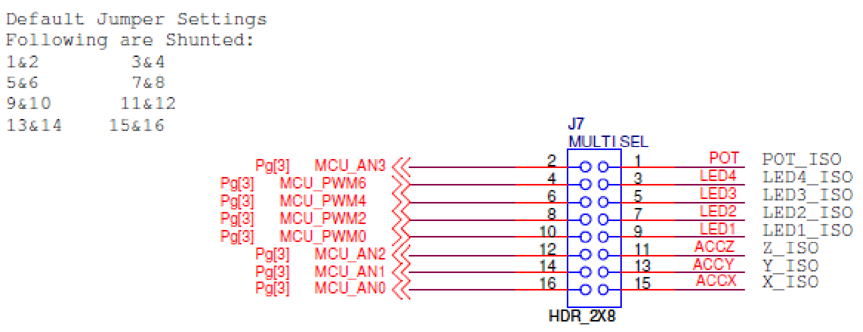
Sw 3	**Sw 4**	***delay***
Off	Off	250 ms
Off	On	350 ms
On	Off	450 ms
On	On	550 ms

Introduction

Look at the TWR-MCF5225X Board Schematic. Turn to page 5 and locate the 4 LED's.

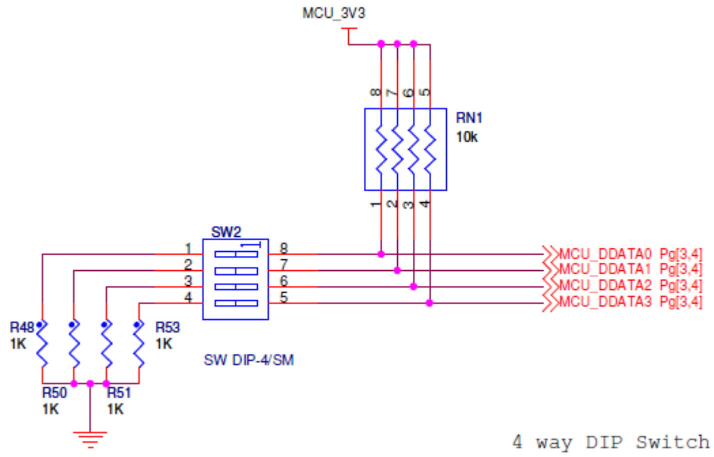


Note that one lead of each LED is connected to ground—through a current-limiting resistor. The other end of each LED is connected to signals labeled LED1, LED2, LED3, LED4. Those signals travel to jumper J7 pins 9, 7, 5, and 3, respectively. Each signal passes through jumper J7 (if the jumpers are on—as they should be) coming out the other side on pins 10, 8, 6, and 4 which are labeled as signals MCU_PWM0, MCU_PWM2, MCU_PWM4, and MCU_PWM**6**. The schematic tells you that these signals are routed to page 3 of the schematic.



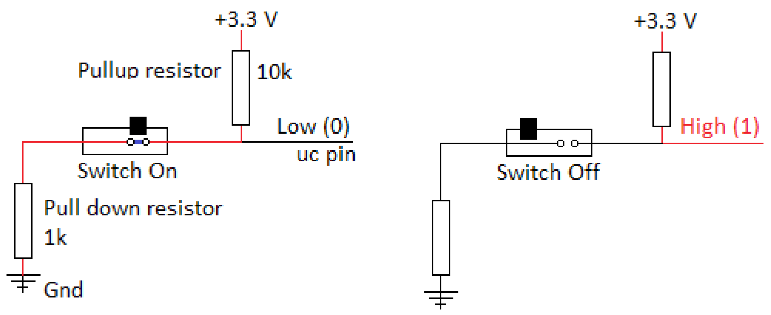
Peripheral Select Jumpers

Turn back to page 5 of the board schematic and locate the DIP switch labeled SW2



4 way DIP Switch

Notice that one lead of each switch (switch 1 is connected to pins 1 and 8, switch 2 to 2 and 7, ..., and switch 4 to pins 4 and 5) is connected to +3.3 V by a pull-up resistor; the other lead of each switch is connected to ground through a pull-down resistor. The DIP switch operates this way,



Moving one of the switches to the On position makes a connection between the two terminals (indicated by the small circles) thus connecting the +3.3 V signal to ground through the pull-down resistor; therefore, the microprocessor will see a logical low (0) on the input pin. Moving one of the switches to the Off position breaks the connection between the two terminals and the current flowing through the pull-up resistor cannot flow to ground so the microprocessor pin will be at a logical high (1). The purpose of the pull- down resister is to provide enough resistance to overcome the weak internal resistor (inside the microcontroller) connected to the input pin, thus pulling the pin low. Therefore,

Switch in On position => 0 on microcontroller pin
Switch in Off position => 1 on microcontroller pin

Now, let's see where these signals MCU_DDATA0, MCU_DDATA1, MCU_DDATA2, and MCU_DDATE3 go (I'll give you a hint: they are connected to four pins on the MCF52259). Notice that the schematic tells you that the signals travel to pages 3 and 4. Turn to page 3 and locate the MCF52259 chip labeled U2A. Look at pins 83–86, each labeled with a signal MCU_DDATA0...MCU_DDATA3 respectively. This tells you that the DIP switches 1–4 are connected to pins 83–86, respectively, of the ColdFire microcontroller.

ColdFire General-Purpose Input/output (GPIO) Module
You will be using the GPIO module for this project; you should study the table 2-1 in chapter 2, as well as, Chapter 15 in the Reference Manual to determine what registers to set and to what values. This information connects to the schematic information you just were walked through. **It's extremely important that you begin to develop the skill of reading and understanding the IMRM because you are unlikely to pass this course if you do not. We will be using this document extensively in this and all future lab projects. By the way, your other best friend in this course is the CodeWarrior debugger. You *have* to learn how to use it, it's not that tough.**

ColdFire DMA Timer (DTIM) Module
DMA is an acronym for Direct Memory Access. DMA is a technique which allows devices (other than the processor) to directly transfer data to or from memory, without processor intervention. This is very useful because while the transfer is taking place the processor can be performing other tasks, some of which may be more important.

The MCF52259 has a DMA module which includes the DMA controller along with four high resolution timers, referred to as DTIM0, DTIM1, DTIM2, and DTIM3. Each timer contains a 32-bit counter which counts up from 0 to some reference count. When the timer matches the reference count an interrupt can be triggered. We will not be using interrupts in this lab project, but we will in future lab projects. For this lab project we need to accurately delay for the delays shown on page 2 of this document. Chapter 26 of the reference manual will contain the information you need to program this timer.

Project
Your project must consist of nine source code and header files,

main.c	Contains, well, um, the, um, main function.
dtim.h	A header file containing public declarations for stuff in dtim.c.
dtim.c	Contains functions for initializing and using the DMA timer DTIM0.
gpio.h	A header file containing public declarations for stuff in gpio.c.
gpio.c	Contains functions for initializing and using the GPIO ports.
uc_dipsw.h	A header file containing public declarations for stuff in uc_dipsw.h.
uc_dipsw.c	Contains functions for initializing and using the microcontroller board DIP switch.
uc_led.h	A header file containing public declarations for stuff in uc_led.c.

uc_led.c Contains functions for initializing and using the microcontroller LED's (LED1, LED2, LED3, and LED4).

<u>Your project files will contain the following functions; you will need to fill in the details to make this program work. You should not require any additional functions or files for this project.</u>

main.c

- static int *dipsw_delay_poll*()
  Polls DIP switches 3, and 4 to see what the *delay* should be set to. Returns the *delay* in ms.
- static void *dipsw_onoff_poll*()
  Polls DIP switch 2 to see if the program should be reversed because it has been moved to the Off position or if the program should go back to its original direction because it has been moved to the On position.
- static void *flash_led_sequence*(int *p_delay*)
  Flash the LED's in the proper sequence per the project requirements.
- static void *init*()
  Calls *uc_dipsw_init()* to initialize the DIP switch, *uc_led_init*() to initialize the LED's, and *dtim0_init*() to initialize DTIM0.
- **int** *main*()
  Calls *init*() to initialize the hardware. Then drops into an infinite loop calling *flash_led_sequence*() forever.

dtim.c

- void *dtim0_delay*(int *p_usecs*)
  Configures the appropriate DTIM0 registers to delay for *p_usecs* microseconds.
- void *dtim0_init*()
  Initializes the DTIM0 timer.

gpio.c

- **int** *gpio_port_dd_get_pin_state*(int *p_pin*)
  Accesses the SETDD register and returns the state of pin *p_pin* (0 or 1).
- void *gpio_port_dd_init*(int *p_pin*)
  Configures PDDPAR and DDRDD so pin *p_pin* of port DD is in GPIO function and the pin data direction is input.
- **int** *gpio_port_tc_get_pin_state*(int *p_pin*)
  Accesses the SETTC register and returns the state of pin *p_pin* (0 or 1).
- void *gpio_port_tc_init*()
  Configures PTCPAR and DDRTC so port TC is in GPIO function and the pin data direction is output.
- void *gpio_port_tc_set_pin_state*(int *p_pin* ,int *p_state*)
  Accesses the SETTC register to set the state of pin *p_pin* to *p_state*.

uc_dipsw.c

- **int** *uc_dipsw_get_state*(int *p_switch*)
  Calls *gpio_port_dd_get_pin_state*() to return the state of the pin connected to *p_switch*.

▪ void ***uc_****dipsw_init*()

Calls *gpio_port_dd_init*() to initialize switches 1, 2, 3, and 4.

uc_led.c

▪ void *uc_led_all_off*()

Calls *uc_led_off*() four times to turn all LED's off.

▪ void *uc_led_all_on*()

Calls *uc_led_on*() four times to turn all LED's on.

▪ void *uc_led_all_toggle*()

Calls *uc_led_toggle*() four times to toggle all LED's.

▪ void *uc_led_init*()

Calls *gpio_port_tc_init*() four times to initalize the pin that each LED is connected to.

▪ void *uc_led_off*(int *p_led*)

Calls *gpio_port_tc_set_pin_state*() to set the state of the pin for LED *p_led* to turn the LED off.

▪ void *uc_led_on*(int *p_led*)

Calls *gpio_port_tc_set_pin_state*() to set the state of the pin for LED *p_led* to turn the LED on.

▪ void *uc_led_toggle*(int *p_led*)

Calls *gpio_port_tc_get_pin_state*() to determine if LED *p_led* is on or off. It it is on, turn it off or if it is off, turn it on.

What to submit for grading

Put a comment header block at the top of each source code file that contains: (1) the name of the source code file; (2) the lab project number; (3) your name (and your partner's name); (4) your email address (and your partner's email address); (5) the course number and name, CSE325 Embedded Microprocessor Systems; and (6) the semester, Spring 2013.

We will export the project to a directory structure. In CodeWarrior, click File |Export on the main menu. In the Export dialog, expand General. Click on File System. Click Next. In the next dialog, click Select All. Enter a destination directory, e.g., C:\Temp. Click Create Directory Structure for Files. Click Finish. The entire project will be exported to C:\Temp\Proj03 (or whatever name you used for your project when you created it).

Zip this entire directory naming the zip archive **cse325-s13-p03-*lastname*.zip** or **cse325-s13-p03-*lastname1-lastname2*.zip** if you worked with a partner.

Upload the zip archive to Blackboard using the project submission link by the deadline. Consult the online syllabus for the late and academic integrity policies.