1 Instructions

You may work in pairs with a partner on this assignment if you wish or you may work alone. If you work with a partner, only submit one project file with both of your names in it; you will each earn the same number of points. Your project files must be uploaded to Blackboard by the assignment deadline. Section 4 describes what to submit and by when; read it now.

2 Lab Project Objectives

1. Configure and use the DTIM module to busy delay for a specific time period.
2. Configure and use the GPIO module to configure pins for digital I/O and primary/secondary/tertiary functions.
3. Configure and use the GPT module for the primary (GPT) function and configure pins for input capture mode.
4. Configure and use the I2C module to communicate with the Wii Nunchuk using the I^{2}C serial interface standard.
4. Configure and use the INTC module for interrupt handling.
5. Configure and use the PIT module to generate periodic interrupts at a specific frequency.
6. Configure and use the QSPI module to interface to the Octopus LED matrices using the SPI serial interface.
7. Configure and use the RNG module to generate random integers.

3 Lab Project

3.1 The Lights Out Game

Lights Out[1] is an electronic game consisting of a 5 × 5 grid of lights, see Fig. 1.

Figure 1. The Lights Out Game

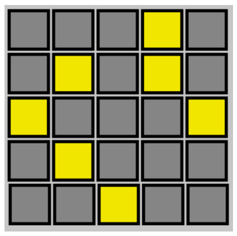

Figure 2. An Initial Game Configuration

Each light can be turned on or off by pushing the light (which is connected to a push button). When the game starts, certain lights will be on and the others will be off, see Fig. 2. The object of the game is to turn all of the lights off. Seems simple enough, but pressing a button to turn a light that is on or off will cause the light and all of the adjacent lights (above, below, left, and right) to toggle.

Let's number the rows and columns in the grid starting at 0 and use the notation (*row*, *col*) to represent the row and column of the light at row *row*, column *col*. Using this notation, the light in the upper left corner of the grid is at (0, 0) and the light in the lower right corner of the grid is at (4, 4).

Suppose we push the button for the light at (1, 3) as shown in Fig. 2. Fig 3 shows the state of the game after the lights toggle. Note that the lights at (1, 3) and (0, 3) were turned off—because they were on—and the lights at (1, 2), (1, 4), and (2, 3) were turned on—because they were off.

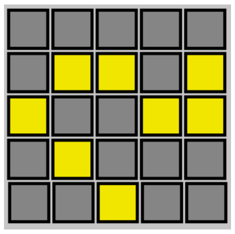

Figure 3. Configuration After Toggling the Light at (1, 3)

1 http://en.wikipedia.org/wiki/Lights_Out_(game)

3.2 Project Requirements for LED's Off

We shall implement the game (which we shall name **LED's Off**[2]) by using the Octopus Project Board 1 LED matrix as the game grid. The player shall interact with the game using the Wii Nunchuk controller which connects to an adapter on Octopus Project Board 3.

Game Play Mode Requirements

1. The game shall be played on a 5 × 5 grid of LED's using rows 0–4 and columns 0–4 of LED matrix 2 (LEDM2).
2. When an LED is on, the LED shall be yellow.
3. When an LED is off, the LED shall not be illuminated.
4. When a new game starts, $5 \le n \le 10$ LED's shall be randomly turned on.
5. The player shall use the Wii Nunchuk joystick to move a "pointer" around the game matrix. The pointer is displayed as a blue LED when the underlying game matrix LED is turned off or green when the underlying game matrix LED is turned on. The purpose of the pointer is to indicate the light that will be turned on or off when Nunchuk button Z is pressed and released (see Requirement 10).
6. When the Nunchuk joystick is in the left position, the pointer shall be moved to the LED to the left of the current pointer position, wrapping around if necessary.
7. When the Nunchuk joystick is in the right position, the pointer shall be moved to the LED to the right of the current pointer position, wrapping around if necessary.
8. When the Nunchuk joystick is in the up position, the pointer shall be moved to the LED above the current pointer position, wrapping around if necessary.
9. When the Nunchuk joystick is in the down position, the pointer shall be moved to the LED below the current pointer position, wrapping around if necessary.
10. When Nunchuk button Z is pressed and released, the LED at the pointer and the four adjacent LED's shall be toggled, i.e., if an LED is on, it shall be turned off and if an LED is off, it shall be turned on.
11. The game shall end when all LED's are turned off. When the game ends, the LED matrix shall display a green smiley face, see Fig. 4.
12. When the smiley face is displayed, pressing and releasing Nunchuk button Z shall start a new game.
13. During the game, pressing and releasing Nunchuk button C shall immediately start a new game.
14. During the game, pressing PB1 on the Tower microcontroller board shall cause the system to enter Test Mode.

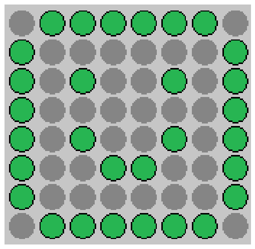
Figure 4. Smiley Face Displayed At Game End

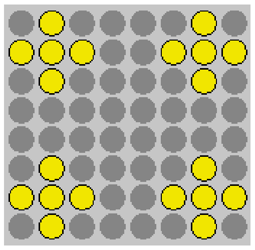
Figure 5. Test Mode Configuration

Test Mode Requirements[3]

1. Upon entering Test Mode, the LED configuration shall be immediately changed to the state shown in Fig. 5.
2. Game Mode Requirements 2, 3, and 5–11 shall function in Test Mode the same as in Game Play Mode.
3. When the last LED is turned off, the system shall return to Game Play Mode and display the smiley face per Requirement 11.

2 I would hate for any of you to get sued for trademark infringement.

3 Because it is next to impossible to get all of those frickin' LED's to turn off. Actually, there is an easily implement strategy for solving the game, see Wikipedia: http://en.wikipedia.org/wiki/Lights_Out_(game) or http://www.logicgamesonline.com/lightsout/tutorial.html.

3.3 Software Design and Implementation Requirements (Shalls)

I am not going to dictate a detailed software design but I am going to require that you partition your code into modules with each module implementing common, specific functionality. The required modules are listed below.

Module	Source Files	Section	Remarks
dtim	dtim.c, dtim.h	§3.3.1	Provided in Lab Project 5
gpio	gpio.c, gpio.inc, gpio.h	§3.3.2	Provided in Lab Project 5 (gpio.inc, gpio.h), Reused (gpio.c)
gpt	gpt.h, gpt.inc, gpt.s	§3.3.3	Provided in Lab Project 5 (gpt.h, gpt.inc), Reused (gpt.s)
i2c	i2c.c, i2c.h	§3.3.4	New
int	int.h, int.inc, int.s	§3.3.5	Provided in Lab Project 5
ledoff	ledoff.c, ledoff.h	§3.3.6	New
main	global.inc, main.c, others...	§3.3.7	Provided (global.inc), New (main.c and others)
oct_ledm	oct_ledm.c, oct_ledm.h	§3.3.8	New
oct_nunchuk	oct_nunchuk.c, oct_nunchuk.h	§3.3.9	New
pit	pit.c, pit.h	§3.3.10	Reused
qspi	qspi.c, qspi.h	§3.3.11	New
rng	rng.c, rng.h	§3.3.12	New
uc_pushb	uc_pushb.c, us_pushb.h	§3.3.13	Reused

3.3.1 dtim module — *dtim.c, dtim.h*

Reuse the code from Lab Project 5. Use *dtim_busy_delay_us*() to implement the required I^{2}C delays as described in the I2C pseudocode.

3.3.2 gpio module — *gpio.c, gpio.inc, gpio.h*

You should be able to reuse your code from previous projects and augment or modify as necessary.

3.3.3 gpt module — *gpt.c, gpt.inc, gpt.s*

You should be able to reuse your code from previous projects and augment or modify as necessary. This module is used by the uc_pushb module to detect Tower microcontroller board push button presses.

3.3.4 i2c module — *i2c.c, i2c.h*

This module contains functions which uses the I^{2}C serial interface standard to communicate with an I^{2}C device. Suggested functionality,

1. Implements *i2c_acquire_bus*() which busy-waits until the I^{2}C bus is idle.
2. Implements *i2c_init*() which initializes the ColdFire I2C module per the pseudocode.
3. Implements *i2c_reset*() which enables the I2C module, makes the MCF52259 a slave-receiver, disables I2C interrupts, and disables the generation of repeated start bits.
4. Implements *i2c_rx*() which will receive n data bytes from an I^{2}C peripheral per the pseudocode.
5. Implements *i2c_rx_byte*() which will receive one byte from an I^{2}C peripheral per the pseudocode.
6. Implements *i2c_send_stop*() to send a stop bit.
7. Implements *i2c_tx*() which will transmit n data bytes to an I^{2}C peripheral.
8. Implements *i2c_tx_addr*() which will transmit the slave address and the read/write bit to an I^{2}C peripheral.
9. Implements *i2c_tx_byte*() to transmit one byte to an I^{2}C peripheral.
10. Implements *i2c_tx_complete*() which returns true if a transfer has completed and false if it has not.

Remarks

I have given you the pseudocode for this module in the lecture notes.

3.3.5 int module — *int.h, int.inc, int.s*

Reuse the code from Lab Project 5. Modify *int.s* to delete the statements on lines 126-128 that write 1 to IMRL [MASKALL].

3.3.6 ledoff module — *ledoff.c, ledoff.h*

Contains functions to implement the LED's Off game per the project requirements.

3.3.7 main module — *main.c, global.inc*, and others

Suggested functionality,

1. Calls initialization functions in other modules to initialize specific MCF52259 peripherals being used.
2. Calls code in the ledoff module to play the game.

3.3.8 oct_ledm module — *oct_ledm.c, oct_ledm.h*

This module contains functions to interface to the Octopus Project Board 1 LED matrices. Suggested functionality,

1. Implements *oct_ledm_init*() to initialize the hardware so we can display patterns on the Octopus project board LED matrix 2 (LEDM2). This function shall configure PIT 0 to generate periodic "LED matrix refresh" interrupts at 500 Hz (every 2 ms). It shall not start the timer.
2. Implements *oct_ledm_off*() to turn off the display of patterns on LEDM2. This is accomplished by deasserting $\overline{\text{LEDM\_OE}}$ (bring it high) to disable the outputs of the 74HC595 and TLC5916 shift registers. This function also disables PIT 0 so LED matrix refresh interrupts are no longer generated.
3. Implements *oct_ledm_refresh*() which is called every 2 ms by the PIT 0 interrupt service routine. This function shall send the appropriates bits for the red, green, blue, and row bytes to turn on/off the LED's in the current row of LEDM2. That is, the first time this function is called, it displays row 0. The next time it is called, it displays row 1. The eighth time it is called, it displays row 7. The ninth time it is called, it displays row 0, and so on.
4. Implements *oct_ledm_refresh_row*() which uses the QSPI module to transmit the red, green, blue, and row bytes for a selected row.
5. Implements *oct_ledm_display_pattern*() which will display an 8 × 8 pattern on LEDM2. This function does not really display the pattern, but rather, it stores the bits for the red, green, and blue bytes of each row. It then enables PIT 0 to generate LEDM refresh interrupts every 2 ms. The actual turning on and off of the LED's is controlled by *oct_ledm_refresh*().

Remarks

I have given you the pseudocode for this module in the Lecture 22 notes.

3.3.9 oct_nunchuk module — *oct_nunchuk.c, oct_nunchuk.h*

This module contains functions to interface to the Wii Nunchuk via Octopus Project Board 3. Suggested functionality,

1. Implements *oct_nunchuk_accel_x*() to return the x-axis acceleration value.
2. Implements *oct_nunchuk_accel_y*() to return the y-axis acceleration value.
3. Implements *oct_nunchuk_accel_z*() to return the z-axis acceleration value.
4. Implements *oct_nunchuk_button_c*() to return the state of button C.
5. Implements *oct_nunchuk_button_z*() to return the state of button Z.
6. Implements *oct_nunchuk_init*() to initialize the Octopus Nunchuk module and the I2C module for communication with the Wii Nunchuk. Configures PIT 1 to generate periodic interrupts at 4 Hz (every 250 ms).
7. Implements *oct_nunchuk_on_button_c*() which will save a callback function pointer for the user's function which is to be called when Nunchuk button C is pressed.
8. Implements *oct_nunchuk_on_button_z*() which will save a callback function pointer for the user's function which is to be called when Nunchuk button Z is pressed.

9. Implements *oct_nunchuk_on_stick_down*() which will save a callback function pointer for the user's function which is to be called when the Nunchuk joystick is moved to the down position.
10. Implements *oct_nunchuk_on_stick_left*() which will save a callback function pointer for the user's function which is to be called when the Nunchuk joystick is moved to the left position.
11. Implements *oct_nunchuk_on_stick_right*() which will save a callback function pointer for the user's function which is to be called when the Nunchuk joystick is moved to the right position.
12. Implements *oct_nunchuk_on_stick_up*() which will save a callback function pointer for the user's function which is to be called when the Nunchuk joystick is moved to the up position.
13. Implements *oct_nunchuk_pos_x*() to return the joystick x-axis (left/right) position.
14. Implements *oct_nunchuk_pos_y*() to return the joystick y-axis (up/down) position.
15. Implements *oct_nunchuk_read*() which is called every 250 ms by the PIT 1 interrupt service routine. This function reads the current state of the Nunchuk, stores the state information in global variables, and calls the user's callback functions as appropriate.
16. Implements *oct_nunchuk_reset*() which resets all of the callback function pointers to null.
17. Implements *oct_nunchuk_tx_cmd*() which transmits a command to the Nunchuk.

Remarks

I have given you the pseudocode for this module in the lecture notes.

3.3.10 pit module — *pit.c, pit.h*

You should be able to reuse your code from previous projects and augment or modify as necessary. Use PIT 0 to generate a periodic interrupt at 500 Hz to refresh the LED matrix. Use PIT 1 to generate a periodic interrupt at 4 Hz to read the state of the Wii Nunchuk buttons and joystick.

3.3.11 qspi module — *qspi.c, qspi.h*

This module contains code to interface with the ColdFire QSPI module. Suggested functionality,

1. Implements *qspi_init*() to initialize the ColdFire QSPI module per the pseudocode.
2. Implements *qspi_tx*() to transfer n data bytes. I polled QIR[SPIF] to determine when to return from this function.
3. Implements *qspi_rx*() to receive n data bytes.

Remarks

I have given you the pseudocode for this module in the lecture notes.

3.3.12 rng module — *rng.c, rng.h*

This module contains code to interface to the ColdFire Random Number Generator (RNG) module. Suggested functionality,

1. Implements *rng_init*() to initialize the ColdFire Random Number Generator (RNG) module.
2. Implements *rng_next*() which returns an unsigned 32-bit random integer.
3. Implements *rng_next_in_range*(int *low*, int *high*) which returns an unsigned random integer in [*low*, *high*].

Remarks

The ColdFire Random Number Generator (RNG) module is documented in Chapter 6 of the IMRM. It is very easy to configure and use so I will let you figure out how to write this code.

3.3.13 uc_pushb module — *uc_pushb.c, uc_pushb.h*

You should be able to reuse your code from previous projects and augment or modify as necessary.

3.4 Build Notes

Make certain to configure your CodeWarrior project to conform to the Warning and Language Settings as discussed on p. 4 of the *CSE325 Code Style Guidelines* (available on the course website). The TA will be building your project using these configuration settings, and if you do not follow these instructions—in particular the Warnings settings—then your code may not compile. This could result in a reduced project score, so please do this.

Be certain to disable the C Standard Library by navigating to *Your Project* > Properties > C/C++ Build > Settings > Librarian and unchecking Enable Automatic Library Configurations and then to *Your Project* > Properties > C/C++ Build > Settings > ColdFire Linker > General and checking No Standard Library.

Assembly language include files—those with a .inc file name extension—are placed in the Project_Headers folder of your project. However, the assembler needs to be instructed to look in that folder for the include files. Navigate to *Your Project* > Properties > C/C++ Build > Settings > ColdFire Assembler > Input. In the User Path area you will see an icon with a green + sign. Click on that icon to open the Add Directory Path dialog. In the dialog, click on Workspace. In the Folder Selection dialog, expand your project folder and select Project_Headers. Click OK, OK, and OK to close the dialogs.

4 What to Submit for Grading and the Assignment Deadline

For each source code file, put a header comment block at the top of the 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, Fall 2013.

We will export the project to a directory structure. In CodeWarrior, click **File | Export** on the main menu. In the **Export** dialog, expand the **General** category. 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\Proj07* (or whatever you named your project).

Upload the zip archive to Blackboard using the project submission link by the deadline which is 4:00am Mon 9 Dec 2013. Consult the online syllabus for the late and academic integrity policies.