Final Project

Sabrina Herman (sh997) and Brian Szczesniak (bs593)

Video Link

https://youtu.be/MkcQW1bn_II

Overview

Original idea: Fairy Lights

Our original idea was to use individually addressable LED light strips to create a fancy multi-color set of fairy lights. We were going to communicate with the LED's using SPI and use the buttons on the board to change the lighting patterns like random twinkle, chasing, strobe/blinking, and change colors. We were going to use a PIT timer to control changes in within a mode, the button interrupts to change modes, and SPI communication to interface with the lights. The buttons SW2 and SW3 would have controlled color and mode respectively.

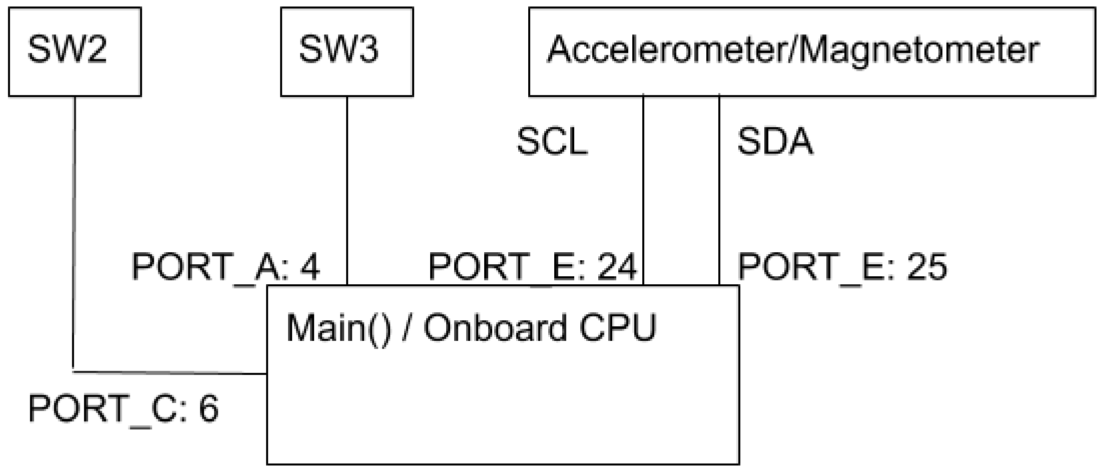
However, we were not able to implement SPI. We used the link at the end of the lab 6 description for CMSIS Driver, but were unable to get that to work. It wasn't properly initializing the Driver_SPI0, and all its pointers were null. We debugged to see that they were null pointers and used an oscilloscope to see nothing was outputting from the board pins. After speaking with several TAs and with Bruce Land, we were unable to find a solution. We switched to the next best project we could come up with, a Bop-It, since that project has the required complexity and uses no additional hardware (as there was not time to get new hardware). As such our project uses no hardware other than the board and thus we have no bill of materials.

If we had more time we would have liked to implement SPI from scratch by directly manipulating SPI registers and pins, but given the time constraints of finals, this wasn't an option.

Actual Project: Bop-It

We used the accelerometer, magnetometer, and the push buttons to create a single-player Bop-It game. The game works in rounds. To win a round, the player will need to hit one of the two pushbuttons (SW2 or SW3), shake the board (measured with accelerometer), or turn the board (measured with magnetometer). The player loses when they fail to complete the exercise within the required time. The game ends when the player loses a round, and then the player gets a score based on how many rounds they were able to win before the failure.

Implementation



Pregame State
Before the round starts, the player can choose their level of difficulty by hitting the SW3 which cycles through easy, then medium, then hard. The colors are as follows:

Green - easy
Yellow - medium
Red - Hard

Then the player hits SW2 to start the round.

In Game State
We used different colored lights to instruct the player. Once the game starts, the players respond to a color with a corresponding action.

Cyan - push the SW2
Blue - push the SW3
Green - shake the board
Purple - turn the board

End Game State
Then when the game ends, the game indicates it and displays the score.

Red - indicates game has ended.
Yellow - The light blinks yellow the number of times that the player completed a round, indicating the player's score.

Interrupt Timers
The PIT0 timer is used to make our random number generator. We mod the value of the PIT0 timer by 4 to determine which action is required of the player (push SW2, push SW3, shake, or turn).
The PIT1 timer set the amount of time for one round game play. If this timer is up before the player completes the action, the player loses.

Pushbuttons
When a button is pressed, it triggers that button's interrupt. SW2 triggers the PORTC_IRQHandler and SW3 triggers the PORTA_IRQHandler. In a pushbutton's interrupt handler, the state of whether or not the button has been pressed gets updated.

Main While Loop
The game is played in a while true loop in main through a series of conditional statements. First, we check if the game has started. This brings the player to either the "pregame state" if the round has not started and to the "in game state" if it has.

If the player is in the pregame stage, and has pressed SW2 or SW3, the appropriate actions will be taken as described in the pregame section. If SW3 is pressed, the difficulty status is increased, and a time is picked to load into the PIT1 timer (decides length of game round). If the SW2 is pressed, we call nextRound() which starts the next round by enabling the PIT1 timer.

If the player is in the game stage, we first check if the player has failed the round (by running out of time or by performing incorrect action). In this case, we change a variable to indicate that the game is no longer occurring, blink the red light, then call displayScore() which will display the score.

If the player hasn't failed yet, we check which type of activity the player needs to complete. (The PIT0 random number generator has determined the activity). Once we've determined the activity, we check if the player has completed the activity. For the pushbuttons we check if the sw2pressed and sw3pressed variables have been updated to reflect that they have been pressed. For the accelerometer and magnetometer we check the x, y, and z values of the accelerometer state or the magnetometer state. The while loop keeps checking to see if the player has completed the activity. If yes, do a new activity. If no then no action is taken, but they will eventually run out of time and fail this activity.

Initializations
In our main function (but before the while loop), we initialize the accelerometer, magnetometer, LEDs, the PIT0 timer interrupt, and the SW2 and SW3 timer interrupts. We set the priorities of the interrupts.

Initialization of the PIT and LEDs interface directly with the corresponding registers. For the accelerometer and magnetometer we use function provided in "board_magnetometer.h" and "board_accelerometer.h" which are part of the board support files for the devices. These provided with two functions each, one which initializes the respective portion of the device (both the magnetometer and accelerometer are a part of the "NXP FXOS8700CQ low-power, six-axis Xtrinsic sensor") and the other of which updates a state struct with current readings. Additionally, the hardware_init() function from the "hardware_init.c" file had to be modified to enable the clock to Port E which is were the sensor connected. This sensor is communicated with using I2C, which is performed by the provided functions.

The LEDs are enabled using the util.c file we borrowed from previous labs. We edited this code to include some more useful functions that were relevant to our project such as delayShort(), blinkGreen(), blinkRed(), blinkBlue(), and blinkYellow().

The PIT0 timer and push buttons were enabled here (in main but before while loop) but the PIT1 timer isn't enabled until the round starts (when nextRound() is called).

The priorities of the interrupts are SW3 > SW2 > PIT1 > PIT0.

The initializing and managing the many different input sources was the most complicated part of this project. Specifically the PIT1 timer is distinct from timers we have used previously in that it does not always run to completion and throw an interrupt. Sometimes the PIT1 timer needs to be stopped manually before it has finished. In the case that an action was performed correctly, we had to manually stop the timer and reset it value. This initially cause trouble because you cannot directly set the countdown back to its start value. You must disable the timer then enable it with the correct load value.

Testing

We tested by playing the game many times. We counted the score to make sure the score was accurate (for zero and multiple times). We tested multiple games without resetting the board. We checked that speed (game difficulty) correctly changes between games if the user selects a new level of difficulty. We tested the round timer (PIT1) to be sure it properly resets between rounds and games. Additionally we tested to figure out what the accelerometer and magnetometer threshold should be in order to make it the right level of difficulty for the user

Work Distribution

We made important project decisions together including specific aspects of the design we want to have. We set meeting times to both work on the code. We coded on Brian's computer because his computer is very fast. For code review we used pair-programming; the partner that doesn't actively type a section of code will review the code of the other partner to check that the code performed what we intend. Testing was done together during our meetings on the board. We collaborated on the written components in a google doc and also collaborated on the video. Both were completed during our meetings. We had equally active roles in all lab aspects including discussing ideas, typing code, and writing this report.

Sources of Information

The only resources we used were class materials. We used the user guide and the user manual. We followed the tutorial for the accelerometer given in the lab handout. We used the discussion slides to understand how to operate the pushbuttons. We also used advice from Piazza. We borrowed the utils.c and utils.h files from previous labs to blink the LED.