Dennis Your and Tim(Ding) Qin

EC450

Homework 6

For this assignment, we used SPI to communicate between two devices, which are two MSP430 boards. One is the master, one is the slave. After the button is pressed on the main board, it should generate a random number, generated via timer. The second board will then guess a number and the main board will tell it if it is less than, greater than, or equal to. The master’s red LED will turn on when guessing is in progress, while the slave’s red LED will do the exact opposite, but will also blink while evaluating the second set of bits. Pressing the button again after a correct guess will reset the game. Our program is able to successfully guess numbers in the range of 2 bytes. There is a global variable that keeps track of how many guess are taken.

SPI is used to communicate between the boards, using the SPI transmit and receive buffers to load the data into shift registers to be sent/received. Using a synchronized clock and the sending of states in between devices, we were able to communicate between the MSP430s. SPI uses 3 ports, one for SOMI, one for SIMO, one for the synchronized clock. The input/outputs use the pins P1.6 and P1.7 while the clock uses pin P1.5. The button uses pin P1.3 and the LED uses pin P1.1.

In the master code, ‘s’ is the starting state, ‘l’ is the load first byte state, ‘u’ is the load second byte state, and ‘g’ is the finishing state. In the slave code, ‘s’ is where the lower bits are loaded, ‘l’ is where the higher bits are loaded, and ‘u’ is the intermediary state between transmit and receive. The master keeps track of if the game is running or not, transmitting or not, and receiving or not through the variables started, tx\_mode, and rx\_mode.

The guesses are performed using a binary search, halfing the possibilities every time it guesses. If the guess is lower than the random number, the guess becomes the lower threshold and the guess is recalculated. Reverse happens when the guess is higher.

We had trouble with the second board guessing the correct number when we implemented two bytes. The program loaded 2 bytes, compared them successfully, and sent it back. However, the initial part of the program appeared to load false data into itself at the beginning, corrupting the upper and/or lower bounds in the binary search sequence. Thus we rewrote the code to better integrate states. One of the issues we were missing the SPI handler for the master code, thus the WDT timings messed with the implementation of the states.

The random number generator uses the MSP’s timer values to generate a 16 bit number.