Microprocessor final project

Ding Qin

Dennis Your

EC450

Introduction

For our final project we decided to implement a lock and authentication system. This system contains two MSP430 boards that communicate with each other, 4 buttons to input the pin code, and 4 leds to show the user what the pin is after the system is authenticated. For simplicity, we will refer to the two chips as master and slave and define the uses of both now. The master handles the button inputs (4 decimal digits, represented in binary), key comparison, and unlock system. The slave sends over its key and pin to the master and if the keys match, will flash the pin that it has on the LEDs (also 4 decimal digits, represented in binary).

One board will act as the master, while the other board will act as the slave. The master board will act as the lock that has a preset key, with the pin determined randomly. The slave will act as the authenticator that sends over a 4 byte long int key and a corresponding pin code. Once the master receives the full 4 byte key, it will confirm whether or not this is the same key that it has stored. If the keys match, then the master will set the pin code to that of the slaves, which the user uses to unlock the system. Once the slave’s key is confirmed, it will display the pin code in binary by blinking the LEDS accordingly. Then the user can input this pin through the buttons that are connected to the master board. If a mistake is made when entering the pin, for that clock cycle, the pin will be randomized, but on the next clock cycle, the user can start to enter the pin again from the beginning. Also, if the user realizes they inputted the incorrect pin, they can wait for a timeout to occur, resetting the position the pin inputting is at back to zero. Once the correct pin has been entered, the LED on the master, which is connected to pin 1.0, will light up indicating the user has been granted access. The device may be hooked up to various lock systems, for instance door locks or lockboxes.

Also if you want to sync another slave that has a different key to the master, the user can connect the two boards and then press all four buttons down at once. This way the master will set its key and pin code to that of the slaves, overwriting the original pin code. Initially, when the new slave is connected to the master, the LEDs will not flash due to the key being incorrect, but upon successful synchronization with the second slave, the leds will start flashing the current pin.

If there is no slave connected to the master, then the master will randomly generate a pin every clock cycle. This means that without a slave, no one, not even the designated user will have access to the system.

Implementation

**Hardware implementation**

To implement the communication between the two boards we will be utilizing SPI. The two boards will be connected in the following way. Pin 1.7 of the slave will be connected to Pin 1.7 of the master as the SPI MOSI. Pin 1.6 of the slave will be connected to pin 1.6 of the master as the SPI MISO. Finally pin 1.5 of the slave will be connected to pin 1.5 of the master in order for the slave to synchronize itself with the master’s clock. Also both boards have a common VCC andground.

The buttons were connected to the pins on the master board, while the LEDs were connected to the pins on the slave board. But they all shared a common ground from the power source. The buttons are connected in series to pins 1.1-1.4 of the master board and in series to ground. The LEDs are connected in series to pins 1.1-1.4 of the slave board and are in parallel with 1k ohm resistors, which are connected to ground.

**Software implementation**

The system uses button interrupts, WDT interrupts, and receive interrupts to operate. When the master and slave boards are connected and the button to start has been pressed, the master will send over chars such as “s, a, b, c, d, e, f, g, =, or !” to indicate the state the master is in to receive the key from the slave. This is implemented in the WDT interrupt.

The master first starts off in state “s”. It will set the variable “data\_to\_send” equal to the char “s” and then send it to the slave using the transmit buffer UCB0TXBUF. The slave will go into the spi\_rx\_handler() and take in the char through the receive buffer UCB0RXBUF. Here the first byte of the key will get transferred to the master through the TX BUF. Once the master detects that it received something, it will take in the value in the receive interrupt and will set the master to stop receiving mode by setting the value of rx\_mode to 0 in the WDT. Then it will set the state to “a”. In the spi\_rx\_handler() for the master, it will then go into the state “a” where it will take the value from the RX buffer and store it into data\_received, which then gets stored into the first\_byte variable. This will be the first byte of the key sent from the slave.

To get the second byte of the four byte key, the master will now send over the char of “a” to the slave through the same method. The slave will go into the state “a” case in the spi\_rx\_handler and send the second byte of the key over to the master. Once the master receives something, the value is stored and the state will be changed to “b” and it will no longer be receiving again. The master will then store the value in the RX buffer to the variable data\_received, which then gets stored into the variable second\_byte.

To get the third byte of the four byte key the master will send over the char of “b” to the slave. The slave will go into the “b” case and send over the third byte of the key to the master. Once the master RX buffer receives something, the value is stored and the state will be changed to “c” and it will not be in the receive state anymore. The value in the RX buffer for the master will be stored into the data\_received variable, which then gets stored in the variable third\_byte.

To get the final byte of the key, the master will send over the the char of “c” to the slave. The slave will go into the “c” case in the spi\_rx\_handler and send over the fourth byte of the key. Once the RX buffer in the master receives it, the value is stored and the state changes to state “d” and it will stop receiving. The value in the RX buffer will be stored in the data\_received variable, which then gets stored in the variable fourth\_byte. This is also where the four separate bytes get combined to become the complete full key.

After the key gets sent, it will be in the “d” state. In this state, the first byte of the 2 byte pin will be sent to the master from the slave. Once the master receives it through the RX buffer, the state will change to “e”, where the value from the RX buffer in the master gets stored in the variable pin1.

To get the last byte of the pin, the master will send over the char of “e” to the slave. The slave will go into the “e” case in the RX handler and send over the last byte of the pin to the master through the TX buffer. Once the master receives something in the RX buffer, the state will be changed to “f” and it will stop receiving. In the rx handler for the master, the “f” case will store the last byte of the key from the RX buffer into the variable called pin2.

The “f” state is an intermediary state to indicate that transmission is finished and the master can now send over the response to the slave.

In state “g” in the WDT interrupt for the master, the key that was sent from the slave is compared to the key stored in the master. If the keys are a match, then the master sets the pin to that of the slaves and sends the slave a char of “=”. Once the slave receives this char, the variable playpin will be set to 1, which will go into the LED playing state in the WDT interrupt for the slave. If the key is wrong, an exclamation mark “!” will be sent to the slave and playpin will be set to 0, thus stopping the LED playing.

In the slave LED playing state there is a switch case that is dependent on the variable pindigit. The variable pindigit parses through the 4 cases, where each case blinks a digit of the pin represented in binary. To make sure the pindigit does not increment every WDT interrupt cycle, we set it so it only increments after 32 WDT interrupts. Every 16 WDT interrupts, the LEDs turn off, giving a pause in between digit flashes. This gives the user enough time to register the information shown on the LED. After 192 WDT interrupt cycles, pindigit will be reset to 0 so it will start repeating the binary pin code from the beginning. This is to ensure the consecutive plays of the total pin will have a pause in between them.

Once the slave receives the char “s” through the UCB0RXBUF, it will transmit the first byte of the key to the master again through the UCB0TXBUF. This segment of code is implemented in the spi\_rx\_handler() in the slave board.

Once the key is confirmed in the master, the pin is set to the slave’s pin and can be inputted through the buttons. The buttons are operated through the button interrupt handler and the WDT interrupt handler. Once a button is pressed, the variable corresponding to the binary digit in the pin, digit1-4, goes to a changed state, along the variable “charged” which indicates if the next decimal digit in the sequence is being inputted. The buttons are debounced by the counter “debounce” which after a button is pressed, resets itself to 32 (WDT interrupts). If the debounce counter times out, the variables digit1-4 are read. First, if the binary representation is 1010 or in decimal, equal to 10, then the temporary variable testvalue is set to 0 (because you can’t have 0 button presses). Next, if the variable is marked as a 1, the value of the binary representation is added. So for instance, if the decimal digit is 9, the button presses would be 1001, which would add 1+8 to a temporary variable testvalue. If testvalue is greater than 9, it is ignored, otherwise it is added to the total decimal pin, with corresponding position determined by a factor of ten multiplier (so if the 9 was in the hundreds place, it would be multiplied by 100). The pinindex is then incremented to the next digit to be inputted.

Finally, if the pinindex reaches 4 and the inputs are read in, the pin stored and the typed pin are compared. If the typed pin is equal to the pin, the LED at P1.0 is turned on, which can also be connected to a locking device, and the LED/lock will stay unlocked for 200 WDT interrupts (lock\_timeout) before locking again. If it is not correct, the typed pin, pin digits, and pin index will all reset for the next unlock attempt.

Assessment of the project

The software implementation of the project was a success. Everything performed as intended. We were able to correctly send over a key from the slave, have the master check to see if it matches. If it did then the master pin code would be set to that of the slaves and the led would start blinking the access code.

The hardware implementation was not as successful. The biggest and only issue was the solenoid lock. The lock that was used operates at 9-12V, which translates to 500-650mA of current. To obtain the power ratings required, we first tried to implement an op-amp to obtain the 9-12V required, obtaining the higher voltage through a 9V battery in series with a 1.5V battery. We successfully got the voltage to a workable voltage, but was only able to drive a current of about 120mA. Next, we attempted to use transistors to obtain the voltage gain instead. However, neither BJTs nor MOSFETs could provide anywhere close to 500-650mA. Finally, we obtained a power BJT to hopefully be able to operate the lock, and even that could not give the results we needed.

Future Improvements

To compensate for the insanely high power requirements, we could have done a multiple stage amplifier to potentially reach the operating thresholds. Or a simpler method would be to use a motor requiring a lower voltage and current to operate to use as a lock.

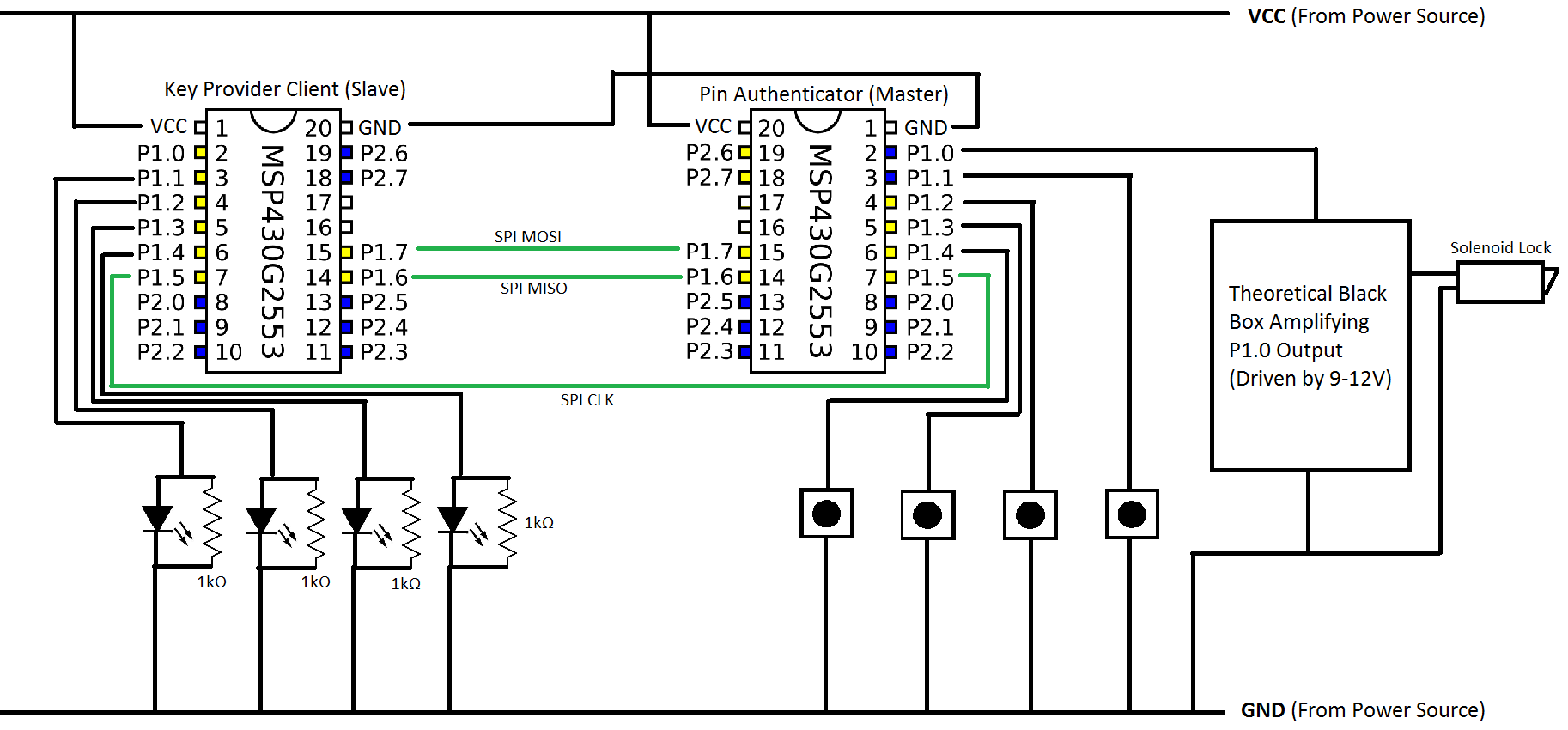
As for functionality, the synchronization with the key device should have not been implemented using the four pin buttons, and instead used the Port 2 pins and another button to synchronize with a different key, as it would be a major security flaw. If someone obtained a slave key device, they would be able to synchronize the authenticator to their own pin at any time, without having unlock access first. The extra button would be located in the locked area, thus requiring any alternate key synchronization other than the one already saved to have access beforehand to save their key into the database.

In addition, the device would be more secure with a randomly generated pin given by the slave rather than a preset pin. Also some form of encryption or key guess protection would be useful, as if someone kept guessing the key of the master (though it would take a long time since it is a long int), they would eventually get the key and be able to set the pin themselves.

Summary of contributions

The way we split up the work was one person did the code for the slave board while the other person did the code for the master board. But we generally worked together on both parts since our final design was a little different from our original design.

Circuit Diagram of the System



Actual picture of the Working Device, with example Solenoid Lock+Batteries to Drive it

