SMART PORTABLE KEY

By

Aashay Shah

Akshay Chanana

Final Report for ECE 445, Senior Design, Spring 2013

TA: Igor Fedorov

1st May 2013

Project No. 45

Abstract

We have designed and implemented a wireless key and lock system that allows the user to wirelessly access multiple locks. Our aim was to build a secure and portable device that is easy to use and carry around.

The key consists of a fingerprint scanner, controller, RF transmitter and a panel of switches. After a successful fingerprint scan, the controller activates the panel of switches following which the user can choose which lock to open from the panel. Once the button is pressed the transmitter sends an encrypted key to the lock side.

The lock consists of a RF receiver, controller and an electromechanical lock with a servo. Once the encryption key is received, it is validated by the controller after which the servo is signaled to unlock the device.

Our final design worked completely with the exception of the re-locking mechanism. We were able to unlock multiple locks using one single key. The portability of the design was not as sufficient as needed and there are many ways this issue is addressed as discussed in the conclusion.

Contents

1. Introduction

This project was chosen because there are no devices presently available in the market that allows the user to wirelessly access multiple locks used for general purposes. There is a high demand for secure portable locks and this project aims to fulfill that need. The main focus is to provide a portable and secure lock system with a smart key that can be used to unlock multiple locks wirelessly.

The key consists of a fingerprint scanner, controller, RF transmitter and a panel of switches as shown in Figure 1. The lock consists of a RF receiver, controller and an electro-mechanical lock as shown in the same figure. The basic design is summarized in the sections below.

1.1 System Block Diagram

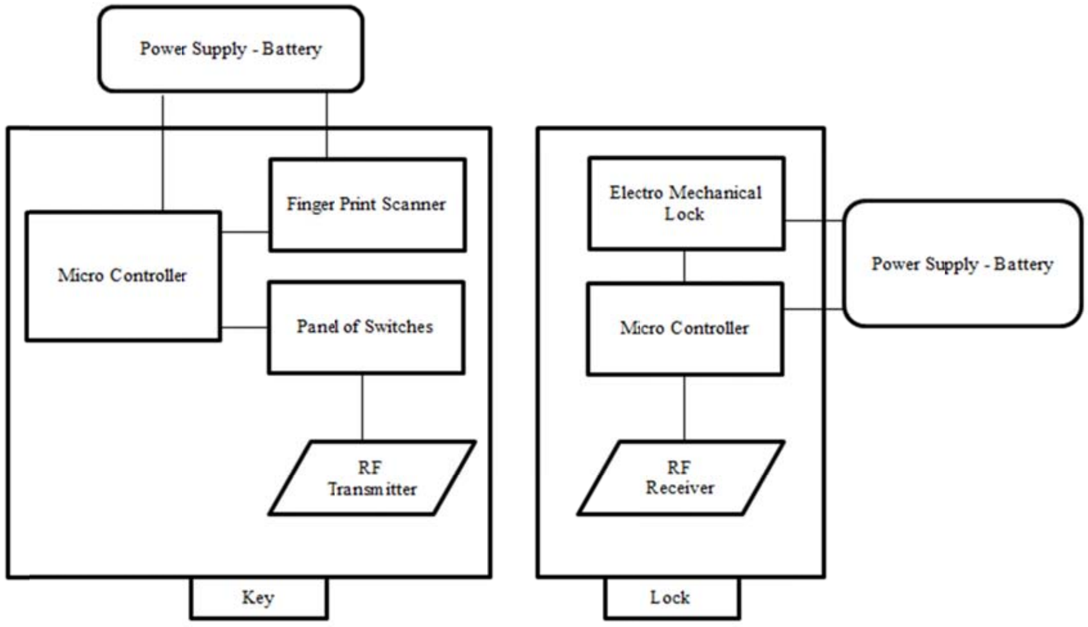


Figure 1: Detailed Block Diagram of the system implemented

1.2 Block Description

The project was divided into many modules, each with its own specific tasks and functions. These are described below with their designs explained in detail in Section 2.0.

1.2.1 Power Supply (Key)

This is responsible for powering the entire circuit on the key side of the design which includes the fingerprint scanner, microcontroller, panel of switches and the RF transmitter. Initially we were planning on using AA batteries but then we changed our design and instead used a 9V battery with a 3.3V regulator.

1.2.2 Power Supply (Lock)

This is just like the supply on the key side. It is used to power the controller, RF receiver and the servo on the electro-mechanical lock. The microcontroller should be able to send a signal to the electromechanical component which will then implement the unlocking mechanism. It consists of a 9V battery with a 3.3V and a 5V regulator. We need a steady 3.3V supply for the RF receiver and the controller whereas the 5V is required for the servo.

1.2.3 Fingerprint Scanner

This is the security measure used on the design of the key module. It is used to record and verify multiple fingerprints from users. Upon validation of the correct user it sends data serially over to the microcontroller connected to it. The microcontroller then checks this data and on a successful validation activates the panel of switches for unlocking the locks. We are using the 3.3V LEM-100 module by Integrated Biometrics.

1.2.4 Panel of Switches

This panel consists of multiple switches that are directly used for unlocking the same respective number of locks. This panel gets activated by the microcontroller after successful verification by the fingerprint scanner. This is achieved by using multiple debounced push buttons connected to the controller's digital pins.

1.2.5 Microcontroller

The microcontroller used is the Texas Instruments MSP430. This is the main control unit of the both the key and the lock modules. The MSP430 is a good option as it has very low power consumption as well as a low cost. It processes the data from the fingerprint scanner to validate that it has read the right fingerprint and only then activate the panel of switches after which an RF signal will be transmitted to a particular lock. Another microcontroller is also present on the lock side to process the incoming signal and then initiate the unlocking mechanism.

1.2.7 RF Transceiver

The key also consists of a RF transmitter, and the lock of a RF receiver to implement the wireless communication between the two components of the design. Thus making use of RF will also let us unlock the device from a distance. This communication module relays data to and from the microcontroller on both, transmit and receive side. We used the XBee RF transceiver to achieve this functionality.

1.2.6 Electromechanical Lock

The electromechanical component on the lock should be implemented such that it clicks open on receiving the correct signal from the controller. The controller sends a PWM signal to the servo on the lock which in turn unlocks it.

1.3 Performance Requirement

• Battery life should be long and thus overall efficiency of power supply should be more than 50%.
• Voltage regulators should efficiently provide a steady 3.3V and 5V supply.
• Instantaneous response after triggering switch on panel.

- Fingerprint scanner holds multiple users.
- Fingerprint scanner relays data to microcontroller correctly for validation.
- Microcontroller should be able to activate the panel of switches within a reasonable delay.
- Panel of switches should be properly debounced.
- Microcontroller should be able to successfully send an encryption key.
- Able to accurately transmit and receive within a certain range (~ 100m).
- Able to rotate servo after providing the required PWM signal.

2 Design

The final design procedure and details are described in the sub-sections below based on what we mentioned in our Design Review [1].

2.1 Power Supply (Key/Lock)

A 9V battery is used for both the key and the lock components. On the key side the battery is used with a 3.3V regulator (UA78M33) as all the components run on the same supply voltage. On the lock side the battery is used with a 3.3V (UA78M33) and a 5V (L7805) regulator. The 3.3V is necessary for the controller and the RF receiver whereas the 5V supply is needed for the servo on the electromechanical lock.

Power Budget: After the calculations below we can successfully say that the 9V battery will be sufficient.

XBee RF Transceiver (1mW): TX peak Current 45 mA @ 3.3 V
RX Current 50 mA @ 3.3 V
Power down Current < 10 uA
Max Power needed = 3.3*0.05 = 0.165 W

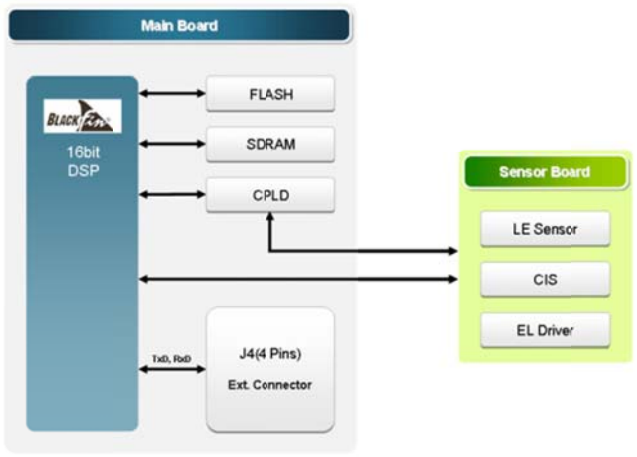
LEM100 Fingerprint Scanner: Typical voltage = 3.3V, maximum voltage = 3.8 V
Supply current - Idle state = 118 mA
Enrollment state = 178 mA
Identification state = 170 mA
Deletion state = 178 mA
UART baud rate = 9600 bps
Max Power needed = 3.8*0.178= 0.6764 W

TI MSP430G2 Microcontroller: Max current = 60 mA
Voltage = between 1.8 V and 3.6 V
Max Power needed = 0.06*3.6 = 0.216 W

Max Power Used by any component = 0.6764 W
Total Power Used = 0.216 + 0.6764 + 0.165 = 1.0574 W
Total Power that we have = 9 * 0.178 = 1.6 W > Total Power Used

2.2 Fingerprint Scanner

We are using the 3.3V LEM100 scanner from Integrated Biometrics. The module can perform storing, identification and deletion of fingerprints using one of the best algorithms one can find in the market today. It is quite a compact module and is very easy to integrate in our system. It comes with an in-built memory system as shown in the block diagram for LEM100 in Figure 2. The scanner comes with a main board and a sensor board as shown in Figure 3.



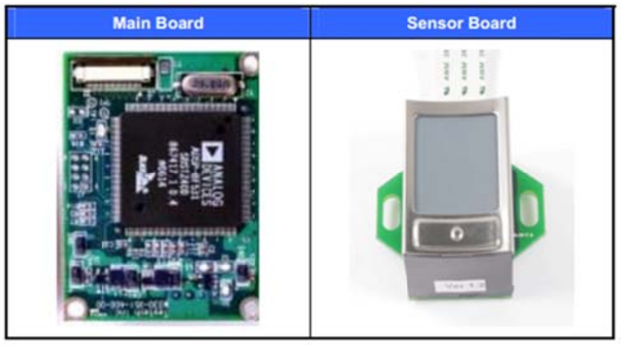


Figure 3: LEM100 Module Board [2]

The sensor board is connected to the main board through the J2 connector. We mainly use the J3 4-pin connector to interact between the controller and scanner. J3 contains pins for: VCC, GND, TXD, and RXD as shown in Figure 4.

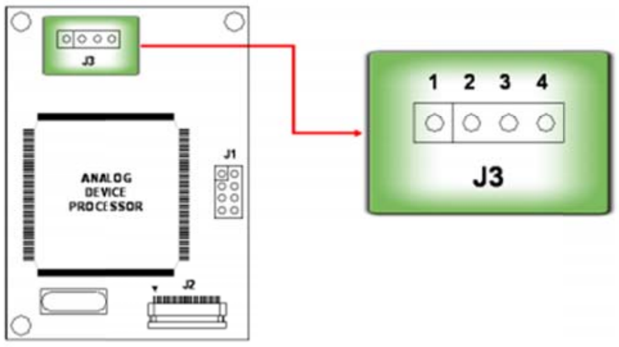
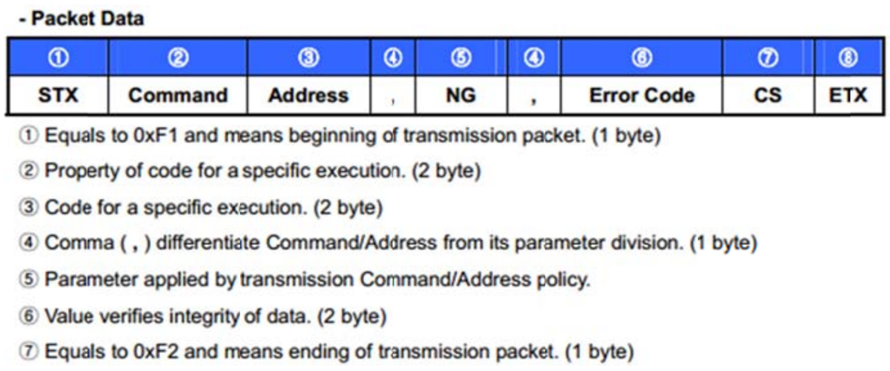


Figure 4: J3 connector LEM100 Module Board [2]

The TXD is used to transmit the data from the scanner to the controller and the RXD is used to receive data from the controller. The communication protocol for the LEM100 is described below in

Figure 5. The packet data has a start of packet byte followed by 2 bytes of which command, some reserved bytes, error code and finally 1 byte to signify end of packet.



- Packet Data

①	②	③	④	⑤	④	⑥	⑦	⑧
STX	Command	Address	,	NG	,	Error Code	CS	ETX

① Equals to 0xF1 and means beginning of transmission packet. (1 byte)

② Property of code for a specific execution. (2 byte)

③ Code for a specific execution. (2 byte)

④ Comma (,) differentiate Command/Address from its parameter division. (1 byte)

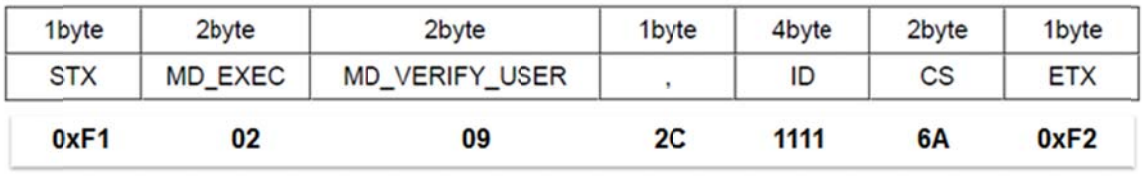
⑤ Parameter applied by transmission Command/Address policy.

⑥ Value verifies integrity of data. (2 byte)

⑦ Equals to 0xF2 and means ending of transmission packet. (1 byte)

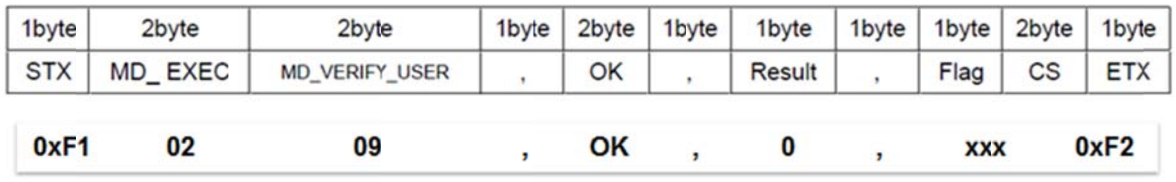
Figure 5: LEM100 Communication Protocol [2]

The control signals sent and received by the controller to/from the scanner are shown in the figures below. The control signal to the scanner basically consists of the command to accept and verify a fingerprint with the ID 1111 as shown in Figure 6. This ID had already been registered before the communication started in the scanner memory. Once the scanner checks for a fingerprint, it responds back to the controller with a signal as shown in Figure 7. If it successfully validates the scan, then we get the “OK , 0” bytes in the packet. For no scan or not validated we get a “NG” (No Good) or “OK , 1” bytes respectively.



1byte	2byte	2byte	1byte	4byte	2byte	1byte
STX	MD_EXEC	MD_VERIFY_USER	,	ID	CS	ETX
0xF1	02	09	2C	1111	6A	0xF2

Figure 6: Control signal sent from Controller to Scanner



1byte	2byte	2byte	1byte	2byte	1byte	1byte	1byte	1byte	2byte	1byte
STX	MD_ EXEC	MD_VERIFY_USER	,	OK	,	Result	,	Flag	CS	ETX
0xF1	02	09	,	OK	,	0	,	xxx		0xF2

Figure 7: Control signal received at Controller from Scanner

2.3 Panel of Switches

This consists of three switches that are directly used for unlocking the same respective number of locks. This panel gets activated by the microcontroller after successful verification by the fingerprint scanner. Once activated, this panel allows the user the option of unlocking any of the three switches. We have push button switches for each of the locks, which on pressing tells the controller to transmit a RF signal with some encryption to the receive side on the lock. These three switches are connected to the controller's digital ports and whenever the controller witnesses a high signal, it sends the signal to the required lock using the RF transmitter.

Since we are dealing with push buttons the issue of bouncing comes into play. Thus these switches need to be debounced properly. A simple debouncing circuit is shown in Figure 8 which helps us make the circuit perform smoothly with no bouncing as shown in our testing in Figure 14 under Section 3.0.

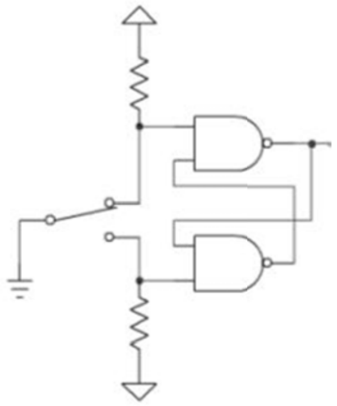


Figure 8: A simple debouncing circuit

2.4 Microcontroller

The microcontroller used in the design is the Texas Instruments MSP430G2553. This will be the main control unit on both the key and the lock sides. Thus this control unit in our design basically has to send and receive data, to and from, all other devices. The basic flow of information is captured in the flowcharts shown below in Figure 9 with the sender and receiver controller code shown in Appendix B. We used Code Composer Studio and the MSP430 Launchpad to basically test out and code our controller.

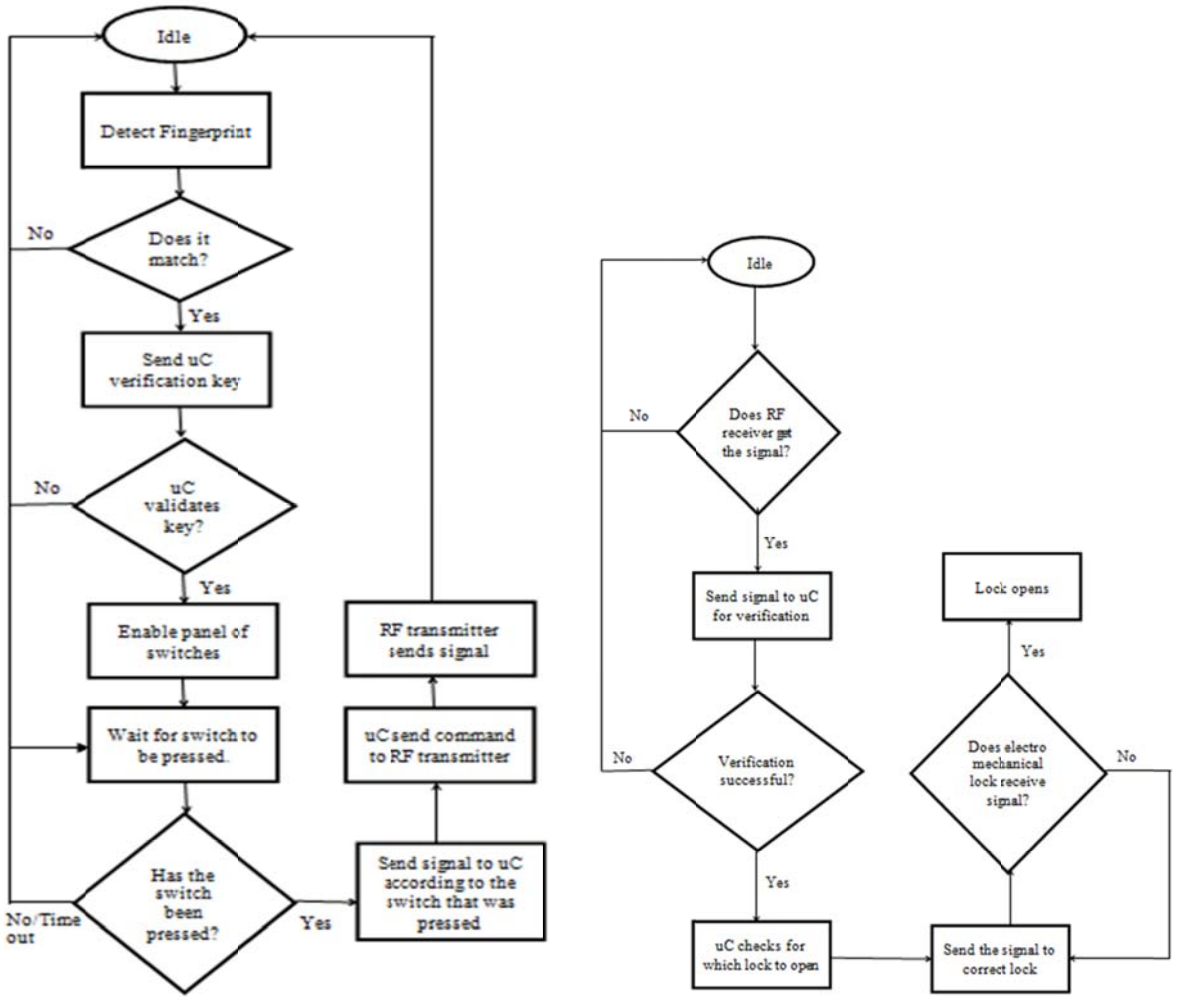

Idle
Detect Fingerprint
Does it match?
No
Yes
Send uC verification key
uC validates key?
No
Yes
Enable panel of switches
Wait for switch to be pressed.
Has the switch been pressed?
No/Time out
Yes
Send signal to uC according to the switch that was pressed
uC send command to RF transmitter
RF transmitter sends signal
Idle
Does RF receiver get the signal?
No
Yes
Send signal to uC for verification
Verification successful?
No
Yes
uC checks for which lock to open
Send the signal to correct lock
Does electro mechanical lock receive signal?
No
Yes
Lock opens

Figure 9: Flowchart of Controller Functionality on the Key (left) and Lock (right) sides

A big factor in our design was to implement a secure key so that there was no chance of any physical hacking possible. To tackle this issue, we had a 2 byte encryption key which was known to both the transmit and the receive side, such that whenever a transmission takes place with this key it gets verified every time for safety. This is not the best way to provide encryption, but with the time we had this is what we got working. Another way which would be harder to hack was to provide an encryption key based on the real time clock that was synchronized between the two ends as discussed in our conclusion.

2.5 RF Transceiver

The RF transceiver we are using is the XBee 1mW Trace Antenna - Series 1 (802.15.4) that works on the ZigBee protocol. Once the button is pressed on the panel of switches, the RF transmitter sends the corresponding frequency with the encryption key provided by the controller. The frequency would be the one matched with the lock the user wants to open. On the lock side when the RF receiver gets this message, it conveys it to the controller which validates the encryption key. Once this key is validated the controller sends a PWM signal for the unlocking mechanism to get activated on the lock. The RF module and controller interface is shown below in Figure 10.

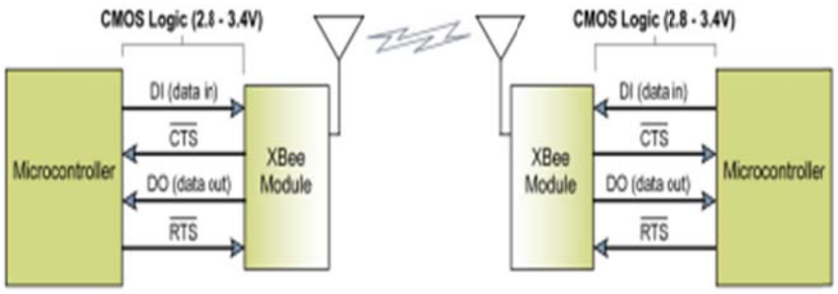


Figure 10: Microcontroller interface with XBee Transceiver [3]

An asynchronous signal is introduced in the module through the Data-In (DI) pin and is idle high when no signal is transmitted through it. The data transmits as 8 bits where the first bit is the Start/Stop bit. When the signal is low, the data starts transmitting into pin DI and when it becomes high it stops. The process is shown in Figure 11.

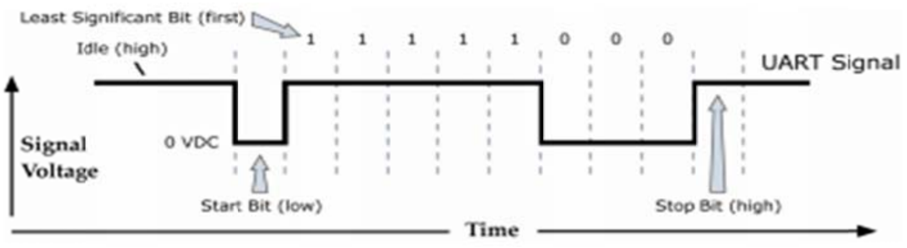


Figure 11: UART data packet as transmitted through RF module [3]

During the time that the RF transceiver doesn't operate, it works in the idle mode. Otherwise, it supports four different modes of operation as shown in the Figure 12 below. The modes used in this project are the - transmit, receive and idle modes.

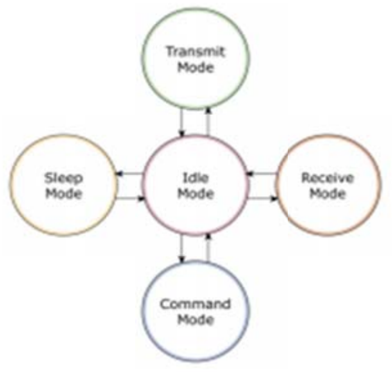


Figure 12: Modes of operation for the RF Transceiver [3]

2.6 Electromechanical Lock

The lock was designed and built by the ECE Machine Shop professionals. It consisted of the Hitec-HS311 servo as the electromechanical portion of the lock. It had three connectors: supply voltage pin (5V), ground pin and the PWM signal pin. The controller is used to rotate the servo and thus start the unlocking process when necessary. There is a switch on our lock circuit as well that locks back the servo to the original position. For safety purposes, once the lock has been unlocked the user has to first lock it in order for him/her to unlock it again. Hence it reduces the chance of the servo malfunctioning.

We had to rotate the servo only by 90 degrees to unlock it and then provide the same PWM signal in reverse to unlock it as shown in the figure below.

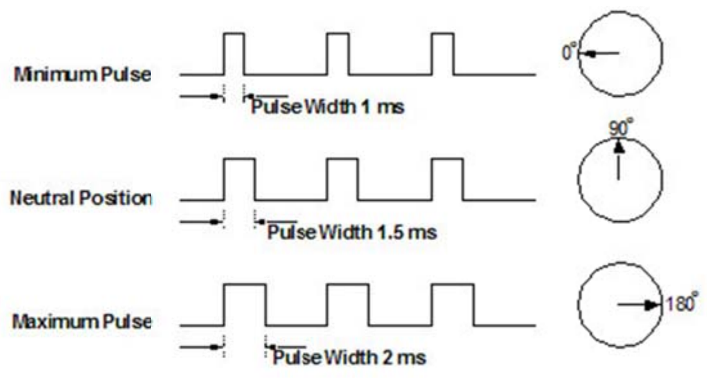


Figure 13: Servo Positioning [5]

3. Design Verification

3.1 Power Supply (Battery and Voltage regulator)

3.1.1 Ripple Voltage within limits

Before using the voltage regulator in the circuit, it was tested using an oscilloscope for any ripple voltage. If the voltage increases too much by chance, the circuit could get damaged. A 9V battery was connected to each of the voltage regulators and the output was connected to the oscilloscope. The oscilloscope showed that the output from both the regulators was within limits of ±0.5V. This case worked for the 5V and the 3.3V regulators.

3.1.2 Current Limit

To prevent reverse current, in case the battery is connected with reverse polarity, a diode was tested. This diode was placed in series with a resistor and current was supplied on the negative side of the diode. Then the current flowing through the resistor was checked on the oscilloscope. It was found that there was no current flowing through it.

3.2 Panel of Switches

3.2.1 Debounced Circuit for Switches

A circuit using a simple SR latch for debouncing was designed. This is shown in Figure 8. Oscilloscope tests were done using the debounced circuit and without it. The results found are shown in Figure 14. Debouncing was important as it could have sent many signals to the receiver on one push of a button.

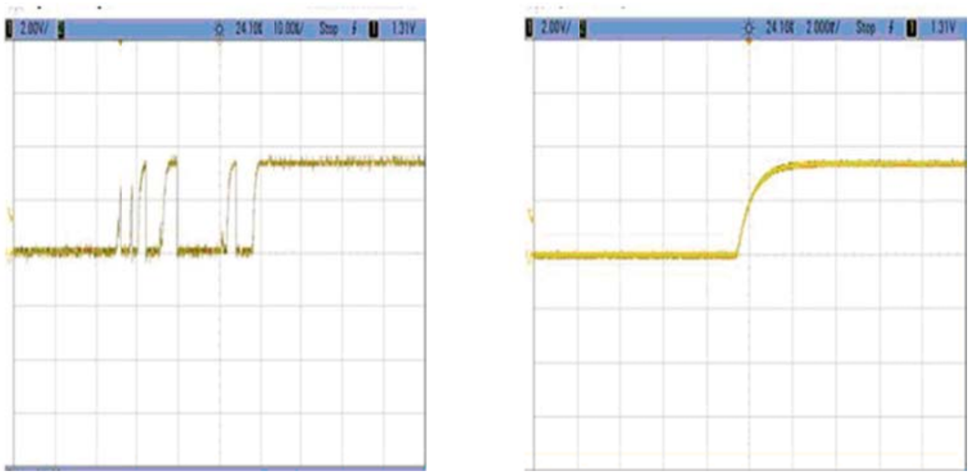


Figure 14: Before & After Debouncing

3.3 RF Transceiver

3.3.1 Accuracy test

This was verified by sending the signal multiple times and checking for failure at the receiving end. It was checked 50 times and it did not fail even once ensuring 100% transmission and retrieval.

3.3.2 Range test

This test was done by trial and error. The distance between the transmitter and the receiver was increased slowly till the signal no longer reached the receiver. This distance and time taken is shown below.

Distance (meters)	Time (sec)	Pass/Fail
20	< 1s	Pass
40	< 1s	Pass
60	< 1s	Pass
80	~ 1s	Pass
92	∞	Fail

3.3.3 Communication speed is 9600 bps

This speed was set using the CoolTerm programmer software and was tested by connecting the XBee and the microcontroller. Then 1kB of data was transferred and was found that the speed was around 9600bps.

3.4 Fingerprint Scanner

3.4.1 Registration & Deletion of Fingerprints

Multiple (~50) fingerprints were scanned and stored in the module successfully. We were able to delete all of the fingerprints as well. Control signals were sent to do the above and also an Emulator program was used to test out the same.

3.4.2 Verification of Registered Fingerprint

Different fingerprints were verified successfully after issuing a control signal from the microcontroller to the scanner. The controller received a control packet back letting it know if it was a successful scan or not. Based on the packet contents, we verified that the scan is successful and we enabled the panel of switches. The control signals are shown in Figures 6 & 7.

3.4.3 Signal Verification Terminal

At first we were using the emulator program to register and delete fingerprints using a USB connection to the computer. But after integrating the scanner with the controller, it was really important to look at if the right packet was being sent and received. Thus we set up a connection from the computer to an Arduino microcontroller to use as a debug tool between our controller and scanner, to figure out if our program was doing the right thing. Later we moved on to using the LEM100 Development kit to do the same.

3.5 Microcontroller

3.5.1 Interface with Scanner & XBee Modules

Multiple control packets for initialization, registration, verification and deletion were sent and the packets received back provided a successful result of the same. The UART interface is used to carry out this communication.

The same test was done with the XBee module. Control signals were successfully sent to the XBee which enabled it to send data over to the receiver which was verified by a simple LED test. For the final design we had the controller send an encrypted key over the transmitter, after a successful scan, which got validated at the receiver end.

3.5.2 Serial communication at 9600bps

Data packets were communicated to and from, between the controller and a computer to test the communication speeds. Multiple packets of 1kB were transmitted and checked for completion and speeds. Perfect packet retrieval at a speed of 9600bps was achieved.

3.6 Lock

3.6.1Pulse Width Test

To check which way the servo turns, a PWM signal was generated using the microcontroller. When the pulse width was 1.5 milliseconds it moved to a 90 degree position rotating clockwise. When the pulse width was 1 millisecond it moved back to the 0 degree position rotating anti-clockwise. This was all set up on the breadboard and tested using an oscilloscope to validate the right PWM signal.

4. Costs

4.1 Parts

Table 1: List of Parts [1]

Part	Manufacturer	Price/Unit	Quantity	Total
MSP430 Microcontroller	TI-430G2	$5.89	4	$23.56
Fingerprint Scanner	Integrated Biometrics-LEM100	$180	1	$180
XBee RF Transmitter/Receiver	Digi- 1mW Trace Antenna - Series 1 (802.15.4)	$22	3	$66
UA78M33 Voltage Regulator	TI	$1.43	6	$8.58
L7805 Voltage Regulator	TI	$1.5	3	$4.5
AA Battery Pack	Energizer	$2.95	2	$5.9
Breakout Board XBee	Sparkfun	$2.3	3	$6.9
Header Pins	ECE Store	$0.15	10	$1.5
LED	ECE Store	$0.10	4	$0.4
XBee Header Pins	Sparkfun	$1.2	8	$9.6
Servo	ServoCity	$5.25	2	$10.5
Deadbolts	Kwikset	$20.25	2	$40.5
Resistors	ECE Store	$0.3	10	$3
			Parts Total	**$360.94**

4.2 Labor

Table 2: Labor Breakdown [1]

Group Member	$/Hour	Hours/Week	Number of Weeks	Multiplier	Total/Person
Aashay	30	15	12	2.5	$13500
Akshay	30	15	12	2.5	$13500

4.3 Total Cost

Table 3: Finalized Total Cost [1]

Labor Cost	$27000
Parts Cost	$360.94
Total Cost	**$27360.94**

5. Conclusion

5.1 Accomplishments

The design worked as per what was proposed. The fingerprint scanner module was integrated with the microcontroller. The fingerprint scanner also successfully added additional users and was able to remove existing ones. The correct input was detected by the fingerprint scanner based on whether the users were registered. The microcontroller was able to send an encryption key via the XBee unit to the lock. On the receiving side the microcontroller was able to validate the correct encryption key and was able to generate the correct PWM output for the servo to turn clockwise such that it unlocks the deadbolt. The RF module could catch the encrypted signal up to a distance of 100m. The transmission and receiving of packets on the RF transceivers were 100% completed every time with very minimal chance of packet drop. Also, as the switches were debounced the RF module never transmitted multiple times on one button press.

5.2 Uncertainties

The main uncertainty we encountered was that the controller did not have an algorithm to send a different real time encryption key to the receiving side. This makes it a little easier to physically hack the device. Another uncertainty encountered was that the servo did not perform the lock function when the user pushed a button. The microcontroller sends a reverse PWM signal such that the servo turns anti-clockwise which locks the deadbolt. This we found later was a result of insufficient power delivered to the servo when the switch is pressed. When this function was performed on the breadboard with a bench power supply it worked perfectly and we were able to rotate the servo in both directions.

5.3 Ethical considerations

Table 4: IEEE Code of Ethics Relevance [1]

IEEE Code of Ethics	Relevance in Design
“1. to accept responsibility in making decisions consistent with the safety, health, and welfare of the public, and to disclose promptly factors that might endanger the public or the environment;”	The purpose of the project is to make a secure lock and key system such that the user can keep their important documents safe. This feature of the device helps the user feel safe about their belongings.
“3. to be honest and realistic in stating claims or estimates based on available data”	Only the authorized user should be able to open the lock and hence the accuracy of the device is important such that no one else tries to open the lock. While doing the project every data taken will be reported honestly even if it isn’t used.
“5. to improve the understanding of technology; its appropriate application, and potential consequences”	Once the project is done, the knowledge about MSP430G2, UART, LEM 100 scanner and the XBee RF transceiver would be gained. This knowledge would help understand these devices and would also help gain technical competence.
“6. to maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent	

limitations;”	
“9. to avoid injuring others, their property, reputation, or employment by false or malicious action;”	It should be made sure that the authorized user is the only one allowed to use the key and no one else would be able to. This would help avoid any kind of theft or loss of property.
“10. to assist colleagues and co-workers in their professional development and to support them in following this code of ethics.”	We will readily provide assistance to colleagues with their professional development and support them in their code of ethics.

5.4 Future work

One of the first things we would do would be to get a PCB for the lock and key in order for it to actually serve as a portable device that a user can carry around with ease. We already had it designed, but we couldn’t physically get it made in time for our demo. To make the design more hack proof, we can have an algorithm such that the RF module sends a different verification key every time the user wants to open the lock. This can be achieved by having a real time encryption key generated on the transmission and receive side with both sides being synchronized in order to do so. Electromechanical locks should be designed such that tampering with the locks would not be possible. The user should be able to use the key from long distances so that they can open the lock for someone else more easily if needed to be. Further multiple keys can be replicated for more than one user.

References

[1] Aashay Shah and Akshay Chanana, “Design Review – Smart Portable Key Team 45.” Available: http://courses.engr.illinois.edu/ece445/projects/spring2013/project45_design_review.pdf

[2] "LEM100 Hardware Development Manual."*Https://integratedbiometrics.zendesk.com/attachments/token/d0btaf2e6bsrvwt/?name=LEM100+Module+Hardware+Development+Manual+_En__Rev1.4.pdf*. Integrated Biometrics, n.d. Web.

[3] "XBee®/XBee-PRO® RF Modules - 802.15.4 - v1.xEx [2009.09.23]."*Http://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf*. Digi International Inc, 23 Sept. 2009. Web

[4] “TI MSP430G2 Data Sheet.” Available: http://www.ti.com/lit/ug/slau318c/slau318c.pdf. Web.

[5] "How Do Servos Work?" *How Do Servos Work?* Servo City, n.d. Web. 15 Apr. 2013. <http://www.servocity.com/html/how_do_servos_work_.html>.

[6] “MSP430 Hardware tools Users Guide.” http://www.ti.com/lit/ug/slau278l/slau278l.pdf. Web.

Appendix A Requirement and Verification Table

Module	Requirements	Verification Procedure	
1. Fingerprint Scanner	1.1. Scanner management system should be able to record, detect existing and delete stored fingerprints.	1.1 The scanner will be tested using the J3 UART connection on the main board. The scanner will interface with the controller using this connection.	Y
	a. Communication interfaces should be initially properly working	a. Check to ensure connection between module and device are well connected. • Check input power to module ~ 3.3 V • Check LED blinking of module after turning on the module • Check serial communication interface LED of module is blinking.	Y
	b. Serial communication speed should be set optimally for synchronization between scanner and controller.	b. Send 1 kB of data and check the rate of transfer.	Y
	c. Store new users that will be able to validate access to the key	c. To start scanning MD_START_CAPTURE is sent to the module. Multiple users (~5-10) will be added by using the MD_ENROLL_FP command and then checking for these registered users in the in-built memory as shown in Figure 2.	Y

	d. Unauthorized users should be denied access	d. Users will be identified using the MD_IDENTIFY_USER command and verified by sending MD_VERIFY_USER.	Y
	e. Existing users should be removed on request	e. Users will be deleted using the MD_DELETE_FP command and checking if the digital output associated with it receives a high signal.	Y
	f. On a successful scan scanner should be able to send a valid data key to controller for verification	f. The scanner will send the signal to the controller via the UART interface. This will be done about 10-20 times, to check if the controller validates the data sent by the scanner, using the LED on the controller to signal for valid data received.	Y
2. Power Supply (Battery and Voltage Regulator)	2.1 Check if voltage regulator gives the correct output that is needed for the fingerprint scanner.	2.1 To test the voltage regulator (MIC5219) a small circuit would be made and different resistor values would be used to see if the given output is the required one (3.3± 0.5 V). These results would be checked using an oscilloscope. This voltage regulator would be used for the microcontroller and the fingerprint scanner.	Y
	2.2 Supply rated voltage and current. a. 3.3 ± 0.5 V	2.2 a. Using an oscilloscope, the voltage will be tested on the power supply. This is very important to check every time as the fingerprint scanner cannot take voltage above 3.6V or less than 2.7 V.	Y

	b. Current doesn't exceed 118mA	b. This precaution is very important because the fingerprint scanner is really expensive. A fuse will be used here so that it prevents it from getting damaged. Also, a diode will be kept such that the scanner doesn't get damaged in case the battery is reversed.	Y
3. Microcontroller (Key)	3.1 Interface with the fingerprint scanner works with synchronization	3.1 The scanner will be connected to the controller serially.	Y
	a. Serial communication speed set at 9600 bps at all times	a. Controller will be set to a default serial communication speed of 9600 bps using the Code Composer software. It will be verified as described in 1.1 (b).	Y
	b. Verification key received on a successful fingerprint scan	b. A test program will be written to ensure that the correct verification key is sent. This will be tested 20 times just to see if it works correctly. When the correct key is received an LED will blink.	Y
	3.2 Detects the change of state of any of the switches on the switch panel a. Panel should be debounced	3.2 Panel switch be connected and tested through the digital ports of the controller a. We will test the switch interface by testing the pins they are connected to on the controller. Every time we switch one on, we will have a LED signal that verifies this interface.	Y
	3.3 Should be able to send a correct signal via the XBee	3.3 The XBee DIN pin will receive information from the	Y

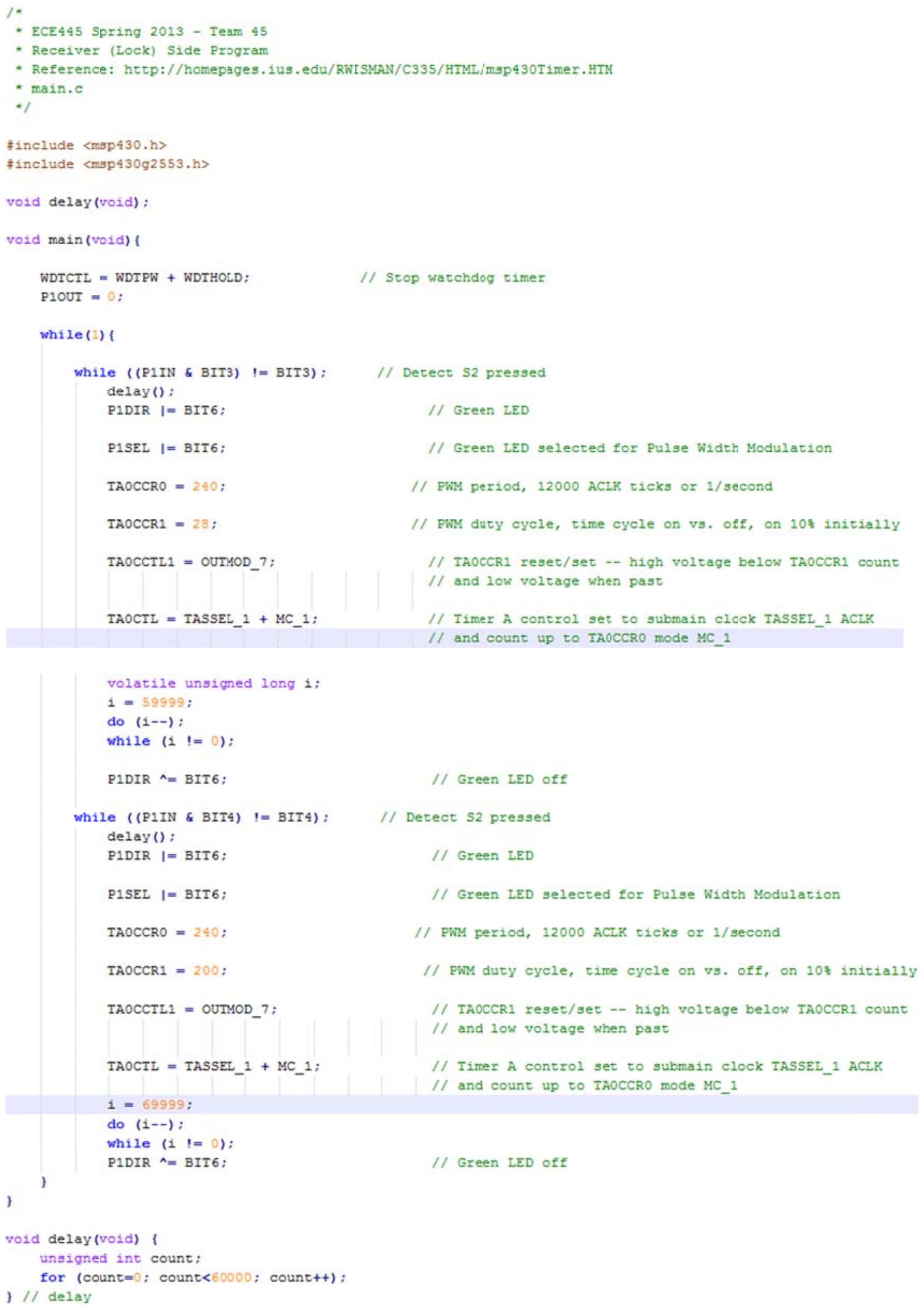
	module to the respective lock after verification from scanner completes and one of the switches is on. a. Wait for the user to pick which lock to be open b. Once a lock is picked, sent signal through the XBee using the UART interface should be to the right lock	controller necessary to signal the respective lock a. This waiting period will be done by programming the controller with a wait function till one of the pins connected to the controller sees a high signal. We will also incorporate a time out so as to not keep it always on once the scanner is successful. Time out will be checked by leaving the system alone and checking if one of the LEDs starts to blink. b. After receiving the signal from one of the switches, send packet to XBee controller based on the XBee datasheet so as to transfer to the right lock. Tested fifty times and will check for right lock by using the controller on the receive side. We will store the message on the controller and read its memory using software for validation of the message received. These tests will ensure that the controller works well with each component on the key.	 Y Y
4. Microcontroller (Lock)	4.1 Receives a valid signal from the XBee receiver and then signals the electro-mechanical lock	4.1 The RF receiver will have to behave the same as the transmitter interfaces with the key controller. This is tested	Y

		the same way as described in section 3.3. Once the signal is validated, we can output a high initially to test an LED so as to know that the signal passes the verification and the lock can now be unlocked. Multiple tests will be done to check the accuracy of this unlocking mechanism.	
5 Panel of Switches	5.1 The switches need to be debounced such that it resets after 1 second.	5.1 This will be done using a simple debouncing circuit and will be checked using an oscilloscope by pressing the button thirty or so times. The oscilloscope should show that the button was hit thirty times with perfect accuracy. Also, the oscilloscope will give a high output when the button is pressed and it resets after a particular time so that the user can open another lock by scanning the fingerprint and activating the panel of switches.	Y
	5.2 Ensure that the correct lock receives signal when the button is pressed.	5.2 When the button is pressed three LED's will be put near the receiver and whenever the corresponding button is pressed the correct LED should light up.	Y
6 RF Transceiver	6.1 Serial connection works perfectly between XBee and the controller	6.1 This will be verified by connecting the transmitting side and receiving side to different computers (using UART) and checking by sending 2kB data and then receiving it to ensure it works 90% of the time. It will be tested 30 times and error calculation will be done.	Y
	6.2 To ensure that it activates when the correct	6.2 To ensure this, a test signal will be received. LED's will be	Y

	frequency is received.	placed such that when the correct signal is received, they would turn on and when an incorrect signal is sent it stays off. This whole process will be handled by the controller and signals will be sent manually to check on its verification.	
	6.3 Range test. Should be able to communicate up to a 30m distance between the transmitter and receiver.	6.3 The range would be checked by sending the signal in different situations where the distance between the transmitted and received side vary. We will send sample test data using the controller and we should be able to receive data with 100% integrity up to a distance of 30m.	Y
	6.4 All interfaces work at a communication speed of 9600 bps. The transceivers should do the same.	6.4 This will be done the same way as described in Section 1.1 (b) by connecting the controller and XBee together to exchange data at the defined rate.	Y
7 Lock	7.1 Ensure enough power is there so that the lock opens when it has to.	7.1 Send a PWM signal via the controller for clockwise rotation to unlock	Y
	7.2 It snaps the lock when signal is received.	7.2 Different voltages from the range of 5V-20V would be given to the lock and would be checked for which ones it opens smoothly.	Y
	7.3 The lock closes when the user hits a push button.	7.3 Reverse PWM for anti-clockwise rotation using the 9V battery and regulators.	N

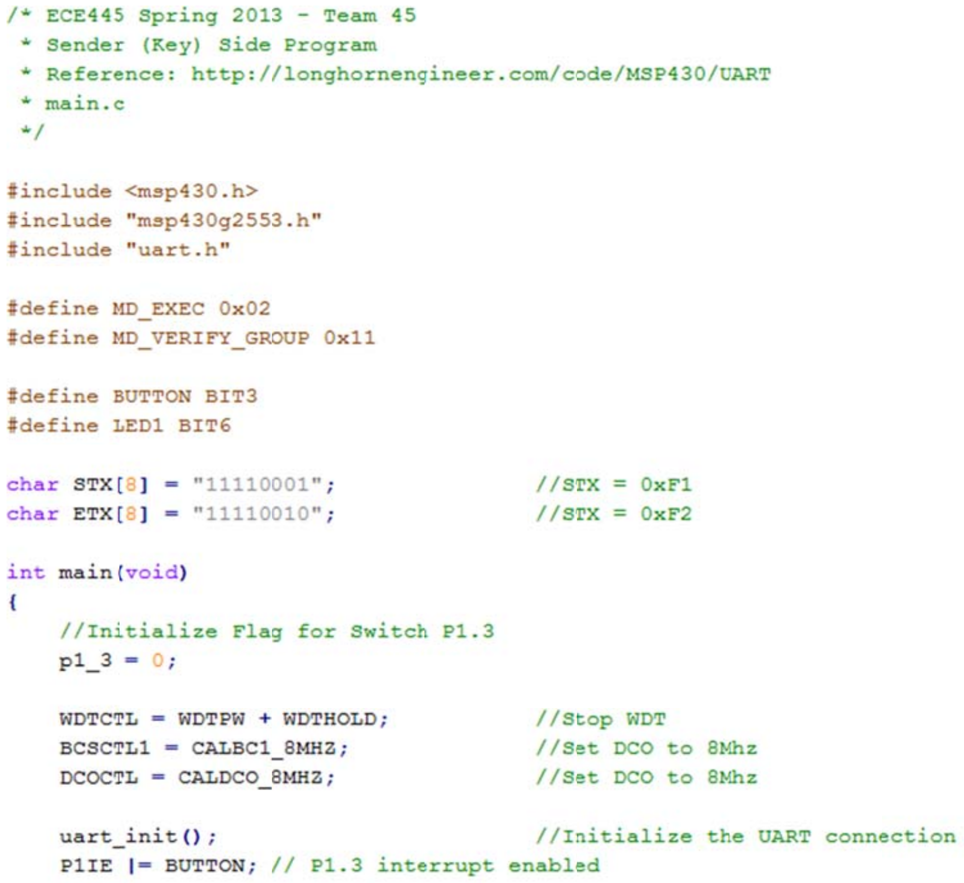
Appendix B   Microcontroller UART Code

B.1 Receiver Program

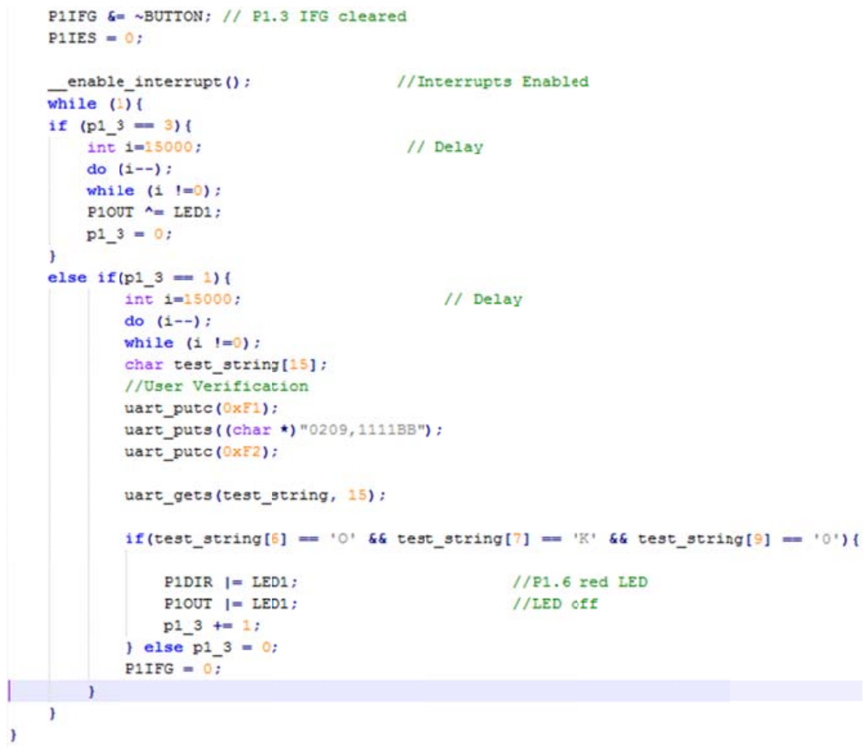


```
/*
 * ECE445 Spring 2013 - Team 45
 * Receiver (Lock) Side Program
 * Reference: http://homepages.ius.edu/RWISMAN/C335/HTML/msp430Timer.HTM
 * main.c
 */

#include <msp430.h>
#include <msp430g2553.h>

void delay(void);

void main(void){

    WDTCTL = WDTPW + WDTHOLD;              // Stop watchdog timer
    P1OUT = 0;

    while(1){

        while ((P1IN & BIT3) != BIT3);      // Detect S2 pressed
            delay();
            P1DIR |= BIT6;                      // Green LED

            P1SEL |= BIT6;                      // Green LED selected for Pulse Width Modulation

            TA0CCR0 = 240;                     // PWM period, 12000 ACLK ticks or 1/second

            TA0CCR1 = 28;                      // PWM duty cycle, time cycle on vs. off, on 10% initially

            TA0CCTL1 = OUTMOD_7;                // TA0CCR1 reset/set -- high voltage below TA0CCR1 count
                                                // and low voltage when past

            TA0CTL = TASSEL_1 + MC_1;           // Timer A control set to submain clock TASSEL_1 ACLK
                                                // and count up to TA0CCR0 mode MC_1


            volatile unsigned long i;
            i = 59999;
            do (i--);
            while (i != 0);

            P1DIR ^= BIT6;                      // Green LED off

        while ((P1IN & BIT4) != BIT4);      // Detect S2 pressed
            delay();
            P1DIR |= BIT6;                      // Green LED

            P1SEL |= BIT6;                      // Green LED selected for Pulse Width Modulation

            TA0CCR0 = 240;                     // PWM period, 12000 ACLK ticks or 1/second

            TA0CCR1 = 200;                      // PWM duty cycle, time cycle on vs. off, on 10% initially

            TA0CCTL1 = OUTMOD_7;                // TA0CCR1 reset/set -- high voltage below TA0CCR1 count
                                                // and low voltage when past

            TA0CTL = TASSEL_1 + MC_1;           // Timer A control set to submain clock TASSEL_1 ACLK
                                                // and count up to TA0CCR0 mode MC_1
            i = 69999;
            do (i--);
            while (i != 0);
            P1DIR ^= BIT6;                      // Green LED off
    }
}

void delay(void) {
    unsigned int count;
    for (count=0; count<60000; count++);
} // delay
```

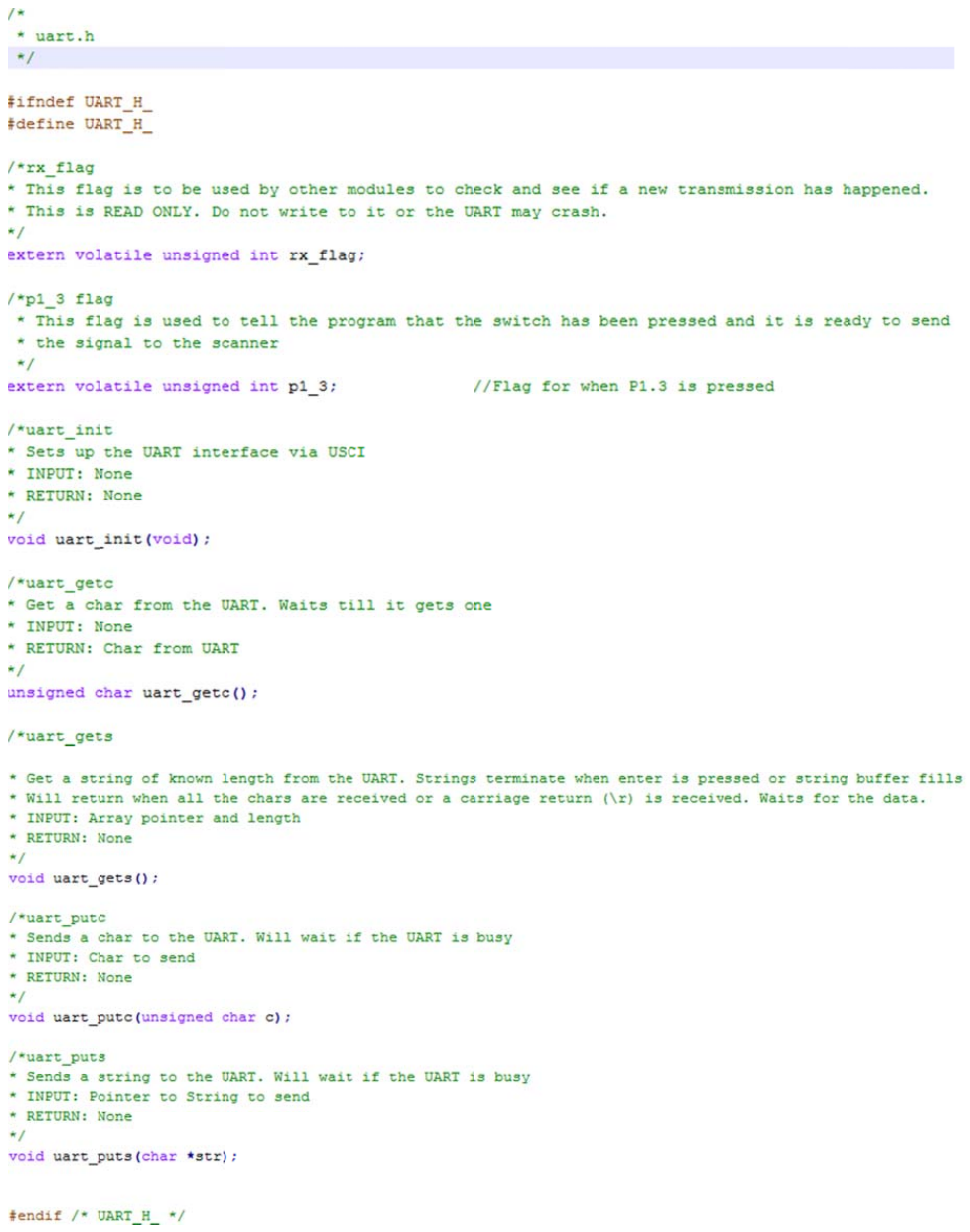
B.2 Sender Program



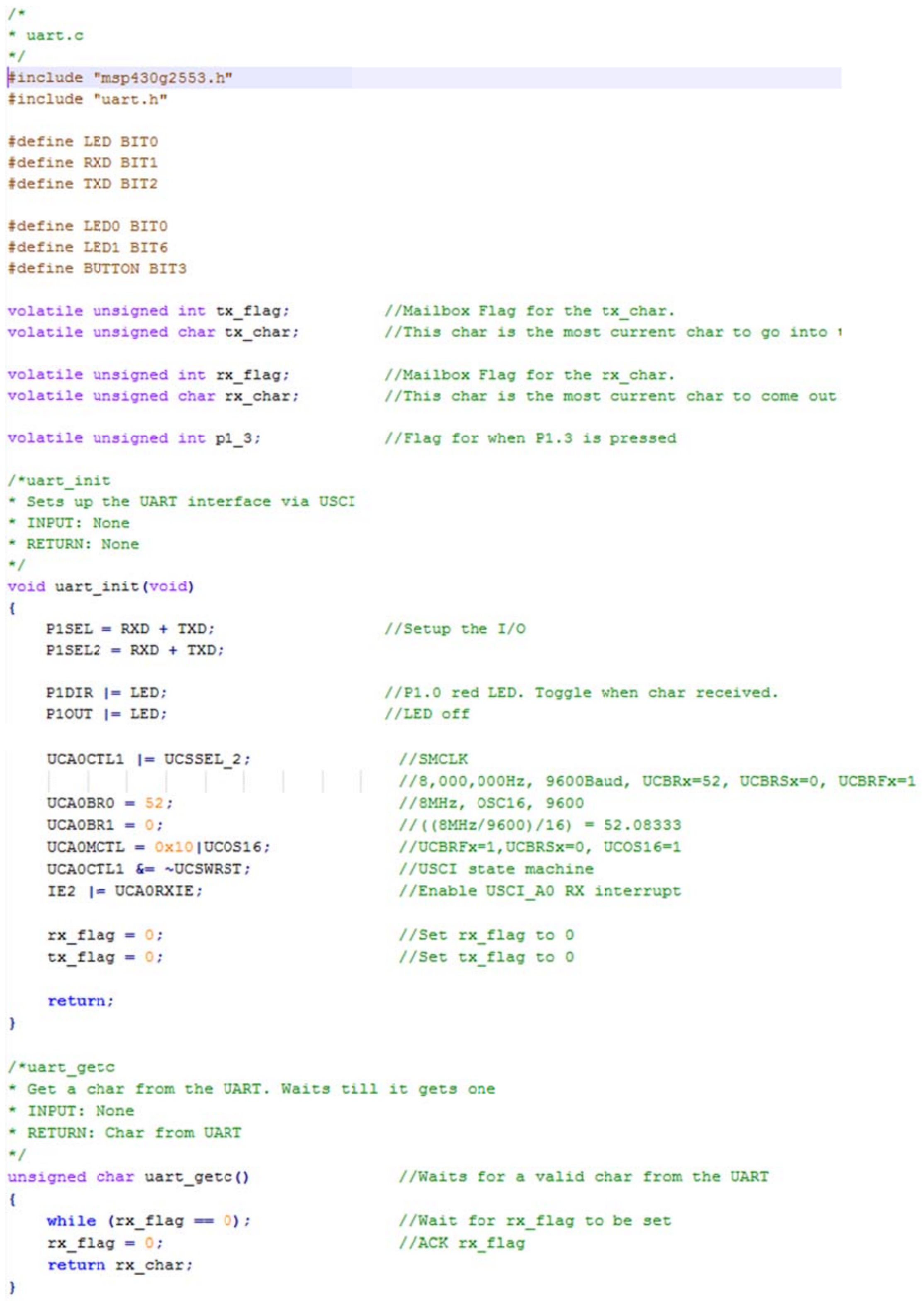
```
/* ECE445 Spring 2013 - Team 45
 * Sender (Key) Side Program
 * Reference: http://longhornengineer.com/code/MSP430/UART
 * main.c
 */

#include <msp430.h>
#include "msp430g2553.h"
#include "uart.h"

#define MD_EXEC 0x02
#define MD_VERIFY_GROUP 0x11

#define BUTTON BIT3
#define LED1 BIT6

char STX[8] = "11110001";                   //STX = 0xF1
char ETX[8] = "11110010";                   //STX = 0xF2

int main(void)
{
    //Initialize Flag for Switch P1.3
    p1_3 = 0;

    WDTCTL = WDTPW + WDTHOLD;               //Stop WDT
    BCSCTL1 = CALBC1_8MHZ;                  //Set DCO to 8Mhz
    DCOCTL = CALDCO_8MHZ;                   //Set DCO to 8Mhz

    uart_init();                            //Initialize the UART connection
    P1IE |= BUTTON; // P1.3 interrupt enabled
```



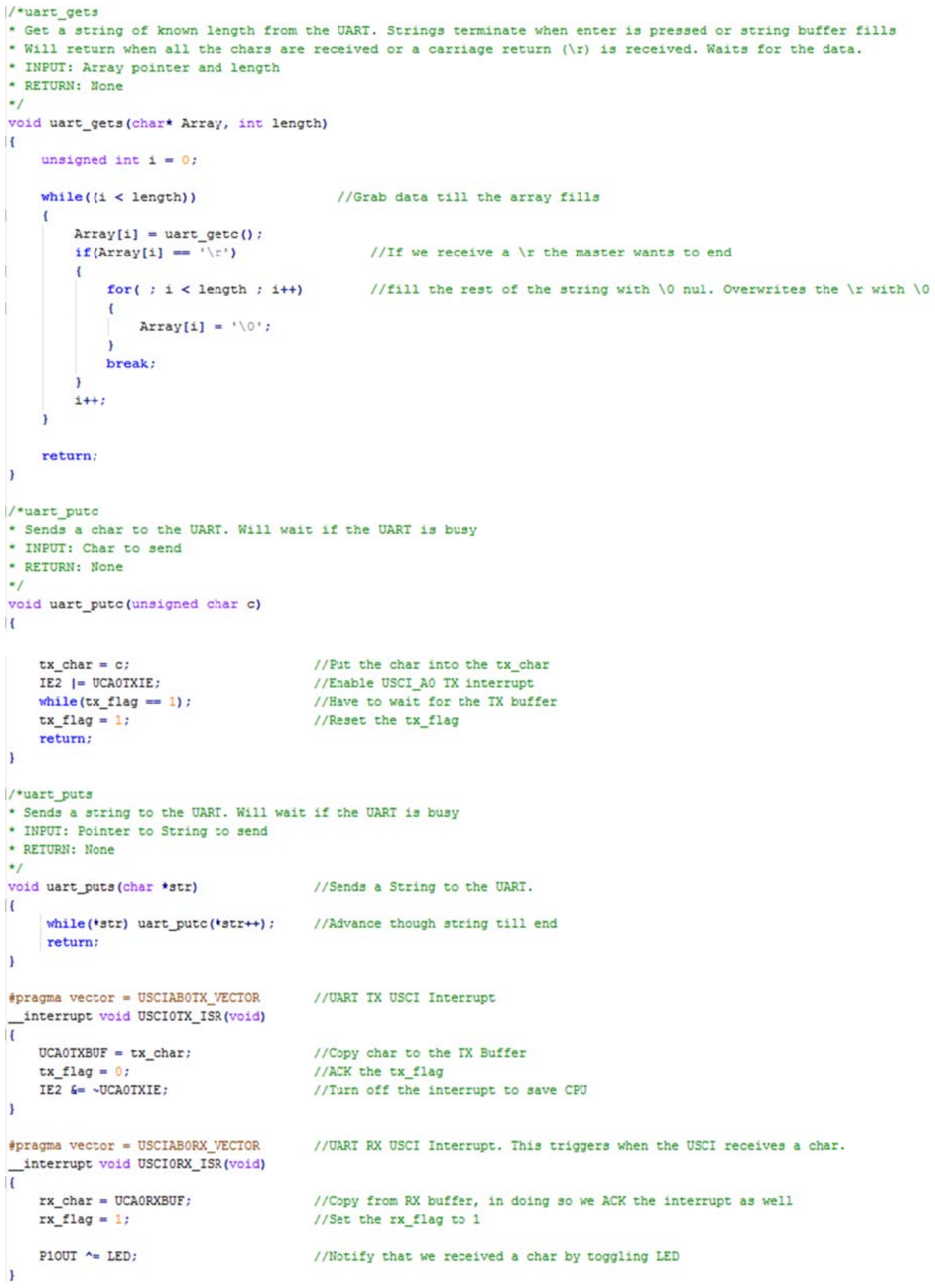
```
    P1IFG &= ~BUTTON; // P1.3 IFG cleared
    P1IES = 0;

    __enable_interrupt();               //Interrupts Enabled
    while (1){
    if (p1_3 == 3){
        int i=15000;                     // Delay
        do (i--);
        while (i !=0);
        P1OUT ^= LED1;
        p1_3 = 0;
    }
    else if(p1_3 == 1){
            int i=15000;                     // Delay
            do (i--);
            while (i !=0);
            char test_string[15];
            //User Verification
            uart_putc(0xF1);
            uart_puts((char *)"0209,1111BB");
            uart_putc(0xF2);

            uart_gets(test_string, 15);

            if(test_string[6] == 'O' && test_string[7] == 'K' && test_string[9] == '0'){

                P1DIR |= LED1;                   //P1.6 red LED
                P1OUT |= LED1;                   //LED off
                p1_3 += 1;
            } else p1_3 = 0;
            P1IFG = 0;
        }
    }
}
```



```c
/*
 * uart.h
 */

#ifndef UART_H_
#define UART_H_

/*rx_flag
* This flag is to be used by other modules to check and see if a new transmission has happened.
* This is READ ONLY. Do not write to it or the UART may crash.
*/
extern volatile unsigned int rx_flag;

/*p1_3 flag
 * This flag is used to tell the program that the switch has been pressed and it is ready to send
 * the signal to the scanner
 */
extern volatile unsigned int p1_3;              //Flag for when P1.3 is pressed

/*uart_init
* Sets up the UART interface via USCI
* INPUT: None
* RETURN: None
*/
void uart_init(void);

/*uart_getc
* Get a char from the UART. Waits till it gets one
* INPUT: None
* RETURN: Char from UART
*/
unsigned char uart_getc();

/*uart_gets

* Get a string of known length from the UART. Strings terminate when enter is pressed or string buffer fills
* Will return when all the chars are received or a carriage return (\r) is received. Waits for the data.
* INPUT: Array pointer and length
* RETURN: None
*/
void uart_gets();

/*uart_putc
* Sends a char to the UART. Will wait if the UART is busy
* INPUT: Char to send
* RETURN: None
*/
void uart_putc(unsigned char c);

/*uart_puts
* Sends a string to the UART. Will wait if the UART is busy
* INPUT: Pointer to String to send
* RETURN: None
*/
void uart_puts(char *str);


#endif /* UART_H_ */
```



```c
/*
* uart.c
*/
#include "msp430g2553.h"
#include "uart.h"

#define LED BIT0
#define RXD BIT1
#define TXD BIT2

#define LED0 BIT0
#define LED1 BIT6
#define BUTTON BIT3

volatile unsigned int tx_flag;          //Mailbox Flag for the tx_char.
volatile unsigned char tx_char;         //This char is the most current char to go into

volatile unsigned int rx_flag;          //Mailbox Flag for the rx_char.
volatile unsigned char rx_char;         //This char is the most current char to come out

volatile unsigned int p1_3;             //Flag for when P1.3 is pressed

/*uart_init
* Sets up the UART interface via USCI
* INPUT: None
* RETURN: None
*/
void uart_init(void)
{
    P1SEL = RXD + TXD;                  //Setup the I/O
    P1SEL2 = RXD + TXD;

    P1DIR |= LED;                       //P1.0 red LED. Toggle when char received.
    P1OUT |= LED;                       //LED off

    UCA0CTL1 |= UCSSEL_2;                //SMCLK
                                         //8,000,000Hz, 9600Baud, UCBRx=52, UCBRSx=0, UCBRFx=1
    UCA0BR0 = 52;                        //8MHz, OSC16, 9600
    UCA0BR1 = 0;                         //((8MHz/9600)/16) = 52.08333
    UCA0MCTL = 0x10|UCOS16;              //UCBRFx=1,UCBRSx=0, UCOS16=1
    UCA0CTL1 &= ~UCSWRST;                //USCI state machine
    IE2 |= UCA0RXIE;                     //Enable USCI_A0 RX interrupt

    rx_flag = 0;                         //Set rx_flag to 0
    tx_flag = 0;                         //Set tx_flag to 0

    return;
}

/*uart_getc
* Get a char from the UART. Waits till it gets one
* INPUT: None
* RETURN: Char from UART
*/
unsigned char uart_getc()                //Waits for a valid char from the UART
{
    while (rx_flag == 0);                //Wait for rx_flag to be set
    rx_flag = 0;                         //ACK rx_flag
    return rx_char;
}
```



```
/*uart_gets
* Get a string of known length from the UART. Strings terminate when enter is pressed or string buffer fills
* Will return when all the chars are received or a carriage return (\r) is received. Waits for the data.
* INPUT: Array pointer and length
* RETURN: None
*/
void uart_gets(char* Array, int length)
{
    unsigned int i = 0;

    while((i < length))                  //Grab data till the array fills
    {
        Array[i] = uart_getc();
        if(Array[i] == '\r')                 //If we receive a \r the master wants to end
        {
            for( ; i < length ; i++)         //fill the rest of the string with \0 nul. Overwrites the \r with \0
            {
                Array[i] = '\0';
            }
            break;
        }
        i++;
    }

    return;
}

/*uart_putc
* Sends a char to the UART. Will wait if the UART is busy
* INPUT: Char to send
* RETURN: None
*/
void uart_putc(unsigned char c)
{

    tx_char = c;                        //Put the char into the tx_char
    IE2 |= UCA0TXIE;                    //Enable USCI_A0 TX interrupt
    while(tx_flag == 1);                //Have to wait for the TX buffer
    tx_flag = 1;                        //Reset the tx_flag
    return;
}

/*uart_puts
* Sends a string to the UART. Will wait if the UART is busy
* INPUT: Pointer to String to send
* RETURN: None
*/
void uart_puts(char *str)               //Sends a String to the UART.
{
     while(*str) uart_putc(*str++);     //Advance though string till end
     return;
}

#pragma vector = USCIAB0TX_VECTOR       //UART TX USCI Interrupt
__interrupt void USCI0TX_ISR(void)
{
    UCA0TXBUF = tx_char;                //Copy char to the TX Buffer
    tx_flag = 0;                        //ACK the tx_flag
    IE2 &= ~UCA0TXIE;                   //Turn off the interrupt to save CPU
}

#pragma vector = USCIAB0RX_VECTOR       //UART RX USCI Interrupt. This triggers when the USCI receives a char.
__interrupt void USCI0RX_ISR(void)
{
    rx_char = UCA0RXBUF;                //Copy from RX buffer, in doing so we ACK the interrupt as well
    rx_flag = 1;                        //Set the rx_flag to 1

    P1OUT ^= LED;                       //Notify that we received a char by toggling LED
}
```