– ECE 09.321: Final Project –

Passcode Protected Smart Safe

Ryan Boylan, Alexandra Jackson, Michael Johns
Alexander Steel, Ben Zalewski
Rowan University

December 16, 2019

1 Design Overview

In this high level design, an MSP430 family micro-controller is used to control a smart safe. The safe is protected with a four digit passcode as well as a locking mechanism using a solenoid. The passcode can be set and changed using UART communication, with these options programmed into the micro-controller. This device also includes a safety feature that sends out a high frequency tone upon the input of an incorrect passcode.

1.1 Design Features

These are the design features:

- Keypad interfaced with laptop to input and change the passcode
- Button locking mechanism using a solenoid
- Signal for incorrect passcode

1.2 Featured Applications

- Protection of valuable items
- Smart locking

1.3 Design Resources

The following link is to the GitHub repository where the code is stored.

- Github Repository
  https://github.com/Intro-To-Embedded-Systems-RU09342/final-project-smart_safe.

1.4 Block Diagram

Figure 1 shows a simple block diagram consisting of the basic components of this system and how they work together.

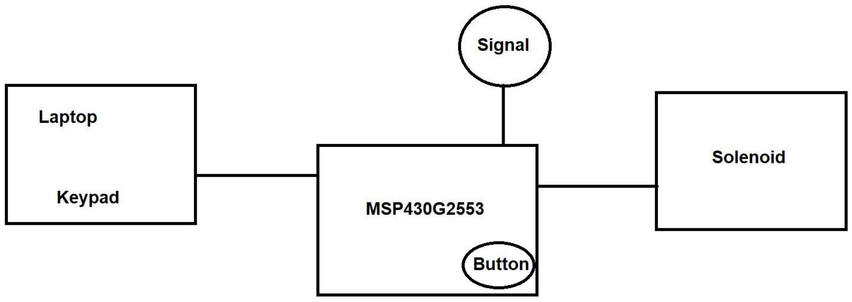


Figure 1: System Block Diagram

1.5 Board Image

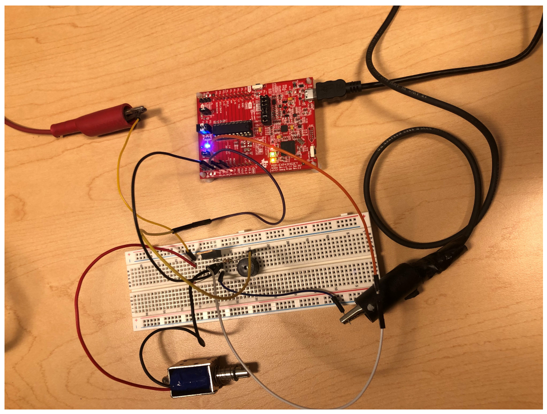


Figure 2: Board Image

2 Key System Specifications

The following table shows the voltage specifications of the devices used in this project.

Parameter	Specifications	Details
MSP430G2553	5V power	Supply via USB
Solenoid	6V Power, 800mA, 10mm throw	Supply via Power Supply
UART baudrate	9600	Set using RealTerm

3 System Description

Storing valuable items knowing that they will be safe is a problem faced by everyone, whether it be documents, digital files, or physical pieces. Passcode protected devices are a way to combat this issue. Many use the passcode protection on their cellphones and laptops to protect their digital data, and many use physical locks or chains to hide important pieces away from the world.

This smart safe combines both ideas, using both a digital passcode and a physical place to store items. The passcode can be modified and changed as much as the owner wants by using simple UART communication straight from their laptop. A solenoid lock moves into and out of place upon proper entry of the passcode. The MSP430G2553 board controls the passcode inputs, the locking mechanism, and the wrong passcode alert tone for the safe.

3.1 Detailed Block Diagram

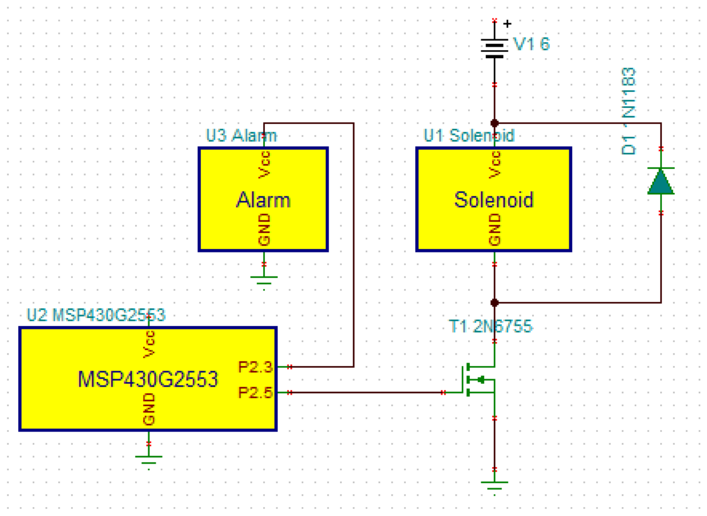


Figure 3: Detailed Block Diagram

3.2 Highlighted Devices

- MSP430G2553
- Solenoid
- Piezoelectric Buzzer

3.3 MSP430G2553

The MSP430G2553 was chosen for this project due to its easily compatible systems and easy to work with design, which includes multiple 16-bit timers, 24 accessible pins, ultra-low power, and UART communication capability. This device utilizes C code to control the solenoid locking, the passcode changing capability, the fingerprint scanner, and the signal. The on board button was used for the button lock, and the on board ground and output pins were used to communicate with the hardware devices used in this design.

3.4 Solenoid

The solenoid chosen for this system was the JF-0630B model due to its small size, its simple configuration, and low cost. This solenoid requires a 6V power, and has a rated current of 800mA. It also features a 10mm thrust. This solenoid has only one wire for power, and one for ground, making it easy to connect and control in the system by using a Low Side Switch.

3.5 Piezoelectric Buzzer

This component was inputted into the system in order to indicate an incorrect passcode. The model used is the PS1420P02CT, and includes only two pins, which can be interchanged at any point. This device also uses very low power.

4 SYSTEM DESIGN THEORY

The design of this system is based off of both hardware and software components. The heart of the system is the MSP430G2553, which drives the system using C code. The code is divided into four main parts; the communication capability, passcode input and change control, solenoid locking control, and piezoelectric buzzer control.

The communication is done through UART, which requires a laptop to send a four digit string of hexadecimal values to the board. The board will process these digits and decide if they match the passcode. The passcode input and change control is done in code by presetting the passcode to ”1234”. The passcode is inputted through UART using a five byte hexadecimal string. Adding a 0xFF to the end of the hexadecimal string will allow for the passcode to be changed. Any other byte at the end of the string will keep the passcode the same.

The solenoid is programmed to lock automatically when the code is run. It will only unlock if the correct passcode is inputted. The piezoelectric buzzer is coded to sound an alert tone whenever the incorrect passcode is inputted, and stay silent when the correct passcode is inputted. The following sections explain in detail how the code works, as well as how the hardware components interface with the rest of the system.

4.1 Pin Setup

The output pins on the micro-controller exist to communicate with the off board circuits. The code is set up so that each of the functionalities can be outputted to its respective circuit. The code consists of three main parts; the UART communication for inputting and changing the passcode, the solenoid locking control, and the piezo buzzer system. The UART communication can be done through USB, so it does not require anything to output or input to the transmit and receive pins. The solenoid is sent to pin P2.5, and the piezo is set to pin P2.3. The on board button is also connected to the solenoid pin, and locks the solenoid when pressed.

Each of the pins on the G2 board come with preset functionalities. Many pins correspond to different timers, some are connected to clocks, and some are able to work with multiple different types of outputs. This system does not require timers or special pin functions, so it was not extremely important which pin was chosen for which function. Figure 4 shows each of the pins that were utilized in this system.

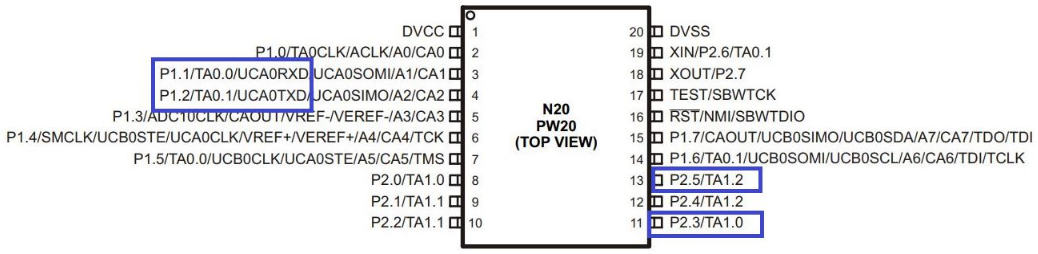

N20
PW20
(TOP VIEW)
DVCC 1
P1.0/TA0CLK/ACLK/A0/CA0 2
P1.1/TA0.0/UCA0RXD/UCA0SOMI/A1/CA1 3
P1.2/TA0.1/UCA0TXD/UCA0SIMO/A2/CA2 4
P1.3/ADC10CLK/CAOUT/VREF-/VEREF-/A3/CA3 5
P1.4/SMCLK/UCB0STE/UCA0CLK/VREF+/VEREF+/A4/CA4/TCK 6
P1.5/TA0.0/UCB0CLK/UCA0STE/A5/CA5/TMS 7
P2.0/TA1.0 8
P2.1/TA1.1 9
P2.2/TA1.1 10
20 DVSS
19 XIN/P2.6/TA0.1
18 XOUT/P2.7
17 TEST/SBWTCK
16 RST/NMI/SBWTDIO
15 P1.7/CAOUT/UCB0SIMO/UCB0SDA/A7/CA7/TDO/TDI
14 P1.6/TA0.1/UCB0SOMI/UCB0SCL/A6/CA6/TDI/TCLK
13 P2.5/TA1.2
12 P2.4/TA1.2
11 P2.3/TA1.0

Figure 4: MSP430G2553 Pins Utilized in this System

4.2 UART Communication

The G2 micro-controller comes with Universal Asynchronous Receiver/Transmitter (UART) communication capability. Using a software called RealTerm, passcode digits are able to be sent to the board from a laptop or PC device.

The UART communication is set up using P1.1 as the receiving output and P1.2 as the transmitting output. The baudrate is then set to 9600, which means it is capable of transmitting 9600 bits per second. Passcode digits are sent through UART using hexadecimal byte format.

4.3 Input and Change Passcode Control

When a user inputs a passcode, it is transmitted through UART into the micro-controller, which runs the code to process the passcode and decide the correct course of action. The code consists of some preset arrays that are compared as new digits are received through UART. The preset passcode is an array of size 4 called ”combo”, which is filled with the digits ”1234”. The ”combo” array is compared to the user input using if statements. The user input is an array of size 9 called ”input”, and is initially set to [0,0,0,0,0,0,0,0,0]. A counter variable is set to an initial value of 0, and it increases by 1 every time the UART receives a hexadecimal byte.

The fifth byte in the ”input” array is the byte that determined if the user wants to change the passcode. A variable called ”reset” is given a value of 255, which is equivalent to the hexadecimal FF. If the fifth byte in the user input is sent as FF, the user may change the passcode. Any other value as the fifth byte will be processed as a normal input, and open or lock the safe as needed. When the ”counter” variable reaches 9, which is the maximum size of the ”input” array, it will check that the fifth byte is FF and then assign the next four bytes to be the new passcode. Table 1 shows these variables and their presets.

Variable	Type	Initial Value	Function
input	9 digit array	{0,0,0,0,0,0,0,0,0}	user input
combo	4 digit array	{1,2,3,4}	correct passcode
counter	integer	0	counts number of inputted bytes
reset	constant integer	255	if fifth byte is 255 (FF), change passcode

Table 1: Passcode Presets within System Code

4.4 Solenoid Locking Control

The code for this system runs through a series of if statements that decide the status of the solenoid lock. If the passcode is correct and the user has opted not to change the passcode, the solenoid unlocks. If the passcode is correct and the user has opted to change the passcode, the solenoid remains locked until the passcode is changed and then inputted. If the on board button is pressed, the solenoid locks. In every other scenario, the solenoid locks or remains locked.

The solenoid is set to pin P2.5, which is referenced throughout the code. Within each of the if statements, there is a line which states if pin P2.5 is on or off. If the solenoid is to lock, the line of code is written as P2OUT |= BIT5. This is used instead of P2OUT = BIT5, as that would overwrite all of the other bits in P2OUT, rather than just changing the 6th bit to 1. If the solenoid is to unlock, it is written P2OUT &= ∼BIT5, to set only the 6th bit to zero.

4.5 Solenoid Convert-o-Box: Low Side Switch

Since the solenoid operates at a voltage of 6V, it requires a convert-o-box to regulate the voltage. The convert-o-box used is called a low side switch, which regulates current by using a MOSFET as a switch. This was done using a Silicon carbide power

MOSFET with the source to ground, drain to load, and gate to supply.[1] The MOSFET used is shown in Figure 5.

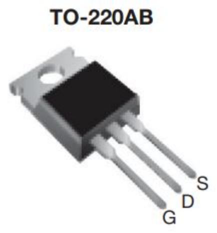


Figure 5: SiC MOSFET

These types of MOSFETs are known for their extremely fast switching and low driving power. They are different from regular MOSFETS in that their gate and drain pins are switched, as seen in the figure above. This allows for the device to support higher voltages and currents running through it.[2]

The low side switch is constructed by connecting the ground wire of the solenoid to the drain of the MOSFET, then connecting the power wire to the gate and running 6V through. The gate also connects to the solenoid's corresponding pin on the MSP430G2553 micro-controller. The source is connected directly to the common ground. The low side switch construction is shown in Figure 6.

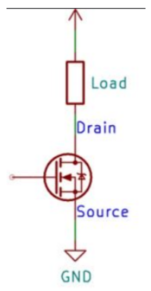


Figure 6: Low Side Switch

4.6 Piezoelectric Buzzer

This device is a small buzzer that sends out a tone when programmed a certain way. In this system, it is wired to pin P2.3, and beeps when an incorrect passcode is inputted. It is constructed by connecting either of the pins to ground, and the other pin to P2.3 on the MSP430 board. It is shown in Figure 7.

Figure 7: Piezoelectric Buzzer

5 Getting Started- Hardware

The following section details the hardware components of this system and how to construct them before testing the device. The hardware pieces included are the solenoid and the piezoelectric buzzer.

5.1 Solenoid and Piezo Setup

The solenoid and piezo can be simply connected to the micro-controller using simple circuits. The solenoid requires a low side switch, as discussed in Section 4.5. The piezo requires one resistor, a pin to common ground, and a pin to the board, as discussed in Section 4.6. The breadboard construction of these circuits is shown in Figure 8.

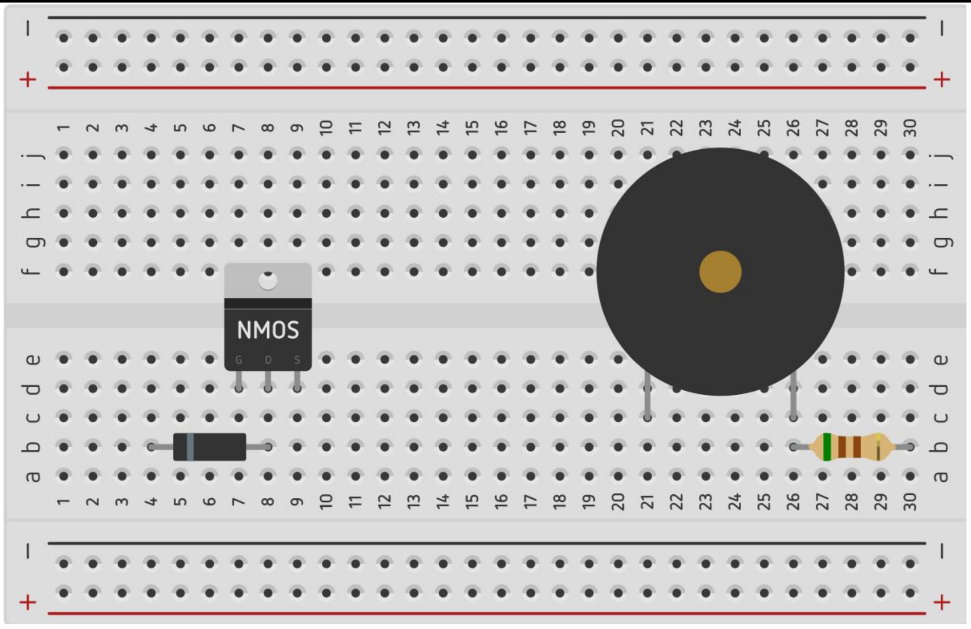


Figure 8: Breadboard Circuit Setup

6 Getting Started- Software

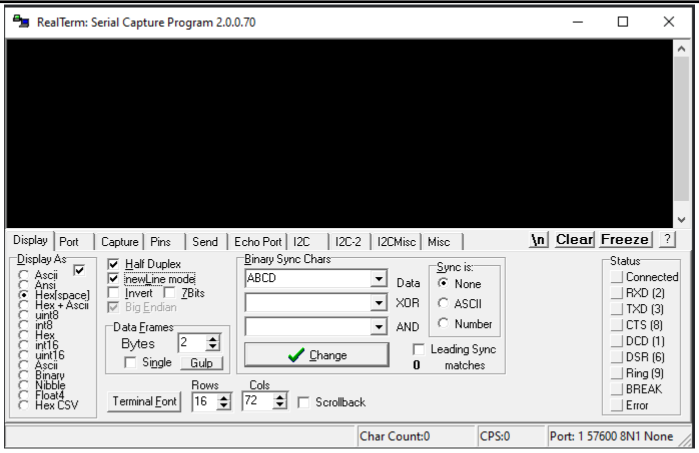
The following section details the software needed in order for this device to be used properly.

6.1 Code Composer

Download the Code Composer application and copy the code into a new project. The code should properly debug using the debug button in the application. Once debugged and adjusted to the specifics of the devices in use, the code can be flashed onto the MSP430 micro-controller, which will allow it to run the code from memory.

6.2 UART Communication Using RealTerm

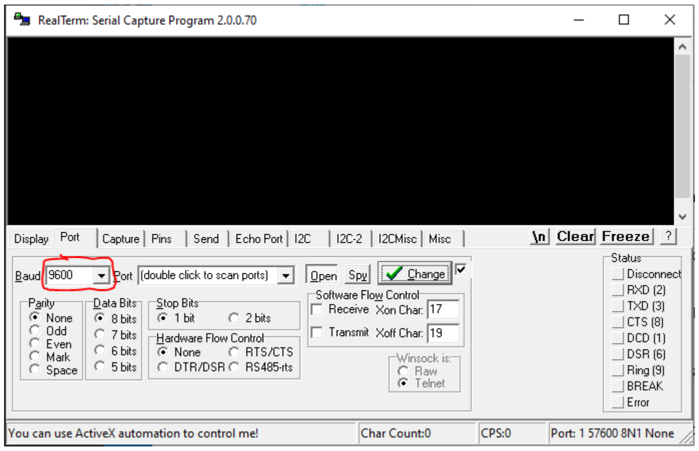
In order for the device to properly communicate, the application RealTerm must be downloaded onto a laptop or PC. When the micro controller is plugged into the computer, it will be able to communicate with the smart safe. To communicate with the MSP430 over UART, first RealTerm must be configured using the steps below.



Display As: Hex[space]
Half Duplex ✓
newLine mode ✓
Invert
7Bits
Big Endian ✓
Data Frames: Bytes 2, Single, Gulp
Rows 16, Cols 72, Scrollback
Binary Sync Chars: ABCD
Data XOR AND
Sync is: None
Leading Sync 0 matches
Char Count:0 CPS:0 Port: 1 57600 8N1 None

Figure 9: Display options

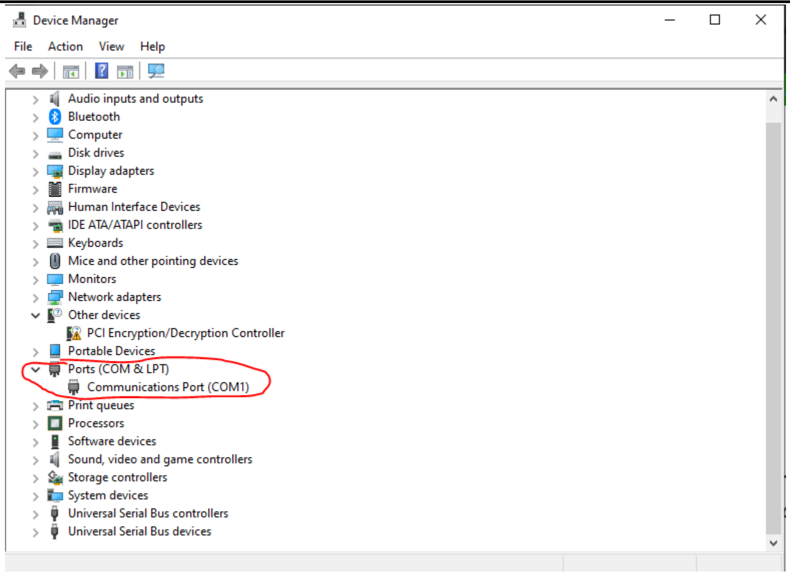
The first step to configuring RealTerm is the setup the display options. For ease of use, the options 'Hex[space]', 'Half Duplex', and 'newLine mode' should be used to display the numbers correctly. The next step is connecting to the same port as the MSP430.



Baud 9600
Port (double click to scan ports)
Parity: None
Data Bits: 8 bits
Stop Bits: 1 bit
Hardware Flow Control: None
Software Flow Control: Receive Xon Char: 17, Transmit Xoff Char: 19
Winsock is: Telnet
You can use ActiveX automation to control me! Char Count:0 CPS:0 Port: 1 57600 8N1 None

Figure 10: Selecting the Correct Port and Baud Rate for the MSP430

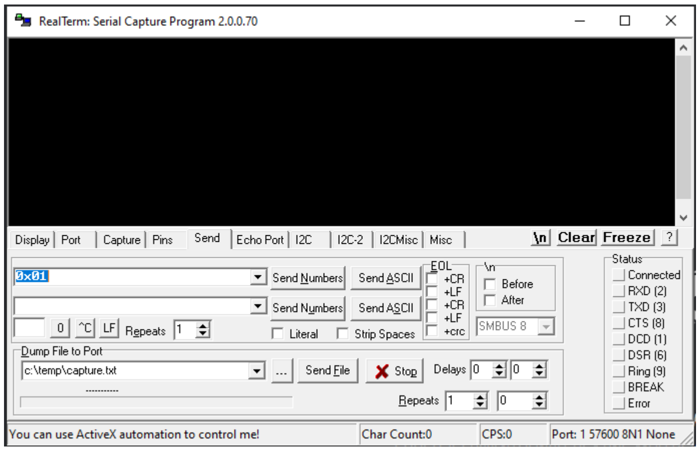
The chosen baudrate is 9600, which is very important otherwise communication will not function as intended. To select the correct port, either search in Windows device manager under Ports as shown below in Figure 11 below, or choose one of the two ports that are shown in RealTerm and later determine if it was the correct port.



Audio inputs and outputs
Bluetooth
Computer
Disk drives
Display adapters
Firmware
Human Interface Devices
IDE ATA/ATAPI controllers
Keyboards
Mice and other pointing devices
Monitors
Network adapters
Other devices
PCI Encryption/Decryption Controller
Portable Devices
Ports (COM & LPT)
Communications Port (COM1)
Print queues
Processors
Software devices
Sound, video and game controllers
Storage controllers
System devices
Universal Serial Bus controllers
Universal Serial Bus devices

Figure 11: Windows Device Manager View

To send a code to the MSP430 and unlock the device, use the 'Send' screen to input the passcode. Since the code is in hex, each number should be in the format of '0xMN' where MN are the digits in hexadecimal. This is shown in Figure 12 below. Another tip is that the two input boxes can be used to load multiple passwords very easily.


RealTerm: Serial Capture Program 2.0.0.70
Display Port Capture Pins Send Echo Port I2C I2C-2 I2CMisc Misc
\n Clear Freeze ?
Send Numbers Send ASCII
Send Numbers Send ASCII
EOL +CR +LF +CR +LF +crc
\n Before After
SMBUS 8
0 ^C LF Repeats 1
Literal Strip Spaces
Dump File to Port
Send File Stop Delays 0 0
Repeats 1 0
Status Connected RXD (2) TXD (3) CTS (8) DCD (1) DSR (6) Ring (9) BREAK Error

0x01

c:\temp\capture.txt

You can use ActiveX automation to control me! Char Count:0 CPS:0 Port: 1 57600 8N1 None

Figure 12: Sending numbers over RealTerm

The last thing that is necessary to get UART communication functional is to reorient two jumpers on the MSP430 silkscreen. The original orientation is shown below in Figure **??**, and the two circled jumpers need to be turned 90 degrees from their original position.

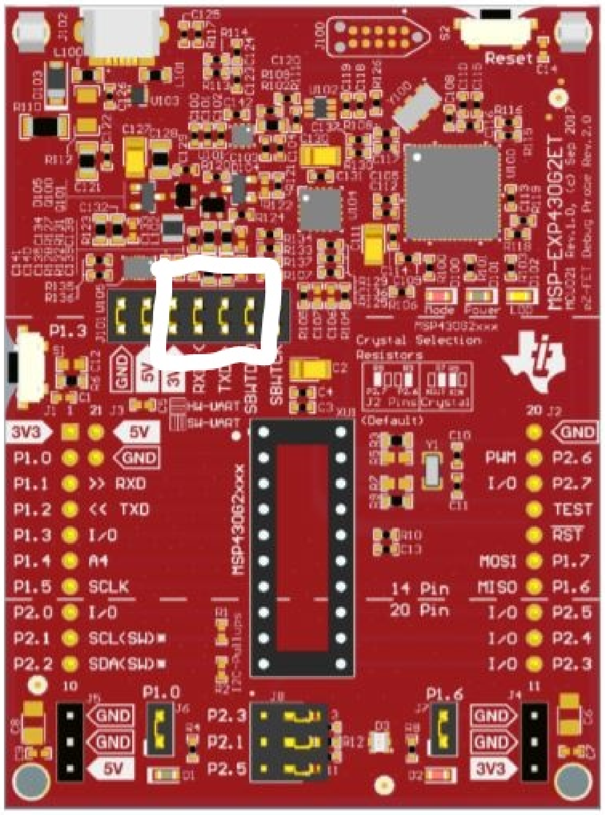


Figure 13: Image showing the original configuration of the MSP430G2553 and showing the specified jumpers in highlighted white box

7 Test Setup

When the circuits are properly built and the code is flashed to the MSP430G2553 micro-controller, the device is ready to be connected together. A jumper wire is run from the gate of the low side switch to pin P2.5 on the G2. Another jumper is run from the piezo pin to pin P2.3 on the G2. Two more jumpers are run from the grounds of the low side switch and the piezo to the common grounds on the G2. The solenoid is connected to the low side switch by powering the red wire and connecting the black wire to the drain of the MOSFET. Figure 14 details each of these connections from the board to the circuits, as well as to the solenoid.

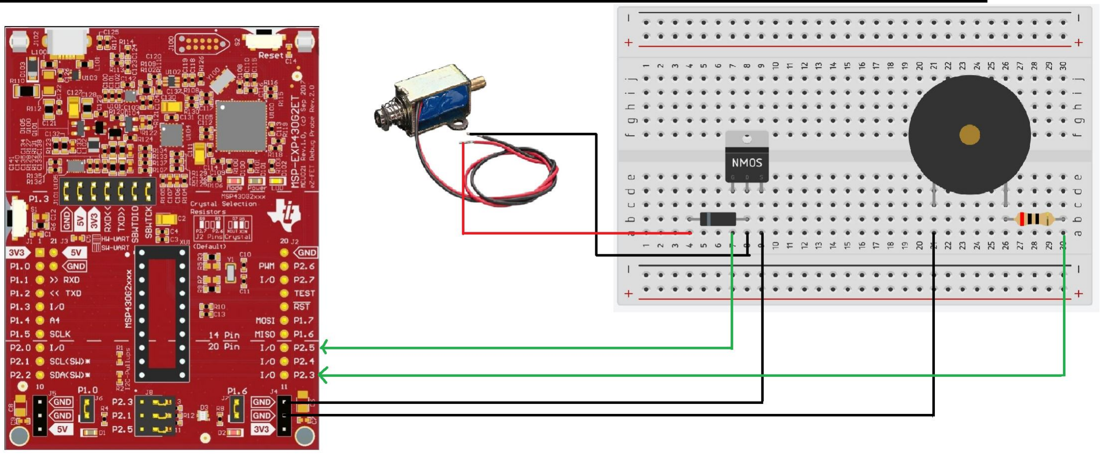


Figure 14: Full System Connections

7.1 Test Data

Input	Output
0x01 0x02 0x03 0x04 0x00	Unlock
0x01 0x02 0x03 0x04 0xFF 0x05 0x06 0x07 0x08	Change Password
0x01 0x02 0x03 0x04 0x00	Alarm
0x05 0x06 0x07 0x08 0x00	Unlock/Disable Alarm

The table above shows a sequential set of operations that were used to test the system. The safe is preset with a passcode of 0x01 0x02 0x03 0x04. Entering this code unlocks the safe. The next input is used to change the password. The user enters the current passcode, followed by a reset byte, indicating if the user is trying to change the passcode. Since this byte is 0xFF, it will reset the password to the next four bytes, 0x05 0x06 0x07 0x08, or whatever 4 digit passcode the user desires. Now that the passcode is changed, the original passcode is no longer effective and if entered will set up the alarm. When the new correct passcode is entered the safe unlocks and disables the alarm.

7.2 Bill of Materials

- JF-0630B Baomain Solenoid
- MSP430G2553 Micro-controller
- PS1420P02CT Peizoelectric Buzzer
- Power supply

- 510ΩResistor
- IRL520 SiC MOSFET
- 1N4148 Signal Diode 1x
- Various jumper wires
- Breadboard

7.3 Data Sheets

Solenoid: https://www.amazon.com/Baomain-Solenoid-Electromagnet-JF-0630B-Stroke/dp/B01K46PX

MSP430G2553: http://www.ti.com/lit/ds/symlink/msp430g2553.pdf

MSP430FR6989: http://www.ti.com/lit/ds/symlink/msp430fr6989.pdf

IRL 520 SiC MOSFET: http://www.vishay.com/docs/91298/sihl520.pdf

Piezoelectric Buzzer: https://www.jp.tdk.com/tefe02/ef532ps.pdf

1N4148 Signal Diode: https://www.vishay.com/docs/81857/1n4148.pdf

References

[1] Lewis, J. (2019, April 17). Low side vs. High side transistor switch. Retrieved December 8, 2019, from https://www.baldengineer.com/low-side-vs-high-side-transistor-switch.html.

[2] Power MOSFET Basics. (n.d.). Retrieved from http://www.aosmd.com/res/applicationnotes/mosfets/powermosfetbasics.pdf.