touchscreen
SECURITY SYSTEM

Devon Harker
Josh Haskins
Vincent Tennant

Table of Contents

- Plan
- Hardware
  - Commander
  - Node
- Possibilities
- Communication
- Issues Encountered
  - IDE was not Ideal
  - GUI plugin was Terrible
  - Multiple OS’s were Required
- Learning Outcomes
  - Learned C
  - Programming Mircocontrollers
  - Learned How To Work Around and With Subpar IDEs
  - How to Connect Different Pieces of Hardware Together
- Compiling

Plan

Initially we started with the simple idea of using a keypad to lock and open a door when the correct code is entered. We built on this idea by considering using a key card reader like what is used in hotels, using an RFID chip to open the door when scanned or using a fingerprint scanner to detect when the correct person is trying the enter the room. For simplicities sake and for a less steep learning curve we decided to go with a key card reader or a simple keypad. When we received the device we would be working with (PIC32) it came with the Multimedia Expansion Board (MEB) attached.
A touch screen was included on the MEB and we decided to go with a touch screen keypad. With the basic idea down we wanted to interface more hardware with the PIC32 to allow for out device to actively show that a door was being opened as well as showcase what could be done. We added a motor, motion sensor and some LEDs as well a another microcontroller, the PIC18, to make interfacing the hardware with the PIC32 easier.

Hardware

Commander

The commander consists of a Microchip PIC32MX795F512L and a Multimedia Expansion Board (MEB). The MEB has many input and output devices directly connected to it, such as touchscreen, 5x LEDs, 1 button, and a joystick.

Node

The node consists of a Microchip PIC18F4550, an LED, a motor and a motion sensor. Both the LED and motor are directly connected to the PIC18, whereas the motion sensor is just connected to the breadboard to use the battery (which was only needed to power the motor and motion sensor as only 3.3 volts was supplied from the Commander). The actual wire runs directly into the Commander.

Possibilities

It would very very easy to add on additional Nodes to the Commander. If we were able to just plug them into the same I/O pins it would work. Unfortunately if you were to tell it to open one door, all the doors would open. It would be very easy to simple to add additional inputs and outputs to the program so that a specific door or light could be turned on. The only limitation would be the PICtail connector, which only has a limited number of I/O pins.

Communication

Communication between the two devices is done through 3 wires. There is one input wire and two outputs. The input is a 1 for when the motion detector is not tripped and a 0 when motion is detected. The motion detector was so sensitive that it was constantly fluctuating between a 1 and 0. Even when the motion sensor was put in a dark box it was still being tripped. The other two wires were output from the Commander to the Node. These controller the motor and light. The two wires send two bits in total, with a total of four different commands, these bits control the motor and light.

Issues Encountered

IDE was not Ideal

The IDE gave us issues while programming. Some of these issues were reporting errors when there actually were not any errors at all. If you were to close the class, then reopen it, the errors would disappear. Once the software was loaded on the microcontroller, it would freeze the computer mouse for about 15 seconds.

GUI plugin was Terrible

The GUI plugin used for MPLabX IDE for the PIC32 MEB didn't import previously saved projects correctly and almost half of the project would be lost when opened. This made editing our GUI to make changes very difficult as they would not open properly. The GUI would have to be rebuilt and designed every time changes had to be made or changes would have to be made directly in the code.

Multiple OS's were Required

Since we were using two different microcontrollers two pieces of software had to be used to program both chips. As it turned out the software used to program and load the programs on to the PIC18 was only available on windows. Binaries were available for manual compiling for Linux and Mac systems but it was just easier to used the Windows .exe.

Learning Outcomes

Learned C

Before this project, no member of the team had any experience in programming with the C language. This project gave us first hand experience with C and we learned that it is rather similar to Java. We also learned the basics of C the 'hard way'. Additionally, we learned that programming GUI's in C is more difficult than Java. For example, 'primitive' GUI elements such are circles and rectangles cannot be moved after they have been drawn.

Programming Mircocontrollers

We learned how to program our two microcontrollers; the PIC32 and the PIC18. More specifically, we learned how to use MPLabX to upload a program onto the PIC32 and we learned how to use the PICkit 2 to program the PIC18. Finally, we learned that programming micro-controllers is not difficult to do when you have the proper software.

Learned How To Work Around and With Subpar IDEs

Refer to the 'IDE was not ideal' and 'GUI plugin was terrible' sections above to learn more about the issues we faced with regards to the IDEs.
We had to overcome these issues in order to complete the project. To accomplish this, we learned how to work with subpar IDEs. This is a valuable skill because it will be useful in the work world as it will allow us to work regardless of the software we are given.

How to Connect Different Pieces of Hardware Together

This project taught us the importance of breadboards as they reduce the amount of soldering or hot glue gunning; which helps to simplify the design and make it more understandable. We also learned that connecting microcontrollers to each other was not difficult to accomplish.

Compiling

Using MPLabX open the project file name.X and you will be able to compile the project from there. MPLabX is available for download from here http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en019469&part=SW007002.

In order to compile Project.X the xc32 compiler is needed, available here http://www.microchip.com/pagehandler/en_us/devtools/mplabxc/. Also the Microchip Application Library will is required, it is available from http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2680&dDocName=en547784.

Also to compile the Node.X project the xc8 compiler is needed, available here http://www.microchip.com/pagehandler/en_us/devtools/mplabxc/.

If you need help compiling the project, we are happy to help. Please email haskins@unbc.ca if aid is required.