2012 Co-op Design Challenge

Force B User's Guide

Texas Instruments

Team Blaster

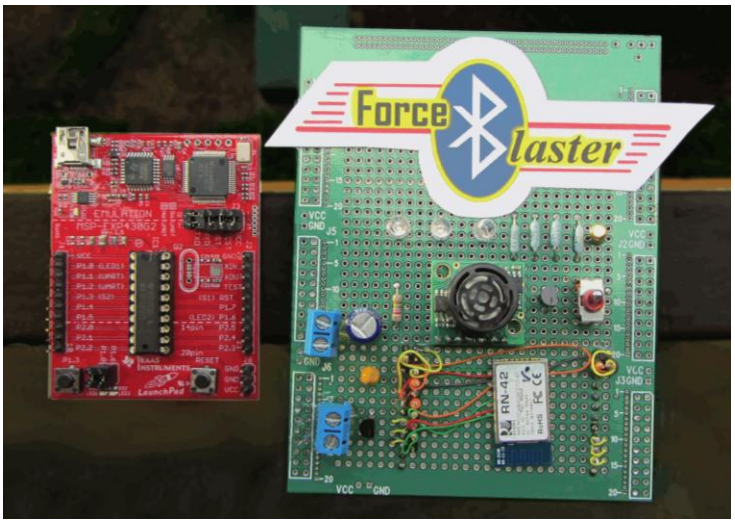
Dorothy Wong
Timothy Reyes
Polina Vorozheykina

Table of Contents

Introduction

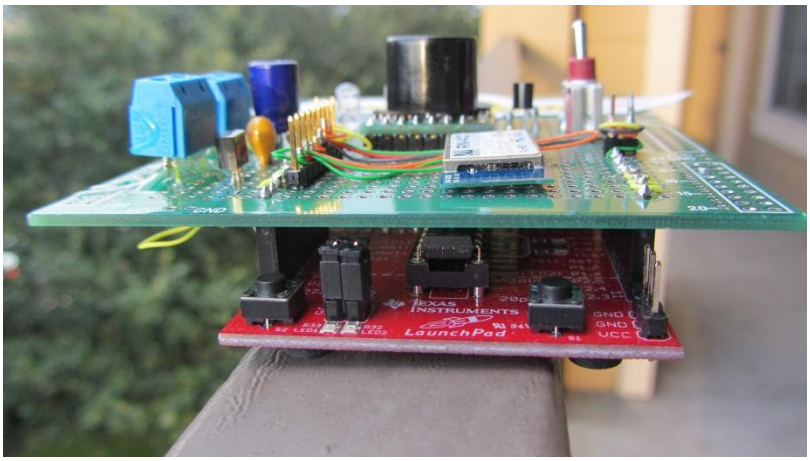
Overview

For the 2012 Co-op Design Project, we designed a boosterpack for the MSP430 Launchpad that we named the Force B. This boosterpack adds Bluetooth wireless connectivity to the Launchpad. It is designed as a base device that can easily be modified with new features by the user.

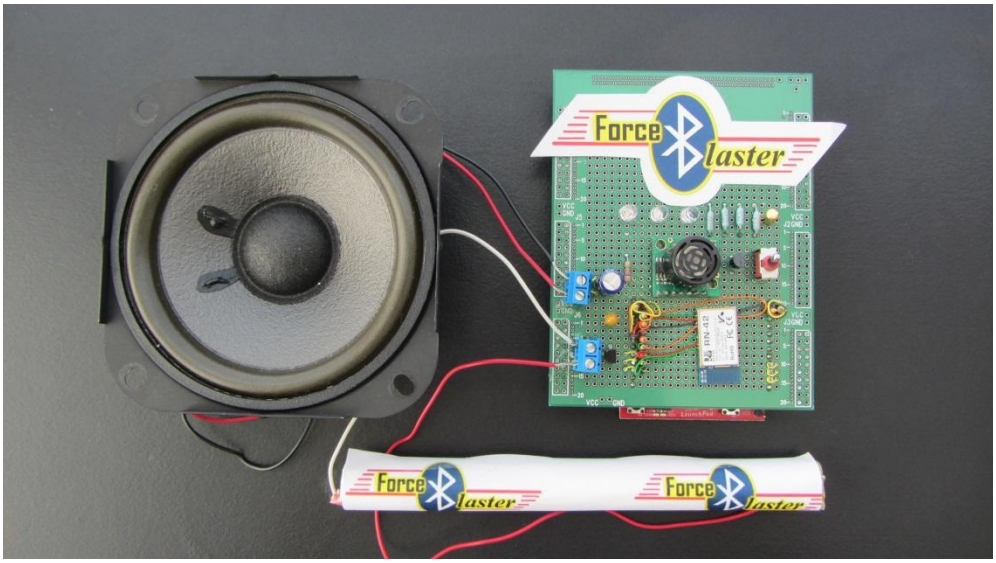


Features:

1. Wireless connectivity through Bluetooth
2. Small form factor: sits right on top of the Launchpad
3. Low cost: base model is only $25.72
4. Goes into low power mode when device is out of range, thus conserving energy
5. Simple, easy-to-use graphical user interface (GUI)
6. Fully customizable per the user's needs
7. Can be powered via USB cable or battery pack



The base model contains a speaker for sound output, and a battery pack. These connect to ports on the Force B and can be easily connected or disconnected depending on use in user-defined modules.



Operation

There are three main parts to the Force B: hardware, MSP430 program, and GUI software. The Force B's hardware facilitates Bluetooth traffic and contains all the sensors and output devices. These devices are controlled by the Launchpad, which processes the sensory input and handles Bluetooth communication. The MSP430 controlled hardware connects to a Bluetooth-enabled computer via a serial communication port. It is through this COM port that the third part of the Force B, the graphical user interface, or GUI, communicates with the MSP430 and the hardware. Communication is made possible through a pre-defined "language" of opcodes. Depending on an output opcode from the GUI, the MSP430 will report sensory data or output data to load devices. The GUI will also update its interface to correspond with input from the MSP430.

Pre-Programed Modules

The Force B comes pre-programmed with three different modules: Security and Home Automation, Entertainment, and Piano.

Security and Home Automation

Ever had a day when you forgot to turn off a light before you headed to work? Or if you wasn't sure if you turned off the air conditioning before you rushed out on an urgent errand? With the security and home automation module installed in your home, these days are far in the past! This module detects when your Bluetooth device is out of range, and automatically locks your door, turns off your stove, and turns off your lights when you go out of range. Figure 1 shows the GUI for this module.

Figure 1: Security and Home Automation Module GUI

By checking the GUI, users can see if whether their home is secure, and if their device is in range. It also shows if their door is open or closed, and if it is locked or unlocked. These indicators update automatically upon sensory input from the Force B hardware. The lock indicator also serves as a button to remotely lock the door just by clicking on the icon.

As a power savings function, the Force B goes into sleep mode when the Bluetooth device goes out of range. It powers down, but is still able to detect input. When the device comes back into range, the Force B wakes back up and resumes normal operation.

This brings us to the second part of the module: Home Automation. By clicking on any of the three switches for stove, air, and lights, users can remotely control the corresponding devices in their homes. Too tired to walk over to the light switch to turn on the lights? Just click the on/off switch to turn them on or off (Figure 2). This is a convenient way to turn off home appliances before going to bed for the night.

Figure 2: Security and Home Automation Module GUI

Entertainment

The second pre-programmed module is “Jedi Party” or Entertainment module. This module provides control for lights, music, and a motor for a disco ball, the three necessary ingredients for any party (Figure 3).

Figure 3: Jedi Party

Sliding the slider bar for lights allows remote control for lights. They can be strobed at various speeds, depending on the position of the slider. Buttons for music and disco allow music and a motor for a disco ball to be turned on and off remotely. At a party, one person can conveniently control the atmosphere of the setting by just operation from one location. A piano is also activated in this module, which provides more excitement to the party by being able to play songs.

Figure 4: Jedi Party in action

Piano

A third module pre-programmed into the Launchpad is the piano module named Jangler. It turns the computer keyboard into, well, a keyboard of the piano type. It offers a full octave chromatic scale, from note C to C.

Figure 5: Piano GUI

When the user presses a key on the keyboard, the corresponding key on the piano lights up as an indicator that a key is pressed, and the note is sent wirelessly to a connected speaker.

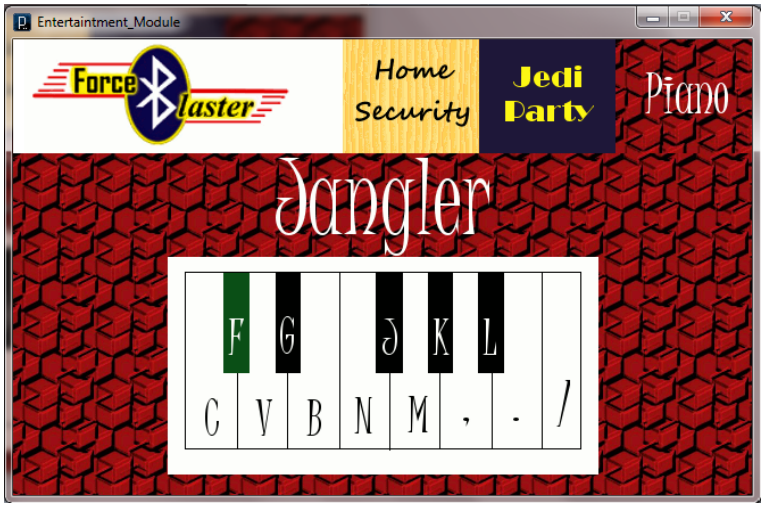


Figure 6: Lighted keys

How to Add Modules

The Force B is designed so that users can easily create and customize their own modules. There are three steps in making a module: hardware, MSP430 software, and Processing software. This tutorial covers the steps to add a simple digital input or output to the Force B.

Hardware

The Launchpad has various input and output ports that can take in various senor input or output signals to control devices. Ports 1.1 and 1.2 are used for the Bluetooth function to communicate with a Bluetooth device (i.e. computer). This leaves Ports 1.4 to 1.7 and Ports 2.0 to 2.5 to be user programmable. Here are the steps to configure a port for input or output:

Input:

Connect the output of the sensor to an available port pin on the Launchpad. In this example, let's say we connect a switch input to pin 1.5.

Output:

Connect the input to a load device to an available port pin on the Launchpad. Let's say we connect an LED output to pin 2.2.

MSP430 Software

We program the MSP430 with TI's Code Composer. Code Composer is a code editor, compiler, programmer, and debugger all in one. It can be downloaded from the TI website, www.ti.com. MSP430 software can be written in either assembly language or C code. For this tutorial, we use C code to demonstrate. More information about Code Composer can be found here: http://www.ti.com/tool/ccstudio.

Input:

1. Program the corresponding bit in the direction register of the input port. We write a '0' to bit 5 of register P1DIR .

```
P1DIR &= ~BIT5;     // Set P1.5 to input direction
```

2. Decide on a set of unique opcodes to facilitate communication between the MSP430 and GUI. Let's say opcode 0x01 means ask for input, 0x02 means high, 0x03 means low.

3. Add a case statement to the code to implement the decided communication protocol. We add code to handle an input of 0x01, and output a 0x02 or 0x03 depending on input from the switch.

```
command = receiveInput();           // Ping user input and store in command
switch(command) {
    case 0x01:
        if ((P1IN & 0x20) == 0x20){     // if P1.5 is high
            sendOutput(0x02);           // return 0x05
        }
        else{
            sendOutput(0x03);           // if P1.5 is low, return 0x06
        }
}
```

Output:

1. Program the corresponding bit in the direction register of the input port. We write a '1' to bit 2 of register P2DIR.

```
P2DIR |= BIT2;     // Set P1.5 to input direction
```

2. Decide on a set of unique opcodes to facilitate communication between the MSP430 and GUI. Let's say opcode 0x04 means set led output high, 0x05 means set led output low.

3. Add a case statement to the code to set the corresponding output register to high or low depending on communication protocol. We add code to write a '1' to bit 2 of P2OUT when a 0x04 is received, or write a '0' to bit 2 of P2OUT when a 0x05 is received.

```
command = receiveInput();   // Ping user input and store in command
switch(command) {
    case 0x04:
        P2OUT |= BIT2;      // set P2.2 high
    break;

    case 0x05:
        P2OUT &= ~BIT2;     // set P2.2 low
    break;
}
```

Processing Software

Force B uses Processing, an open source programming language primarily designed to create graphical user interfaces. It is based on Java, and requires a Java runtime environment to run. It can be downloaded from http://www.processing.org (recommended version 2.1 or higher). The input serial library will need to be used in order to communicate with the MSP430 via serial port. To establish the port, the user needs to manually locate the MSP430 device and establish a serial port (procedure different for different operating systems). Afterword, the user will be using a comprehensive built in Serial library in processing to communicate to the device.

Setup:

1. For the GUI to communicate with the MSP430, a serial port will be used. To establish the port, user needs to manually locate the MSP430 device and establish a serial port via built in Bluetooth. (The procedure is different for different operating systems, and is only necessary to complete once).
2. Afterward, the GUI will use a comprehensive built in Serial library to communicate the device. The library needs to be imported, and a Serial port must be established within the code:

```
import processing.serial.*;

/* Create object from Serial class */
Serial myPort;

void setup() {
  String portName = "COM41";
  myPort = new Serial(this, portName, 115200);
}
```

3. Serial library functions can be found here: http://processing.org/reference/libraries/serial/

Input:

1. Create switch variables for your input and match them to the set of opcodes decided on during the input of the MSP430 software. If you are using an input switch and an output LED, you may choose to pick variables such as:

```
boolean switch1 = false;
boolean led1 = false;
```

   For the inputs, you will only be working with the input variable, in this case the switch1 variable. If coded correctly, the MSP430 will send opcodes to the GUI. The preprogrammed GUI then decides on course of action with the input.

2. Opcodes generated by the MSP430 can be captured by GUI via the readChar() function. Since the port stores sent information, the data can be read continuously (non-interrupt based). From the example above (0x02 means high, 0x03 means low), their code in the capture opcode section should resemble:

```
//============================================================================
//                                Input opcode handler
//============================================================================
while (myPort.available() > 0) {
  char inByte = myPort.readChar();

  /*  if GUI reads in opcode of 0x02, indicate that switch is true (high)  */
  if (inByte == 0x02) {
    switch1 = true;
  }
  /*  if GUI reads in opcode of 0x03, indicate that switch is false (low)  */
  if(inByte == 0x03)
    switch1 = false;
}
```

   Based on the value of the switch variable, the program then may adjust the LED light up status accordingly by triggering the value of led1.

Output:

1. In order to display the status of the switch on the GUI, you will need to provide the GUI with proper picture files for both on and off status. Those are not restricted to just 2 statuses.
2. For proper operation, the GUI needs to be able to communicate signals back to the MSP430. This can be done by using the write function of the Serial port:

   ```
   myPort.write(0x02);
   ```

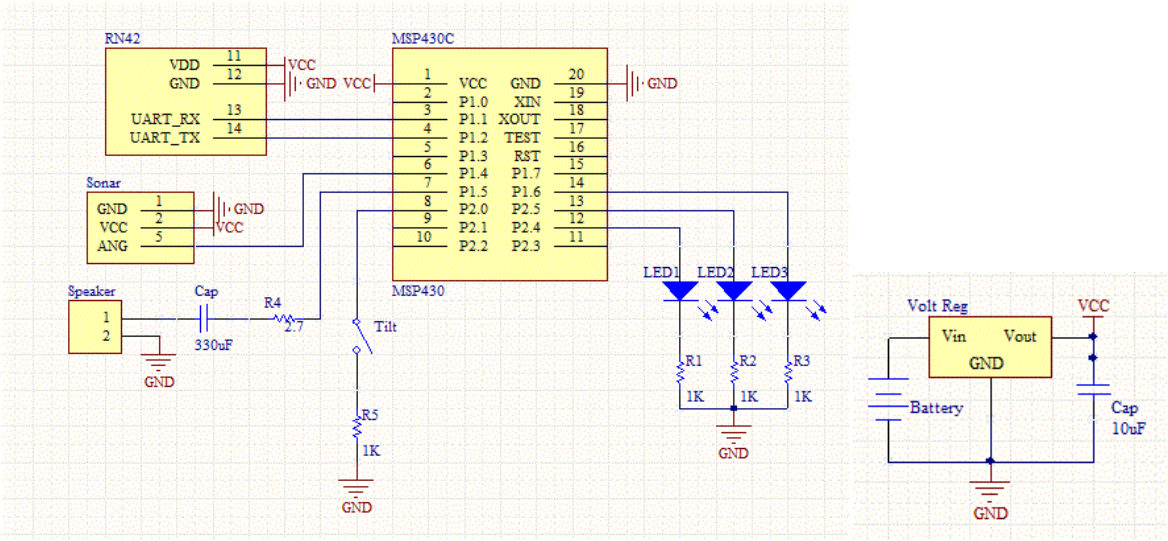
   This function can be used to write proper opcodes back to the MSP, and can be called both during continuous processes draw(), or during interrupts, such as mouseClicked().
3. To display the buttons on screen, PImage objects are used. First, an image of interest must be added to the data folder inside the sketch, which can be either created and modified manually, or can be modified from within the program (Sketch -> Add File).
4. The PImage object can be created and displayed in many ways. Example:

   ```
   PImage piano = loadImage("button on.gif");
   imageMode(CORNER);
   tint(255,255,255,255);
   image(piano,100, 100);
   ```

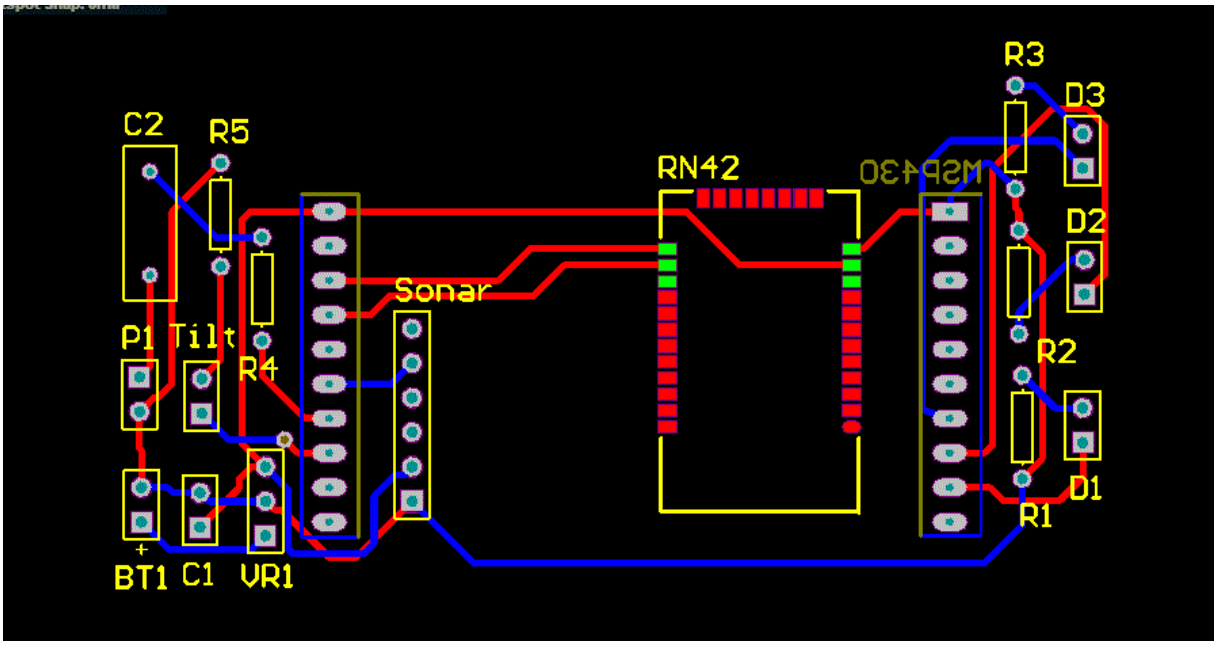
   Comprehensive review of PImage object can be found here:
   http://processing.org/reference/PImage.html

Documentation

Schematic:



PCB Design:



Bill of Materials:

Module	Product	Part Number	Distributor	Quantity	Cost per unit	Total
Base	MSP430 Launchpad	MSP-EXP430G2	Texas Instruments	1	4.30	
	Bluetooth Chip	RN-42	Sparkfun	1	15.95	
	Protoboard	PRT-08815	Sparkfun	1	4.50	
	3.3V Regulator	LM2937-3.3	Digikey	1	0.77	
	10 uf Capacitor	ECA-1CM100	Digikey	1	0.20	25.72
Home Security	Tilt Sensor	SEN-10289	Sparkfun	1	1.95	
	IR Rangefinder	GP2Y0A21YK0F	Adafruit	1	15.00	
	LED	TLHK5800	Digikey	3	0.43	
	1K Ω Resistor	ERD-S2TJ112V	Digikey	4	0.09	17.74
Entertainment	Motor	ROB-09608	Sparkfun	1	1.95	
	LED	TLHK5800	Digikey	3	0.43	
	Speaker	COM-09151	Sparkfun	1	1.95	
	2.7 Ω Resistor	ERD-S2TJ2R7V	Digikey	1	0.09	
	330 uF Capacitor	FK18X7R1H332K	Digikey	1	0.29	
	1K Ω Resistor	ERD-S2TJ112V	Digikey	3	0.09	5.84
Piano	Speaker	COM-09151	Sparkfun	1	1.95	
	2.7 Ω Resistor	ERD-S2TJ2R7V	Digikey	1	0.09	
	330 uF Capacitor	EEU-FR0J331B	Digikey	1	0.49	2.53

Cost of entire hardware (all three modules) = $48.60