## Software

### Ground Control Program

The ground control program of this project is the program which runs on the MSP430 housed in the handheld controller. This program takes inputs from the user using 2 joystick potentiometers and 3 buttons. These inputs are then interpreted by the program then sent to the blimp controller via a wireless interface. The components of this program can be seen below.

Main

The main function of this program consists mostly of function calls. This allowed us to easily make changes to specific parts of the code as well as keep track of where certain initializations were in the program. In main we first declare 5 variables. The first 4 of these are control variables used to store state information about the 3 motors and the LEDs on our blimp. The 5$^{th}$ variable is a state variable that is used in the main function state machine. The next 3 calls in main are for initialization steps. Further information about each of these steps can see seen in the function explanations below. The program then enters the main function state machine. This state machine is used to cycle through reading each input, then sending this data wirelessly to the blimp controller. The order of reading is left joystick, right joystick, up and down buttons, then LED button. After all of the inputs have been read, the program then sends these updated values. This state machine repeats as long as the controller is on.

configureMSP430

The configureMSP430 function includes initializations needed to set up the MSP430 for this program. This includes disabling the watchdog timer, setting the DCO to 1 MHz, enabling the input pins, and setting pull up resistors for the buttons. The inputs were P2.3 for the LED button, P2.4 for the DOWN button, and P2.5 for the UP button.

configureAIR

The configureAIR function was taken from sample code distributed by Texas Instruments for use with the AIR module. This code was found in the initiation steps of the main function of the sample program. This code was copied into this function so that it was clearly separate from other initializations.

configureADC

The configureADC function includes the initializations needed to set up the ADC for reading from the 2 joystick potentiometers. In this initialization the ADC is set to use $V_{CC}$ as its reference voltage and the ADC interrupt is enabled. P1.3 is set up was the input for the right channel and P1.4 is set up as the input for the left channel. These pin assignment were made using a definition of RIGHTPIN, LEFTPIN, RIGHTCHANNEL, and LEFTCHANNEL. This allows the input pins of the joysticks to by easily changed at the begin of the program.

readLeftJoystick and readRightJoystick

The readLeftJoystick and readRightJoystick functions are very similar except they pass different channel identifiers to the readADC and range functions. The channel identifier for the left joystick is 0 and the channel identifier for the right joystick is 1. Each of these functions first calls the readADC function, which will return the value of that joystick as read through the ADC. This value is stored in a temporary variable. The function then called the range function and passes it the correct channel and the value returned by the ADC. This function will then return a value between 0 and 6, which is returned by the read function.

readADC

The readADC function reads in a value from the joystick indicated by the channel identifier using the ADC and returns that value. It takes 1 parameter. This is the channel indicator. To do this, the function first interprets whether to read from the left or right joystick. Since the channel identifiers are 1 and 0 a simple if else statement can be used to determine which channel to set the ADC up to use. If the identifier is a logical TRUE, meaning it was a 1, the ADC is set up to use the right channel, however if the identifier is not TRUE this means it was a 0 and the ADC is set to use the left channel. After the correct channel has been chosen, the function begins a sampling and conversion cycle and waits in low power mode 0 until it is finished. When the conversion has finished the ADC interrupt service routine will trigger which exits low power mode. Once the function exits low power mode it returns the value of the ADC.

range

The range function takes the value read in by the ADC and determines which of 7 ranges that value fits into. It then returns the value of that range. The function takes 2 parameters. These are the channel indicator and the value of the ADC. The function first chooses which channel it is dealing with in the same way at the readADC function; a logical TRUE means right and a FALSE means left. After the correct channel has been selected the function then compared the ADC output to a series of values stored in an arrays corresponding to each channel. The values in these arrays are the cutoffs between each range. For example LEFTRANGE[0] contains the cutoff between range 0 and range 1.

These cutoffs were determined using a testing program that allowed us to output the values returned by the ADC onto a computer screen. By moving the joysticks to 3 positions equally spaced forward of neutral and 3 positions equally spaced backward from neutral we were able to see what the ADC values at those positions were. We these positions as the midpoints of each range and set the cutoffs accordingly. The cutoffs for each joystick can be seen in Table 1 and 2.

| Range | Low Cutoff | High Cutoff |
|---|---|---|
| 0 | 0 | 287 |
| 1 | 287 | 372 |
| 2 | 372 | 457 |
| 3 | 457 | 561 |
| 4 | 561 | 669 |
| 5 | 669 | 770 |
| 6 | 770 | 1023 |

Table 1: Range Values for Left Joystick

| Range | Low Cutoff | High Cutoff |
|---|---|---|
| 0 | 0 | 241 |
| 1 | 241 | 347 |
| 2 | 347 | 458 |
| 3 | 458 | 572 |
| 4 | 572 | 668 |
| 5 | 668 | 750 |
| 6 | 750 | 1023 |

Table 2: Range Values for Right Joystick

To determine what range the value is in, the function compares it to the low cutoff for that range and the high cutoff for that range. If the value is above the low cutoff but bellow the low cutoff then that range is returned. This analysis begins at the lowest range, testing if the value is lower than the lowest cutoff and continues until the value is higher than the highest cutoff in which case a rang of 6 is returned.

readUpDown

The readUpDown function reads in the inputs from the up and down buttons and interprets these inputs into a command that can be sent to the blimp. To do this it first stores if the up

button has been pressed by calling the readUp function and storing this value in a variable. It then does the same with the down button by calling the readDown function. The variable will be 1 if the button is pressed and 0 if it is not. The function then determines which of 3 commands should be sent to the blimp. If neither button is pressed or both buttons are pressed then a 0 command is returned meaning the motor should be off. If only the up button is pressed then a 1 is returned meaning the motor should turn in such a way that it pushed the blimp up. If only the down button is pressed then a 2 is returned meaning the motor should spin in such a way that it pulls the blimp down.

readUp and readDown

The readUp and readDown functions are very similar, the only difference being which pin they read from. These functions read in from the push buttons and return a 1 if the button is pressed and a 0 if the button is not pressed. Since the push buttons are active low, if the pin is HIGH the function returns a 0, if not then the pin is LOW and the function returns a 1.
*We choose to not debounce these pushbuttons because with the additional interrupts and code our program was too large to fit within the program size limit imposed by our free version of Code Composer. We determined that the buttons were polled frequently enough that any bouncing would be quickly corrected and not affect the operation of the blimp.

readLED

The readLED function reads in from the LED button and returns a 1 if the LEDs should be one and a 0 if they should be off. This function takes in one parameter containing a 1 if the LEDs are currently on and a 0 if they are off. Each time the button is pressed the LEDs toggle between on and off. To achieve this toggling effect it is necessary for the previous state of the LEDs to be passed to the function as the parameter mentioned. Also this function utilizes a global variable called LEDFLAG. Since the LEDs should be toggled when the button is pressed and released the LEDFLAG stores if the LED button was pressed the last time the function was called so that if it was pressed and is now not pressed the program will toggle what value it returns. If this is not the case the program returns the previous state.

To accomplish this in code, the function first determines if the LED button was previously not pressed and is now pressed, or if the button was previously pressed and is now not pressed. If the first of these cases is true it means that the button has been pressed and the function should remember this in the LEDFLAG until the button is released. In this case the program returns the previous value. If the second case is true instead it means that the button was released and the function should toggle the value it returns. It does this by returning the opposite of the previous value.

runTX

The runTX function and its accompanying support functions were borrowed from a sample code distributed by TI for the AIR module. Its purpose is to transmit data from the ground controller to the blimp controller wirelessly using the AIR module. This function was slightly modified for use in the project. Part of this modification was passing the function the 4 control variables. Also the original code used a button press to determine when to transmit, however this was removed so that the function will transmit once and exit.

The runTX function first creates a transmit buffer to hold the data that will be sent. It then sets up an interrupt that will be triggered when the full packet of data has been sent. Next the function creates a packet of data to be sent and transmits this data to the AIR module using a Serial Peripheral Interface. After the data has been sent the function waits for an interrupt

service routine to set a flag meaning the data has been fully transmitted. After this the function flashes an LED to show the user that data has been sent.

radioRxTxISR and registerConfig

Both the radioRXTXISR and registerConfig functions were borrowed directly from the TI sample code with no modification. The radioRxTxISR is the interrupt service routine, which is called when the AIR module has finished transmitting. In this ISR a flag is set to let the runTX function know that transmission has occurred and to proceed.

The registerConfig function sets up the radio using SPI to write to control registers which control the settings of the AIR module radio.

createPacket

The createPacket code was borrowed from the sample code for the AIR module from TI. This function takes an array which serves as a transmit buffer and fills this buffer with the desired data. The original function filled this buffer with random data, however the function was modified to take 4 parameters for the 4 control variables and place these 4 variables into the transmit buffer. The index of each variable is important because this is how the blimp controller knows which variable controls which motor or the LEDs. The first index of the array is filled with the packet length, the subsequent indexes are filled with the left, right, up and LED control variables respectively.

## Hardware

### Ground Control Hardware

The hardware for the ground control is a handheld controller with 3 push buttons and 2 joystick potentiometers. The push buttons and potentiometers were salvages from a VEX robotics controller. The MSP430 Launchpad, AIR module, joysticks, pushbuttons and batteries were all housed in a cardboard housing for this prototype.

Joysticks

The joysticks have 2-axises of movement, but only the forward and backward axis was used in this project. The two joysticks are wired together so that one $V_{CC}$ source and one GND go to both joysticks. The output from the left potentiometer was connected the P1. 4 and the output of the right potentiometer was connected to P1.3.

Buttons

The buttons for this project were also taken from the VEX controller and are situated in pairs. Only the bottom button on the left side of the controller is used and controls the LEDs on the blimp. This button is connected to P2.3 on the Launchpad. The buttons on the right side controller control the up and down motion of the blimp. The top button makes the blimp go up and is connected to P2.5. The lower button makes the blimp go down and is connected to P2.4.

LED

A small LED is connected to the board to show the user that power is supplied and that controller is transmitting. This LED is wired directly to P1.0 on the Launchpad board.

AIR Module

The CC110L RF Module BoosterPack is a circuit board containing a radio transmitter and receiver, which we used to transmit commands to our blimp. This BosterPack is installed

directly on top of the LaunchPad board using pin headers. It communicates with the MSP430 using SPI protocol. There is one AIR module in the ground-based controller and one in blimp. These modules are able to communicate with the ground-based bosterpack transmitting and the blimp based one receiving.

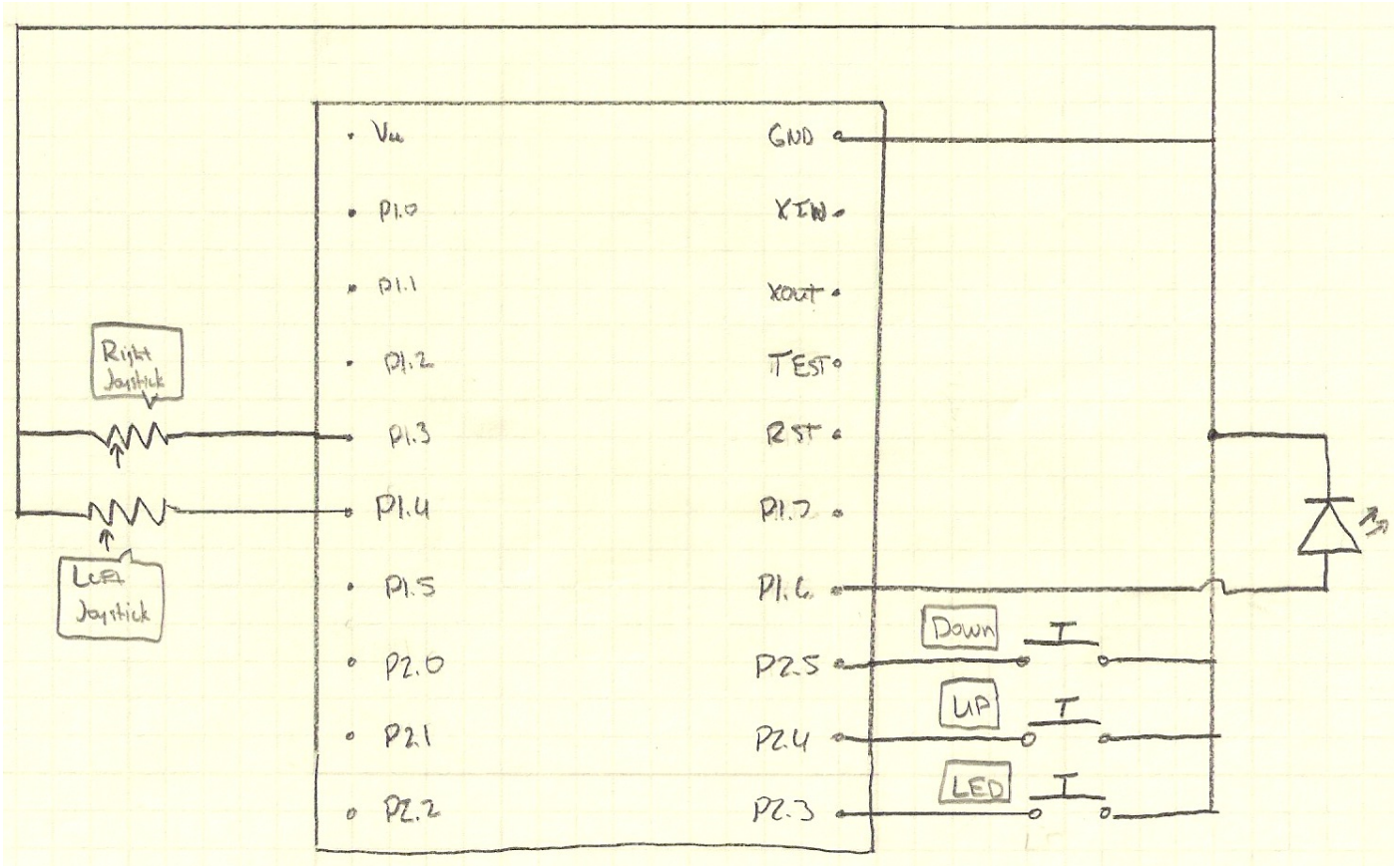


Figure 1. Ground Control Circuit Diagram

3.5V
100pF
LED
Right Black
Right Red
Left Black
Left Red
Up Red
Up Black
5.2V