CSE466 Blimp Project – Phase Two

AUTUMN 2011

OBJECTIVES

In this lab you will DO the following:

- Implement a wireless RUN/STOP controller for the blimp
- Attach the gondola (PCB and fans) to the blimp envelope
- Control the motors
- Control the IR proximity sensing subsystem

In this lab you will LEARN the following:

- How to work with the multichannel timer peripherals on the F5510
- How to control motors using PWM and an H-bridge driver IC
- Theory of IR proximity sensing/ranging
- How to distribute these tasks among team members…

DELIVERABLES

At the beginning of the next lab period you will demo any sensors and peripherals you were able to get working**. This demo will not be graded; the purpose of the demo will be to check in with the TA and keep your project on track.**

RESOURCES

These documents and web resources will be useful in completion of the lab and/or in answering the questions posed. Additional resources are described in the procedures below.

- Class discussion board
- BlimpBot schematics and component datasheets on course website
- MSP430F5510 Datasheet
- MSP430X5XX Family User's Guide
- MSP430F2274 Datasheet
- MSP430X2XX Family User's Guide
- MSP430 Code Examples
- MSP430 Optimizing C/C++ Compiler User's Guide

IMPORTANT NOTES

There are several very important rules to follow when using the blimp hardware. If you do not follow these rules, there's a good chance you'll release the magic smoke and have to wait to have your hardware repaired.

1. **ALWAYS POWER ON THE BLIMP BATTERY SWITCH BEFORE CONNECTING THE DEBUGGER**. Never connect the debugger when the battery switch is OFF or when the battery is disconnected or dead.
2. **NEVER short the motor or battery terminals.** The motor driver ICs and battery are very good at sourcing HUGE currents, and things will be damaged. Use caution when probing VBAT or motor terminals.
3. **ALWAYS plug in the battery with the proper polarity.** There is no protection against a reversed battery!
4. **ALWAYS keep the battery voltage above 3.4V.** When you notice the regulated voltage on VDD starting to drop below 3.3 V, it is time to charge the battery (HINT: Your code should watch for this "drop out" condition...)
5. **ALWAYS disconnect the battery when not using the blimp.** This helps ensure the blimp will not be left ON and deplete the battery.

PART 3: REMOTE CONTROL RUN/STOP

In this section you will add a RUN/STOP feature to the command line interface (CLI) developed in Part 2. The purpose of this feature is to make the act of programming your blimp feasible by allowing you to STOP the operation of the fan motors and other peripherals while you are handling or loading code onto the blimp gondola. You can then remotely RUN the system once the debugger has been safely disconnected and the gondola is attached to the envelope.

The main requirement for the STOP mode is that it **disables operation of the fan motors and the IR/E-field sensing peripherals**. In STOP mode, these peripherals need to be shut off in such a way that they consume little to no current (i.e., the IR LEDs need to be off and the fans must not be spinning). In RUN mode, normal operation may resume with all peripherals enabled.

There are at least two ways in which RUN/STOP may be implemented, with the difference being in the details of the STOP mode. In the simplest of implementations, the STOP mode could simply **shut off peripherals and block code execution** until a RUN command is received. In a more advanced implementation, the STOP mode could **set flags which prevent the operation of peripherals without preventing code execution**. This more elaborate method would allow for your blimp firmware to "go through the motions" of full operation but would selectively disable the fans and sensing peripherals, allowing for debugging of the system to take place even while in STOP mode. It is your choice which of these methods to implement!

<u>GOALS:</u>

1. Add a RUN/STOP feature (callable from your CLI) which either:
    a. Disables motors and sensing peripherals on STOP and blocks code execution until RUN received
    b. Disables motors and sensing peripherals on STOP without blocking code execution. Resumes normal operation on RUN.

PART 4: ATTACH GONDOLA TO ENVELOPE

Now that the framework for remote control of the blimp has been developed, you are ready to start working with the fan motors and sensing peripherals. To do this, you will first need to attach the gondola to the envelope, balance the blimp, and attach ballast to achieve a neutrally buoyant blimp.

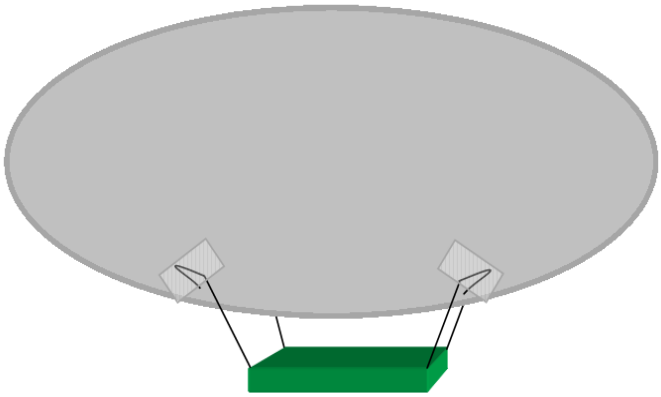

Figure 1. Attach the gondola to the envelope as shown using monofilament and tape.

PROCEDURE:

1. Get two 2' lengths of monofilament (fishing line).
2. Pass each monofilament length through the two outer holes on the gondola such that the ends of the monofilament emerge on the top side of the PCB (side with battery).
3. Using **masking tape** (easy to remove), attach the gondola to the blimp envelope as shown in the figure above. The gondola should hang about 2 - 3" below the envelope.
4. Ballast the blimp by attaching paperclips, etc. uniformly to the front and back until it is neutrally buoyant.
5. Reposition the gondola until the blimp is balanced front to back. This ensures that horizontal motion will not cause the blimp to ascend/descend.
6. Once the blimp is balanced front to back and is neutrally buoyant, mark the monofilament and envelope at the four points where they meet.
7. Replace the masking tape with strips of packaging tape for a more permanent attachment. Align the marks on the monofilament and envelope. Be sure that the monofilament doubles back under the tape strips for extra security as shown in the figure.
8. Clip the excess monofilament.

PART 5: MOTOR CONTROL

In this section you will develop a module for PWM control of the fan motors. Ultimately this module should be interfaced with your CLI for wireless control of each motor. **Don't forget to include the RUN/STOP functionality when developing motor control.**

HARDWARE PRIMER

The three fan motors, titled LFAN, RFAN, and VFAN (Left, Right, and Vertical) on the schematic diagram, are each connected to one channel of an LV8405 H-bridge driver IC. The purpose of this IC is to provide high-current switching for motor control as well as to protect the rest of the circuit from the large voltage spikes produced the inductive load of the motors. Inputs and outputs 1 & 2 of the motor driver make up the first channel of motor control, and 3 & 4 make up the second channel. The truth table below details the four modes of operation of each motor driver channel: Standby, Reverse, Forward, and Brake.

Truth Table

IN1 (IN3)	IN2 (IN4)	OUT1 (OUT3)	OUT2 (OUT4)	Charge pump	Mode
H	H	Z	Z	ON	Standby
H	L	L	H		Reverse
L	H	H	L		Forward
L	L	L	L		Brake

- : denotes a don't care value. Z : High-impedance

Figure 2. Truth table for LV8405 Motor Driver

Each channel of the motor driver IC is connected to a pair of pins on PORT1 of the MSP430F5510. Figure 3 shows the relation of the MSP outputs to each of the fan motors. In this diagram, "_H" and " _L" stand for the "High" and "Low" sides of the motor and correspond to the 1st and 2nd inputs for each channel of the motor driver. For instance, setting VFAN_H = 0 and VFAN_L =1 will put the vertical fan in Forward. **To initialize the motor control subsystem, you should drive all 6 output pins HI, which puts each motor channel in Standby.**

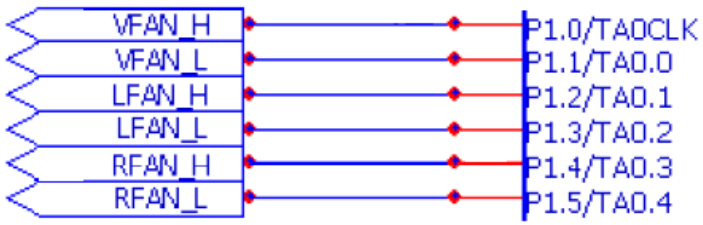

Figure 3. Motor driver connectivity diagram, from main blimp schematic.

PWM DETAILS

Pulse-width modulation will be applied here to vary the speed of the motors. To implement PWM, you will need to configure each channel of the TIMERA0 peripheral, which is capable of producing four independent PWM signals with unique duty cycles. The goal is to configure the PWM in such a way that the motor driver **quickly toggles between the Standby and Forward states (to go forward) or between the Standby and Reverse states (to go backwards). Your motor controller should NEVER switch to the "Brake" state, as it will cause unnecessary current consumption.** Keep in mind that the period (1/frequency) of the PWM signal for every channel of TIMERA0 is controlled by the first timer module register (TA0CCR0). The PWM frequency for this application should be set fairly low (several hundred Hz to several kHz) as the motor driver may not respond well at high frequencies.

The vertical fan will not be possible to control in the same manner, as outputs P1.0 and P1.1 cannot produce PWM signals via direct hardware control. You will need to develop a creative solution to vary the speed (or the effective thrust) of the vertical fan motor. Periodically pulsing the motor or implementing a form of software PWM are two possible methods of varying the effective thrust. Keep in mind that software PWM can burn a lot of CPU cycles if the frequency is set high.

GOALS:

1. Develop a motor control module which can:
    a. Initialize the motor control subsystem (all control lines HI).
    b. Update the direction and speed for horizontal fans LFAN and RFAN.
    c. Update the direction and effective thrust for the vertical fan VFAN.
2. Make this module callable from your CLI.

PART 6: IR PROXIMITY SENSING

In this section you will develop a module to control the IR proximity sensing subsystem. This sensor is comprised of five directional channels which allow for ranging of objects below, in front of, behind, and to the left and right of the blimp.

THEORY OF OPERATION

Similar to your E-field sensor from lab 3, this device has multiple transmitters and a single receiver. The transmitters are narrow-angle IR emitters (LEDs), of which four are mounted on the periphery and one in the center of the PCB. The receiver is a single wide-angle infrared photodiode mounted near the center of the PCB. The analog signal from the detector is buffered by an operational amplifier circuit and routed to P6.0 of the F5510 (Channel A0 of the ADC, labeled IR_RCV1_LO in the blimp schematic).

The intensity of the reflected infrared signal, as measured using the photodiode, will be used to estimate the range of a nearby object. As with the E-field sensor, the output will be toggled and the result accumulated via synchronous detection. However, there is one major difference between the E-field sensor and the IR proximity sensor in **that the received IR signal received is NOT high-pass filtered**; if we transmit a square wave, we will receive a square wave rather than the brief transient spikes produced by the E-field receiver. Also**, the frequency response of the IR detector is much lower than that of the e-field detector**, so you will need to pause after toggling the IR LED before you sample the detector. An efficient way to implement this is to sample just *before* toggling the IR LED state (in contrast to the E-field sensor in which you sampled just *after* toggling the transmitter state).

It should be noted that reflectance of surfaces in the infrared spectrum can vary significantly, causing inconsistent ranging results with different surfaces.

CONNECTIVITY

Because the IR emitters need high currents (50-80mA) to produce adequate amounts of IR radiation, external drive circuitry is required. Each IR emitter has a dedicated driver circuit utilizing a Darlington pair bipolar transistor and a current limiting resistor.

Due to a lack of pins on this MSP430F5510, the IR emitters are controlled individually through a demultiplexer/decoder IC, the 74HC238. The signal to be transmitted should be produced by P2.0 on the F5510, which is connected to the Enable input of the 74HC238. A specific IR emitter may then be selected by configuring PJ.1, PJ.0, and P1.6 as specified in Table 1. Please reference the main blimp schematic for a better understanding of the IR driver and receiver hardware.

LED Number	{PJ.1, PJ.0, P1.6}	Direction
LED1	{0,0,0}	Forward
LED2	{0,0,1}	Reverse
LED3	{0,1,0}	Right
LED4	{0,1,1}	Left
LED5	{1,0,0}	Down

Table 1. IR LEDs with decoder select values and emitter direction

GOALS:

1. Develop an IR sensing control module which can polls a selected sensor and reports the resulting ranging data.
2. Make this module callable by your CLI.

HINTS:

- The ADC10_A peripheral on the F5510 is NOT the same as the ADC10 peripheral you have used on the F2012. Register names have changed a little, and at least one configuration bit will be different.

EXTRA CREDIT: LINEAR MAPPING OF RANGING DATA

As you will certainly realize after completing Part 6, the output of your IR proximity sensor is very nonlinear. In order to linearize the ranging data and make possible the use of more interesting control algorithms, you can map the output of your proximity sensor to a linear distance scale using empirical data from your sensor.

A look-up table (LUT) will be useful for this mapping purpose. You may choose to perform linear interpolation for sensor values that fall between entries in the LUT, or you may simply choose to round to the nearest LUT entry. Be sure to declare your LUT as “const“, or else it will be copied onto the heap during initialization!

GOALS:

1. In your IR sensing module, add a LUT-based map (based on experimental data) which produces a linear distance reading given any possible sensor value.