Build and Program Your Own Flight Computer with Arduino

By Anthony Claiborne

Introduction

We rocketeers are living in a Golden Age of commercially produced flight electronics for our hobby. Dozens if not hundreds of companies now offer a wide variety of inexpensive lightweight electronic devices for recording altitude and acceleration, controlling recovery deployment and performing many other data-driven tasks to add depth to our knowledge and experience in rocketry. Why would any hobbyist want to build their own rocket electronics?

For me, a big part of the challenge and reward of our hobby is simply building things that work. It’s satisfying to see that late night brainstorm materialize into a custom rocket that flies straight and true. So it is with flight electronics: just building stuff that works is cool, and having it work in your own rocket is very cool indeed.

In addition, programming these devices is as much fun as building them. Home-built flight computers can be programmed in myriad ways, enabling flexible, customizable functionality that is not possible with most commercial flight electronics. In this article, I describe a simple design approach that you can use as a hobbyist to build and program your own general purpose flight computer.

In Part I, I outline the construction of a general purpose flight computer. I don’t give detailed step-by-step instructions, but rather provide design guidance for electronically-minded hobbyists to create their own computer hardware according to a fairly simple architecture. To guide you in programming a flight computer with this architecture, in a follow-on article I show software approaches and sample code that I have used.

PART I – HARDWARE DESIGN

Microcontroller

At the heart this design is the venerable Arduino microcontroller. Introduced in 2005, Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino hardware is manufactured in many forms. For this project, I use an Arduino Micro because it combines small size with full-featured functionality and relatively hassle-free USB communication. The flight computer comprises this microcontroller connected to small boards (sometimes called “breakouts”) and other devices for sensor input, memory storage, and effector output, along with electrical power for operation.

Sensor input

The computer senses and records altitude and acceleration. For altitude sensing, I use a pressure sensor breakout that senses and reports barometric pressure and temperature – with these two parameters, we can calculate altitude. For acceleration, I use an accelerometer breakout that senses and reports acceleration in the range of interest. For low- to mid-powered rockets, a range of +/- 16 gees works well.

Memory storage

Computer memory is either volatile or non-volatile. Data recorded in volatile memory is lost when power is removed from the computer while data in non-volatile memory is retained even when powered off. We need non-volatile memory to hold flight data, so we can retrieve the data after the flight computer has been turned off. While the Arduino Micro alone does have non-volatile memory on board (1K of EEPROM memory), these are relatively few bytes and, over time, the EEPROM becomes unreliable for memory storage. Another option for non-volatile memory storage are micro-SD cards, widely used in commercial products such as altimeters for memory storage. While there are advantages in using an SD card for storage, for reliability and speed I choose to use non-volatile electronic storage in the form of a FRAM breakout (FRAM is ferromagnetic random access memory, a relatively new form of non-volatile memory). Data is directly written to and read from the FRAM. I use a FRAM board with a capacity to store 32K bytes of data.

Effectors

To provide information about the state of the flight computer, I use a red and a green LED and a small buzzer. To turn on current for delayed recovery deployment, in-flight ignition, etc., I use a two-relay breakout, providing two sources of current that can be switched on independently.

Power supply

The microcontroller is powered by a single A23 battery. This small cell, about half the height of a AAA cell, weighs only about 8 grams and supplies 12 volts. The Arduino receives and regulates this voltage for its own use, in turn supplying regulated power at 3.3 volts for the memory and sensors and 5.0 volts for the effectors. A separate 9 volt transistor radio battery supplies the current to be switched through the relays.

Connecting the parts

The general connection of the parts in this computer is shown in fig. 1 below.



**Figure 1 - General connection of parts in the flight computer**

The positive terminal of battery A23 connects through switch SW to the Arduino’s V-in connector, and the negative terminal connects to Arduino ground.

The pressure sensor, the accelerometer and the FRAM breakout boards all communicate via the i2C bus. i2C is a hardware and software standard that uses four lines: 3.3V power, ground, serial data (SDA) and serial clock (SCL). These lines connect to corresponding connectors on the microcontroller, with SDA and SCL lines connected to digital connectors 2 and 3 respectively on the Arduino Micro board. The i2C protocol enables the computer to communicate with each of the three boards independently via only the SDA and SCL lines.

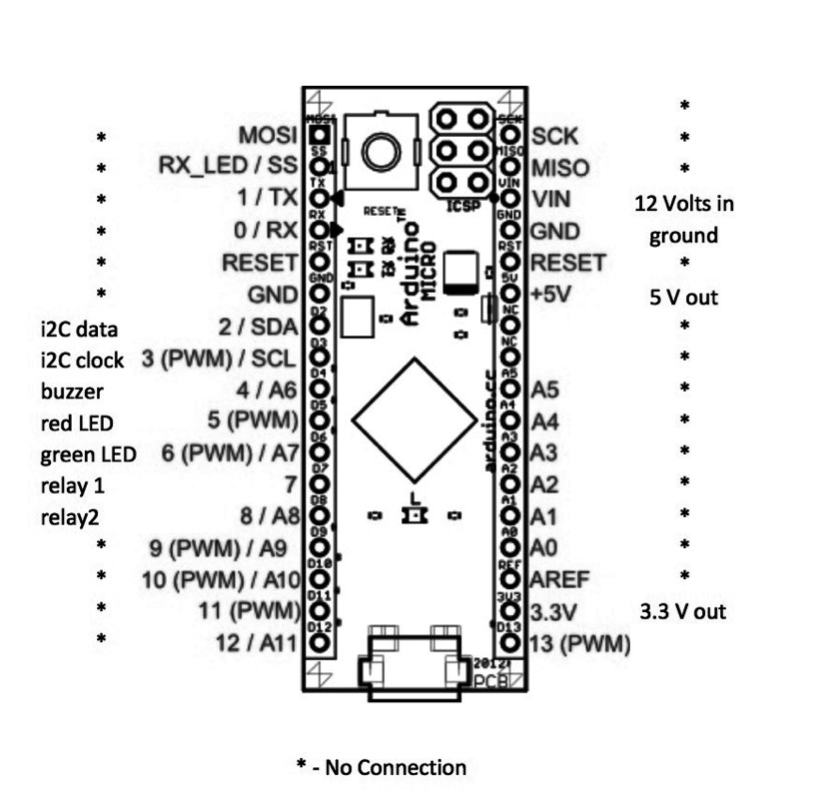
Each of a green LED, a red LED and a buzzer are connected to their own digital lines, sharing a common ground with the Arduino. A 330 ohm resistor is placed between each LED and the Arduino, to limit current draw by the LEDs.

For my flight computer, I use a breakout board having two relays. The two relays each have their own digital control lines and are supplied with 5V and ground from the Arduino.

The armature of each relay is connected to the positive terminal on the 9V transistor battery. The normally open contact of each relay is connected to its own pin on a three pin terminal connector CN, the third pin of connector CN being connected to the negative terminal on the 9V battery.

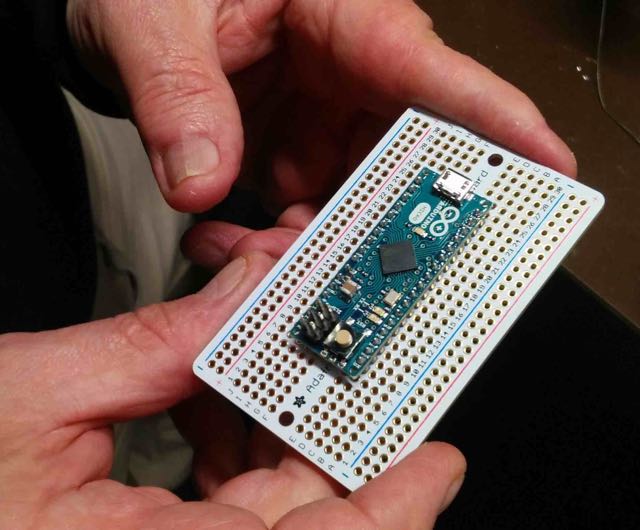
Connector CN hooks the computer up to devices that need ignition, like delayed deployment charges or engines for in-flight ignition. For ignition, a line is run from a relay contact pin on CN and a second line is run from the pin on CN that is connected to the negative 9V terminal. These two lines are connected to your device needing ignition. When a relay receives a signal from the Arduino on its digital control line, the relay’s armature connects to the relay’s normally open contact, providing voltage to its corresponding pin on connector CN, causing current to flow and ignite your device. You can program the computer to send an ignition signal at a given time or when the sensors indicate certain events have occurred.

A summary of all these connections is shown in table 1. Note that I show connections I have chosen - you can select the digital pins you use for the LEDs, the buzzer and the relays from any of the fourteen available digital pins on the Arduino Micro except pins 2 and 3, which are dedicated to i2C.

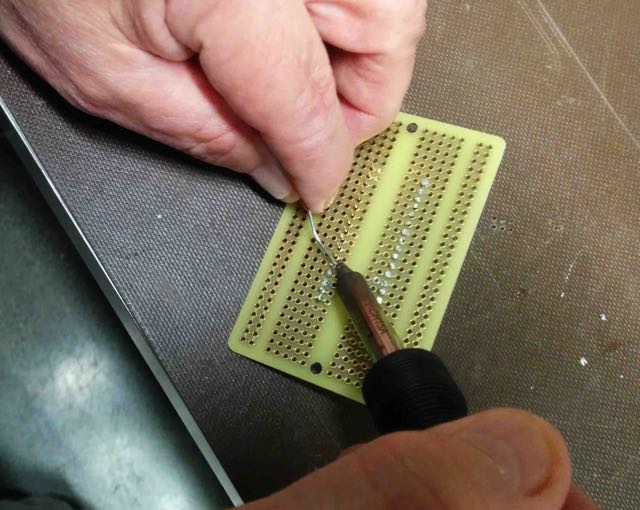


**Table 1 - Summary of connections to Arduino Micro**

To simplify wiring, I use a small breadboard from Adafruit. An Arduino Micro with header pins is positioned across the midline of the board, as illustrated in fig. 2. Each pin is then soldered on the bottom side of the breadboard, as illustrated in fig. 3.

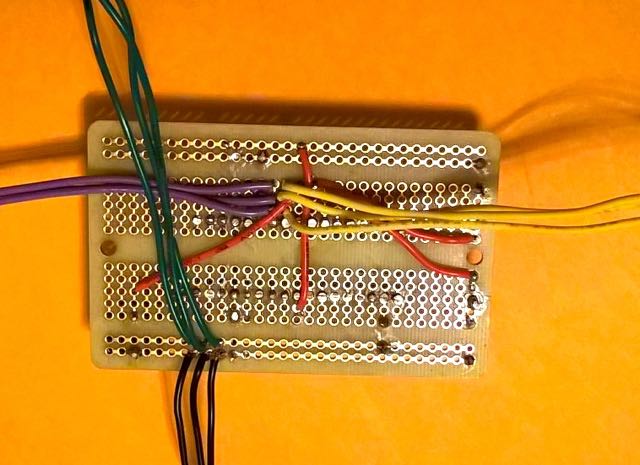
****

**Figure 2 - Arduino on breadboard**



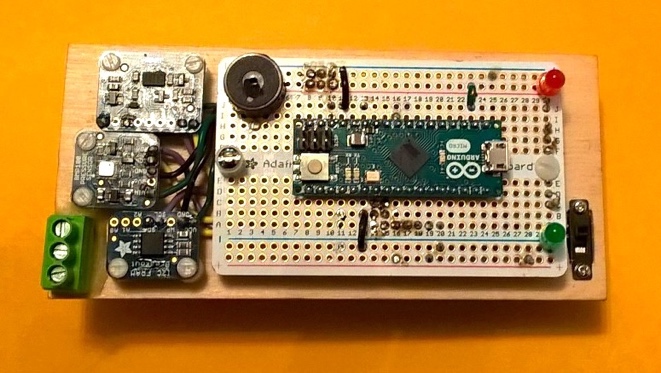
**Figure 3 – Soldering to breadboard**

Wiring is less confusing when different colors of wire are used. For general wiring, I use black wire for ground and red wire for other connections. With the i2C bus, three breakouts with four lines each make 12 wires to connect, which can be pretty complicated. To wire i2C, I use green for 3.3 V, black for ground, purple for SDA and yellow for SCL, as shown in fig. 4.



**Figure 4 – Color-coded wires for i2C**

How you lay out and assemble your components depends on the configuration you need for your rockets. I laid my flight computer out on both sides of a 3/16 inch plywood piece measuring 5 1/8 by 2 1/4 inches. Fig. 5 shows the side with Arduino, LEDs, buzzer, top side of switch and the three i2C boards. Fig. 6 shows the side with the batteries and a breakout board with the two relays. In my layout, the wiring from the Arduino to the relay board is not visible but passes through a hole in the middle of the plywood piece. See the Parts List for all parts used for my computer.

****

**Figure 5 – Arduino side of computer**



**Figure 6 – Relay side of computer**

Testing the flight computer

Software is used to test the assembled flight computer. You will need the Arduino IDE (Integrated Developer Environment), available from <http://www.arduino.cc>, to program the Arduino generally. Guidance on getting started with Arduino is available online at

<https://www.arduino.cc/en/Guide/HomePage>.

You will also need libraries with code for working with the sensors and the FRAM. These are available from Adafruit’s website, shown in listing 1. For additional guidance on installing and working with Arduino libraries generally, go to <https://learn.adafruit.com/adafruit-all-about-arduino-libraries-install-use>.

|  |
| --- |
| **Wire.h**  Wire.h is a standard library included with Arduino IDE. You do not need to download it  **Adafruit\_Sensor.h**  Adafruit Unified Sensor Library:  <https://github.com/adafruit/Adafruit_Sensor>  **Adafruit\_ADXL345\_U.h**  ADXL Accelerometer Library:  <https://github.com/adafruit/Adafruit_ADXL345>  **Adafruit\_BMP085.h**  BMP180 Pressure Sensor Library:  <https://github.com/adafruit/Adafruit_BMP085_Unified>  **Adafruit\_FRAM\_I2C.h**  FRAM Library:  <https://github.com/adafruit/Adafruit_FRAM_I2C>  **TimerOne.h**  Interrupt Library:  <https://github.com/PaulStoffregen/TimerOne> |

**Listing 1 – Library files to be downloaded**

When you have installed the Arduino IDE and the libraries, program the computer with the code shown in listing 2 (also available for download from <https://github.com/Aclaiborne1/Board_Test)>.

Create a folder named **Board\_Test** in the folder you use for your Arduino sketches and copy the **Board\_Test.ino** codeinto it. Connect your flight computer to your PC, bring up the IDE, open **Board\_Test.ino** and upload the code to your flight computer. When the IDE indicates that upload is complete, click on the serial monitor icon (magnifying glass) on the upper right of your screen to open the serial monitor. After a short pause, if all is well, you will see the messages in listing 3. Check wiring if any of the effectors do not turn on when tested or if any of the sensors or FRAM fail testing.

|  |
| --- |
| #include <Wire.h> // needed for i2C  #include <Adafruit\_Sensor.h> // needed for i2C  #include <Adafruit\_ADXL345\_U.h> // for ADXL 345 accelerometer  #include <Adafruit\_BMP085.h> // for pressure sensor  #include <Adafruit\_FRAM\_I2C.h> // for FRAM  #define beepPin 4  #define redLEDPin 5  #define greenLEDPin 6  #define relay1Pin 7  #define relay2Pin 8  Adafruit\_ADXL345\_Unified accel = Adafruit\_ADXL345\_Unified(12345); // for accelerometer  Adafruit\_BMP085\_Unified bmp = Adafruit\_BMP085\_Unified(10085); // for altimeter  Adafruit\_FRAM\_I2C fram = Adafruit\_FRAM\_I2C(); // for FRAM  sensor\_t sensor; // required for testing sensors  void setup()  {  Serial.begin(9600); //set up serial monitor    // configure effector pins for output  pinMode(beepPin, OUTPUT);  pinMode(redLEDPin, OUTPUT);  pinMode(greenLEDPin, OUTPUT);  pinMode(relay1Pin, OUTPUT);  pinMode(relay2Pin, OUTPUT);    // test effectors  delay(2000); // pause 2 seconds for user to get serial monitor  Serial.println("FLIGHT COMPUTER TEST"); Serial.println();  delay(2000);  Serial.println("Testing buzzer"); effectorTest(beepPin);  Serial.println("Testing red LED"); effectorTest(redLEDPin);  Serial.println("Testing green LED"); effectorTest(greenLEDPin);  Serial.println("Testing relay 1"); effectorTest(relay1Pin);  Serial.println("Testing relay 2"); effectorTest(relay2Pin);  // test sensors  Serial.print("Testing accelerometer...");  accel.getSensor(&sensor); // attaching the accelerometer  if (accel.begin()) {Serial.println(" good.");} // can we start it?  else {Serial.println(" bad.");}  Serial.print("Testing pressure sensor...");  bmp.getSensor(&sensor); // attaching the accelerometer  if (bmp.begin()) {Serial.println(" good.");} // can we start it?  else {Serial.println(" bad.");}  // test FRAM  Serial.print("Testing FRAM...");  if (fram.begin()) {Serial.println(" good.");}  else {Serial.println(" bad.");}  }  void effectorTest(int pinNo)  {  digitalWrite(pinNo, HIGH);  delay(2000);  digitalWrite(pinNo, LOW);  }  void loop() {} |

**Listing 2 – Board\_Test.ino**

|  |
| --- |
| FLIGHT COMPUTER TEST  Testing buzzer  Testing red LED  Testing green LED  Testing relay 1  Testing relay 2  Testing accelerometer... good.  Testing pressure sensor... good.  Testing FRAM... good. |

**Listing 3 – Board\_Test output**

Summary

This flight computer uses an Arduino microcontroller and the i2C interface to obtain sensor data on rocket altitude and acceleration, storing data in non-volatile FRAM memory. The computer can convey information regarding its state via two LEDs and a buzzer. The flight computer can be programmed to ignite one or two flammable devices (e.g. delayed deployment charges, in-flight ignition engines) independently. Components are readily available and relatively easily assembled.

While heavier than modern commercially available flight computers (mine weighs in at 152 grams with batteries), the fact that you can program this computer on many levels gives a flexibility for hobbyist applications that is generally impossible with commercial products.

Part II of this article describes programming the flight computer, with guidance for going further.

PART II – SOFTWARE

Introduction

In Part I of this article, I described the hardware design for a flight computer. It is designed around the Arduino microcontroller because the Arduino allows easy programming. I assume in Part II of this article that you have general familiarity with the Arduino and basic concepts in computer programming. Guidance on the basics of writing code for the Arduino is available at <http://arduino.cc>. Before we get into the program, though, you need to know how we are going to store and retrieve the flight computer’s data.

Memory map

All the useful data from a flight is stored in the FRAM board on the computer. As you recall, the FRAM board is non-volatile memory and the one we are using has a capacity of 32k bytes. Since a computer “k” is actually 1024 bytes, 32k is actually 32,768 bytes. We will be storing flight data in 4k chunks, or “banks”. The whole FRAM contains 8 banks, each bank having 4096 bytes. In each bank, we will use the lower 4,000 bytes for recording altitude and accelerometer data. We will use some of the remaining 96 bytes for recording other important information.

In each bank, 4,000 bytes are the maximum number of bytes into which altitude and accelerometer data are written (**maxBytes**). In each bank, the **maxBytes** of FRAM are divided in two equal parts, one for the altitude data and one for the accelerometer data, each then having 2,000 bytes. We will be storing the data as integers. Each integer takes two bytes, so the bank has a capacity to store 1,000 velocity data points and 1,000 acceleration data points, which you will see later is more than enough for our purposes. We start writing altitude data at FRAM address 0 from the beginning of the bank (which we call **altStart**) and we start writing accelerometer data in the middle of the bank (at an address which we call **geesStart**). A memory map of this arrangement is depicted in fig. 7.



**Figure 7 – Memory map**

Simple Program

We’ll start with a simple flight computer program and then add functionalities to it to make it more useful. Listing 4 shows a flowchart for this program.



**Listing 4 – Flowchart for simple flight computer program**

After initializing the computer hardware and software, this simple program has two parts. The first part, **flight()**, starts with a one minute delay, during which time both LEDs are on steady. This delay is to give you time to close up the computer in your rocket’s e-bay and get to the launch control before recording starts. Launch your rocket one minute after you have powered up the computer. After one minute, the buzzer sounds and the computer starts recording altitude and acceleration data. During the recording time, the red LED is turned off, leaving only the green LED on to indicate that recording is taking place. Recording continues until **maxBytes** are filled with data. **flight()**then calculates and stores the total time of recording in seconds. Next, **flight()**stores a value in FRAM that sets up the program so that control is always passed to the second part of the program, **menu(),** until later redirected by the user. When recording is complete, the buzzer sounds, the green LED is turned off and the red LED is flashing. Total recording time is about 65 seconds. Do not power down or reset the computer until the red LED is flashing. On reset or power cycle, the flight computer will now bring up the **menu()**routine.

After your flight, connect your PC to the flight computer’s USB port. Reset the flight computer and then select the serial port on your PC’s Arduino IDE. The reset of the flight computer brings up **menu(),** which turns the red LED on solid. After a short delay, **menu()**retrieves the recording time and uses it to calculate the time interval between each data record for altitude and acceleration. **menu()**then reports the elapsed time, recorded altitude and acceleration at each interval for the total recording time. When it has finished reporting, **menu()**prompts the user to set up the program to redirect control to be passed back to **flight()**. If you respond to the prompt, a value is set in FRAM indicating control will be passed to **flight()**on reset or power cycle, and the LEDs alternately flash on and off. If you don’t respond to the prompt in **menu()**, the red LED remains lit and the program remains set for control to be passed back to **menu()**again on reset or power cycle.

Download the code for this program at <https://github.com/Aclaiborne1/Super_Simple_Board> . Create a folder named **Super\_Simple\_Board** in your Arduino sketches folder. Copy all the downloaded files into this folder. Open up the IDE on your PC and open the file **Super\_Simple\_Board.ino** for uploading to your flight computer.

General Program

The simple program has very limited functionality. It simply records all data in **flight** mode, whether or not the rocket is flying, filling the entire memory bank with data, and it simply reports altitude and acceleration data from all the allocated memory, whether or not there is any actual flight data in it. It doesn’t report any other values that the computer can store or calculate, such as launch temperature and maximum velocity. Also, it doesn’t take advantage of the flight computer’s relays during flight to provide voltage for delayed recovery deployment, in-flight ignition, etc. A more general program is needed to provide the key functionalities of a flight computer. The general program described below. Like the simple program, this program also has a **flight** mode and a **menu** mode. However, the general program has greatly enhanced functionality.



**Listing 5 – Flowchart for flight mode of the general program**

General Program flight mode.

Listing 4 shows a flowchart for the **flight** mode of this general program. In this mode, the general program has three stages: pre-flight operations; the in-flight loop; and post flight operations.

Prior to flight, the program does several things in a manner similar to the simple program above. It initializes the Arduino pins used for effectors to provide digital output, it checks the sensors and the FRAM to verify that they are operating properly, and it initializes variables used in later parts of the program (in particular, zeroing the flight counter variable). It also analyzes sensor data to determine launch temperature and the elevation of ground level, which it records in FRAM.

After everything is initialized and checked, this more general program sounds a long beep and turns the green LED on solid. It pauses for 15 seconds to give time for you to close the ebay. It then gives another long beep and turns the green LED on flashing, indicating that the flight computer is now trying to detect launch. In this state, the computer repeatedly tests altitude to see if the rocket has launched. When launch is detected, the green LED is turned on solid and the program proceeds to the in-flight loop. Also, importantly, upon detecting launch the program sets a reset flag in memory indicating that flight is recorded. When this flag is set, the program will enter **menu** mode when the computer is next powered on or reset.

In the in-flight loop, the flight counter is first incremented in anticipation of receiving acceleration and altitude data. The computer then obtains the rocket’s current acceleration and altitude data, recording each in a location in FRAM indicated by the flight counter. The computer repeats this process, storing subsequent acceleration and altitude data according to the flight counter, until the computer determines that the rocket has landed.

While the rocket is in-flight, the computer also tests to see whether user specified conditions have been met to activate the first relay. When a specified condition is met, the Arduino sends a signal to activate the relay.

When the program detects that the rocket has landed, both the red and green LEDs are turned on solid, indicating that the in-flight loop has finished. The program then examines the altitude data to determine the rocket’s maximum altitude. The program also calculates the rocket’s maximum velocity. Finally, the program loops to report the apogee and maximum velocity for the flight in “beep-coded decimal” form, first sounding a long beep, then beeping out the digits for the altitude in feet, pausing and then beeping out the digits for the maximum velocity in miles per hour. Because the reset flag was set when launch was detected, the computer will enter **menu** mode when power cycled or reset.

General Program menu mode

The program’s **menu** mode is for user input and displaying flight data. You should be in **menu** mode the first time you run the general program. The flight computer also enters **menu** mode on power cycle or reset after a flight.

To use **menu** mode, power down the flight computer and connect it to your PC running the Arduino IDE. Reset the Arduino with the reset button located on the Arduino Micro board and select the serial interface on the IDE. The red LED will be flashing on the flight computer, indicating that it is in **menu** mode. You should see the menu in fig. 8 displayed in the serial interface on your PC’s IDE.

|  |
| --- |
| Altimeter Menu  1. Dump data  2. Zero out  3. Enter atmospheric pressure  4. Enter deploy altitude  5. Prepare for launch  Enter option: |

**Figure 8 – Menu options**

To display the most recent flight data, type 1 and press enter. The Arduino reports all the recorded altitude and acceleration data, along with the additional data shown in fig. 9 below. You can copy and paste this data into spreadsheet programs for further analysis.

|  |
| --- |
| Flight time = 52.60 seconds; interval = 0.065740; datapoints = 791  Sea level atmospheric pressure set to 1016.20 millibars.  Ground level = 53 feet.  Launch temperature = 68 degrees Fahrenheit.  Acceleration on launch: 9.04 gees.  Maximum post-launch acceleration = 9.31 gees at 10.72 seconds.  Maximum altitude = 1063 feet  Maximum velocity = 181 mi/hr  Deploy timing count = 536 = t+35.03 seconds |

**Figure 9 – Dump data summary information**

To use the altimeter menu to get the flight computer ready for the next launch, first zero out the memory bank by selecting menu option 2. Next, select menu option 3 to enter a value for the current sea level equivalent pressure for your location. If you don’t know that value, just use standard pressure of 1013.9 millibars. Next, select menu option 4 to provide an altitude in feet for activating the first relay (deploy altitude). When these values have been entered, select option 5, “Prepare for launch”, from the menu. When you confirm that you wish to launch, the red and green LEDs will flash alternately, indicating the computer is prepared for flight. At this time, the program unsets the reset flag in memory, so that the computer will enter **flight** mode on next power cycle. You may now unplug the flight computer from the PC. The next time the flight computer is powered up, it will be in **flight** mode.

You can download code for this more general program at <https://github.com/Aclaiborne1/Delay_Deploy_Modular> . Make a folder named **Delay\_Deploy\_Modular** in your Arduino sketches folder. Copy all the downloaded files into this folder. Open up the IDE on your PC and open the file **Delay\_Deploy\_Modular.ino** for compiling and uploading to your flight computer.

Going further

There is much more you can do with the flight computer:

* First, you can always improve software code – you can make the source code clearer and the running code faster and more efficient (the velocity routine in particular is very inefficient as I wrote it).
* In addition, you will note that we use only one of eight possible memory banks for storing data. You can add functionality to the general program so that all of the memory banks are used, enabling you to store data for up to eight flights in the computer.
* You will also note that we use only one of the relays in the general program. You can add code to use the second relay for purposes such as apogee deploy or in-flight ignition.
* To simplify operation, you can add a hardware switch to select between **flight** mode and **menu** mode.
* You can add a sensor and enhance the software to record temperature near the engine in the rocket.
* The flight computer can operate servo motors for manipulating flight surfaces to control trajectory.
* Much more compact flight computers can be constructed with similar architectures. I have several smaller altimeters built around this architecture that weigh less than 50 grams, measuring no more than 1 X 1½ X 3 inches.
* And on and on, limited only by imagination.

Conclusion

The flight computer is larger, heavier and a bit more expensive than many of the fine commercially available altimeters/flight computers out there. But the transparency of the flight computer’s hardware design and access to its source code give you the flexibility to customize the flight computer for your specific own rocketry needs.