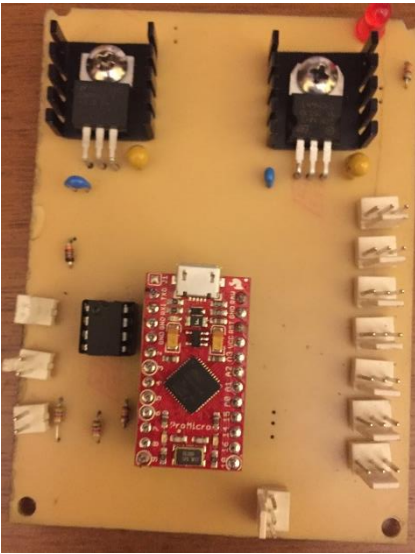


Flight Data Recorder Software Description, Version 1.1

By R. G. Sparber

Purpose



This software measures up to 7 analog voltages¹ plus the battery voltage² every two seconds³ for up to about 4 ½ hours. It can then output this data into a file readable by Excel.

Although originally designed to record sensor outputs during flight, it can be used anywhere long term monitoring is needed. For example:

- measure temperature and/or humidity variation inside a package while it is being shipped
- run time of a water pump
- power usage in the home



¹ 0 to 5 volt

² 7 to 10 volts

³ This value can be changed. If set to one second, data is collected for up to 2 hours 16 minutes 30 seconds. If set to 60 seconds, it will run for 136 hours 31 minutes.

Future Features

These features are candidates for later development.

1. User defined sample rate – at the present time the sample rate is set with a constant within the software. It could be set with a user accessible command. The longer the time between samples, the longer data can be collected.
2. Port disabling – the current design records data from all 7 ports plus battery. By disabling unused ports, memory can be saved and the maximum run time can be increased. For example, if only one port is needed we could increase the run time by a factor of four.
3. Port data conditioning – as the time between samples increases, it becomes more important to know what is going on between recorded data points. The user could specify, on a port basis, if the intervening data is to be averaged, searched for a maximum, or searched for a minimum. The data would be read at least every 0.01 seconds.

Contents

Purpose.....	1
Future Features.....	2
File Map	3
High Level Hardware Description	4
High Level Software Description	4
User's Guide	5
Initial Run of Software.....	7
Software Overview	15
Physical Environment Overview	16
Computing Environment.....	17
Command Details.....	18
Software Details	27
Adding Digital Sensors	37
Acknowledgement	42
Appendix: Tera Term.....	43

File Map

The article you are reading:

<https://rick.sparber.org/FlightDataRecorderSoftwareDescription.pdf>

The Hardware Description:

<https://rick.sparber.org/FlightDataRecorderHardwareDescription.pdf>

The Software in a .txt File:

<https://rick.sparber.org/FDR.txt>

The Circuit Board Artwork:

<https://rick.sparber.org/FDRartwork.pdf>

High Level Hardware Description

The heart of the Flight Data Recorder is a Pro Micro Computer System. The Pro Micro contains a computer with built-in data collection hardware plus memory to hold the code even when power has been removed. It also has hardware which permits the user to communicate via a standard micro-B USB cable. Collected data is stored in a separate memory device that is also able to save all information when power is removed. Board power comes from a 7.4 volt Li-Po battery. The Pro Micro and associated external memory can be run from the USB cable plugged into a laptop or PC.

High Level Software Description

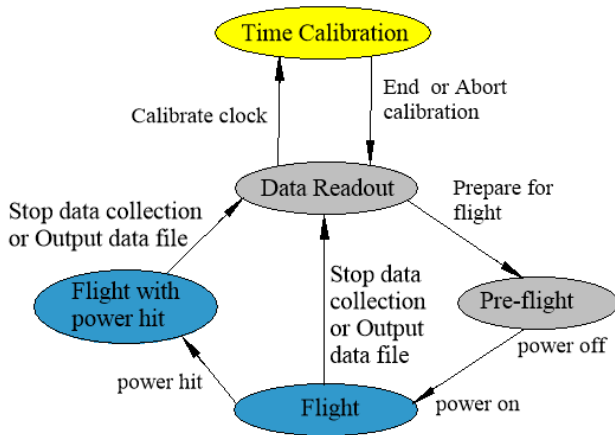
The software that runs on the Pro Micro is able to communicate with the user while on the ground and collect data during the flight. If power is momentarily lost (called a power hit) during the flight, the software will return to collecting data when the power is restored.

User's Guide

Support Tools

You will need to have a program able to send your keystrokes to the Flight Data Recorder plus display information from it. This data must also be able to be saved in a file for later use by Excel. One such program is called Tera Term (<https://ttssh2.osdn.jp/index.html.en>). See the Appendix for details on how to install it.

Overview of Operation



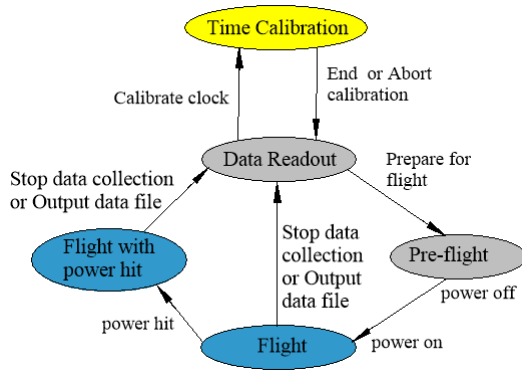
Sorry to state the obvious, but the payload is either on the ground or in the air. When on the ground, we can do a **Data Readout** or be in **Pre-flight**.

In the air, we can either be in **Flight** or be in **Flight after a power hit**⁴. In both cases, we are collecting data.

There is also an auxiliary function called **Time Calibration** which takes place on the ground.

⁴ A power hit is when power is disrupted. This causes the computer to restart the same way as when power has been off for a long time.

Events or commands move us between these states. If in the **Data Readout** state, the *Prepare for flight* command moves us to **Pre-flight**. Then *power is turned off*. Just before launch we *apply power* and move to the **Flight** state. If there is a *power hit during flight*, we move to the **Flight with power hit** state. Although unlikely, if the data recording memory becomes full, we stop collecting data in order to preserve what has already been recorded.



Once back on the ground, a *Stop data collection* or a request to *Output the data file* moves us to the **Data Readout** state. Note that no data can be recorded except when in the **Flight** or **Flight with power hit** states.

Not shown here is a diagnostic function which can tell us the current state.

Initial Run of Software

The software relies on a “control block” in order to know what took place before power was previously lost. Think of it as a scrap of paper that has notes on it about what job was last performed. When the Flight Data Recorder is powered up, it looks here to see what, if anything, it was doing before power was removed.

If a new memory device is installed, the control block may need to be initialized. The software detects this condition and attempts to correct it. Then you will see

```
EEPROM was initialized. Power Cycle  
to complete task. If you get this  
message again after the power cycle,  
you have a hardware fault.
```

It will appear on the Tera Term window every 5 seconds until you turn the Flight Data Recorder’s power off and then back on (“Power Cycle”).

The memory device is called an EEPROM which stands for Electrically Erasable Programmable Read Only Memory. Data is stored on this device and will not be lost when power is removed.

Summary of Commands

All commands are entered from the PC's keyboard in the Tera Term window. They can be entered at any time except during clock calibration and immediately after "P" (prepare for launch) has been entered.

Enter "M".

Flight Data Recorder
by Rick Sparber

Version 1.0 ←will be a different version

M This menu.
D Display all port voltages and set the pre-flight state.
P Prepare for launch!
A Use only while in flight: Display all port voltages.
O Output data file.
S Stop data collection.
C Calibrate internal hardware clock.
H Diagnostic: Output control block.
T Diagnostic: Generate test pattern in data memory.

LED semaphore

1 flash per cycle - normal data collection
2 flashes per cycle - normal data collection after power disruption
3 flashes per cycle - memory locked either manually or because it is full
4 flashes per cycle - pre-flight
5 flashes per cycle - hardware or software fault detected

You enter any of the letters to get the associated action.

The external LED signals the system's status with a pattern of ½ second pulses followed by 5 seconds of darkness except when preparing to launch or calibrating the timer. In these states, the LED will flash at a much higher rate.

Enter **M**⁵ at any time except when preparing to launch in order to see the available commands.

⁵ or any character not listed as a valid command

During Development

Enter “D” to Display all port input voltages plus the battery’s voltage. Expect to see something like this:

```
***Display of all Port Voltages and Pre-flight State***
```

```
Battery: 7.40 volts
```

```
Port      Voltage
====      =====
0         5.00
1         5.00
2         0.09
3         0.74
4         0.82
5         0.63
6         4.99
```

Time calibration is now showing 1000 internal millisecond counts per actual second.

If not between 900 and 1100, run time calibration.

Unterminated ports will read between 4.98 and 5.00 volts.

If the battery is not connected, you will see

```
Battery: disconnected
Sensors do not have 5V.
```

In place of the battery voltage.

Before Starting to Collect Data

Enter “P” to Prepare for data collection. Software will respond with:

You asked to prepare to collect data. This will erase all data. Are you sure? Y/N

Enter “Y” if you want to proceed or “N” to abort. Software will respond with:

Ready to collect data. Turn off power now.
No further commands will be accepted until after power cycle.

The LED will flicker until power is turned off.

Disconnect the USB cable from the PC. Turn off power. ***Do not turn power back on until ready to record.***

Starting to Record

1. Turn on power and see 0.2 second flash of the external LED.
2. Observe the external LED Flash Rate. All LED flashes are ½ second on and ½ second off. Between a series of pulses is 5 seconds off. Then the cycle repeats.
 - a. Sunny Day⁶: **One flash**: signifies normal data collection is in process.
 - b. Rainy Day⁷:
 - **Two flashes**: If power was applied and removed after the software was prepared to record, you will see two flashes. Recording can proceed but you will have to look at the data file to see if any other power disruptions occurred. The time count resets to 0 minutes 0 seconds after a power hit.
 - **Three flashes**: memory is full. This only happens when the software has been collecting data for more than about 4 ½ hours given one sample every 2 seconds.
 - **Four flashes**: in the pre-flight state. This says you did not correctly set up for the flight state.
 - **Five flashes**: a hardware or software failure has occurred. This may prevent the collection of data but at this late date, what do you have to lose?

⁶ Sunny Day means that all is working normally.

⁷ Rainy Day means that one or more things are not right.

Live Data Display while Logging

If you wish to monitor the input voltages and have the PC connected via the USB, you can enter “A” during the flight state to display the readings being stored. For example:

Port readings will be displayed as they are collected.

```
Port0 Port1 Port2 || Port3 Port4 Port5 || Port6 battery
2.02  2.14  1.61  ||  1.47  1.59  1.24  ||  1.41  4.78
2.10  2.02  1.59  ||  1.57  1.53  1.22  ||  1.45  4.78
1.98  2.33  1.80  ||  1.45  1.65  1.35  ||  1.41  4.78
2.02  2.02  1.57  ||  1.49  1.55  1.22  ||  1.41  4.78
1.92  2.16  1.69  ||  1.35  1.61  1.31  ||  1.35  4.78
```

When data is being displayed, you can enter “A” at any time to print out the heading again. For example:

Port readings will be displayed as they are collected.

```
Port0 Port1 Port2 || Port3 Port4 Port5 || Port6 battery
1.84  2.16  1.76  ||  1.27  1.61  1.39  ||  1.31  4.78
1.86  2.14  1.75  ||  1.29  1.59  1.37  ||  1.31  4.78
1.88  2.14  1.73  ||  1.33  1.59  1.35  ||  1.31  4.78
```

If you stop data recording, you will see

Data collection has been stopped. The EEPROM is now locked.

If you then enter “A”, you will see

Memory is locked so there is no new data to display.

After Landing

1. Turn off the board's power.
2. Turn on the PC and bring up Tera Term or similar program⁸.
3. Select the correct port.
4. Set Tera Term to log the incoming data stream to a .txt file.
 - a. Select the **File** tab
 - b. Click on the **log...** menu item
 - c. Select the path to this file and enter the File name with .txt at the end
 - d. Click Save
5. Enter "O" and the collected data will dump to both the screen and to the log file. You will be able to see the first number in the line fly past. This is the number of minutes into the flight.
6. Note: once "O" has been entered, the data memory is locked and becomes read-only. It can be unlocked only by entering "P" which will erase all data.
7. When the screen stops scrolling data, move the cursor over the Tera Term icon and click in **Tera Term: Log**.
8. Modify Filename and/or Fullpath if needed and then click **Close**.
9. Open Excel.
10. From within Excel, open the text log file you created. In the lower right corner of the file open window, select **All Files** so it will display your .txt (text) file.
11. Excel will take you through the conversion process. Your file uses commas to separate data and is therefore called a Comma Separated Volume (CSV). Column data format can be **General**. Excel should then show you a spreadsheet of your data with column headings. All port and battery numbers are in volts. Remember that the time count resets to 0 minutes 0 seconds after a power disruption.

⁸ The remainder of this procedure assumes you are using Tera Term. The same tasks with probably different key strokes will be performed with any similar program.

If the payload experienced any power disruptions (i.e. power hits), you will see warning text in the data file like this:

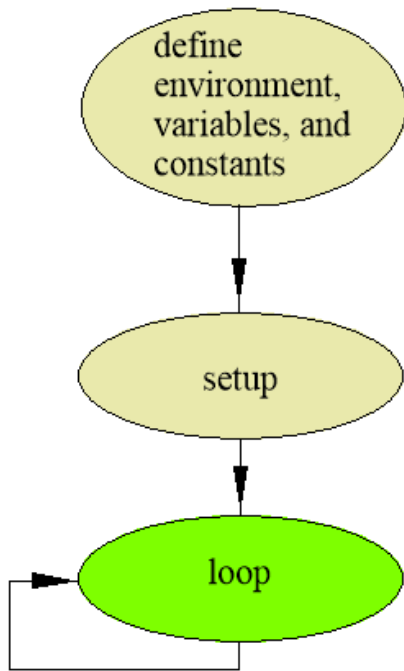
```
Minutes,Seconds,Port0,Port1,Port2,Port3,Port4,Port5,Port6,Battery,  
0,1,5.00,5.00,5.00,4.99,4.99,4.99,4.99,4.77,  
0,2,5.00,5.00,5.00,4.99,4.99,4.99,4.99,4.77,  
...  
35,4,5.00,5.00,5.00,4.99,4.99,4.99,4.99,4.77,
```

Power has been disrupted during flight. ← **warning text**

```
0,1,5.00,5.00,5.00,4.99,4.99,4.99,4.99,4.77, ←time stamp resets  
0,2,5.00,5.00,5.00,4.99,4.99,4.99,4.99,4.77,  
...  
63,42,5.00,5.00,5.00,4.99,4.99,4.99,4.99,4.77,
```

The above data set was taken with a one second sampling rate but the default for the code is every two seconds.

Software Overview



The Pro Micro's code⁹ is structured into three groups. When power is applied, parameters related to the physical structure of the processor board are read so the software knows what is connected. Variables and constants accessible by all code are also defined.

Next, we execute code within the setup grouping. This code only runs once.

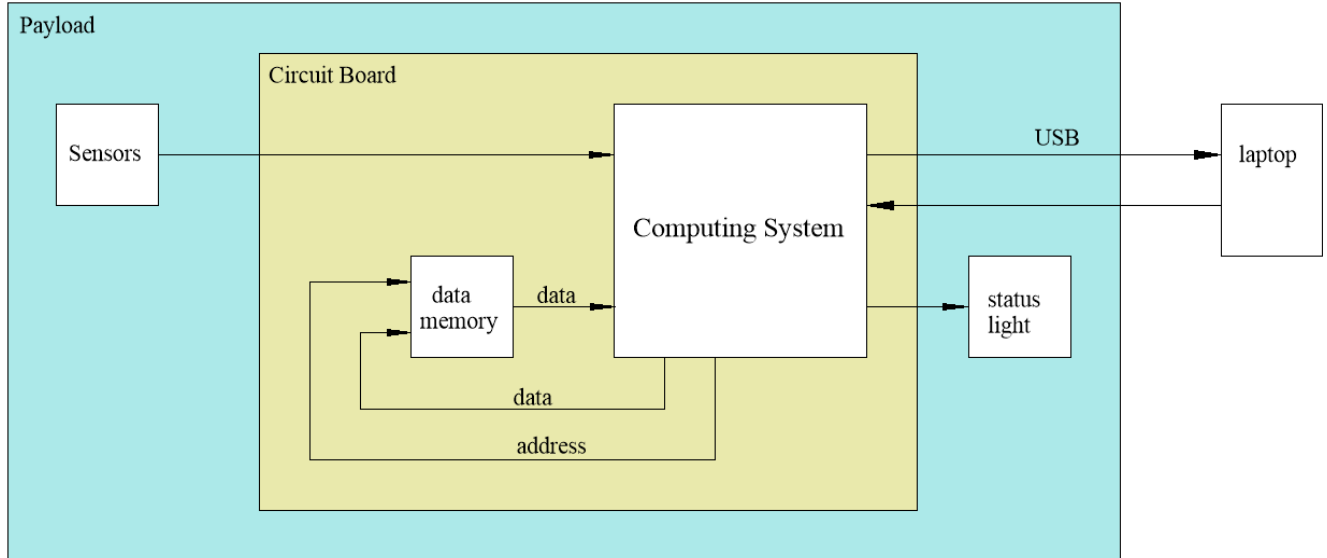
Code within the loop grouping is continuously executed until power is removed.

Major functionality first presented in the User's Guide is:

- S Scan all ports and display
- P Prepare for launch
- O Output collected data in Comma Separated Volume (CSV) format

⁹ The Pro Micro is a member of the Arduino family and runs C/C++.

Physical Environment Overview



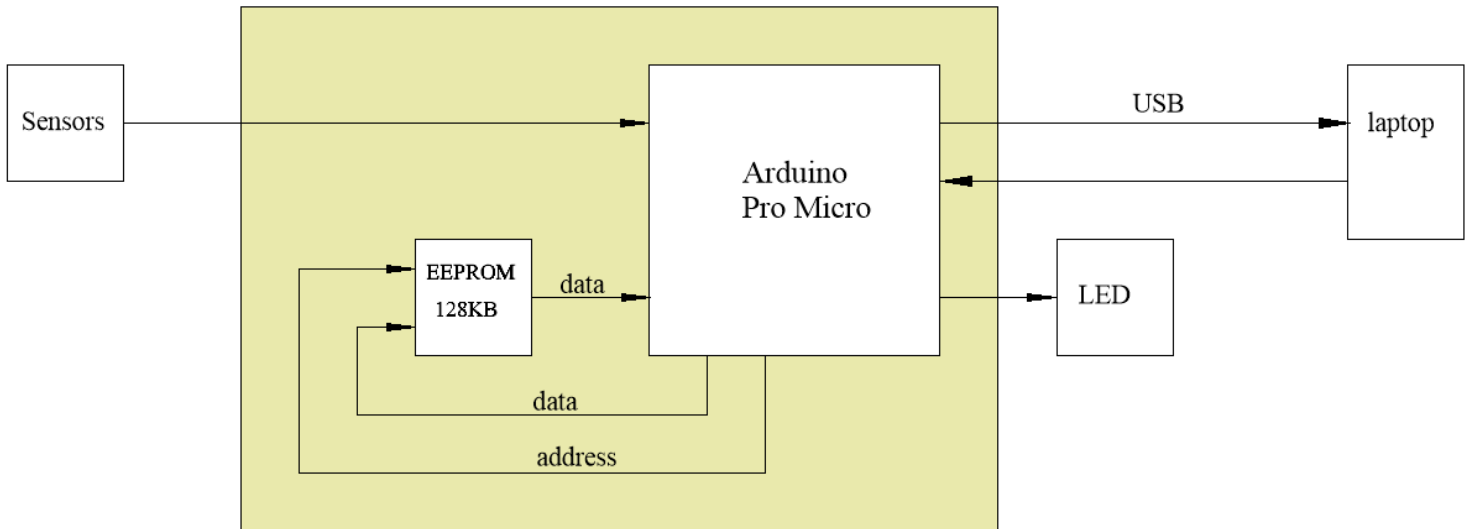
Within the payload we have sensors, our circuit board, an externally mounted status light, and a USB cable that can be connected to a PC.

On the circuit board is a Computing System (our Pro Micro) and a memory device that retains information even when power is removed (our Electrically Erasable Programmable Read Only Memory or EEPROM).

The Computing System can read data from the sensors in the form of DC voltages, convert this data to numbers, store the numbers in memory or send it via the USB to the PC. It can also drive the status light.

Not shown in the diagram is the power source which is a 7.4 volt Li-Po battery.

Computing Environment



Our Computing System is an Arduino compatible device called a Pro Micro. The EEPROM has a capacity of 128 KB (kilobytes) and is part number 24LC1025. From a logic standpoint, the Pro Micro sends an address to the EEPROM along with a read/write command. If a read is requested, data is returned. If a write is requested, the Pro Micro supplies the data and it is stored at the specified address.

The EEPROM has a 17 bit address so can store $2^{17} = 131,072$ bytes. Note that “kilo” in memory means 1024 and not 1000 so 128 K equals 131,072. The first address is 0 so the last address is 131,071 for a total of 131,072 bytes.

Inside the Pro Micro is another EEPROM which holds the software.

The status light is a high intensity LED that is visible even in direct sunlight. It is mounted on the exterior of the payload.

All sensors output a voltage between 0 and 5 volts.

Command Details

When you enter “M” in the Tera Term command window, you will see

Flight Data Recorder
by Rick Sparber

Version 1.0 ←will be a different version

M This menu.
D Display all port voltages and set the pre-flight state.
P Prepare for launch!
A Use only while in flight: Display all port voltages.
O Output data file.
S Stop data collection.
C Calibrate internal hardware clock.
H Diagnostic: Output control block.
T Diagnostic: Generate test pattern in data memory.

LED semaphore

1 flash per cycle - normal data collection
2 flashes per cycle - normal data collection after power disruption
3 flashes per cycle - memory locked either manually or because it is full
4 flashes per cycle - pre-flight
5 flashes per cycle - hardware or software fault detected

D *Display all Ports*

A single scan of all ports is displayed along with the Time Calibration constant. If any hardware or software errors were detected, they will be displayed too.

```
***Display of all Port Voltages and Pre-flight State***
```

```
Battery: 7.40 volts
```

```
Port      Voltage
====      =====
0         5.00
1         3.51
2         0.00
3         0.74
4         0.82
5         0.63
6         0.83
```

Time calibration is now showing 1000 internal millisecond counts per actual second.

If not between 900 and 1100, run time calibration.

The hardware and software error code should be written down and brought to the attention of the instructor. In all cases, this means a fatal error has occurred but some data may have been collected.

error codes	meaning
1	Out Of Range Read
2	Out Of Range Write
3	ControlTheFlightParameters ReadyForLaunch Read Back Failure
4	ControlTheFlightParameters InFlightFlag Read Back Failure
5	JustWait Read Back Failure
6	PrepareForLaunch Read Back Failure
7	WriteEEPROM Attempt Made To Write To Locked Memory
8	Attempt Made To Write To OneSecondMilliseconds Constant
9	USB's receiver return code was Out Of Range (negative value)
10	Read Back From EEPROM Mismatch

P Prepare for Launch

When the user wants to prepare the payload for flight, the Pro Micro sets flags in the EEPROM to indicate we are ready for flight. These flags are read the next time power is applied and the software knows it must start to collect sensor data and store it in the EEPROM.

Before entering P, you can see the state of the Control Block by entering H. One possible output is:

```
Control Block
=====
Memory is locked. In Data Readout.
There are no detected faults.
Start of Next Memory Block = 131072 (100% full)
Time calibration value = 1000

Data will be recorded every 2 seconds for a total
run time of 4.55 hours.

Compilation date is Nov 20 2018

C:\Users\Owner\Documents\FDR10.ino
```

We are in data readout mode. The start of the next memory block depends on the amount of data collected on the last run. Here you see that the memory is full.

The time calibration value should be within 10% of 1000. If it is not, calibration must be performed by entering “C”.

After entering P and then Y to confirm this choice, you must turn Flight Data Recorder power off. Turn it back on just before launch. Then you can enter H to see something like this:

```
Control Block
=====
In flight.
There are no detected faults.
Start of Next Memory Block = 48 (0% full)
Time calibration value = 1000
```

Data will be recorded every 2 seconds for a total run time of 4.55 hours.

Compilation date is Nov 20 2018

C:\Users\Owner\Documents\FDR10.ino

Note that memory is nearly empty¹⁰. Data is sampled every 2 seconds which translates to a data collection run time of 4.55 hours which is 4 hours 33 minutes.

¹⁰ Each data block is 16 bytes and the control block is 16 bytes. That “next memory block” address of 48 means we will write block $(48-16)/16 = 2$ next.

Live Data Display while Logging

While testing the payload, it is useful to be able to continuously display the port readings while also logging them.

At power-up, live data display is turned off. You can turn it on via the menu command “A” but only if you are already logging port readings. In other words, you are in the “in-flight” state. To get to this state, you must first enter “P” and follow the instructions.

Once in the “in-flight” state, you can monitor the sensors by entering “A”. Expect to see something like this:

Port readings will be displayed as they are collected.

```
Port0 Port1 Port2 || Port3 Port4 Port5 || Port6 battery
2.02  2.14  1.61  ||  1.47  1.59  1.24  ||  1.41  4.78
2.10  2.02  1.59  ||  1.57  1.53  1.22  ||  1.45  4.78
1.98  2.33  1.80  ||  1.45  1.65  1.35  ||  1.41  4.78
```

When data is being displayed, you can enter “A” at any time to print out the heading again. For example:

Port readings will be displayed as they are collected.

```
Port0 Port1 Port2 || Port3 Port4 Port5 || Port6 battery
1.84  2.16  1.76  ||  1.27  1.61  1.39  ||  1.31  4.78
1.86  2.14  1.75  ||  1.29  1.59  1.37  ||  1.31  4.78
1.88  2.14  1.73  ||  1.33  1.59  1.35  ||  1.31  4.78
```

If you stop data recording with the “S” command, you will see

Data collection has been stopped. The EEPROM is now locked.

If you then enter “A”, you will see

Memory is locked so there is no new data to display.

O Output collected data in CSV¹¹ format

Upon landing, the user can request that all data be dumped to the PC. This action locks the EEPROM so none of the collected data is erased. The EEPROM will only be unlocked when the software receives a request to prepare for a new flight. See page 13 for how to capture this data into a log file.

S Stop Data Collection

The EEPROM will be locked and no further collection of the data will occur until the next time “P” is entered. This command is handy immediately after landing when you may want to verify there were no error codes and data was collected but do not want to sit through the download of many blocks of data.

C Calibrate Internal Hardware Clock

The Pro Micro has a hardware based clock that derives its accuracy from a crystal built into the Pro Micro board. This function lets you calibrate this clock to give the best possible accuracy. This value is stored in EEPROM.

There are 1000 milliseconds in a second. If the Pro Micro hardware is accurate to 4 places, then the calibration value will be 1000. The software allows a hardware error of +/- 10% which means it recorded a value of 900 to 1100 milliseconds per second. This value is used to adjust the timing so we collect sensor data every second.

If the Pro Micro board or the EEPROM device are changed on the Flight Data Recorder, it is essential that the hardware clock be calibrated.

When you enter “C”

```
***Clock Calibration Function***
```

```
WARNING: This task takes 3 hours to run. Do not proceed  
unless you are willing to complete the job.
```

```
Type 'A' now to Abort.
```

```
Use https://time.is/ for the best timing source.
```

```
When you are ready to Go, type 'G'.
```

¹¹ CSV means Comma Separated Volume.

If you enter “G”

G

```
***Time calibration has just started.***  
Type 'E' after exactly 3 hours to End calibration  
interval. Type 'A' to Abort.
```

Every minute another “.” will appear. After every hour you will see "About one hour has passed."

The LED will flash once every second.

After exactly 3 hours¹², enter “E” to end the calibration interval. Any calculated Clock Calibration value more than $\pm 10\%$ from the ideal value of 1000 will be rejected.

If you start the calibration procedure but cannot finish, all is not lost. While in the timed interval, you can enter “I”. This will give you the opportunity to initialize the Clock Calibration value to 1000. Doing this is better than nothing but is not a substitute for performing the calibration procedure.

¹² By running for 3 hours, the typical response time of the user is a small enough percentage of the total time that is does not show up in the calibration time. 3 hours equals 10,800,000 milliseconds. If the user took 4,000 milliseconds to press “E”, then the count would be 10,804,000 milliseconds. 10,804 divided by 10,804,000 equals 1000.4. Rounded to 4 places give 1000. So a response time of up to 4 seconds is harmless when we run calibration for 3 hours.

H Diagnostic Output Control Block

When “H” is entered you will see something like this:

```
Control Block
=====
In flight with power hit.
There are no detected faults.
Start of Next Memory Block = 64 (0% full)
Time calibration value = 1000
```

Data will be recorded every 2 seconds for a total run time of 4.55 hours.

Compilation date is Nov 20 2018

C:\Users\Owner\Documents\FDR10.ino

In flight with power hit - software is in flight mode and has experience at least one power hit. Look at the timer in the output data for details of when this happened.

There are no detected faults. - If any faults were detected, they would be spelled out here. These could be hardware or software faults. There is nothing the user can do about these faults except report them to the instructor. If the fault is an intermittent hardware or software fault, replacing the Pro Micro and/or EEPROM would be in order.

Start of Next Memory Block = 64 - This is the address of the next block of data. The first data block starts at 16 and the last possible data block starts at 131,060. The memory is 0% full.

Time calibration value = 1000 - This is the number of hardware based millisecond counts in one externally measured second. Each of these counts is ideally 1 millisecond so ideally, this value is 1000. The software allows the user to adjust this value by as much as $\pm 10\%$ due to variation in the hardware. See page 23 for the calibration procedure.

Control Block

=====

In flight with power hit.

There are no detected faults.

Start of Next Memory Block = 64 (0% full)

Time calibration value = 1000

Data will be recorded every 2 seconds for a total
run time of 4.55 hours.

Compilation date is Nov 20 2018

C:\Users\Owner\Documents\FDR10.ino

Next we see the data collection rate and maximum run time. Given that we record data every 2 seconds, we can collect data for 4.55 hours which is 4 hours 33 minutes.

The rest of the information relates to a check on the program's version number. You see the day the program was compiled and the location of the program file in the development environment.

Software Details

There are three guiding principles used to write this software:

1. Structured – similar functions are grouped into the same subroutine.
2. People readable first – it is far more important that the code be readable by people and only secondarily readable by the computer
3. Layered functionality – keep breaking down the tasks into subroutines

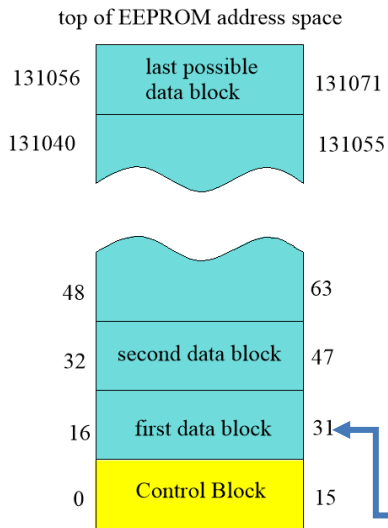
The software is dealing with the EEPROM so let's look at it first.

The EEPROM has an address space of 17 bits. This means there are a total of 2^{17} bytes available. Since we start counting from 0, the highest address is $2^{17} - 1 = 131,071$.

The control block occupies the first 16 bytes so its addresses go from 0 to 15. At the present time, we are only using the first 11 bytes. The rest is for future growth. Any control information that must survive a loss of power is stored in the control block.

Our first data block starts at address 16. One data block is 16 bytes (0 – 15) so this data block ends at $16+15 = 31$. We

continue this pattern until we read the end of memory. The last possible data block starts at 131,056 and ends at 131,071. When we run out of memory, it will be locked with a next block starting address of 131,072.



The maximum number of data blocks is

$$\frac{\textit{maximum address} - \textit{start of data} + 1}{\textit{data block size}}$$

We add 1 because we want to know the total count and not the difference¹³.

$$\frac{131071 - 16 + 1}{16} = 8191$$

We generate a data block every two seconds so if we fill the EEPROM, it will hold 8191 X 2 seconds of data. This is 4 hours, 33 minutes, and 2 seconds of run time.

¹³ For example, say I have items 1 through 5 so 5 items. I must subtract 1 from 5 to see the different and then add 1 to get the count.

Now we can start talking about the software.
At the highest level we have:

```
void loop()
{
    Time(); //applies on the ground and in the air
    OnTheGround(); //tasks only done on the ground
    InTheAir(); //tasks mostly done in the air
}
```

Within Time () we have

```
    RefreshTimeCalibrationConstantQ();
    LapseTimerSeconds();
    LED_Cadences();
```

The Arduino compiler does not permit special characters so I use a “Q” to represent a “?”. RefreshTimeCalibrationConstantQ () deals with ensuring that the hardware timer is running accurately even when there has been a momentary power loss. More on this later.

LapseTimerSeconds () deals with the time since the last time power was turned on. Built into the name of this subroutine are its units.

LED_Cadences () generates a number of different cadences used to flash the external LED. From the “D” command, you saw:

```
LED semaphore
1 flash per cycle - normal data collection.
2 flashes per cycle - normal data collection after power disruption.
3 flashes per cycle - memory locked either manually or it is full.
4 flashes per cycle - pre-flight.
5 flashes per cycle - hardware or software fault detected.
```

These flashing patterns let the user know the state of the payload without connecting up the PC. This will be useful when the payload is retrieved from the landing site.

```

void loop()
{
    Time(); //applies on the ground and in the air
    OnTheGround(); //tasks only done on the ground
    InTheAir(); //tasks mostly done in the air
}

```

The second subroutine in loop() is OnTheGround() which contains

```

OnTheGround() {
    ControlTheFlightParameters();
    ScanForCommand();
    OutputMenuQ(); //M command
    SingleDisplayDataQ(); //D command
    PrepareForLaunchQ(); //P command
    ContinuousDisplayDataQ(); //A command CR1.7
    OutputDataFileQ(); //O command
    StopCollectingDataQ(); //S command
    CalibrateClockQ(); //C command
    DiagnosticOutputControlBlockQ(); //H command
    GenerateTestPatternQ(); //T command
    FlashLED();
}

```

I will discuss ControlTheFlightParameters() later and address ScanForCommand() next.

ScanForCommand() looks for single letter commands from the PC's keyboard:

M This menu.
D Display all port voltages and set the pre-flight state.
P Prepare for launch!
A Use only while in flight: Display all port voltages.
O Output data file.
S Stop data collection.
C Calibrate internal hardware clock.
H Diagnostic: Output control block.
T Diagnostic: Generate test pattern in data memory.

These commands are sent to all of the subroutines ending in "Q".

```
OutputMenuQ(); //M command  
SingleDisplayDataQ(); //D command  
PrepareForLaunchQ(); //P command  
ContinuousDisplayDataQ(); //A command CR1.7  
OutputDataFileQ(); //O command  
StopCollectingDataQ(); //S command  
CalibrateClockQ(); //C command  
DiagnoticOutputControlBlockQ(); //H command  
GenerateTestPatternQ(); //T command
```

If this is painfully obvious, then I've done my job. The software should be "People readable first".

GenerateTestPatternQ () can be used to exercise the steps from outputting the data to graphing it in EXCEL. When you enter “T” you will see

```
You asked to generate a test pattern in the data.  
This will erase all existing data. Are you sure? Y/N
```

```
Test Pattern generation about to start.  
.....  
Done.
```

Each “. ” means a data block has been written.

Then if you enter “O” to output the data you will see

```
Minutes, Seconds, Port0, Port1, Port2, Port3, Port4, Port5, Port6, Battery,  
0, 2, 0.00, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 5.07,  
0, 4, 0.08, 0.09, 0.10, 0.11, 0.12, 0.13, 0.14, 5.15,  
0, 6, 0.16, 0.17, 0.18, 0.19, 0.20, 0.21, 0.22, 5.23,  
0, 8, 0.23, 0.24, 0.25, 0.26, 0.27, 0.28, 0.29, 5.31,  
0, 10, 0.31, 0.32, 0.33, 0.34, 0.35, 0.36, 0.37, 5.39,  
0, 12, 0.39, 0.40, 0.41, 0.42, 0.43, 0.44, 0.45, 5.46,  
0, 14, 0.47, 0.48, 0.49, 0.50, 0.51, 0.52, 0.53, 5.54,  
0, 16, 0.55, 0.56, 0.57, 0.58, 0.59, 0.60, 0.61, 5.62,  
0, 18, 0.63, 0.64, 0.65, 0.65, 0.66, 0.67, 0.68, 5.70,  
0, 20, 0.70, 0.71, 0.72, 0.73, 0.74, 0.75, 0.76, 5.78,
```

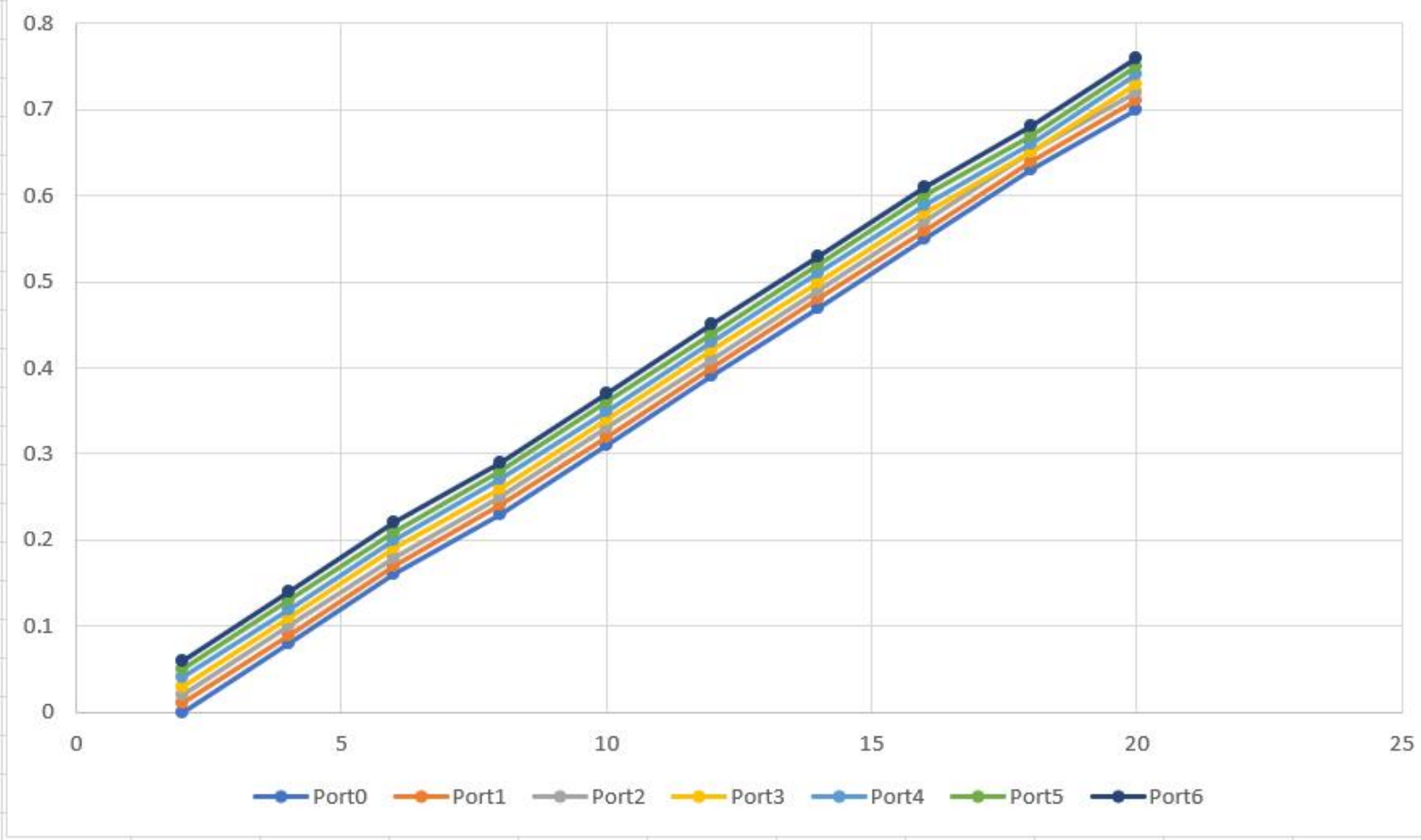
As currently set¹⁴, you get 10 blocks of data starting at 2 seconds and going through 20 seconds.

The first two fields are the time stamp. Ports 0 through 6 follow with an increase of 0.01 volts per port. The last entry is 5 plus the count. For example, in the first record, port 6 equals 0.06 volts and Battery is 5.07 volts. Port 0 in the next block is 0.08. You will see round off error in this data of ± 0.01 volts.

¹⁴ SamplingRateByte determines how often the program records and plays back the data.

Graphed in EXCEL, the port data looks like this:

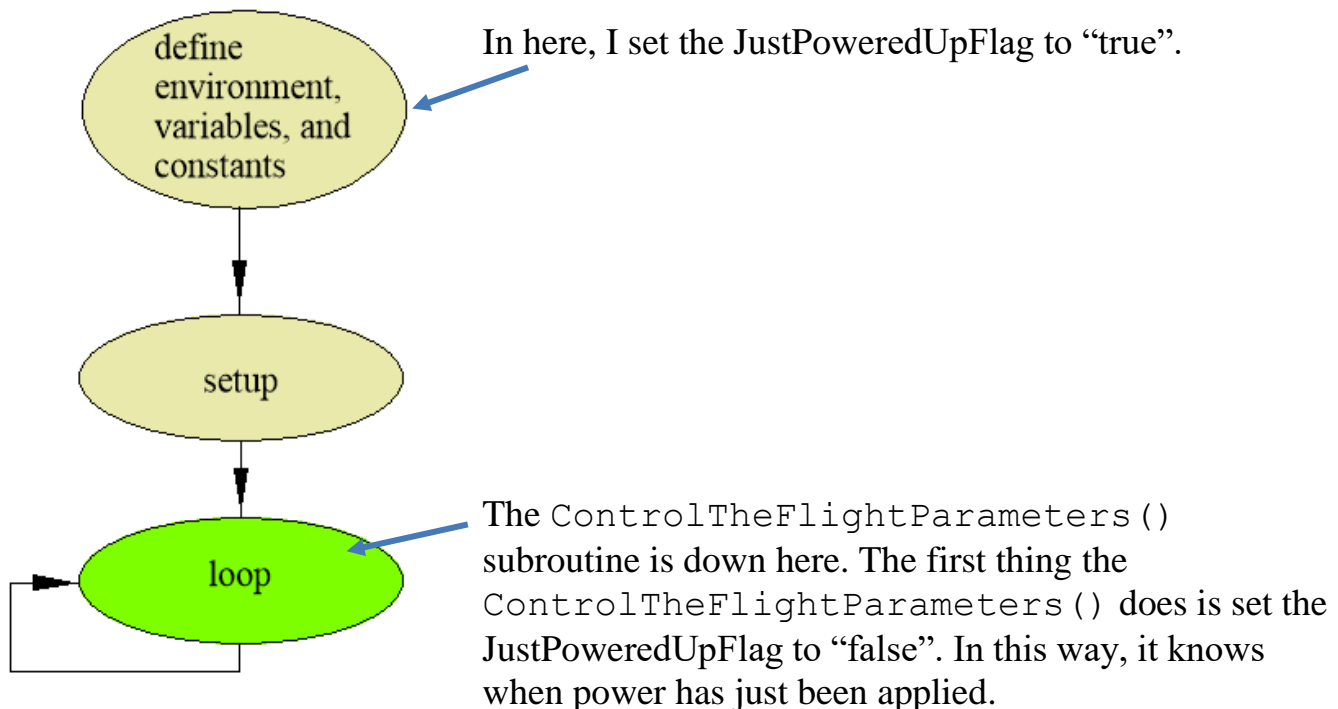
Test Pattern



This regular pattern makes it easier to identify logic errors in EXCEL that processes this data.

The last subroutine in `OnTheGround()` is `FlashLED()` which depends on the `Time()` subroutine plus `Flight Parameters` to flash the external LED at the correct cadence.

Going back to the first subroutine of `OnTheGround()` we have `ControlTheFlightParameters()` which solves the problem of what to do when power is first applied to the Arduino.



Then it must figure out what needs to be done *next*. The options are:

- We *were* set to “just wait” so *now* continue to just wait.
- We *were* set to “ready to launch” so *now* are “in flight”.
- We *were* “in flight”, took a power hit so we have *now* experienced “disrupted power” so must get back to being “in flight”.

The Control Block stored in the EEPROM tells the software where it was before power was lost.

We automatically get into “just wait” when we ask for a Display of the port voltages. Data is not collected, we do not prepare to launch, and we do not continue to collect data after power is restored.

```
void loop()
{
    Time(); //applies on the ground and in the air
    OnTheGround(); //tasks only done on the ground
    InTheAir(); //tasks mostly done in the air
}
```

So far we have addressed `Time()` and what to do `OnTheGround()`. The last top level subroutine is what to do `InTheAir()`. This subroutine must know when to process data. The rule is that data is collected and saved only when we are in flight, the memory is not locked, and it has been two seconds since the last data cycle. “In flight” is the time between power just re-applied after we prepared for flight and the time we stop collecting data with either a D, S, or O command.

Where from here?

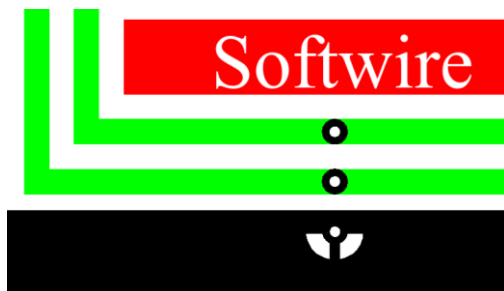
You are ready to look at the code and understand the details of how it works plus how it was structured and documented. It will take time to comprehend even though all constants, variables, and subroutine names are designed to be descriptive.

Adding Digital Sensors

The first line of the program is

```
///#define SoftwareActive 1 //comment out if ports 5 and 6 are to be used
```

Those first two forward slashes (//) make this line invisible to the compiler. Remove those forward slashes and you transform sensor ports 5 and 6 into a I2C bus.



The circuit board has a three pin connector with Software etched into the copper. This connector has

SCL
SDA
and ground.

A cable plugs in here and goes over to a new board that holds digital sensors. As needed, run +5V sensor power too.

The software needed to support this interface is called Software. Search the program for “SoftwareActive” and you will see how it turns off ports 5 and 6 and also turns off the port scanning subroutine. Along with selecting digital sensors, you must locate their drivers and supporting documentation. Ultimately you need to address each sensor and have it return its data. This data is then stored in EEPROM. If you limit each data point to two byte, it will fit into the existing storage and readout scheme as described on page 27.

Indirect Addressing

This is an essential software technique that is not always so easy to comprehend.



Let me try to sneak up on this topic by starting with a physical example: candy and a cup. The candy is a constant and the cup is a place to put things.

cup



Say I want to write a command that says to put the candy in the cup.

```
candy → cup
```

cup

In the language used to program the Arduino, C/C++

```
C = 4;
```

4 is a constant just like the candy and C is the cup.



bowl



cup

But what if I want to have a more flexible rule that says to put whatever is in the bowl into the box?

```
what is in the bowl → cup
```

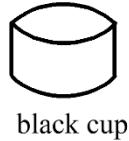
```
C = B;
```

or even

```
Cup = WhatIsInTheBowl;
```

This will let me have a fixed rule yet change the contents of the bowl and it will always end up in the cup.

What if I had three cups? I might want the flexibility to do



RedCup = WhatIsInTheRedBowl;

Or



BlackCup = WhatIsInTheBlueBowl;

Or

BlueCup = WhatIsInTheBlackBowl;

I could write three commands and select between them but as the number of cups increases, this becomes impractical. What about adding a box that will hold a piece of paper with the name of the cup on it?

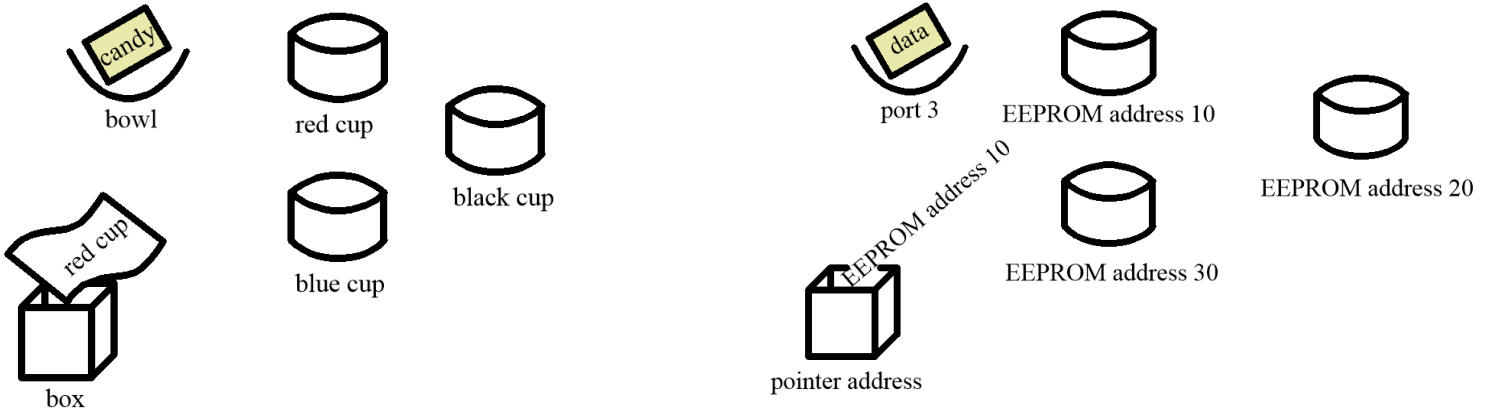


1. Read note in box to find which cup to use
2. Pick up what is in that bowl
3. Put it into that cup



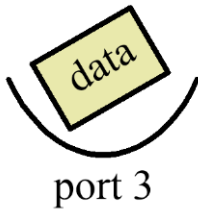
By changing the note in the box, I change which cup to fill. This is called *indirect addressing*.

Now we are ready to apply this to the program.

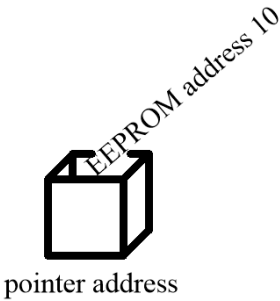


Our “candy” is now our *data* and the “bowl” is now a *port*. The “box” is now our *pointer address*. The “note” marked *red cup* is now our *EEPROM address pointer*. Instead of having red, black, and blue cups we have *EEPROM address 10, 20, and 30*.

Let’s try this out.



I scan port 3 and have data.

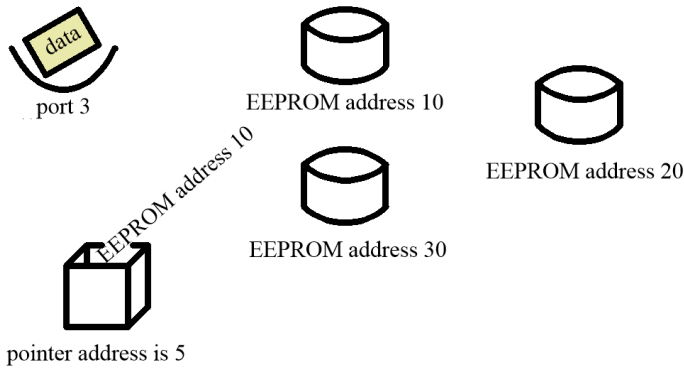


Looking in the pointer address, I see that the next available EEPROM address pointer is 10.



The data is moved to EEPROM address 10.

We can then increment the EEPROM address pointer to 20 and start the cycle over. The next data to arrive will end up being stored in EEPROM address to 20 with no change in the logic.



What can be confusing is that one address, pointer address, is used to find another address, EEPROM address 10.

Now we are ready to look at “pseudo” code. This is English text which can be translated to Arduino code.

```

Setup: //done once
Data; //define a variable called data which holds the value collected from a port

StartOfNextMemoryBlockPointerAddress = 10; //The memory location called “StartOfNextMemoryBlockPointerAddress” has been set to the value 10;

```

Logic:

//get the address of the next available EEPROM address which is 10

```

10 StartOfNextMemoryBlockPointer = Read(StartOfNextMemoryBlockPointerAddress);

```

//write the data to this next available EEPROM address so data is put in location 10
Write(StartOfNextMemoryBlockPointer, data);

//increment the EEPROM memory pointer value by 10 so it becomes 20

StartOfNextMemoryBlockPointer = StartOfNextMemoryBlockPointer + 10;

//save the updated StartOfNextMemoryBlockPointer back in its memory location

```

20 Write(StartOfNextMemoryBlockPointerAddress, StartOfNextMemoryBlockPointer);

```

StartOfNextMemoryBlockPointerAddress

We are now ready to get the next port’s data. All logic will operate the same except this data will be stored in EEPROM memory location 20.

Acknowledgment

Thanks to José Inzunza and John Herrmann for reviewing this document and providing invaluable insights.

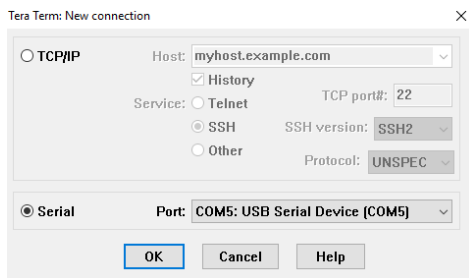
Rick Sparber

Appendix: Tera Term

Tera Term is a program able to send keystrokes to the Flight Data Recorder and received data from it. This data is displayed on the PC screen plus can be saved in a file for later processing. One such program is called Tera Term (<https://ttssh2.osdn.jp/index.html.en>).

Load it on a PC where you have Admin privileges. The PC will be used to send commands to the software and receive both status information and data dumps to a file.

PC Setup



1. Connect the USB cable between the processor board and the PC.
2. Turn on power to the Data Logger.
3. Bring up Tera Term.
4. Watch

<https://www.youtube.com/watch?v=rGrKRpol7PA> for how to set up Tera Term and save it as a short cut. You will then only have to double click this short cut to have all parameters set to the correct values. All of the following menu items are on the Setup tab. Click OK when leaving each screen.

Terminal	Terminal size 80 X 30
Window	enter a Title for the short cut and choose the look of the command window. Set Scroll buffer to 10000 lines.
Font	select a font and size you like
Serial port	select the COM port you are using. The Baud rate is 19200.
Save setup	save all of the setting you just performed in a file called DataLogger. Remember where you put this file and you will need it next

Go to the folder holding the Tera Term application and create a short cut for the desktop. Then right click on the short cut and select Properties. In the Target window, add to the end

```
/f= DataLogger
```

Then click OK. When you want to access the Flight Data Logger, just click on this short cut and it will come up with your desired parameters preset.