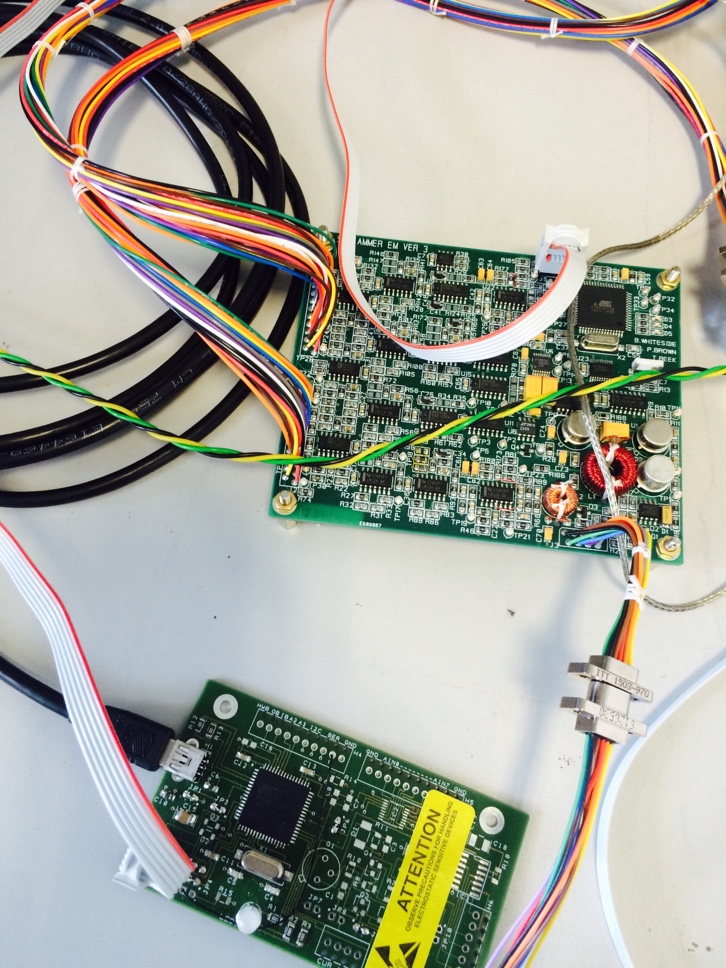
**SUNJAMMER\_DEMO USER GUIDE   
  
  
  
  
  
  
02 September 2015  
Version 1.0.0  
   
  
  
  
  
  
  
  
  
1  
Software Revisions**

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Changes** |
| **02/09/2015** | **1.0.0** | **Initial Version** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**2  
Table of Contents  
  
1) Introduction ……………………………………………………………. 4  
 1.1) Scope and Purpose ……………………………………………… 4  
 1.2)Program Overview ………………………………………………. 4  
2) Setting Up ……………………………………………………………….. 5  
3) Program flow …………………………………………………………… 7  
 3.1) Commands ………………………………………………………….. 8  
 3.2) Telementery bits …………………………………………………. 10  
 4) Design Considerations and Limitations …………………….. 11  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
3  
  
  
  
  
1 INTRODUCTION  
  
1.1 Scope and Purpose**The main purpose of this program is to read and process inputs from connected SPI devices via an off chip Analogue to Digital converter.   
  
This guide will cover the functionalities of the program, how to compile and load the program onto a chip, how to add extra features to the program and possible limitations. **1.2 Program Overview**8 12 bit values are read from the SPI device (in this case a sensor) and then written to a circular buffer on the microcontroller periodically by the use of an interrupt service routine. After being written to the buffer several things can be done to this data **4  
  
  
  
2 SETTING UP  
  
Step 0:** Connect the programmer to the board as shown below.  
  
Notice the position of the red wire in the white ISP connector linking the programmer to the board. It is important to have this at the right position as failure to do so may result in destruction of various components of the board.  
 **Step 1:** Put on the power supply to the Sunjammer Engineering Model board. Press the recall button twice to recall the power   
**5**  
settings for the board. The voltage should be about 15V and current just over 100 mA. Ensure the serial is connected to the computer and the sensor (or other SPI device) is connected to the board as well. **Step 2**: Make sure you’re running a version of Arduino with the Crumbuino additions installed (this may require installation of a previous version of the Arduino software). This is important because the Crumbuino uses the same CPU as the current Sunjammer EM Board (ATMega128) hence the hex file produced on compilation will be compatible with the Sunjammer board. The Arduino environment does not include compilation for the ATMega128 directly.   
  
The easiest way to go is to dump a copy of Tim Oddy’s environment (<http://balius.sp.ph.ic.ac.uk/Sunjammer/>) onto a machine and just create some shortcuts to the Arduino executable. The Crumbuino additions are already installed.   
  
**Step 3**: Open the Arduino program, go to Tools -> Board and select Crumbuino-128 w/ ATMega128  
  
**Step 4**: Go to Sketch -> Export Compiled Binary. A green bar will appear on the bottom right of the application, with the words “compiling sketch…” on the left. Wait till the green bar fills up. This should take 2-3 seconds.   
  
When the bar is full, a compiled version of the current Arduino sketch will be created in the same directory the sketch is located.  
   
**Step 5**: Open command prompt and cd to the directory containing the sketch. Type this command **"C:\WinAVR-20100110\bin\avrdude.exe" -P COM3 -p m128 -c arduino -U flash:w:"{{sketch\_name}}.cpp.crumbuino128.hex":I**   
**6**  
  
A series of status bars should subsequently fill up indicating the program being read and written to the microcontroller, after which your program should now work on the microcontroller.  
  
Usually for serial input and output communication, PuTTY is the standard application, however there is a serial monitor provided by Arduino at Tools -> Serial Monitor which may be simpler to use.  
  
**3 Program Flow**  
  
Upon loading, the program shines the red LED then outputs some copyright information and asks the user to select a mode to start.  
  
The modes are:  
0 - Normal Data Mode  
1 - Raw Data Mode  
2 - Telementery Mode 16Hz  
3 - Telementery Mode 0Hz  
  
Normal Data Mode  
This is one of the two “pull” modes. Data is sampled at 1 sample per minute. Every interrupt, when data is sampled, the vector is written to the buffer. The vector is also displayed on the serial monitor when written to buffer. This feature can be turned off by pressing ‘f’, as it makes it seem like it also pushes data as well. To pull data, ‘r’ can be pressed to perform a buffer read. Information on all possible commands can be found further on in this manual.  
  
Raw Data Mode  
This is basically the same as the Normal data mode except that data is now sampled at 16 samples per second. Commands can be used to   
**7**  
alter this mode as well  
  
Telementery Mode 16Hz  
This is the first “push” mode. Both push modes follow the telementery format which will be discussed in detail further on in this manual. Data is sampled at 16Hz but a status packet is sent out every second showing a few details, including the last 16 vectors read from the SPI device.  
  
Telementery Mode 0Hz  
Basically the same as the 16Hz. Data is still sampled at 16Hz except this time the packets are only sent every minute.  
  
Upon selection of the desired mode, interrupts begin. The orange LED on the board will turn on for 1/60Hz modes and the yellow LED will turn on for 16Hz modes.   
  
**3.1 Commands**  
  
While the program is running, the user can send input characters to the program for certain commands, and can also switch between modes. The possible commands are:  
  
r - Read from buffer");  
k - Read hosekeeping data");

h - Help");

o - Telementary on");

f - Telementary off");

g - Graphical Display on");

n - Normal Telementary on ");  
 **8**

d - Data dump");

v - Voltage Display on");

0 - Normal Data Mode");

1 - Raw Data Mode");

2 - Telementery Mode 16Hz");

3 - Telementery Mode 0Hz");

4 - Switch Inboard Range");

5 - Switch Outboard Range");  
  
r – Reads the earliest item in the circular buffer, removes it from the buffer, shows it on screen and stops vectors being written from showing on the serial monitor to prevent clutter. Only works in Raw and Normal data modes as it is useless in the other two  
  
k – Shows some basic housekeeping data for the Raw and Normal data modes such as time run so far, current mode, outbard and inboard range, vcc 5v, ref 5v and the buffer read and write positions. Will work with the telementery modes but not needed  
  
h – Shows the full list of commands.   
  
o – Allows data to be printed on the serial monitor during the Raw and Normal modes. Has nothing to do with the actual telementery modes, just a clash of definitions  
  
f – Stops data from being written to the screen during Raw and Normal modes. No effect in the others  
  
g – Changes the data shown on the serial monitor during Raw and   
 **9**  
Normal modes from engineering units (b/w 0 and 4095) to x-axis graphs which change as the value from the SPI device changes.  
  
n – Changes the displays back to engineering units during Raw and Normal modes.  
  
d - Does a buffer read of the earliest 16 vectors. Useful if the machine doing a pull has missed a few seconds worth of data and wants to get back up to speed. Not useful in the telementery modes.  
  
v – Changes display to voltages (0.0 – 5.0) durning Raw and Normal modes.

4 – Switches the value on the inboard range from high to low or vice versa. This determines the range the SPI device in board reads.  
  
5 – Switches the value on the outboard range from high to low or vice versa. This determines the range the SPI device out board reads.  
  
**3.2 Telemetery bits**The telementery modes push a packet of data containing 200 bytes with the following format:

|  |  |
| --- | --- |
| Byte | Significance |
| 1 | Packet Count |
| 2 | Status |
| 3 | Command Count |
| 4 – 195 | 16 x 8 x 12bit vectors |
| 196 | 5V Ref Value |
| 198 | 5V VCC Value |
| 200 | Carriage Return |

**10**

Packet count – Number of packets sent so far  
Status – Status byte.

|  |  |
| --- | --- |
| Bit | Significance |
| 0 | Auto range or manual ranging |
| 1 | Current OB range |
| 2 | Current IB range |
| 3 | Mode (16Hz or 1 min) |
| 4 | IB Data Sent |
| 5 | OB Data Sent |

Command Count – Number of commands sent so far  
Vectors – Every packet contains 16 vectors, each vector contains 8 axis values in engineering units (12 bits). Hence 1536 bits = 192 bytes  
Ref Value – Reference value  
VCC Value - Power supply value  
Carriage Return – New line character.  
  
**4 DESIGN CONSIDERATIONS AND LIMITATIONS**  
  
This piece of code is designed for the ATMega128 microcontroller. The microcontroller has 4KB of internal RAM. As a result certain features of the code have to be limited in size. Since each vector is 12 bytes I have limited the maximum buffer size to 128 (1536 bytes = ~1.4 kB).   
  
Also I have tried to minimise the number of strings in the code (i.e in the help, housekeeping methods). It is worth looking into flash strings if you want to include large bodies of text.  
  
  
 **11**  
Occasionally the chip might crash when the power is turned on and the current will drop to about 0.015mA. In this case just turn the output off and back on again.  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
 **12**