BlueSense2

0. History

- 2.11.2016: updated compilation instructions

1. General technical data

30x30mm
Atmel AVR 1284P processor
MPU-9250 motion sensor
SD card
Bluetooth 2.1: Roving Networks RN-41
USB: FTDI FT200X
Extensible

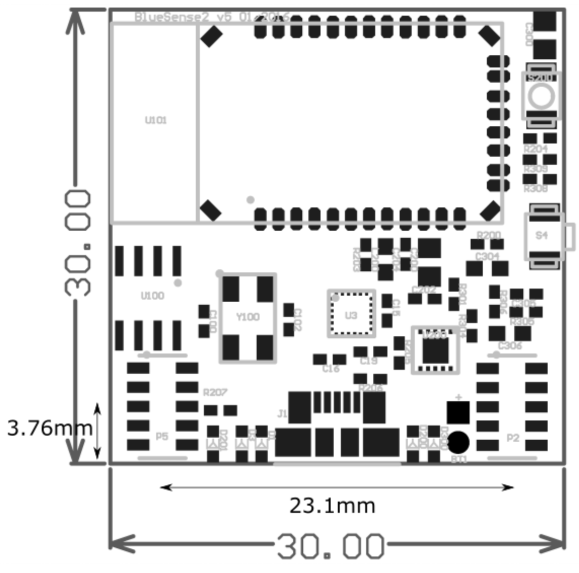
1.1 Hardware revisions

HW5
Hangs when powering up without connection to USB due to FTDI latch-up.
Fully functional if powered on with USB power and USB connection is then removed.
HW6
First version without blocker bugs.
HW7

- Changed pinout of expansion connector: removed I2C on P5 and substituted by RTC_INT and X_ADC7
- The resistors/capacitors R308/R309/C308 for VBatHalf on ADC7 removed to allow for X_ADC7
- Added solder pad J200 connected to Bluetooth module GPIO10
- Changed color of LED2 to green
- Moved USB connector outwards by 0.1mm
- Larger via hole and pad for battery
- Hardware version on silkscreen below antenna
- Increase R308/R309 voltage divider to minimize losses: changed from 470K to 1.5M each

1.2 Expansion connector top: v5, v6

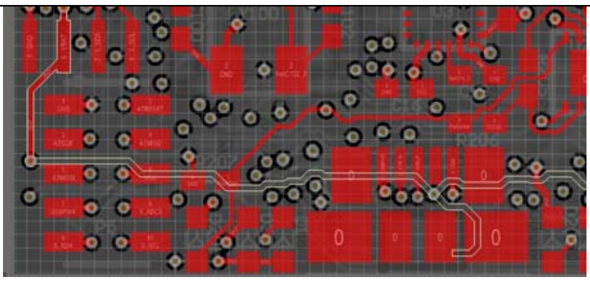
Top view:

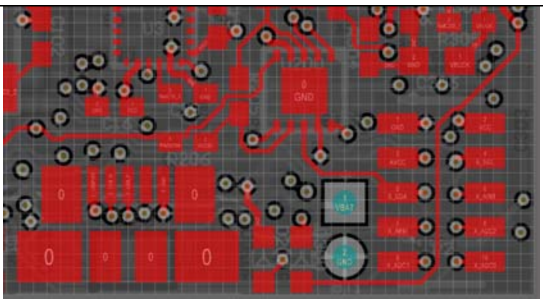


Connector on board: CLP-105-02-L-D
Connector on extension: Samtec FTSH-105-03-L-DV
Pinout:

Top left (P5)			Top right (P2)		
	GND	ATRESET		GND	VCC
	ATSCK	ATMISO		AVCC	S_SCL
	ATMOSI	VCC		S_SDA	X_AIN1
	USBPWR	X_ADC3		X_AIN0	X_ADC2
	S_SDA	S_SCL		X_ADC1	X_ADC0

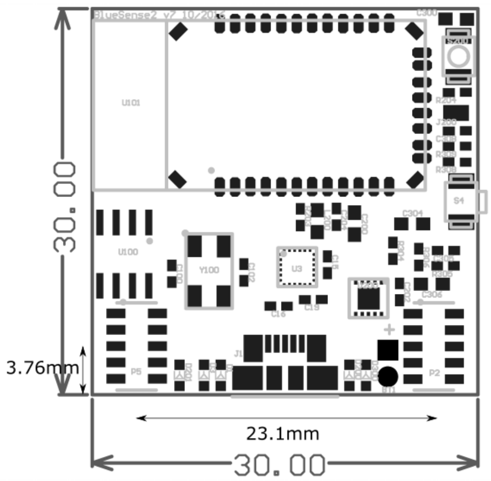
Screenshot:


Top left (P5)


Top right (P2)

1.3 Expansion connector top: v7

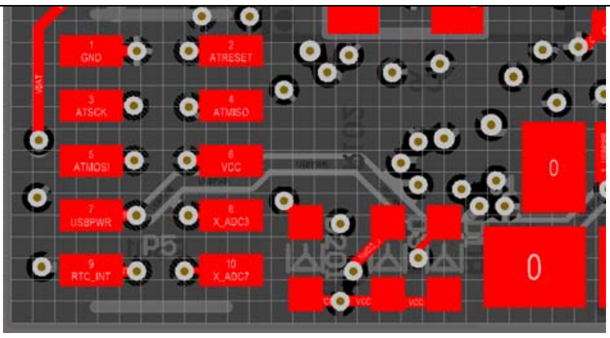
Top view:



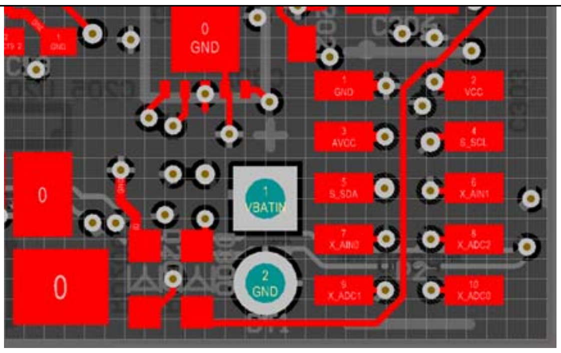
Connector on board: CLP-105-02-L-D
Connector on extension: Samtec FTSH-105-03-L-DV
Pinout:

Top left (P5)			Top right (P2)		
	GND	ATRESET		GND	VCC
	ATSCK	ATMISO		AVCC	S_SCL
	ATMOSI	VCC		S_SDA	X_AIN1
	USBPWR	X_ADC3		X_AIN0	X_ADC2
	RTC_INT	X_ADC7		X_ADC1	X_ADC0

Screenshot:



Top left (P5)



Top right (P2)

1.4 Expansion connector bottom

The bottom connector is pin-to-pin aligned with the top connector for all hardware versions.

2. Development environment setup

2.1 Compiler

- AVR GCC5 is needed to compile the firmware (available in /tools).
    - Suggested installation location: c:\avr-gcc-5
    - Ensure the bin directory is in the path (e.g. C:\avr-gcc-5\bin)

2.2 Development environment

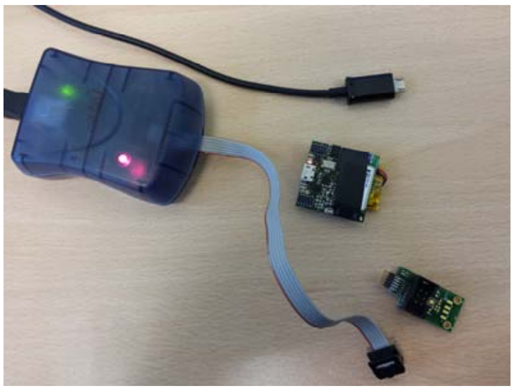
Any IDE or text editor can be used.

A project for use with Programmer's notepad (http://www.pnotepad.org/) is provided in the firmware. Tooling for Programmer's notepad is provided in the distribution available at /tools/pn.zip: it allows for 'make compile', 'make' and 'make program' directly from the GUI assuming avr-gcc is installed in c:\avr-gcc-5

2.3 Programmer and programmer's drivers

The following hardware is required to program the sensor:

- AVRISP MKII (or equivalent)
- 6-pin header to BlueSense2 programming adapter
- Micro-USB cable to recharge and communicate with the sensor over USB
- A PC with Bluetooth to communicate with the sensor over Bluetooth



Sensor node with the AVRISP MKII programmer, the programming adapter and the micro-usb charging/communication cable.

The drivers for the AVRISP MK2 are available on tools/driver-atmel-bundle-7.0.888.exe.

In order to program the sensor directly from Programmer's notepad the avrdude programmer is used which requires libusb to be installed. Libusb is available in tools/libusb-win32-bin-1.2.6.0.zip. First install the Atmel drivers and then libusb selecting to install it for the device "AVRISP mkII". Under windows 10 this requires to reboot into advanced mode (holding shift and selecting the reboot option in the start menu) and enable the installation of unsigned drivers.

2.4 Terminal

A terminal must be installed to communicate with the sensor, either over Bluetooth or USB.
Recommended:

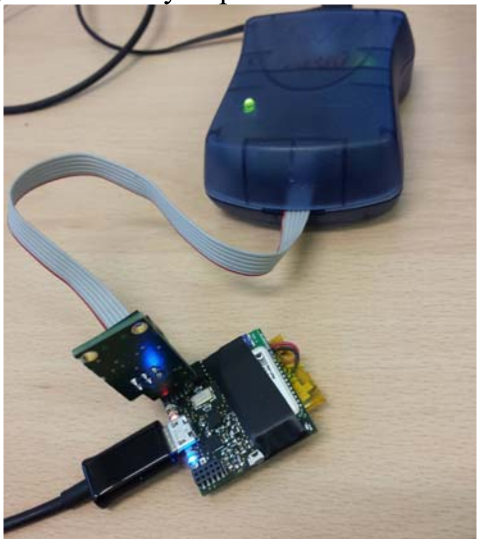
- Termite 3.2 (or better): http://www.compuphase.com/software_termite.htm

Alternative:

- TeraTerm 4.89 (or better): https://en.osdn.jp/projects/ttssh2/releases/

3. Connecting the sensor to the computer

- Connect a microusb cable from the sensor to the PC
- Connect the AVRISP MKII to the programming adapter
- Connect the programming adapter to the sensor node. Be careful to use the correct connector and correct orientation! The programming connector is on the side of the USB connector and Bluetooth chip, and it is on the side of the quartz (silver colour component) and the real-time clock (8-pin SOIC). Make sure you put the connector in the right orientation!



The AVRISP MKII programmer is connected to the sensor node and the micro-USB for recharging/communication.

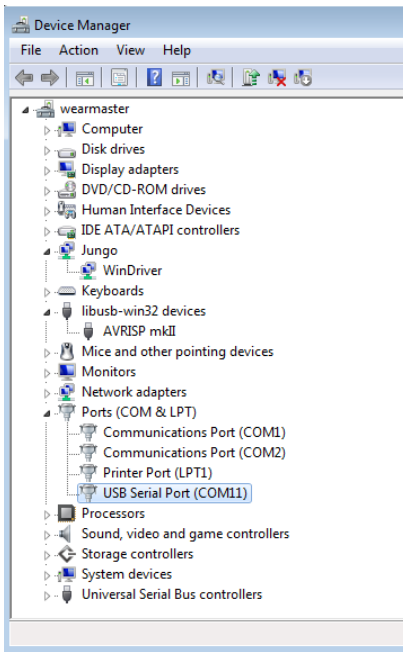
If the sensor is on and the programming connector properly placed then a green light will turn on the programmer.

4. Verifying driver availability

Drivers are required for the programmer (AVRISP MKII) and the USB chip on the node. On a new computer it may be required to manually install these.

Go in the control panel/system and security/system/device manager and check that the drivers are installed:

- The AVRISP MKII is recognised if it appears either under the header "Jungo" or "libusb-win32 devices".
- The USB chip of the node appears under Ports as "USB Serial Port".



Device Manager

```
wearmaster
  Computer
  Disk drives
  Display adapters
  DVD/CD-ROM drives
  Human Interface Devices
  IDE ATA/ATAPI controllers
  Jungo
    WinDriver
  Keyboards
  libusb-win32 devices
    AVRISP mkII
  Mice and other pointing devices
  Monitors
  Network adapters
  Ports (COM & LPT)
    Communications Port (COM1)
    Communications Port (COM2)
    Printer Port (LPT1)
    USB Serial Port (COM11)
  Processors
  Sound, video and game controllers
  Storage controllers
  System devices
  Universal Serial Bus controllers
```

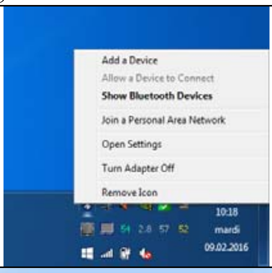
The AVRISP MKII and the FTDI USB interface chip are recognised. Communication over USB occurs through COM11 on this computer.

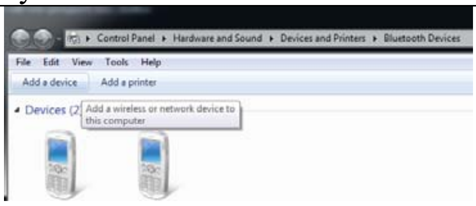
5. Setting up communication and checking ports

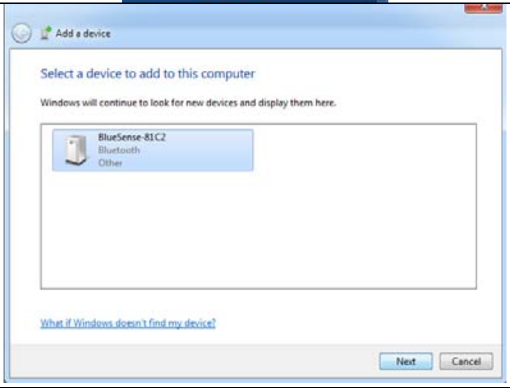
Communication with the node occurs over two virtual serial ports through the Bluetooth chip or the USB cable. In order to communicate with the sensor the port number corresponding to these two interfaces is required.
The device manager (see above) shows the port number corresponding to the USB chip (COM11) in the figure.

For Bluetooth, several steps are required:

- Add the bluetooth device (using the control panel or bluetooth icon in the system tray). It will appear as "BlueSense-xxxx". Select it and provide the pin code, which is "0000"
- Windows will install a serial port profile driver and indicate which COM port corresponds to the Bluetooth link. Usually two COM ports appear and the lower numbered one is the correct one, however in case of communication issues try the other number.




Control Panel ▸ Hardware and Sound ▸ Devices and Printers ▸ Bluetooth Devices
File Edit View Tools Help
Add a device
Add a printer
Devices (2
Add a wireless or network device to this computer

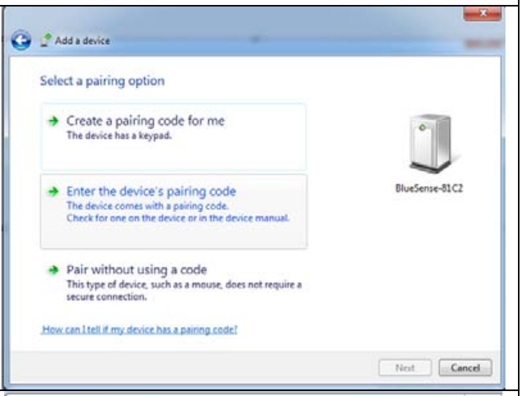

Add a device

Select a device to add to this computer

Windows will continue to look for new devices and display them here.

BlueSense-81C2
Bluetooth
Other

What if Windows doesn't find my device?


Add a device

Select a pairing option

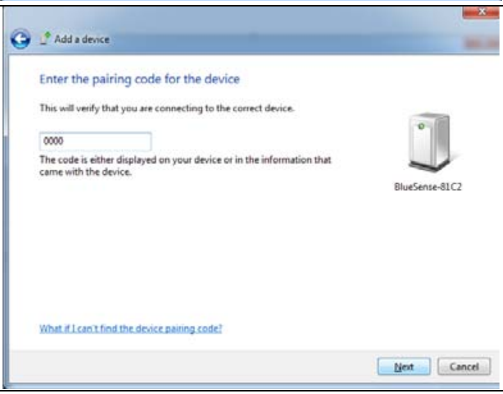
Create a pairing code for me
The device has a keypad.

Enter the device's pairing code
The device comes with a pairing code.
Check for one on the device or in the device manual.

Pair without using a code
This type of device, such as a mouse, does not require a secure connection.

BlueSense-81C2

How can I tell if my device has a pairing code?


Add a device

Enter the pairing code for the device

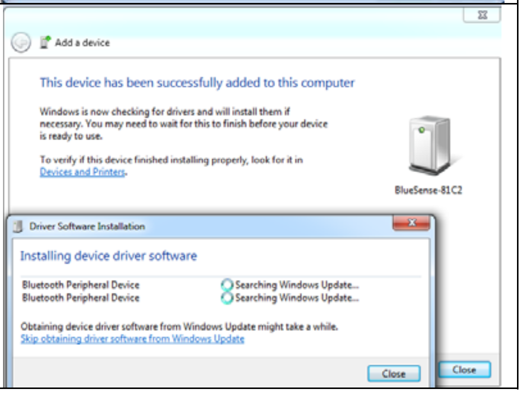
This will verify that you are connecting to the correct device.

0000

The code is either displayed on your device or in the information that came with the device.

BlueSense-81C2

What if I can't find the device pairing code?


Add a device

This device has been successfully added to this computer

Windows is now checking for drivers and will install them if necessary. You may need to wait for this to finish before your device is ready to use.

To verify if this device finished installing properly, look for it in Devices and Printers.

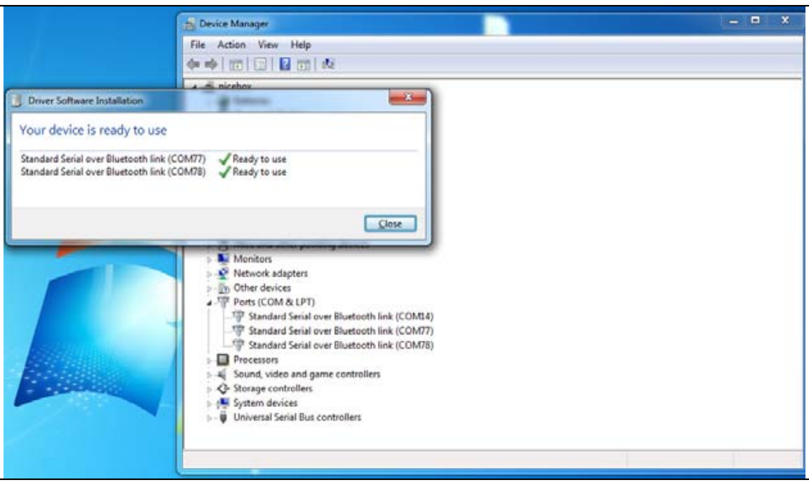
BlueSense-81C2

Driver Software Installation

Installing device driver software

Bluetooth Peripheral Device	Searching Windows Update...
Bluetooth Peripheral Device	Searching Windows Update...

Obtaining device driver software from Windows Update might take a while.
Skip obtaining driver software from Windows Update

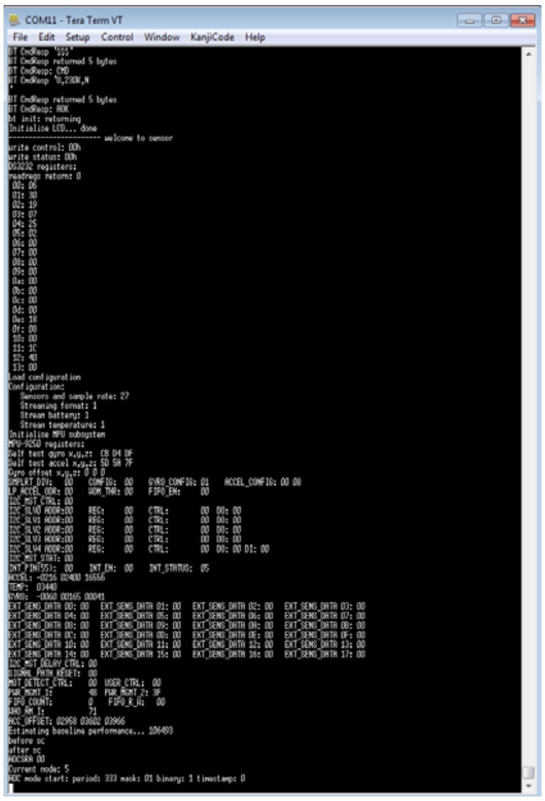

Your device is ready to use

Standard Serial over Bluetooth link (COM77) Ready to use
Standard Serial over Bluetooth link (COM78) Ready to use
Device Manager
File Action View Help
Driver Software Installation
Close
Monitors
Network adapters
Other devices
Ports (COM & LPT)
Standard Serial over Bluetooth link (COM14)
Standard Serial over Bluetooth link (COM77)
Standard Serial over Bluetooth link (COM78)
Processors
Sound, video and game controllers
Storage controllers
System devices
Universal Serial Bus controllers

6. Turning on and testing the sensor

The sensor turns on when plugged into the USB port. Alternatively press the power button a half second to turn it on.
To turn off the sensor press and hold the power button for 10 seconds.

Connect the sensor over USB and launch the terminal (Termite or TeraTerm) and connect to the COM port corresponding to the USB interface.
When turning on, you should see a lot of "boot information" appearing on screen. Finally you can interact with the node (e.g. try typing "H" and Enter for help).



```
BT CmdResp '$$$'
BT CmdResp returned 5 bytes
BT CmdResp: CMD
BT CmdResp 'U,230K,N
'
BT CmdResp returned 5 bytes
BT CmdResp: AOK
bt init: returning
Initialise LCD... done
---------------------- welcome to sensor
write control: 00h
write status: 00h
DS3232 registers:
readregs return: 0
 00: 06
 01: 30
 02: 19
 03: 07
 04: 25
 05: 02
 06: 00
 07: 00
 08: 00
 09: 00
 0a: 00
 0b: 00
 0c: 00
 0d: 00
 0e: 18
 0f: 08
 10: 00
 11: 1C
 12: 40
 13: 00
Load configuration
Configuration:
   Sensors and sample rate: 27
   Streaming format: 1
   Stream battery: 1
   Stream temperature: 1
Initialise MPU subsystem
MPU-9250 registers:
Self test gyro x,y,z:  CB D4 DF
Self test accel x,y,z: 5D 5A 7F
Gyro offset x,y,z: 0 0 0
SMPLRT_DIV:    00    CONFIG:  00    GYRO_CONFIG: 01    ACCEL_CONFIG: 00 08
LP_ACCEL_ODR: 00    WOM_THR: 00    FIFO_EN:     00
I2C_MST_CTRL: 00
I2C_SLV0 ADDR:00    REG:     00    CTRL:        00  DO: 00
I2C_SLV1 ADDR:00    REG:     00    CTRL:        00  DO: 00
I2C_SLV2 ADDR:00    REG:     00    CTRL:        00  DO: 00
I2C_SLV3 ADDR:00    REG:     00    CTRL:        00  DO: 00
I2C_SLV4 ADDR:00    REG:     00    CTRL:        00  DO: 00 DI: 00
I2C_MST_STAT: 00
INT_PIN(55):  00    INT_EN:  00    INT_STATUS:  05
ACCEL: -0216 02400 16556
TEMP:  03440
GYRO:  -0060 00165 00041
EXT_SENS_DATA 00: 00   EXT_SENS_DATA 01: 00   EXT_SENS_DATA 02: 00   EXT_SENS_DATA 03: 00
EXT_SENS_DATA 04: 00   EXT_SENS_DATA 05: 00   EXT_SENS_DATA 06: 00   EXT_SENS_DATA 07: 00
EXT_SENS_DATA 08: 00   EXT_SENS_DATA 09: 00   EXT_SENS_DATA 0A: 00   EXT_SENS_DATA 0B: 00
EXT_SENS_DATA 0C: 00   EXT_SENS_DATA 0D: 00   EXT_SENS_DATA 0E: 00   EXT_SENS_DATA 0F: 00
EXT_SENS_DATA 10: 00   EXT_SENS_DATA 11: 00   EXT_SENS_DATA 12: 00   EXT_SENS_DATA 13: 00
EXT_SENS_DATA 14: 00   EXT_SENS_DATA 15: 00   EXT_SENS_DATA 16: 00   EXT_SENS_DATA 17: 00
I2C_MST_DELAY_CTRL: 00
SIGNAL_PATH_RESET:  00
MOT_DETECT_CTRL:    00  USER_CTRL:  00
PWR_MGMT_1:         48  PWR_MGMT_2: 3F
FIFO_COUNT:         0    FIFO_R_W:   00
WHO_AM_I:           71
ACC_OFFSET: 02958 03602 03966
Estimating baseline performance... 106493
before sc
after sc
ADCSRA 00
Current mode: 5
ADC mode start: period: 333 mask: 01 binary: 1 timestamp: 0
```

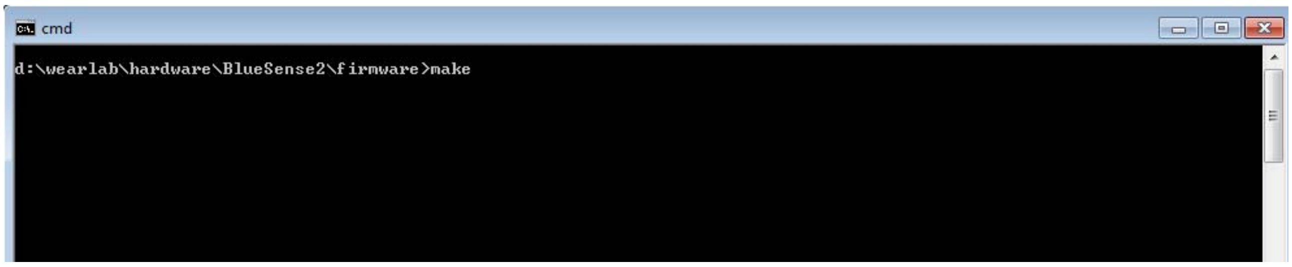
Note that the firmware is under development, and the message

7. Compiling the firmware

This assumes that avr-gcc-5 is placed in your path (typically C:\avr-gcc-5\bin must be in the path). Use the control panel to add these directories to the path if it isn't the case.

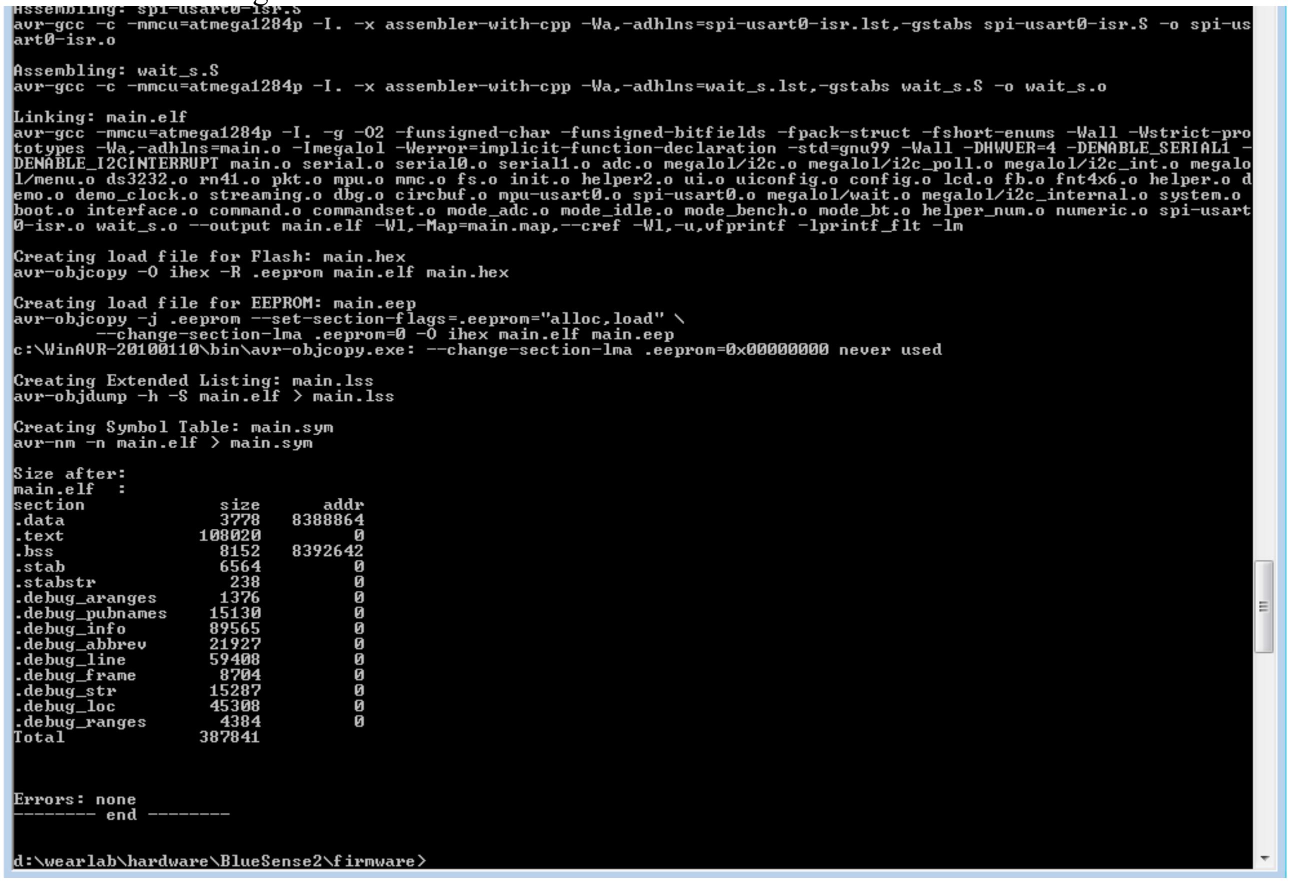
Compiling the firmware:

- Launch the command line (cmd.exe) and navigate to the folder where the firmware is located.



```
d:\wearlab\hardware\BlueSense2\firmware>make
```

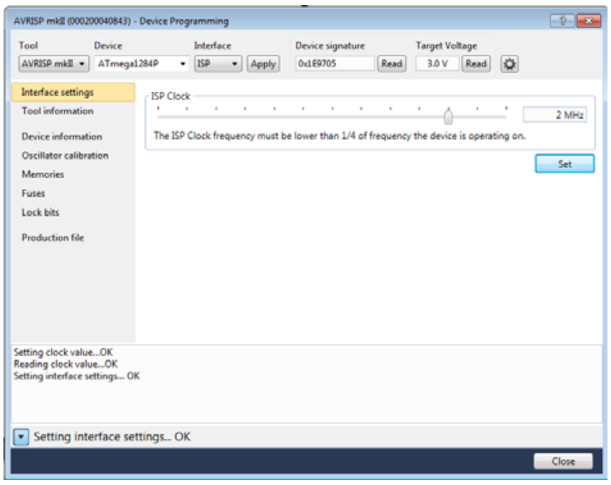
- type "make" (to ensure the project is entirely rebuilt you need to type "make clean" followed by "make"). Note that there may be a lot of warnings, as the code is under development.
- The resulting file is called main.hex



```
Assembling: spi-usart0-isr.S
avr-gcc -c -mmcu=atmega1284p -I. -x assembler-with-cpp -Wa,-adhlns=spi-usart0-isr.lst,-gstabs spi-usart0-isr.S -o spi-us
art0-isr.o

Assembling: wait_s.S
avr-gcc -c -mmcu=atmega1284p -I. -x assembler-with-cpp -Wa,-adhlns=wait_s.lst,-gstabs wait_s.S -o wait_s.o

Linking: main.elf
avr-gcc -mmcu=atmega1284p -I. -g -O2 -funsigned-char -funsigned-bitfields -fpack-struct -fshort-enums -Wall -Wstrict-pro
totypes -Wa,-adhlns=main.o -Imegalol -Werror=implicit-function-declaration -std=gnu99 -Wall -DHWVER=4 -DENABLE_SERIAL1 -
DENABLE_I2CINTERRUPT main.o serial.o serial0.o serial1.o adc.o megalol/i2c.o megalol/i2c_poll.o megalol/i2c_int.o megalo
l/menu.o ds3232.o rn41.o pkt.o mpu.o mmc.o fs.o init.o helper2.o ui.o uiconfig.o config.o lcd.o fb.o fnt4x6.o helper.o d
emo.o demo_clock.o streaming.o dbg.o circbuf.o mpu-usart0.o spi-usart0.o megalol/wait.o megalol/i2c_internal.o system.o
boot.o interface.o command.o commandset.o mode_adc.o mode_idle.o mode_bench.o mode_bt.o helper_num.o numeric.o spi-usart
0-isr.o wait_s.o --output main.elf -Wl,-Map=main.map,--cref -Wl,-u,vfprintf -lprintf_flt -lm

Creating load file for Flash: main.hex
avr-objcopy -O ihex -R .eeprom main.elf main.hex

Creating load file for EEPROM: main.eep
avr-objcopy -j .eeprom --set-section-flags=.eeprom="alloc,load" \
        --change-section-lma .eeprom=0 -O ihex main.elf main.eep
c:\WinAVR-20100110\bin\avr-objcopy.exe: --change-section-lma .eeprom=0x00000000 never used

Creating Extended Listing: main.lss
avr-objdump -h -S main.elf > main.lss

Creating Symbol Table: main.sym
avr-nm -n main.elf > main.sym

Size after:
main.elf  :
section                 size      addr
.data                   3778   8388864
.text                 108020         0
.bss                    8152   8392642
.stab                   6564         0
.stabstr                 238         0
.debug_aranges          1376         0
.debug_pubnames        15130         0
.debug_info            89565         0
.debug_abbrev          21927         0
.debug_line            59408         0
.debug_frame            8704         0
.debug_str             15287         0
.debug_loc             45308         0
.debug_ranges           4384         0
Total                 387841


Errors: none
-------- end --------


d:\wearlab\hardware\BlueSense2\firmware>
```

Programming the firmware:

- Launch Atmel studio and select Tools / Device Programming
- Select "AVRISP mkII and click Apply. You can see if the communication is successful by reading the device signature with the button "Read" under "Device Signature.
- To ensure that programming is fast, set the ISP clock to 2MHz. Above 2MHz will not work.


AVRISP mkII (000200040843) - Device Programming

Tool: AVRISP mkII
Device: ATmega1284P
Interface: ISP
Device signature: 0x1E9705
Target Voltage: 3.0 V

ISP Clock: 2 MHz

The ISP Clock frequency must be lower than 1/4 of frequency the device is operating on.

```
Setting clock value...OK
Reading clock value...OK
Setting interface settings... OK
```

Setting interface settings... OK

- Navigate to "Memories" and select the Flash file "main.hex" and click "Program"
- The program will automatically start once the device is programmed.

8. Firmware structure

As the firmware is under development there is no documentation at this stage.
The main file is called main.c. At the end of the firmware enters a continuous loop to enter different "modes". Some of the existing modes are "streaming sensor data" and "streaming ADC data". You will have to create your own additional modes for your project.

9. Communication interfaces

The communication between the node and computer can occur over USB or Bluetooth.

This is done using commands such as:

```
fprintf(file,"This is a string and a number: %d\n",123);
```

The "file" to use indicates whether the data is sent to USB or Bluetooth. If the data must be sent to a specific interface, use file_usb and file_bt to send to USB or Bluetooth.

However, in many cases only one "primary" interface is going to be used, and the other one is either disconnected or used for debugging information.
BlueSense node automatically detects if it is connected by USB, Bluetooth or both. It defines a "primary" interface and a "debug" interface. The "primary" interface is the interface which is first connected to. The debug interface is the one which is connected to afterwards (see interface.c). For instance, if the sensor is turned on when connected to USB but not Bluetooth, then the USB interface is primary, and Bluetooth is debug. If afterwards USB is disconnected (no connection active at this stage), and later a Bluetooth connection occurs, then the Bluetooth interface becomes primary, and USB becomes debug.
You can use file_pri and file_dbg to send data to the primary and to the debug interface.

10. Modes

The sensor firmware operates in "Modes".
By default it enters the idle mode. In most modes you can type "H" followed by enter to see the list of available functions. Some functions allow to configure the sensor, other functions allow to enter different modes.
In order to introduce or test new functionalities, the best approach consists in creating a new mode.

10. Data acquisition (ADC or motion data modes)

Two existing modes allow to stream raw data to a computer for logging. One mode allows to stream motion sensor data (accelerometer, gyroscope, or combination). Another mode allows to send the readings from the ADC.
The ADC streaming mode is entered with the command "A.....".
The motion streaming mode is entered with the command "M......".
Two streaming format exist:

- Text mode: the data is sent in space separated format, with one line per sample and one column per channel.
- Binary format: the data is sent in binary format, with a header, channel data, and a checksum for each sample.

The software DScope (http://gna.org/projects/dscope) can be used to acquire the data and visualise it or store it in a text file. The software SensHub (http://gna.org/projects/senshub) can be used to acquire the data and store it in a text tile. Refer to the documentation of these software on how to use them.

11. Documentation and API

BlueSense2 is under development. Therefore the firmware is frequently changing with new functions or change of behaviours. There is currently limited documentation available.

The best approach is to gain an understanding from looking at the code and if needed asking pertinent questions.

12. Creating a new mode

While the firmware can be modified freely (e.g. just putting stuff in the "main" function), it is better to create a new "mode" that can be entered and left to test a new functionality, while leaving the other modes as is.

As a way to distinguish the files you create from the original ones in the firmware it is recommended to put all files in the "project" subfolder.

The following point walk through briefly on the principles of creating a new mode.

12.1 Main loop

The main() in main.c has a forever loop that enters different modes based on system_mode:

```
while(1)
{
      switch(system_mode)
      {
            case 0:
                  mode_idle();
                  break;
            case 1:
                  //stream();
                  //mpu_testsleep();
                  system_mode=0;
                  break;
            case 2:
                  main_log();
```

In main.h different constants are defined for modes:

```
#define APP_MODE_CLOCK 3
#define APP_MODE_ADC 5
#define APP_MODE_BENCHIO 6
#define APP_MODE_BT 7
#define APP_MODE_MOTIONSTREAM 8
```

Define a constant for the new mode, which isn't used yet. Recommended: start at 100:

```
#define APP_MODE_MINE 100
```

Then modify the main loop to add a new mode entry:

```
                  case APP_MODE_MOTIONSTREAM:
                        mode_motionstream();
                        system_mode=0;
                        break;
                  case APP_MODE_MINE:
                        mode_mine();
                        system_mode=0;
                        break;
```

This calls mode_mine(), a function you must create, when this mode is selected.

12.2 Modifying the "idle" mode to allow entering the new mode

By default, the firmware goes in the idle mode, whose code is in mode_idle.c.

The idle mode must be modified to detect a command to enter in APP_MODE_MINE. There is a command processing engine that simplifies detecting such commands.

In mode_idle, the list of available commands is in an array CommandParsersIdle:

```
#define CommandParsersIdleNum 15
const COMMANDPARSER CommandParsersIdle[CommandParsersIdleNum] =
{
      {'R',CommandParserBT,help_r},
      {'L',CommandParserLCD,help_l},
      {'T', CommandParserTime,help_t},
      {'D', CommandParserDate,help_d},
      {'0', CommandParserZero,help_h},
      {'1', CommandParserOne,help_h},
      {'2', CommandParserTwo,help_h},
      {'H', CommandParserHelp,help_h},
      {'A', CommandParserADC,help_a},
      {'C', CommandParserClock,help_c},
      {'B', CommandParserBench,help_b},
      {'I', CommandParserIO,help_i},
      {'M', CommandParserMotion,help_m},
      {'W', CommandParserSwap,help_w},
      {'O', CommandParserOff,help_o}
};
```

A new command is created by adding one line, for example to react to the command 'C' for "custom", add:

```
      {'C', CommandParserCustom,help_c}
```

CommandParserCustom is used to parse the command (e.g. if it is of the form C,xxx,yyy) and help_c is a string providing help about the command.

Open commandset.c and commandset.h to define the help string. In commandset.c:

```
const char help_c[] PROGMEM ="My custom function";
```

In commandset.h:
```
extern const char help_c[];
```

12.3 New mode file

We need to create a new file which will have CommandParserCustom and mode_mine. We create this file in "project" and call it mode_mine.c (and mode_mine.h). The file mode_mine.h must be included in mode_idle.c as it will comprise the declaration of the parser. See other mode_xxx for examples.

In mode_mine.c we define CommandParserCustom:
```
unsigned char CommandParserCustom (unsigned char *buffer,unsigned char size)
{
      system_mode = APP_MODE_MINE;
      CommandQuit=1;

      return 0;
}
```

We also define the mode function. Usually the mode function will also use a command parser to detect commands, such as to leave the mode:
```
void mode_mine(void)
{
      while(1)
      {
            while(CommandProcess(CommandParsersMine,CommandParsersMineNum));
            if(CommandShouldQuit())
                  break;

            fprintf(file_pri,"This is a hello world on the primary
interface\n");
            fprintf(file_dbg,"This is a hello world on the debug interface\n");
            _delay_ms(1000);
      }
}
```
The commands available depend on what you want to offer. At minimum quit is needed:
```
#define CommandParsersMineNum 1
const COMMANDPARSER CommandParsersMine[CommandParsersMineNum] =
{
      {'!', CommandParserQuit,help_quit}
};
```
As CommandParserQuit and help_quit are already defined, nothing else is needed.

Note that the header file must be written to declare all the functions and constants that must be accessible from other files.

12.4 Makefile

As a new file was added to the project, it must be added to the Makefile. In Makefile there is a list of source files:
```
SRC += mode_idle.c
SRC += mode_bench.c
SRC += mode_bt.c
SRC += mode_motionstream.c
SRC += motionconfig.c
```

Add the new file mode_mine.c after the last SRC line:

```
SRC += mode_mine.c
```

13 Motion logging

The sensor can store motion data to the SD card. Only SDHC cards are supported (8GB has been tested, other sizes no).

In order to log, the SD must first be formatted using the firmware of the sensor node (formatting on a PC will not work properly). This is only required once.

13.1 SD card formatting

In the main (idle) mode, check the available options:

```
H
Got message of len 1: 'H'
Available commands:
        Z       Z[,<hh><mm><ss>]: Resets the absolute time counter to zero, and optionally sets the
RTC time
        Y       Test sync
        R       RN-41 terminal
        L       L,<en>: en=1 to enable LCD, 0 to disable
        T       T[,<hh><mm><ss>] Query or set time
        D       D[,<dd><mm><yy>] Query or set date
        H       Help
        A       A,<hex>,<us>: ADC mode. hex: ADC channel bitmask in hex; us: sample period in
microseconds
        C       Clock mode
        V       Demo mode
        I       I<period>,<txslots>: Sets USB IO parameters. IO routine is called at
(period+1)/1024Hz; txslots transmit slots are used before receiving data.
        M       M[,<mode>] stream motion data in specified mode; use M for available modes
        N       MPU test mode
        W       Swap primary and secondary interfaces
        O       O[,sec] Power off and no wakeup, or wakeup after sec seconds
        F       F,<bin>,<pktctr>,<ts>,<bat>,<label>: bin: 1 for binary, 0 for text; for others: 1 to
stream, 0 otherwise
        i       i,<ien> Enables additional info when ien=1
        Q       Coulomb counter test mode
        X       SD card test mode
        P       P,<low>: with low=1 the power supply enters low-power (max 10mA) mode; low=0 for
normal operation
        S       S,<us>: test streaming/logging mode; us: sample period in microseconds
        b       b[,run;a boot;script] Prints the current bootscript or sets a new one; multiple
commands are delimited by a ;
        ?       Identify device by blinking LEDs
CMDOK
```

Type X to enter in SD card test mode, then H to see the available options:

```
X
Got message of len 1: 'X'
CMDOK
Idle mode end
Current mode: 10
H
Got message of len 1: 'H'
Available commands:
        I       Low-level SD card initialisation
        W       W,<sector>,<value>: Fills a sector with the given value (all data in decimal)
        S       S,<sector>,<value>,<size>,<bsize>: Writes size bytes data in streaming mode with
caching; sends data by blocks of size bsize
        R       R,<sector>: Reads a sector (sector number in decimal)
        V       Initialise volume
        F       F,<numlogfiles>: Format the card for numlogfiles and initialise the volume
        L       L,<lognum>,<sizebytes>,<char>,<bsiz>: Writes to lognum sizebytes character char in
bsiz blocks
        B       B,<benchtype>
        H       Help
        !       Exit current mode
CMDOK
```

If the card has already been formatted you can use the command 'V' to check the volume, otherwise use command 'F' to format.

Format the card with: 'F,10'. The number (10) is the number of log files that are desired (only a fixed number of log files is possible, from 1 to 13 max):

```
F,10
Got message of len 4: 'F,10'
Formatting with 10 log files
uFAT: Init SD...
uFAT: Writing MBR...
uFAT: Writing bootsect...
uFAT: Writing bkp bootsect...
uFAT: Clearing ROOT and FAT
uFAT: Writing root...
uFAT: Writing FAT for root...
uFAT: Writing FAT for log 0
uFAT: Writing FAT for log 1
uFAT: Writing FAT for log 2
uFAT: Writing FAT for log 3
uFAT: Writing FAT for log 4
uFAT: Writing FAT for log 5
uFAT: Writing FAT for log 6
uFAT: Writing FAT for log 7
uFAT: Writing FAT for log 8
uFAT: Writing FAT for log 9
uFAT: Reading MBR...
uFAT: Reading bootsect...
uFAT: key info:
       Card capacity: 30228480 sectors
       Partition start sector: 8192
       Partition capacity: 30220288 sectors
       FAT1 start sector: 8224
       FAT2 start sector: 0
       Cluster start sector: 11913
       Sectors/clusters: 64
       Root cluster: 2
       Number of data clusters: 472133
uFAT: id: E5 checksum: 27 (obtained: 27)
uFAT: numlogs: 10 startcluster: 128 sizecluster: 47104 sizebytes: 1543503872
uFAT: Logs:
       LOG-0000.LOG start sector: 19977 start cluster: 128 size: 00000000h
       LOG-0001.LOG start sector: 3034633 start cluster: 47232 size: 00000000h
       LOG-0002.LOG start sector: 6049289 start cluster: 94336 size: 00000000h
       LOG-0003.LOG start sector: 9063945 start cluster: 141440 size: 00000000h
       LOG-0004.LOG start sector: 12078601 start cluster: 188544 size: 00000000h
       LOG-0005.LOG start sector: 15093257 start cluster: 235648 size: 00000000h
       LOG-0006.LOG start sector: 18107913 start cluster: 282752 size: 00000000h
       LOG-0007.LOG start sector: 21122569 start cluster: 329856 size: 00000000h
       LOG-0008.LOG start sector: 24137225 start cluster: 376960 size: 00000000h
       LOG-0009.LOG start sector: 27151881 start cluster: 424064 size: 00000000h
CMDOK
```

Now you can check the volume with command 'V':

```
V
Got message of len 1: 'V'
uFAT: Init SD...
uFAT: Reading MBR...
uFAT: Reading bootsect...
uFAT: key info:
       Card capacity: 30228480 sectors
       Partition start sector: 8192
       Partition capacity: 30220288 sectors
       FAT1 start sector: 8224
       FAT2 start sector: 0
       Cluster start sector: 11913
       Sectors/clusters: 64
       Root cluster: 2
       Number of data clusters: 472133
uFAT: id: E5 checksum: 27 (obtained: 27)
uFAT: numlogs: 10 startcluster: 128 sizecluster: 47104 sizebytes: 1543503872
uFAT: Logs:
       LOG-0000.LOG start sector: 19977 start cluster: 128 size: 00000000h
       LOG-0001.LOG start sector: 3034633 start cluster: 47232 size: 00000000h
       LOG-0002.LOG start sector: 6049289 start cluster: 94336 size: 00000000h
       LOG-0003.LOG start sector: 9063945 start cluster: 141440 size: 00000000h
       LOG-0004.LOG start sector: 12078601 start cluster: 188544 size: 00000000h
       LOG-0005.LOG start sector: 15093257 start cluster: 235648 size: 00000000h
```

```
        LOG-0006.LOG start sector: 18107913 start cluster: 282752 size: 00000000h
        LOG-0007.LOG start sector: 21122569 start cluster: 329856 size: 00000000h
        LOG-0008.LOG start sector: 24137225 start cluster: 376960 size: 00000000h
        LOG-0009.LOG start sector: 27151881 start cluster: 424064 size: 00000000h
uFAT: initialised
CMDOK
```

You should see 10 log files.

At this stage, you can take the SD card out and put it in a PC and you will see the 10 log files. Make sure never to delete, create or modify files on the SD card directly from the PC. This modifies the FAT structure and the node will not be able to write log files anymore and the card must be formatted again.

After formatting, when you put the card back in the node, there is no need to format it again.

Leave the SD mode with '!'.

13.2 Motion logging

Assuming the card is formatted, enter the motion streaming mode:

```
M
Got message of len 1: 'M'
Motion modes:
[0]        Motion off
[1]          500Hz Gyro (BW=250Hz)
[2]          500Hz Gyro (BW=184Hz)
[3]          200Hz Gyro (BW= 92Hz)
[4]          100Hz Gyro (BW= 41Hz)
[5]           50Hz Gyro (BW= 20Hz)
[6]           10Hz Gyro (BW=  5Hz)
[7]            1Hz Gyro (BW=  5Hz)
[8]         1000Hz Acc  (BW=460Hz)
[9]          500Hz Acc  (BW=184Hz)
[10]        200Hz Acc  (BW= 92Hz)
[11]        100Hz Acc  (BW= 41Hz)
[12]         50Hz Acc  (BW= 20Hz)
[13]         10Hz Acc  (BW=  5Hz)
[14]          1Hz Acc  (BW=  5Hz)
[15]        1000Hz Acc  (BW=460Hz) Gyro (BW=250Hz)
[16]         500Hz Acc  (BW=184Hz) Gyro (BW=250Hz)
[17]         500Hz Acc  (BW=184Hz) Gyro (BW=184Hz)
[18]         200Hz Acc  (BW= 92Hz) Gyro (BW= 92Hz)
[19]         100Hz Acc  (BW= 41Hz) Gyro (BW= 41Hz)
[20]          50Hz Acc  (BW= 20Hz) Gyro (BW= 20Hz)
[21]          10Hz Acc  (BW=  5Hz) Gyro (BW=  5Hz)
[22]           1Hz Acc  (BW=  5Hz) Gyro (BW=  5Hz)
[23]         500Hz Acc low power
[24]         250Hz Acc low power
[25]         125Hz Acc low power
[26]        62.5Hz Acc low power
[27]       31.25Hz Acc low power
[28]           1Hz Acc low power
[29]         100Hz Acc  (BW= 41Hz) Gyro (BW= 41Hz) Mag 8Hz
[30]         100Hz Acc  (BW= 41Hz) Gyro (BW= 41Hz) Mag 100Hz
[31]         200Hz Acc  (BW= 92Hz) Gyro (BW= 92Hz) Mag 100Hz
[32]         100Hz Acc  (BW= 41Hz) Gyro (BW= 41Hz) Mag 100Hz Quaternions
[33]         200Hz Acc  (BW= 92Hz) Gyro (BW= 92Hz) Mag 100Hz Quaternions
[34]         100Hz Quaternions
[35]         200Hz Quaternions
CMDINV
M,13
Got message of len 4: 'M,13'
CMDOK
Idle mode end
Current mode: 8
mpu_mode_accgyro: 1 0 1 1 1 1
mpu_mode_acc: 1 6 99
00001 0002465167 00000 00000  00407  00378  00718
00002 0002465266 00000 00000  08974  00578  13622
00003 0002465366 00000 00000  09310  00583  14152
00004 0002465465 00000 00000  09326  00598  14190
00005 0002465564 00000 00000  09330  00583  14183
00006 0002465664 00000 00000  09329  00593  14163
00007 0002465762 00000 00000  09317  00568  14158
00008 0002465862 00000 00000  09328  00589  14184
00009 0002465960 00000 00000  09335  00582  14192
00010 0002466159 00000 00000  09321  00595  14183
00011 0002466258 00000 00000  09324  00582  14186
00012 0002466357 00000 00000  09323  00576  14182
00013 0002466457 00000 00000  09317  00591  14187
00014 0002466556 00000 00000  09323  00582  14159
00015 0002466655 00000 00000  09317  00576  14166
00016 0002466754 00000 00000  09314  00573  14178
00017 0002466853 00000 00000  09323  00581  14164
00018 0002466953 00000 00000  09316  00579  14185
00019 0002467051 00000 00000  09324  00591  14184
00020 0002467151 00000 00000  09325  00594  14189
00021 0002467250 00000 00000  09331  00574  14173
00022 0002467349 00000 00000  09321  00566  14185
```

In this mode the sensor streams the data (if the data is binary, use the F command to change the format).

The following commands are available:
```
03581 0002844429 00000 00000  09336  00607  14215
03582 0002844529 00000 00000  09329  00592  14189
03583 0002844627 00000 00000  09327  00582  14189
H
Got message of len 1: 'H'
Available commands:
       H      Help
       F      F,<bin>,<pktctr>,<ts>,<bat>,<label>: bin: 1 for binary, 0 for text; for others: 1 to
stream, 0 otherwise
       L      L[,<lognum>]: Stops streaming and start logging to 'lognum' (0 by default); if
logging is already in progress stops logging and resumes streaming
       Z      Z[,<hh><mm><ss>]: Resets the absolute time counter to zero, and optionally sets the
RTC time
       i      i,<ien> Enables additional info when ien=1
       N      N,<number>: sets the current annotation
       !      Exit current mode
CMDOK
03585 0002844826 00000 00000  09317  00590  14215
03586 0002845025 00000 00000  09321  00576  14156
```

The command F is used to select which information to stream and whether the format is text or binary. E.g. use F,0,0,1,0,1 to enable text mode and stream the timestamp and label.

To enable logging, use the 'L' command with the log file number (e.g. 'L,0' to store in log 0). Information related to the card initialisation is displayed, then, if the i command enabled this, every 10 seconds a message indicates that logging is ongoing. At this stage, the data is stored to the SD card:

```
04863 0002981419 00000 00000  09316  00590  14209
04864 0002981519 00000 00000  09302  00601  14204
L,0
04865 0002981618 00000 00000  09311  00593  14180
Got message of len 3: 'L,0'
Starting log on 0
uFAT: : streaming write at sector 19977
Current log: 0
       size: 0
       sector: 19977
CMDOK
```

In order to stop logging, type 'L' again:

```
L
Got message of len 1: 'L'
Terminating logging
ufat_log_close
Current log: 0
       size: 10710
       sector: 19977
uFAT: Writing root...
CMDOK
05076 0003003735 00000 00000  09289  00584  14209
05077 0003003834 00000 00000  09331  00593  14162
05078 0003003933 00000 00000  09320  00577  14181
```

At this stage, you can exit the streaming mode (command '!') or simply take the SD card out and put it in the PC.

13.3 Reading logging data with Matlab

Copy the log file to your hard drive.

In Matlab you can use the following commands (here loading log 2) to display the sensor signals; this script assumes the timestamps are in the first column:

```
a=load('LOG-0002.LOG');
dt = a(2:end,1)-a(1:end-1,1);
figure(1);
clf;
plot(dt);
figure(2);
clf;
plot(a(:,1));
figure(3);
clf;
plot(a(:,1),a(:,2),'r-');
hold on;
plot(a(:,1),a(:,3),'g-');
plot(a(:,1),a(:,4),'b-');
```

13.4 Note on programming with SD card in the node

When programming the sensor node the same pins are used for the processor programming as for the communication with the SD card. If the programming fails, remove the SD card.

Alternatively, make sure to initialise the SD card (command 'I'), as this will make the SD card ignore the programming signals. Every time the SD card is plugged back into the node it must be initialised, for programming to work.

As a safe default, remove the SD card when programming the node.

14 Using the LCD module

Keep in mind that LCD display is very slow; therefore avoid continuously writing to the display, especially if this is in a loop with tight real-time constraints.

The following functions are basic LCD display functions:

void lcd_writestring(char *str,unsigned char x,unsigned char y,unsigned s,unsigned short cbg,unsigned short cfg)

str: string to display
x,y: coordinate of bottom-left pixel of string
s: scale factor, 1 is normal size, 2 is double size
cbg: background color, e.g. 0x0000 for black
cfg: foreground color, e.g. 0xffff for white

void lcd_pixel(unsigned short x, unsigned short y, unsigned short color)

x,y: pixel coordinate
color: color in RGB

fprintf(file_fb,char *format string, ...)

The LCD can also be used as a frame buffer.

fprintf(file_fb,"Hello world\n");

15 Adding activity recognition function

All activity recognition can be prototyped in the module "mode_motionrec.c" and "mode_motionrec.h".

This mode can be entered from the idle mode with the command G. Use G? to see the list of commands. For instance, G,13 enters sampling of acceleration at 10Hz.

Thus you must enter the sample mode that is suited to the recognition task (selecting the adequate channels and sample rate).

The main recognition loop has to be implemented in the function *void mode_motionrecog(void).* It is as follows:

```
        while(1)
        {
                // Check if the user enters something on keyboard to quit (e.g. !)
                while(CommandProcess(CommandParsersMotionRecog,CommandParsersMotionRecogNum));

                if(CommandShouldQuit())
                        break;
                //_delay_ms(1);

                /*if(system_motion_newsample)
                {
                        printf("ns wk %ld. lvl: %d
wrptr %d\n",wakeup,stream_buffer_level(),system_motion_wrptr);
                        system_motion_newsample=0;
                }*/

                // Get how many samples are in the buffer
                unsigned char l = stream_buffer_level();
```

```
		totsamples+=l;

		// Process each sample; remember to call stream_buffer_rdptrnext to get the next 
sample.
		for(unsigned char i=0;i<l;i++)
		{
			print_sample(file_stream);
			stream_buffer_rdptrnext();

		}

		// Print some status information at regular intervals.
		// Remove this if it takes too much CPU time
		if((t_cur=timer_ms_get())-t_status>10000)
		{
			fprintf_P(file_pri,PSTR("Streaming since %lums\n"),timer_ms_get()-t_start);
			fprintf_P(file_pri,PSTR(" Tot samples %lu.\n"),totsamples);
			fprintf_P(file_fb,PSTR("Streaming since %lums.\n"),timer_ms_get()-t_start);
			fprintf_P(file_fb,PSTR(" Tot samples %lu.\n"),totsamples);

			t_status = timer_ms_get();
		}
	}
```

The key elements are:

- The motion sensor data is stored in a small size buffer (about 16 samples). It is the responsability of this loop to read the data as quickly as possible, otherwise data will be lost. If the main loop does not read fast enough, the oldest data is erased (circular buffer)
- stream_buffer_level indicates how many samples are available.
- The current sample is stored in the circular buffer system_motion_ax[system_motion_rdptr], system_motion_ay[system_motion_rdptr], system_motion_az[system_motion_rdptr] and similarly for the gyro system_motion_gx[system_motion_rdptr], etc.
- The function print_sample shows how to read the samples from the motion buffer.
- After accessing a system_motion_?? buffer, call stream_buffer_rdptrnext which updates system_motion_rdptr to point to the next sample.
- If system_motion_rdptr is called more often than there are samples in the buffer (as indicated by stream_buffer_level) then old (already used) data will be returned.