BlueSense: Extensible Wearable Motion Sensing Platform and IoT platform

1. Project and product description

BlueSense (figure 1) is a 30x30mm extensible wearable/IoT platform designed to be functional out-of-the-box yet extensible. It's primary purpose is to be an inertial measurement unit for wearable applications based on the Invensense MPU9250 and an AT1284p microcontroller. Yet it is extensible for IoT applications with a number of features making it appealing for this purpose.

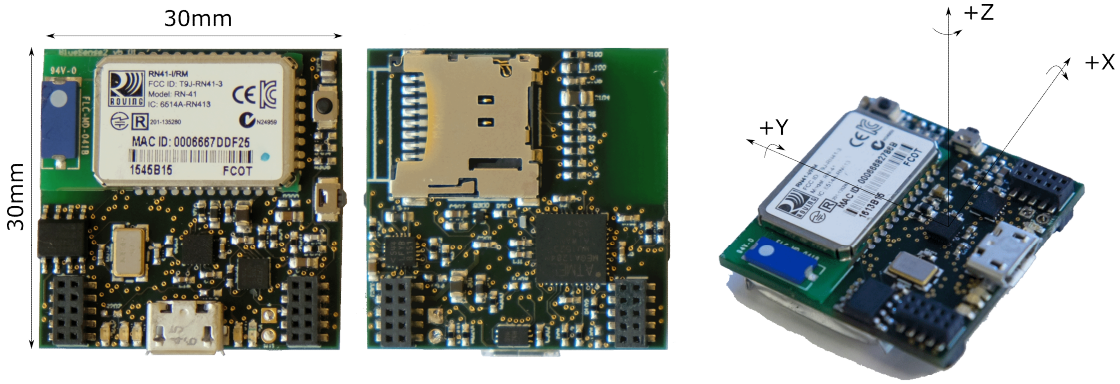

Figure 1. BlueSense PCBs top (left) and bottom (middle), and BlueSense fitted with a 160mAh battery (right).

As a wearable platform, BlueSense is a tiny device which can be used to capture body movement and orientation with a 9DoF motion sensor. The data can either be logged on an SD card or streamed over USB or Bluetooth 2. BlueSense could be used to prototype a custom fitness tracker. Its extension ports allow to plug-in additional sensor modalities for research purposes. A display is being developed converting it into a smartwatch.

As an IoT device it's main appeal is a true *hardware off* which allows to put the device entirely to sleep (everything is powered down, including voltage regulators), yet wake up at programmed intervals thanks to a real-time clock wakeup. This allows to achieve very long battery life, as the device can wake up at desired times (once an hour, once a day, ...) to acquire and send sensor data. The extension ports allow to plug in custom sensor modalities which make this device highly versatile.

The application firmware has been designed to be useful for a wide range of applications out of the box. It has been optimised for high-speed motion data logging and streaming (500Hz), and high-speed external ADC acquisition (1+KHz), informed by the need of wearable motion tracking and activity recognition applications.

Besides USB, Bluetooth 2 and SD card interface with a FAT32 compatible filesystem, BlueSense has a real-time clock, a coulomb counter to measure it's own battery level and power consumption, true hardware off, and connectors for extensions. This makes

the platform also suitable for sensor research, IoT applications, or as an compact alternative to other microcontroller boards.

2.1 Functionality

2.1.1 Motion sensing

Out-of-the-box, BlueSense is a highly versatile inertial measurement unit (IMU) and attitude heading reference system (AHRS), primarily designed for wearable motion sensing and motion tracking applications.

BlueSense is built around a 9DoF motion sensor (Invensense MPU9250). The application firmware allows to sample the motion data in a highly configurable manner, including:

- which combination of channels to acquire among acceleration, rate of turn, magnetic field and device orientation (quaternions or Euler angles)
- Device orientation is computed using a modified Madgwick algorithm [Madgwick]. This algorithm is open-source which makes it suitable for experimentation and research, unlike the binary blob algorithm provided by Invensense for the MPU9250 which is proprietary.
- which sampling rate, from 1Hz to 500Hz (guaranteed, including in AHRS mode) and 1KHz (best effort)
- which metadata to assign to samples, among timestamps, sample counter, label, battery capacity, etc.
- whether to stream the data over Bluetooth, over USB or whether to store the data on the SD card.

Among others, BlueSense has been used in sports tracking applications where high sample rate is desired [Cuspinera16].

2.1.2. Extensibility

BlueSense is extensible through two mirrored Samtec connectors on the top and bottom side of the PCB (Figure 2). These connectors offer power, analog inputs (ADC, analog comparator), digital I/O, SPI and I2C interfaces. In addition, one of the connector is also used for programming.

This makes the platform versatile for other applications, such as sensor research or IoT applications, or more generally as a compact alternative to other microcontroller platforms.

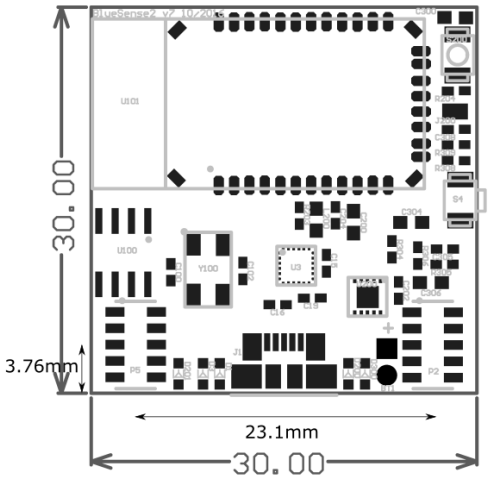

Figure 2. Top view showing the placement of the Samtec extension connectors on either edges of the PCB.

The pinout of the extension connectors is indicated in figure 3. Figure 4 shows BlueSense with a sensor extension module plugged on top to measure ambient electric fields.

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

Figure 3. Pinout of the top-side connectors. The bottom side is mirrorred.

Figure 4. BlueSense with a sensor extension module [Roggen16]. The height is indicated without the battery.

2.1.3. Additional features

True hardware off and alarm power-on

BlueSense has a true hardware off which allows to power-off the device entirely: everything is powered down, including the voltage regulator. In true off, only the real-time clock is powered and some discrete logic to turn on the voltage regulator. Power consumption is <70µw in true off mode.

The device can be programmed to wake up at programmed time/date thanks to a real-time clock wakeup. This allows to achieve very long battery life, as the device can wake up at desired times (e.g. once an hour, once a day, once a week) to do some computation, acquire and send sensor data, etc. This makes the platform interesting for Internet of Things applications which need long battery life.

Power-up can also be triggered from the extension connector (RTC_INT pin) which is an open-drain input.

Coulomb counter

BlueSense has a coulomb counter measuring the charge/discharge of the battery. This allows to measure battery voltage, battery capacity and instantaneous and average power use from within the platform itself.

The coulomb counter allows to easily measure power use of extensions, or to compare different implementations of algorithms to see how much power they use.

2.1.4. Application firmware

The application firmware has been designed to be functional out of the box for a wide range of applications, primarily focusing on motion sensing and external ADC data acquisition.

In particular:

- The sensor node can be interfaced easily by a human with a command line interface, and that interface is also easily suited for programmatic control.
- Multiple sensors can be easily synchronised thanks to the RTC (open-loop)
- Data can be timestamped
- Data can logged to the SD card out-of-the-box
- An "annotation"[1] value can be stored alongside timestamp and sensor data. This allows to have data and annotations stored in the same file and simplifies recordings. In this case, an external device (e.g. a mobile phone) would connect to BlueSense and send a command indicating which annotation value to set. Obviously, recordings without annotation are also possible.
- Care has been taken to ensure low jitter. ADC can be sampled at 1KHz with less than 50µs jitter and higher sample rate is possible. Similarly, all motion sensor data including orientation and metadata can be sampled and stored at 500Hz without data loss

BlueSense user interface

The user can control BlueSense over Bluetooth or USB using a "command line" mode[2]. Figure 5 illustrate the options available when the user issues the *help*

---

[1] Annotation refers to the process of marking the start and end time of "interesting" events. In applications such as activity recognition, annotations would indicate the start and end time of activities of interest (e.g. "walk") so that machine learning algorithms can be trained on this data to recognise these activities.

[2] This command line mode is a text-based command/response protocol also suitable for programmatic control.

command (H). Figure 6 illustrates how to query and set date and time from the RTC and battery status.

```
H
Available commands:
        H       Help
        T       T[,<hh><mm><ss>] Query or set time
        D       D[,<dd><mm><yy>] Query or set date
        Z       Z[,<hh><mm><ss><dd><mm><yy>]: synchronise time.
                No parameters: reset local time/date; otherwise sets the RTC and local
time/date to hhmmss ddmmyy
        Y       Test sync
        R       RN-41 terminal
        t       Time-related tests
        A       A,<hex>,<us>: ADC mode. hex: ADC channel bitmask in hex; us: sample
period in microseconds
        I       I<period>,<#tx>: Sets USB IO parameters. IO at (period+1)/1024Hz.
                Use #tx slots for transmission before reception.
        M       M[,<mode>[,<logfile>[,<duration>]]: without parameters lists available
modes, otherwise enters the specified mode.
                Optionally logs to logfile (use -1 not to log) and runs for the
specified duration in seconds.
        m       MPU test mode
        W       Swap primary and secondary interfaces
        O       O[,sec] Power off and no wakeup, or wakeup after sec seconds
        o       Display power used in off mode; if the node was turned off with O
        F       F,<bin>,<pktctr>,<ts>,<bat>,<label>: bin: 1 for binary, 0 for text; for
others: 1 to stream, 0 otherwise
        i       i,<ien> Prints battery and logging information when streaming/logging
when ien=1
        Q       Long-term battery info
        q       Short-term battery info
        c       Lists timer callbacks
        X       SD card test mode
        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
        ~       Clear boot counter
CMDOK
```

Figure 5. Output of the *help* command in the main interface.

```
?
My name is 835E
CMDOK
D
12.05.18
CMDOK
T
13:45:20
CMDOK
T,135000
Time: 13:50:00
CMDOK
T
13:50:02
CMDOK
q
#V=4142 mV; I=-23 mA; P=-95 mW
-95 -94 -95 -95 -97 -95 -93 -93 -98 -95
CMDOK
```

Figure 6. Output of the some of the other commands in the main interface.

Motion data acquisition

Multiple motion sensing modes are available (figure 7). Figure 8 shows streaming of acceleration, gyro and magnetic field in text mode.

```
M
Motion modes (x indicates experimental modes; sample rate not respected):
[0]     Motion off
```

```
[1]       500Hz Gyro (BW=250Hz) x
[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) x
[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) x
[16]      500Hz Acc  (BW=184Hz) Gyro (BW=250Hz) x
[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]     1000Hz Acc  (BW=460Hz) Gyro (BW=250Hz) Mag 8Hz x
[30]      500Hz Acc  (BW=184Hz) Gyro (BW=250Hz) Mag 8Hz x
[31]      500Hz Acc  (BW=184Hz) Gyro (BW=184Hz) Mag 8Hz
[32]      100Hz Acc  (BW= 41Hz) Gyro (BW= 41Hz) Mag 8Hz
[33]       50Hz Acc  (BW= 20Hz) Gyro (BW= 20Hz) Mag 8Hz
[34]     1000Hz Acc  (BW=460Hz) Gyro (BW=250Hz) Mag 100Hz x
[35]      500Hz Acc  (BW=184Hz) Gyro (BW=250Hz) Mag 100Hz x
[36]      500Hz Acc  (BW=184Hz) Gyro (BW=184Hz) Mag 100Hz
[37]      200Hz Acc  (BW= 92Hz) Gyro (BW= 92Hz) Mag 100Hz
[38]      100Hz Acc  (BW= 41Hz) Gyro (BW= 41Hz) Mag 100Hz
[39]      500Hz Acc  (BW=184Hz) Gyro (BW=184Hz) Mag 100Hz Quaternions
[40]      200Hz Acc  (BW= 92Hz) Gyro (BW= 92Hz) Mag 100Hz Quaternions
[41]      100Hz Acc  (BW= 41Hz) Gyro (BW= 41Hz) Mag 100Hz Quaternions
[42]      500Hz Quaternions Qsg=(qw,qx,qy,qz); rotates vector in earth coords G into
sensor coords S
[43]      200Hz Quaternions
[44]      100Hz Quaternions
[45]      100Hz Tait-Bryan/aerospace/zx'y", intrinsic (yaw, pitch, roll)
[46]      100Hz Quaternions debug (angle, x,y,z)
CMDOK
```

Figure 7. Available motion sensing modes

```
M,33
CMDOK
<IDLE
SMPLMOTION>
Acc scale: 3
Gyro scale: 3
Sample rate: 50
-00061 -01870  00787 -00015  00028 -00001  00073  00106 -00030
-00071 -01870  00789 -00009 -00017 -00013  00073  00106 -00030
-00082 -01879  00767  00011 -00010 -00001  00073  00106 -00030
-00064 -01897  00755  00013  00006  00020  00073  00106 -00030
-00040 -01887  00766  00001  00010  00019  00074  00111 -00030
-00054 -01883  00763  00000 -00006 -00006  00074  00111 -00030
-00078 -01886  00772  00002  00003 -00013  00074  00111 -00030
-00073 -01878  00776 -00006  00003  00000  00074  00111 -00030
-00062 -01883  00776  00004  00004  00010  00074  00111 -00030
-00056 -01880  00764  00009  00004  00012  00074  00111 -00030
-00061 -01888  00763  00006 -00009  00001  00071  00106 -00035
```

Figure 8. Streaming of acceleration, gyroscope and magnetometer data in text mode.

ADC data acquisition

BlueSense allows to acquire the data of the ADC inputs exposed on the expansion connector, and either stream over USB/Bluetooth or store the data to an SD-card.

The firmware has been optimised to allow high ADC sample rate acquisition and streaming/storage. Up to 1 KHz sampling can be achieved with low jitter (<50µs).

Figure 9 illustrates how data acquisition is started from the user interface. The channels to acquire are specified with a bitmask.

```
A,1,1000
mask: 01. period: 1000
CMDOK
<IDLE
ADC mode: period: 1000 mask: 01 binary: 0 timestamp: 0 battery: 0 label: 0
00531
00586
00669
00686
00604
00520
00412
00403
00408
00531
00616
00646
00629
00580
```

Figure 9. Streaming of ADC channel 0 (specified by the bitmask 1) acquired every 1000 µs.

Common data acquisition parameters

Motion and ADC data can be acquired and streamed as-is, or with additional metadata, such as sample number, timestamp, or 'label'. A label is numerical identifier stored along samples which can be used to identify different sections of a recording. The label can be set by sending a command prior or during data acquisition.

In figure 10, the F command (F,0,1,1,0,1) is used to indicate a that samples should be streamed/logged in text mode and should include a sample counter, timestamp, and label. ADC acquisition is then started with 1Hz sample rate. During the recording the user sets the label to 10 with the N command, which is reflected in the 3rd column.

```
F,0,1,1,0,1
bin: 0. pktctr: 1 ts: 1 bat: 0 label: 1
CMDOK
A,1,1000000
mask: 01. period: 1000000
CMDOK
<IDLE
ADC mode: period: 1000000 mask: 01 binary: 0 timestamp: 1 battery: 0 label: 1
00000 3432604465 00000 00569
00001 3433604484 00000 00563
00002 3434604471 00000 00520
00003 3435604470 00000 00501
00004 3436604469 00000 00478
00005 3437604472 00000 00458
00006 3438604470 00000 00426
00007 3439604465 00000 00393
N,10
00008 3440604482 00000 00394
CMDOK
00009 3441605193 00010 00377
00010 3442605211 00010 00377
00011 3443605205 00010 00369
```

```
00012 3444605197 00010 00359
00013 3445605195 00010 00374
00014 3446605202 00010 00392
!
00015 3447605197 00010 00419
CMDOK
ADC mode end. Samples: 16 in 16002 ms (0 samples/sec). Samples not transmitted: 0
(0 %)
IDLE>
```

Figure 10. The format in which the data is streamed or logged can be modified with the F command. Here, a sample number, timestamp and label is sent alongside the sample value.

A binary mode is also available for data streaming/logging. The binary mode reduces the amount of data transferred/stored compared to the text mode, which can be useful to increase sample rate, for instance, or decrease power usage.

SD card storage

BlueSense allows to log data locally to the SD card in log files. The SD card can be formatted in a FAT32 compatible filesystem from within BlueSense. The data can then be copied to a computer after logging is completed.

The application firmware has been optimised to achieve high sample rate (e.g. guaranteed 1KHz ADC, or 500Hz motion sensing). This requires to use a special layout of the files in the filesystem. In particular, log files are pre-layed out during formatting [Roggen18].

Figure 11 shows how the SD card is formatted (command X followed by F) and data logged to the SD card log file 5 (command M,21,5), until interrupted with the ! command.

```
X
CMDOK
<IDLE
SD>
F,14
Formatting with 14 log files
uFAT: Init SD...
CSD:
          CSD: 1
          TAAC: E
          NSAC: 0
          TRAN_SPEED: 32
          CCC: 5B5
          READ_BL_LEN: 9
          READ_BL_PARTIAL: 0
          WRITE_BLK_MISALIGN: 0
          READ_BLK_MISALIGN: 0
          DSR_IMP: 0
          C_SIZE: 0000EE9D (31276032 KB)
          ERASE_BLK_EN: 1
          SECTOR_SIZE: 7F
          WP_GRP_SIZE         : 0
          WP_GRP_ENABLE: 0
          R2W_FACTOR: 2
          WRITE_BL_LEN: 9
          WRITE_BL_PARTIAL: 0
          FILE_FORMAT_GRP: 0
          COPY: 0
          PERM_WRITE_PROTECT: 0
          TMP_WRITE_PROTECT: 0
          FILE_FORMAT: 0
          CRC: 9
SDSTAT:
          DAT_BUS_WIDTH: 00
          SECURED_MODE: 00
          SD_CARD_TYPE: 0000
          SIZE_OF_PROTECTED_AREA: 05000000
          SPEED_CLASS: 04 (class 10)
          PERFORMANCE_MOVE: 00
          AU_SIZE: 09 (4096KB)
          ERASE_SIZE: 0008
          ERASE_TIMEOUT: 04
          ERASE_OFFSET: 01
          UHS_SPEED_GRADE: 01
          VIDEO_SPEED_CLASS: 00
```

```
          VSC_AU_SIZE: 00
          SUS_ADDR: 00
          APP_PERF_CLASS: 00
          PERFORMANCE_ENHANCE: 00
          DISCARD_SUPPORT: 00
          FULE_SUPPORT: 00
uFAT: Erasing card
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: Writing FAT for log 10
uFAT: Writing FAT for log 11
uFAT: Writing FAT for log 12
uFAT: Writing FAT for log 13
uFAT: Reading MBR... uFAT: Reading bootsect...
uFAT: key info:
          Card capacity: 62552064 sectors
          Partition start sector: 8192
          Partition capacity: 62543872 sectors
          FAT1 start sector: 8224
          FAT2 start sector: 0
          Cluster start sector: 15859
          Sectors/clusters: 64
          Root cluster: 2
          Number of data clusters: 977128
uFAT: id: E5 checksum: 04 (obtained: 04)
uFAT: numlogs: 14 startcluster: 128 sizecluster: 69760 sizebytes: 2285895680
uFAT: Logs:
          LOG-0000.LOG start sector: 23923 start cluster: 128 size: 00000000h
          LOG-0001.LOG start sector: 4488563 start cluster: 69888 size: 00000000h
          LOG-0002.LOG start sector: 8953203 start cluster: 139648 size: 00000000h
          LOG-0003.LOG start sector: 13417843 start cluster: 209408 size: 00000000h
          LOG-0004.LOG start sector: 17882483 start cluster: 279168 size: 00000000h
          LOG-0005.LOG start sector: 22347123 start cluster: 348928 size: 00000000h
          LOG-0006.LOG start sector: 26811763 start cluster: 418688 size: 00000000h
          LOG-0007.LOG start sector: 31276403 start cluster: 488448 size: 00000000h
          LOG-0008.LOG start sector: 35741043 start cluster: 558208 size: 00000000h
          LOG-0009.LOG start sector: 40205683 start cluster: 627968 size: 00000000h
          LOG-0010.LOG start sector: 44670323 start cluster: 697728 size: 00000000h
          LOG-0011.LOG start sector: 49134963 start cluster: 767488 size: 00000000h
          LOG-0012.LOG start sector: 53599603 start cluster: 837248 size: 00000000h
          LOG-0013.LOG start sector: 58064243 start cluster: 907008 size: 00000000h
CMDOK
!
CMDOK
<SD
IDLE>
M,21,5
CMDOK
<IDLE
SMPLMOTION>
Logging on 5
uFAT: Erase sectors 22347123-26811762
uFAT: Streaming write at sector 22347123
Acc scale: 3
Gyro scale: 3
Sample rate: 10
!
CMDOK
Terminating logging
uFAT: Writing root...
uFAT: Erase sectors 58064243-62528882
uFAT: Streaming write at sector 58064243
Total errors: 0/94 samples (0 ppm). Err stream/log: 0, err MPU busy: 0, err buffer full: 0
uFAT: Writing root...
#t=9452 ms; V=4201 mV; I=2 mA; P=9 mW; wps=1160; errbsy=0; errfull=0; errsend=0; spl=94; log=0 KB; logmax=2232320
KB; logfull=0 %
MPU acquisition statistics:
 MPU interrupts: 94
 Samples: 94
 Samples success: 94
 Errors: MPU I/O busy=0 buffer=0
 Buffer level: 0/64
 Spurious ISR: 0
MPU Geometry time: 0 us
Total errors: 0/94 samples (0 ppm). Err stream/log: 0, err MPU busy: 0, err buffer full: 0
<SMPLMOTION
IDLE>
```

Figure 11. Example of SD card formatting followed by motion logging to log file 5, until interrupted with the command !.

2.1.5. Support tools

Several support tools exist which are either compatible with BlueSense or designed specifically for BlueSense.

Bs2Mgr

Bs2Mgr (figure 12) is a dedicated management software for BlueSense developed in Qt which runs under Linux, Windows and Android (Mac not tested). It is available under GPLv2 on GitHub [bs2mgr].

Bs2Mgr allows to synchronise the real-time clock of multiple BlueSense nodes, identify a node by blinking its LEDs, setup logging/streaming format, format SD card, and start or stop logging/streaming.

Bs2Mgr allows to easily start logging data on multiple nodes which are time-synchronised. The Android applications makes it particularly suitable for field recordings.

Figure 12. Bs2Mgr is set-up to control two Bluetooth-connected nodes (left). Among the available commands (middle) the identify command (?) is issued with the nodes reporting their ID and blinking their LEDs (right).

DScope

DScope is a generic digital oscilloscope which can be used to visualise the data streamed by BlueSense. It is compatible with the text and binary mode of BlueSense (figure 13). DScope is under GPL2 on GitHub [dscope].

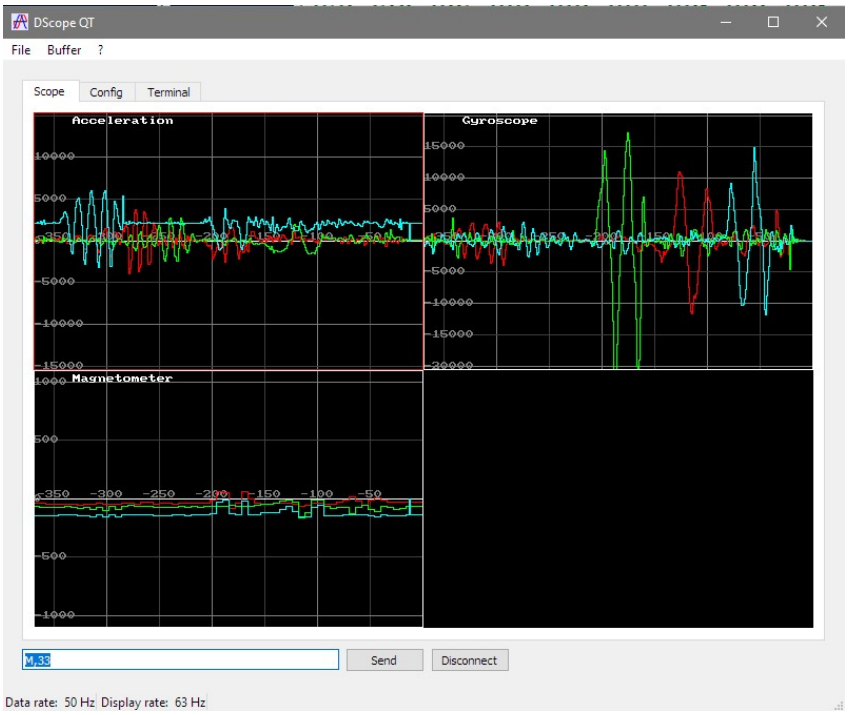

Figure 13. DScope is a digital oscilloscope which can be used to visualise the data acquired by BlueSense.

2.2 Technical details:

- 30x30mm
- LiPo battery (165mAh typically, exchangeable)
- 3V regulated supply
- <200mW motion streaming at 500Hz; <100mW motion logging at 500Hz, 70 µW off
- ATmega1284P core (128KB flash, 16KB RAM)
- USB interface for charging, device interaction
- Bluetooth 2 (Microchip RN41) interface for device interaction
- SD card (SDHC standard) with FAT32 compatible filesystem
- Highest accuracy RTC (real-time clock) on the market: +/-5ppm over the entire temperature range, 0.6ppm measured at ambient temperature.
- 9DoF motion sensor (MPU9250) including software attitude and heading reference system at up to 500Hz (quaternion output). The system is capable of acquiring and storing raw motion data at 1KHz (best effort), which is useful for high frequency applications (sports, vibration analysis)
- Extension ports with analog inputs, digital I/O, I2C, SPI, timekeeping, etc.
- Coulomb counter allowing precise measurement of battery charge/discharge and characterisation of real-time current consumption by software

- True hardware off and alarm wake up: the system can turn itself off entirely, using microamps, and program itself to turn on at a specified date/time thanks to the real-time clock

3. References

[Cuspinera16] Cuspinera, Uetsuji, Morales, Roggen, *Beach volleyball serve type recognition,* Proc. of ACM International Symposium on Wearable Computers, 44-45, 2016

[Roggen16] Roggen et al., *Electric field phase sensing for wearable orientation and localisation applications*, Proc. of the ACM International Symposium on Wearable Computers, 52-53, 2016

[Roggen18] Roggen et al., *BlueSense: designing an extensible platform for wearable motion sensing, sensor research and IoT applications*, Proc. International Conference on Embedded Wireless Systems and Networks, 2018

[dscope] https://github.com/droggen/dscope

[bs2mgr] https://github.com/droggen/bs2mgr

[Madgwick] http://x-io.co.uk/open-source-imu-and-ahrs-algorithms, accessed 26 May 2018