# Development and Evaluation of a ZigFlea-based Wireless Transceiver Board for CUI32

Jim Torresen and Øyvind N. Hauback
Department of Informatics
University of Oslo
P.O. Box 1080 Blindern
N-0316 Oslo, Norway
jimtoer@ifi.uio.no

Dan Overholt
Department of Architecture, Design and Media Technology
Aalborg University, Denmark
Niels Jernes Vej 14, 3-107
dano@create.aau.dk

Alexander Refsum Jensenius
Department of Musicology
University of Oslo
Pb 1017, Blindern, 0315 Oslo, Norway
a.r.jensenius@imv.uio.no

## ABSTRACT

We present a new wireless transceiver board for the CUI32 sensor interface, aimed at creating a solution that is flexible, reliable, and with little power consumption. Communication with the board is based on the ZigFlea protocol and it has been evaluated on a CUI32 using the StickOS operating system. Experiments show that the total sensor data collection time is linearly increasing with the number of sensor samples used. A data rate of 0.8 kbit/s is achieved for wirelessly transmitting three axes of a 3D accelerometer. Although this data rate is low compared to other systems, our solution benefits from ease-of-use and stability, and is useful for applications that are not time-critical.

## Keywords

wireless sensing, CUI32, StickOS, ZigBee, ZigFlea

## 1. INTRODUCTION

Wireless sensing is an important issue in many new digital interfaces for musical expression. The ideal solution here is to build devices that allow for connecting a lot of sensors, and that at the same time are small, fast, inexpensive, accurate/precise, consume little battery power, and, perhaps the most important: are reliable in all sorts of performance contexts. While there are numerous *cabled* sensing platforms to choose from these days, e.g., Arduino, CUI32, Phidgets, there are not many easy-to-use *wireless* sensing solutions that are stable enough for music-related applications.

The de facto standard for short range wireless transmission is Bluetooth (IEEE 802.15.1), a technology which is currently embedded in a variety of commercial devices, ranging from computer mice to mobile phones, cameras, printers, etc. While it certainly works in many contexts, our experience is that Bluetooth is not reliable enough for music-related applications. Then WLAN (IEEE 802.11) solutions are much more powerful and reliable, but such systems are also typically larger, more expensive, and more power-consuming than Bluetooth systems. An alternative solution is to use ZigBee (IEEE 802.15.4), which allows for creating systems that are smaller and less power-consuming than Bluetooth, while at the same time being reliable.

In this paper we report on a project aimed at creating a low-rate and low-power wireless sensor platform, which should at the same time be easy to use also for people that are more interested in creating musical applications than the electronics itself. A sensor collection system is always a compromise between throughput of data and power consumption. High speed is often critical when designing musical *instruments*, where a total latency of approx. 10 ms may be the upper limit [18]. However, in adaptive music devices, aimed at giving the user an active listening experience, the tolerance for latency may be considerably higher [6, 12]. On the other hand, in such devices power consumption is often more important, and this is what we are mainly targeting with the solution discussed in this paper.

The paper starts with a brief overview of the core components that our platform is based on: CUI32, StickOS, ZigBee and ZigFlea. Then our custom-built wireless extension board for the CUI32 is presented, followed by a report on a series of experiments to test its performance with different types and numbers of sensors.

## 2. BACKGROUND

### 2.1 CUI32

The CUI32 is a do-it-yourself USB sensor interface [4], based on the original CREATE User Interface (CUI) [13]. It is an open hardware platform based on the 32-bit PIC microcontroller (PIC32MX440F512H) [11], with a 512 KB flash memory, 32 KB RAM, and 16 analog sensor inputs. The two SPI (Serial Peripheral Interface) connections on the controller make it easy to connect a transceiver for wireless communication.

### 2.2 StickOS

The CUI32 ships with StickOS pre-installed [3], which makes it easy to get started and taking full advantage of the powerful 32-bit PIC microcontroller. StickOS is a full operating system with an on-board BASIC compiler, line editor, debugger, profiler, and a simple file system to create new firmware programs, save them and run them. All this can be done without installing any software on the host computer, only using a standard terminal emulator for the communication. StickOS also supports wireless data transmission, and for logging data to a USB memory stick. For our intended use, creating sensing solutions for music-related applications, StickOS is a good choice: the easy syntax and built-in help system make it possible to get started quickly, and advanced users may appreciate it as a quick prototyping environment.

### 2.3 ZigBee

ZigBee (IEEE 802.15.4) is a low-cost, low-power, wireless network standard [10]. It is somewhat similar to Bluetooth (see comparison in [9]), but overcoming some of the prob-


*NIME'12,* May 21 – 23, 2012, University of Michigan, Ann Arbor.

lems often experienced in Bluetooth devices, e.g. (re)connection to other devices. The ZigBee protocol has a raw transfer rate of around 250 Kbps and 16 channels in the 2.4 GHz band, with the protocol headers and tails included. This maximum transmission range is possible between 10 and 75 m, dependent on the surroundings (walls, etc.).

While ZigBee has been used in some musical projects, e.g. [2, 5, 8], there have not been many systematic studies of its suitability for musical applications. We have previously undertaken some work on determining the possible throughput of ZigBee for an accelerometer-based sensor system using an Atmel AVR microcontroller [17]. In this paper we propose a more general ZigBee-approach for the CUI32/PIC platform.

## 2.4 ZigFlea

ZigFlea is a subset of the ZigBee protocol, implemented in StickOS. It equals ZigBee regarding the physical layer, but is a point-to-point protocol focused on sending in only one direction at a time (half-duplex). The benefit of this is that the protocol stack is as little as 3 KB, as compared to the 30 KB of ZigBee. Our solution is based on the Freescale MC13201 transceiver, which supports the IEEE 802.15.4 standard used by ZigBee and ZigFlea, as well as the Serial Peripheral Interface (SPI) standard for synchronous serial data transfer. The transceiver runs at 2.4 GHz and has support for 16 channels and a speed of up to 250 Kbps. Both point-to-point networks and star networks are possible.

# 3. A NEW TRANSCEIVER BOARD

The aim was to design a wireless transceiver board that could be easily connected to the CUI32. After testing several possible solutions, we settled on an integrated design where the transceiver board can easily be connected on top of the CUI32, as can be seen in Figures 1 and 2. The eagle files for the board are freely available [1].

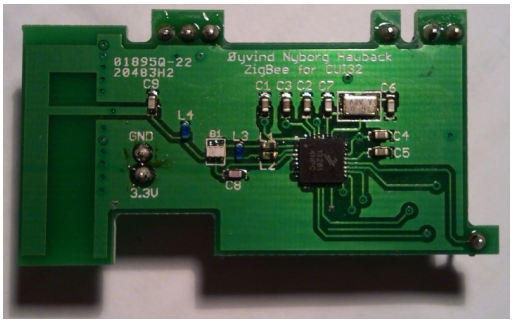


**Figure 1: Our custom-built ZigFlea board.**

The dimensioning of the antenna is based on the Freescale guidelines [15] and [16]. It was a challenge to find the best possible placement of the antenna. Placing it right above the microcontroller on the CUI32 board led to the shortest operation range, probably due to the ground plane covering the (CUI32) board – and disturbing the antenna signal. Thus, the antenna was placed slightly on the side of the board, as can be seen in Figure 2.

# 4. EXPERIMENTS

A number of tests have been undertaken to determine the capability of the wireless platform, and will be presented in the following sessions: (a) continuous reading of one and multiple sensors, (b) infrequent sensor reading, (c) different types of network topologies. All the results that will be presented are average values of 10 runs.

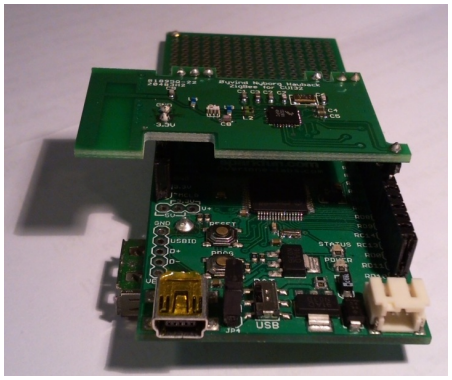


**Figure 2: ZigFlea board mounted on top of the CUI32 board. The antenna is the part sticking out on the left side.**

## 4.1 One Sensor

In the first experiments, we tested with only one sensor, a 3D accelerometer (ADXL335), to get an indication of the highest possible wireless transmission speed. As shown in Table 1, the polling time increases linearly with the number of data samples being read, each containing data from one accelerometer channel. When reading data from all three axes of the accelerometer, the update frequency is 8.3 Hz. If one sample is read and placed in a local variable, not to be transmitted wirelessly, the program uses around 2 ms per reading. This illustrates that the wireless transmission takes the largest fraction of the time for each reading, and that the microcontroller is not a bottleneck in the system.

**Table 1: Measurements of update time and frequency for sampling different numbers of sensor channels of a 3D accelerometer.**

| # samples | Time (ms) | Freq. (Hz) | Time/sensor (ms) |
|---|---|---|---|
| 3 | 120.1 | 8.3 | 40.0 |
| 2 | 78.9 | 12.7 | 39.5 |
| 1 | 38.1 | 26.3 | 38.1 |
| 0 | 0.8 | 1250.0 | — |

The stability of the system was very good in this setup, and it reconnected quickly if the two connected transceivers got out of range.

## 4.2 Multiple Sensors Connected

The second setup was used to test how the platform handles reading several different sensors concurrently. The following sensors were used:

- One orientation (inertial) sensor with a combination of magnetometer, accelerometer and gyroscope (CHrobotics CHR-6dm)
- One distance sensor based on sonar (XL Maxson-EZ1, MB1210)
- One RFID sensor (ID-12)

The distance sensor was connected to the microcontroller through the analog inputs on the CUI32 while the two other sensors were connected through universal asynchronous receivers/transmitters (UARTs). The RFID sensor is based on interrupt control while the orientation sensor is based on polling. For each data polling, 15 samples from the orientation sensor and one sample from the distance sensor were read, respectively.

Table 2 shows the results of the experiment. It took about 652.5 ms (1.5 Hz) to read and transmit data from all the 16

sensors. When only 3 sensor readings were sent, however, the update frequency was 8.3 Hz. Thus, the difference in time per sensor data was minimal compared to the first setup with only a single 3D accelerometer. This tells us that the wireless transmission takes up much more of the time than reading from the sensor locally in the sensor unit. Also, the time needed for reading and transmitting data scales almost linearly with the number of sensor data samples. So an easy way of increasing the overall frequency is to simply limit the number of sensors being read.

**Table 2: Measurements of update time and frequency for sampling different numbers of sensor channels for multiple sensors.**

| # samples | Time (ms) | Freq. (Hz) | Time/sensor (ms) |
|---|---|---|---|
| 16 | 652.5 | 1.5 | 40.8 |
| 15 | 611.8 | 1.6 | 40.8 |
| 14 | 569.9 | 1.8 | 40.7 |
| 13 | 529.6 | 1.9 | 40.7 |
| 12 | 488.7 | 2.0 | 40.7 |
| 11 | 448.0 | 2.2 | 40.7 |
| 10 | 406.7 | 2.5 | 40.7 |
| 9 | 365.8 | 2.7 | 40.6 |
| 8 | 325.1 | 3.1 | 40.6 |
| 7 | 283.6 | 3.5 | 40.5 |
| 6 | 243.2 | 4.1 | 40.5 |
| 5 | 202.2 | 5.0 | 40.4 |
| 4 | 161.3 | 6.2 | 40.3 |
| 3 | 120.4 | 8.3 | 40.1 |
| 2 | 79.1 | 12.6 | 39.6 |
| 1 | 38.7 | 25.8 | 38.7 |
| 0 | 8.9 | 112.4 | - |

In terms of stability, this setup had a tendency to get stuck when the RFID sensor issued an interrupt. To solve this problem, we introduced a short break (a few ms) between each polling.

## 4.3 Infrequent Sensor Reading

The goal of this experiment was to test how the platform works with sensors that are not dependent on frequent reading and high speed wireless transmission.

Here we first tested a setup based on only one RFID reader, connected to the CUI32 through the serial UART interface. The aim was to find the time needed for reading an RFID tag and transferring the whole tag ID wirelessly. This is also a way of testing the interrupt system of StickOS without other sensors interfering.

The results show that, on average, it took 16 ms to read the tag and another 408 ms to transfer all the RFID tag data. Since the RFID tag consists of ten characters, it was sent as ten variables in StickOS. This means that the time it takes to send a character per variable is approximately the same as measured in the experiment in Section 4.2.

To study how adding a second sensor would influence the data, we connected a pulse sensor to the analog input of the CUI32, while keeping the RFID reader connected. Such a system could be relevant for using the RFID to detect the location of a person in a room while the pulse sensor could represent the "quantity of motion" of the person.

Sending the raw pulse data wirelessly would result in varying delays and a slightly erroneous pulse frequency. Instead we measured the time between pulses in the CUI32, calculated the pulse frequency, and only sent the frequency over the wireless connection every 2s. In this setup, the pulse frequency as well as the RFID data were successfully transferred. It may therefore be possible to conclude that the setup works well for non-time-critical wireless sensing.

## 4.4 Star Network Topology

In addition to testing our platform with different numbers of sensors, we were also interested in seeing how different types of network topologies would influence the performance. One such type of topology is a *star network*, in which multiple sensor devices send data to a shared main unit. This is typically useful if several sensors are distributed on different parts of the body. Here our setup consisted of two sensor devices, one with a 3D accelerometer connected (the same as in Section 4.1), and another device toggling between sending "0" and "1".

We started testing with both sensor devices transferring data continuously. This resulted in one of the devices being blocked and not able to transmit data, a result of the main unit being busy with receiving data from the other sensor device. To overcome this problem, we introduced pauses in the transmission. Here we found that it was necessary to introduce a pause of 100 ms to ensure that both sensor units got data sent. Even with such a pause, we experienced some dropouts, and that one unit sent out data more frequently than the other. It was therefore not so easy to measure the average time it took to transfer a variable, so an average of 20 rounds of sensor data collection was applied (i.e. 60 sensor data samples).

The results from testing transmission with pauses of 100, 150 and 200 ms can be seen in Table 3. The table compares the time used in a star network setup with a setup having just a single sensor device transmitting data. We see that the time *difference* decreases as the pause interval increases. Thus, the star network creates less conflict as the pause length increases. The optimal interval depends on the response requirement needed for the given application.

**Table 3: Comparing a star network with two sensor devices and a network with only one sensor unit. All numbers in ms.**

| Pause | Star network | | One sensor device | | Diff. |
|---|---|---|---|---|---|
| | 3 samples | 1 sample | 3 samples | 1 sample | |
| 0 | — | — | 121.4 | 40.4 | — |
| 100 | 358.6 | 119.5 | 192.0 | 64.0 | 55.5 |
| 150 | 375.7 | 125.2 | 242.0 | 80.7 | 44.5 |
| 200 | 417.9 | 139.3 | 295.0 | 98.3 | 41.0 |

We also tested with an RFID reader connected to the second sensor device. In this setup the accelerometer sent data continuously, with sporadic interrupts by the RFID sensor. Here we found that the RFID sensor used the same time as in Section 4.3, i.e. 16 ms for reading the sensor and 408 ms for the wireless transmission. This will, obviously, influence the transmission of accelerometer data, which will be delayed by the same 408 ms.

## 4.5 Point-to-Point Network

A different network topology is a *point-to-point connection*, where one sensor device transmits its sensor data through another sensor device to reach the main unit. The sensor setup in this experiment was the same as the first one in the previous section: a 3D accelerometer in the first sensor unit, and a toggle (0/1) in the second device. The latter was receiving data from the first sensor and transmitted this as well as its own data to the main unit.

During testing it quickly became obvious that the second unit (the middle) got problems with receiving and sending data at the same time. The communication bottleneck became larger than in the previous setup, even when pauses were introduced. In fact, it was difficult to record sufficient timing data for this test since the transmission was so un-

predictable. Even with a pause of 300 ms it took, at best, 455 ms per data variable to be transmitted. So we can safely conclude that this point-to-point network was less suitable than the star network.

## 5. DISCUSSION

The results show that the ZigFlea implementation in the current version of StickOS (version 1.92) is not optimized. For example, it was found that a frequency of 8.3 Hz is all that is possible for wirelessly transmitting three axes of a 3D accelerometer. Because ZigFlea and ZigBee are half-duplex protocols, they require StickOS (or any other firmware) to wait for an acknowledgement after transmitting data. In order to allow the remote node enough time to switch from "receiving" to "transmitting" mode (to reply with the acknowledgement of the received packet), a 40 ms delay is currently hard-coded into the StickOS ZigFlea protocol, and can be seen in the StickOS source code. Future tests should explore the possibility of re-compiling StickOS with this hard-coded delay set to approximately one order of magnitude less (i.e. 4 ms), in order to optimize the performance of ZigFlea.

It can also be seen that ZigFlea variables in StickOS are transmitted one-at-a-time, as soon as they are updated in the user's BASIC program, which explains why the observed transmission times linearly increase with the number of samples being sent. The overall data rate of ZigFlea in StickOS could easily be improved by packaging multiple variables together before sending them. This would obviously come at the cost of slightly increased latency, albeit not much, as gathering multiple channels of sensor data can be accomplished in less than 1 ms in StickOS. This packaging or "bundling" of sensor data is something that will be examined in future versions of StickOS, in order to achieve the full throughput of ZigFlea (250 Kbps). Nonetheless, the achievable actual data throughput will still be less than this maximum specified bit rate, due to required packet overhead. A similar approach has already been taken for non-wireless communication with StickOS. For example, the CUI32 is capable of sustaining the throughput of full-speed USB even with small packets, again achieved by bundling data together before transmission.

Another optimization possibility is to allow sending 16-bit values over ZigFlea, when the current standard of 32-bit values are not needed. Further, the need for introducing pauses when two units try to send concurrently indicates lack of support for synchronization between multiple transmitters. So some system is needed to ensure stability and predictability with multiple transmitters.

When comparing with alternative platforms, our setup benefits from an integrated add-on board for the CUI32 and the ease-of-use through a simple programming language. As such, the system is suitable for applications in need of a low-power and low-rate sensor. If speed and latency are critical, however, other solutions would be preferable. Implementing a full ZigBee solution is one possibility here, as XBee modules can be attached to a CUI32. Or it might even be possible to use WiFi (802.11b/g), which allows OpenSound Control (OSC) packets to be sent directly from the CUI32 [14].

## 6. ACKNOWLEDGMENTS

The research has been funded by the Norwegian Research Council through the project *Sensing Music-related Actions* (proj. no. 183180) and by the EU FP7 *EPICS* project (grant no. 257906). The paper is based on the M.Sc. thesis of the second author [7].

## 7. REFERENCES

[1] Zigflea eagle files. `http://code.google.com/p/cui32/source/browse/#svn/trunk/eagle`.

[2] F. Bevilacqua, F. Guédy, N. Schnell, E. Fléty, and N. Leroy. Wireless sensor interface and gesture-follower for music pedagogy. In *Proc. Int. Conf. on New Interfaces for Musical Expression*, pages 124–129, New York, 2007.

[3] CPUstick. Cpustick and stickos – embedded systems made easy. `http://cpustick.com/`.

[4] CUI32. `http://code.google.com/p/cui32/`.

[5] E. Fléty and C. Maestracci. Latency improvement in sensor wireless transmission using IEEE 802.15.4. In *Proc. of Int. Conf. on New Interfaces for Musical Expression*, pages 409–412, Oslo, 2011.

[6] M. Goto. Active music listening interfaces based on signal processing. In *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, volume 4, pages IV–1441. IEEE, 2007.

[7] Ø. Hauback. *Technology for sensor data collection for media player*. M.Sc. thesis (in Norwegian), University of Oslo, Norway, 2011.

[8] R. Jacobs, M. Feldmeier, and J. A. Paradiso. A mobile music environment using a PD compiler and wireless sensors. In *Proc. of Int. Conf. on New Interfaces for Musical Expression*, Genova, 2008.

[9] J. Kooker. Bluetooth, ZigBee, and wibree: A comparison of WPAN technologies. `http://johnkooker.com/blog/wp-content/uploads/2009/05/jkooker_BTZigBeeWibree.pdf`, 2008.

[10] J. Lee. Performance evaluation of IEEE 802.15.4 for low-rate wireless personal area networks. *IEEE Transactions on Consumer Electronics*, 52(3):742 – 749, 2006.

[11] Microchip. Pic32mx440f512h. `http://www.microchip.com/wwwproducts/Devices.aspx?dDocName=en535591`.

[12] B. Moens, L. van Noorden, and M. Leman. D-jogger: Syncing music with walking. In *Proc. of Sound and Music Computing Conference*, pages 451–456, Barcelona, 2010.

[13] D. Overholt. Musical interaction design with the CREATE USB interface teaching HCI with CUIs instead of GUIs. In *Proc. of Int. Computer Music Conference*, pages 425–430, New Orleans, LA, 2006.

[14] D. Overholt. Musical interaction design with the CUI32Stem: Wireless options and the grove system for prototyping new interfaces. In *Proc. of Int. Conf. on New Interfaces for Musical Expression*, Ann Arbor, MI, 2012.

[15] F. semiconductor. Compact integrated antenna. `http://www.freescale.com/files/rf_if/doc/app_note/AN2731.pdf`, July 2006.

[16] F. semiconductor. Mc13201. `http://cache.freescale.com/files/rf_if/doc/data_sheet/MC13201.pdf`, Apr. 2008.

[17] J. Torresen, E. Renton, and A. R. Jensenius. Wireless sensor data collection based on ZigBee communication. In *Proc. of Int. Conf. on New Interfaces for Musical Expression*, pages 368–371, Sydney, 2010.

[18] D. L. Wessel and M. Wright. Problems and prospects for intimate musical control of computers. In *Proc. of Int. Conf. on New Interfaces for Musical Expression*, Seattle, WA, 2001.