# The BSNOS Platform: A Body Sensor Networks Targeted Operating System and Toolset

Joshua Ellul
*Department of Computing*
*Imperial College London*
*London, UK*
*Email: jellul@imperial.ac.uk*

Benny Lo
*Deparment of Computing*
*Imperial College London*
*London, UK*
*Email: benny.lo@imperial.ac.uk*

Guang-Zhong Yang
*Deparment of Computing*
*Imperial College London*
*London, UK*
*Email: g.z.yang@imperial.ac.uk*

***Abstract*—Body sensor networks face different challenges than those faced in traditional wireless sensor networks. Challenges faced include fewer, more accurate sensor nodes and an increased requirement of context awareness, however, body sensor networks are relieved from high scalability. Programming sensor networks, in general, can be a daunting task due to the limited computation, memory and energy resources available. Operating systems for wireless sensor networks have been proposed which focus on the challenges they face. We believe that an operating system and toolset which focuses primarily on the challenges and properties of body sensor networks could help ease the burden of programming body sensor network applications. In this paper, we present an operating system which is focused on facilitating body sensor network application development.**

***Keywords*-body sensor networks; operating systems;**

## I. Introduction

Body sensor networks have emerged as a promising platform to enable scientists to further understand how the human body operates with a means of remotely monitoring different environmental conditions accurately. Operating systems for wireless sensor networks have been the primary tool in developing body sensor node applications in the past. Developers require expertise in C and embedded programming to be able to create applications. Even seasoned programmers find the task of programming such applications daunting and time consuming. Domain scientists rely on and wait for such applications to be developed before they can extract any useful information from the environment they wish to investigate.

The challenges faced in body sensor networks vary to that of traditional wireless sensor networks. Body sensor networks usually consist of smaller networks of short range communication, whilst traditional wireless sensor network research has been heavily focused on routing and MAC protocols to facilitate scalability and longer range communication.

In this paper, we present the BSNOS platform, an operating system and platform designed to especially meet the requirements of body sensor network applications. The operating system exposes a Java programming environment enabling novice programmers to develop applications with ease. Also, in that it is an operating system, we believe that it should be usable without requiring any low level development. Thus, we have included in the operating system a default application which allows domain experts to monitor environments and configure the sampling and transmission rates of collected data.

The primary contribution of this paper is that we have designed the first operating system targetted especially for body sensor networks, which utilizes run-time compilation of bytecode to provide an efficient Java execution platform for resource constrained devices.

The remainder of this paper is structured as follows. Section II provides an overview of related work. Section III describes the motivation behind why this work is necessary. We provide an overview of the system design in Section IV, followed by implementation details in Section V. We evaluate our work in Section VI and then conclude in Section VII.

## II. Related Work

Operating systems commonly used for sensor network development include TinyOS [1] and Contiki [2]. TinyOS exposes an event driven execution model programmed using a component model. TinyOS applications are programmed using the nesC language developed specifically for TinyOS. Contiki also provides an event-driven programming model, however, programs are coded in C. It is very hard for non-embedded developers to write programs for these operating systems due to the level of embedded and event-driven programming experience required.

More recent initiatives to enable Java for sensor nodes aims to facilitate an easier programming paradigm for sensor nodes [3] [4] [5] [6]. The Squawk virtual machine [4] allows for Java code to be executed on Sun SPOT devices. Sun SPOT devices have substantially more resources than that of more popular sensor nodes (especially those used for body sensing). The Squawk virtual machine has a program memory footprint of 270kB which is larger than that available for most body sensor nodes available.

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-144-1 

Darjeeling [6] and TakaTuka [5] are two virtual machines which provide a Java interpreter for resource constrained devices. Interpreters are infamous for their high execution overheads, and as can be seen in [6], the overhead is quite large. To overcome such interpretation overheads, run-time compilation of bytecode [3] was proposed for sensor nodes, which provides substantially faster execution of bytecode compared to interpreted versions.

## III. Motivation

Body sensor networks provide a different environment and paradigm to that of traditional wireless sensor networks [7]. The size of the environment (the body) to be monitored is much smaller and is geometrically the same to any other (body) deployments, unlike the environments which traditional sensor networks are deployed in (which vary from one deployment to another). This results in smaller sized networks with nodes placed in, usually, known places. A body sensor network rarely grows (or shrinks) in size. Also, once a node is placed, it is very unlikely to move. Transmission ranges of sensor nodes do not require long distances. Additionally, nodes are usually one hop away from every node in the same body sensor network. These properties of body sensor networks relieve body sensor nodes from complex MAC and routing protocols which is a primary requirement for traditional wireless sensor networks.

Although body sensor networks will consist of a smaller numbers of nodes, relieving the nodes from scalability issues, this also means that nodes will not be able to rely on neighbouring nodes for redundancy. Thus, body sensor networks require more accurate and robust sensing algorithms. More so, typical body sensor networks do not usually have the luxury of long sleep periods since events must be caught immediately due to the repercussions of missing a vital event, which in turn affects the lifetime of the sensor node. However, most body sensor network deployments do not require long lifetimes since they are usually used temporarily to analyse or detect specific conditions or applications.

Due to the geometrical, environmental and requirement similarities between different body sensor network deployments and applications, a substantial amount of application logic is common amongst body sensor network applications. Such logic includes context awareness both of the internal environment (the body) and the external environment, logging of data (either distributed or centrally), uploading of data to a pc or server either real-time or post-collection, and sending alerts when an interesting event has occurred.

Programming sensor network applications is a challenging task which most often requires experienced embedded developers to develop such applications and also may involve adopting new programming languages and models. This is primarily due to the operating systems, languages and tools which are currently used including C based operating systems [1][2].

The motivation behind our work is to provide an operating system which facilitates ease of development of body sensor network applications for novice programmers and domain experts. We plan to achieve this by focusing on the intrinsic properties of body sensor network applications. Whilst at the same time, sacrificing features required for traditional wireless sensor networks which are not inherent in body sensor networks in aim of delegating such relieved resources for ease of programmability and body sensor network specific functionality. By exposing an easy to use standard API we envisage that algorithms can seamlessly be shared, switched and compared. Thus, enabling an open development research platform for body sensor network applications.

## IV. Design

One of the main motivations behind our work is to provide an easy to use programming environment for developing body sensor network applications for novice programmers. Most available tools for sensor network applications rely on C (or flavours thereof such as nesC) as a programming language. However, we have decided to opt for a Java programming environment to facilitate an easy to use programming environment. As mentioned in Section II, several initiatives have been made to port Java to sensor nodes, most of which use an interpreter to execute bytecode. As shown in [6], interpretation of bytecode suffers from great execution overheads. We have, therefore decided to utilise a run-time compiler similar to [3] in aim of offering a Java programming environment without incurring substantial execution overhead. An overview of the toolchain and operating system is shown in Figure 1. We will follow by describing each component of the toolchain and operating system.

*1) Scripting:* We would like to target not only programmers but also domain experts who more than likely do not have a large amount of experience programming. Thus, we have included in our architecture a means of being able to control how a body sensor network application would work on a higher level. For those with slight knowledge of programming we have provided a scripting tool which can be used to create simple applications by configuring the main execution thread using a Java like syntax without having to deal with the intrinsic properties of Java. The scripting source is passed into the Java Source Generator which creates equivalent Java source of the specified application, which can then be passed through the toolchain as normal Java source. The reason for doing this is that we can facilitate such abstractions without incurring any extra overhead on the sensor node itself.

*2) Parameter Configuration:* For domain experts without any knowledge of programming, we have provided a tool to configure body sensor network applications' sensing and transmission rates, which should allow domain scientists

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-144-1 

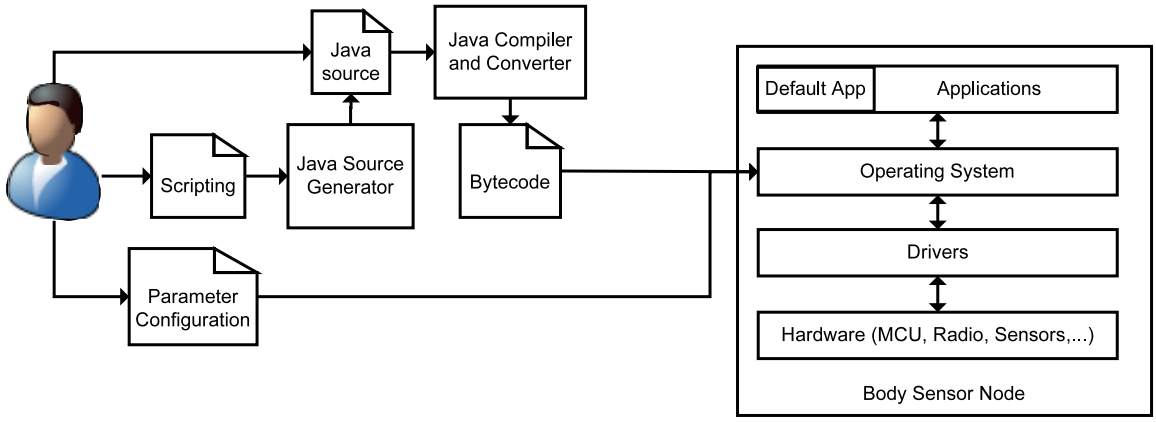


Figure 1. An overview of the toolchain and operating system.

to extract initial data before moving on to more complex applications. BSNOS is initially loaded with the Default Application. The Default Application will sample the connected sensors according to the parameters set using the parameter configuration tool. The data will then be sent to a base station at a period also defined by the parameter configuration tool.

*3) Java for Sensor Nodes:* Java source is passed into the Java compiler and converter which generates an altered version of Java bytecode which is more suitable for resource constrained devices. The details of this compilation and conversion process is similar to that described in [3]. The generated bytecode differs to that of standard Java bytecode in that String literals relating to class and function names are completely removed. Also, Java bytecode is based on a 32 bit stack width, which for majority of sensor network applications is not required. Thus, the converter also changes the bytecode to use a 16 bit stack width. The bytecode generated can then be disseminated to sensor nodes over a serial connection or over-the-air. The benefits of transmitting bytecode instead of native code is that bytecode is smaller in size, and thus, less energy is required to transmit code updates.

*4) Applications to Hardware:* At the lowest level of the BSNOS stack lies the underlying hardware including the microcontroller, radio and sensors. Driver code installed on the microcontroller will expose the different peripherals and hardware features which are required to be used. The operating system can then expose the hardware via the drivers to provide common lower level services required by applications.

*5) The BSNOS Kernel:* Figure 2 provides a view of the different operating system and facilitating tool components. The BSNOS Kernel encapsulates services which are essential to provide an easy to use Java programming and operating system environment. The main kernel components are outlined below:

**Java Run-time Environment**: The run-time compiler generates native code which mimics the Java operand stack used by the original bytecode. Due to this a Java run-time environment is required. The run-time environment exposes basic Java functionality such as array manipulation, object creation, exception handling amongst other features.

**Garbage Collector**: Java relieves programmers from the task of memory management. However, after objects or arrays are no longer required, their memory is still being occupied, and thus a check is required to release such memory that is no longer being used. This is the job of the Garbage Collector.

**Run-time Compiler**: By providing a Java programming environment, the learning curve for novice programmers to develop applications for body sensor networks will be decreased. However, since the early days of Java, it's slow speed was notable as was summed up in [8] with their statement "Java isn't just slow, it's really slow, surprisingly slow." This was primarily due to the fact that Java's execution platform was completely based on interpreters. In aim of providing an efficient execution platform for Java based applications, we have opted to use a run-time compiler similar to [3] which compiles the bytecode it receives to native code which the underlying microcontroller can execute without incurring any interpretation overheads.

**Software Manager**: Body sensor networks like other wireless sensor networks and traditional computing systems require updates from time to time due to various reasons including bug fixes, new features or complete new applications. The Software Manager enables efficient remote updating by providing dynamic loading and unloading of code at the granularity of classes or functions.

**Scheduler**: At the core of the kernel lies the Scheduler. The Scheduler is responsible for scheduling execution of threads and events. Figure 3 depicts how events and threads are scheduled. Interrupts (IRQ) raised on microcontrollers have higher priority than the main thread of execution. Similarly, we have designed events in BSNOS to have a higher

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-144-1 

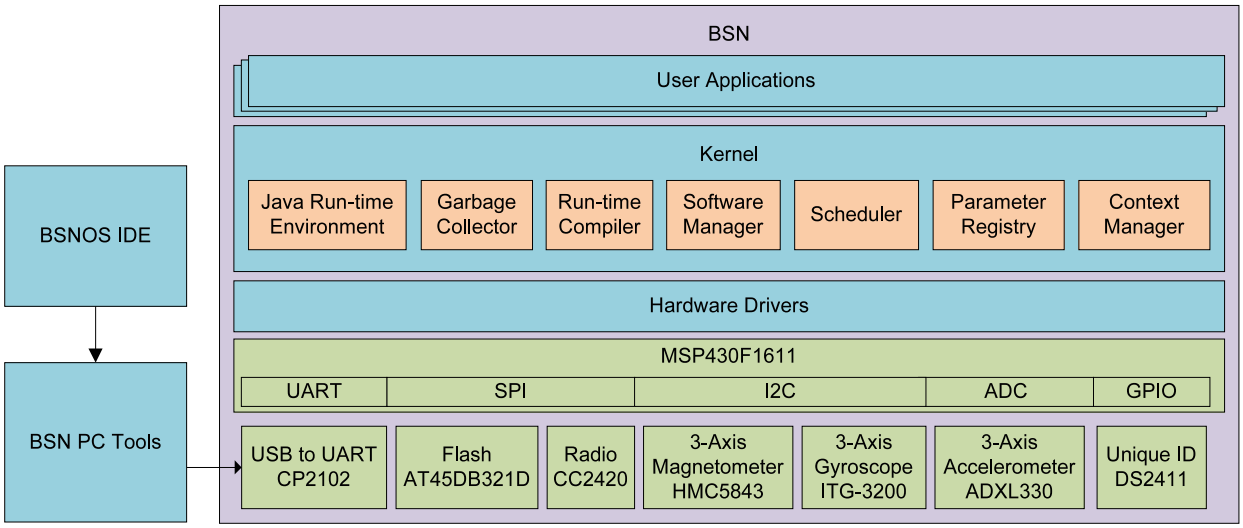


Figure 2. The operating system and facilitating tool components.

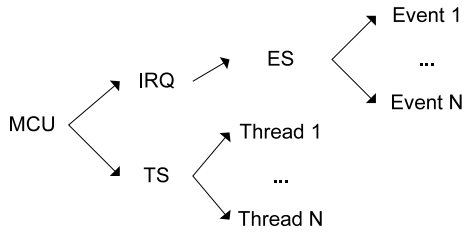


Figure 3. The scheduler execution policy is demonstrated above. Preference is given to microcontroller interrupts (IRQ) which in turn then invoke system events by the Event Scheduler (ES). The Thread Scheduler (TS) is invoked when no interrupts are pending, which in turn executes threads in a round-robin fashion.

priority than other threads since the primary aim of body sensor network applications is to monitor the external events. Thus, when an interrupt is raised the BSNOS kernel buffers any related events and then invokes the Event Scheduler (ES) which will then dispatch any events which are waiting to be scheduled.

The Thread Scheduler (TS) is responsible for scheduling thread execution. We have implemented a round-robin scheduler which provides equal execution slots to each thread in aim of providing a lightweight threading API. However, in future we will investigate whether more sophisticated scheduling techniques should be adopted.

**Parameter Registry**: Sensor networks application users often require to change parameters which alter the way applications would operate. It is to the best of the authors knowledge that other sensor network operating systems have however not provided any API to cater for such a common feature, leaving it up to the application developer to provide a mechanism for storing, altering and retrieving of parameters. The Parameter Registry is a storage area in the operating system whereby parameters can be stored, retrieved and updated by different applications using a standard API.

**Context Manager**: Context is an element that is predominant in body sensor networks. We would like to encapsulate context aware logic that is common to body sensor networks in the operating system. The Context Manager provides a built in mechanism to support applications in their contextual needs. The human body provides an environment which (usually) consists of the same structure amongst different subjects. Thus, we have created an API to facilitate developers in defining the position of sensor placement.

## V. Implementation

The BSNOS kernel is written in C and compiled using Texas Instruments' Code Composer Studio v 4.0.1 for the MSP430F1611 microcontroller. As an initial target node, we have decided to port the operating system for the BSN platform [9]. A detailed view of BSNOS, hardware and facilitating tools is depicted in Figure 2. We have implemented drivers for Atmel's AT45DB321D 4MB DataFlash and Texas Instruments' CC2420 Radio which communicate with the microcontroller over SPI. The $I^2C$ protocol is used to communicate with Honeywell's HMC5843 3-Axis Magnetometer and InvenSense's ITG-3200 3-Axis Gyroscope. ADC channels are used to sample the analog readings for Analog Device's ADXL330 3-Axis Accelerometer and a unique ID is provided to the operating system using Dallas Semiconductor's DS2411 chip which is communicated with using the proprietary 1-Wire protocol.

The ROM footprint for each module is listed in Table I, totalling to just over 12 KB.

We have implemented a lightweight API which provides an easy to use interface. A sample of the BSNOS API is

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-144-1

Table I
MODULE FOOTPRINT

| Module | ROM |
|---|---|
| Run-time System | 2 KB |
| Java Run-time | 500 B |
| Run-time Compiler | 8 KB |
| Threading | 400 B |
| Class Loader | 400 B |
| Drivers | 1 KB |

listed in Table II. To demonstrate the API, a sample program is listed in Figure 4.

## VI. EVALUATION

Evaluation of run-time compilation code compared with interpreted code and native code generated from C source code has been evaluated in [3]. The aim of BSNOS is to provide an easy to use platform which is focused primarily on higher data rates due to the nature of common body sensor network applications. We have conducted evaluation as regards to the maximum throughput that can be achieved using default configurations of BSNOS and TinyOS. Figure 5 depicts the maximum transmission rates of messages of size 1, 50 and 100 bytes for BSNOS and TinyOS. As can be seen from the graph BSNOS achieves a substantial increase in maximum throughput when compared with TinyOS. Obviously, however, this is due to the radio driver implementation and the minimal MAC layer that was used. Were a similiar radio implementation to be used with TinyOS, than a similiar throughput would be achieved.

## VII. CONCLUSION

In this paper we have presented, BSNOS, a new lightweight operating system designed specifically for body

Table II
SAMPLE API

| Module | Function |
|---|---|
| Accelerometer | `void performAccelSample()` |
| | `float getAccelX()` |
| | `float getAccelY()` |
| | `float getAccelZ()` |
| Context | `BodySegment getBodySegment()` |
| Flash | `void writeByte(byte b)` |
| | `byte readNextByte()` |
| | `void dumpFlashToSerial()` |
| Gyroscope | `void performGyroSample()` |
| | `float getGyroY()` |
| Parameters | `short getShortParameter(Parameter p)` |
| Radio | `void msgAppendFloat(float f)` |
| | `void msgAppendShort(short s)` |
| | `void msgAppendInt(int i)` |
| | `void sendMsgToNode(short nodeid)` |
| Serial | `void serialSendByte(byte b)` |
| | `byte serialReceiveByte()` |
| Threading | `void sleepMS(short ms)` |
| Unique ID | `long getUniqueID()` |

```
public class SampleAndSend {

    @BStart()
    public static void main() {
        while(true) {
            BSN.performAccelSample();
            float x = BSN.getAccelX();
            float y = BSN.getAccelY();
            float z = BSN.getAccelZ();

            BSN.newMsg();
            BSN.msgAppendFloat(x);
            BSN.msgAppendFloat(y);
            BSN.msgAppendFloat(z);
            BSN.sendMsg();

            BSN.sleepMS(1000);
        }
    }

}
```

Figure 4. Sample source for a sample and send application which samples the accelerometer and sends the data to the base station.

sensor networks. We have implemented BSNOS on the BSN node as the first target platform. The aims of the operating system is to allow novice developers to focus on the specific application rather than the intrinsic properties of embedded programming. We aimed to achieve this by exposing functionality common in body sensor network applications such as parameter configuration and context. We envisage that by providing a standard API and a Java programming environment, developers would be able to seamlessly share and compare different implementations and algorithms. The contributions of this paper are as follows:

- We have presented the first operating system targeted

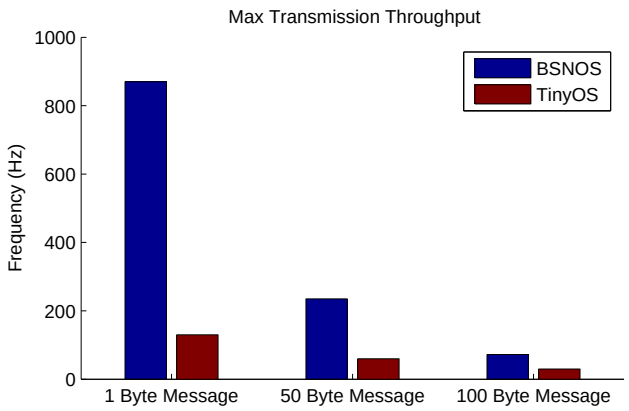


Figure 5. Maximum transmission throughput for BSNOS and TinyOS using default configurations.

for body sensor network applications.

- We have proposed a parameter registry for sensor node operating systems which releases the developer of implementing such functionality.
- We have implemented first steps towards integrating body sensor network context awareness into an OS.
- We have designed the first operating system which utilizes run-time compilation of bytecode to provide an efficient means of executing Java applications for resource constrained embedded devices.

We plan to continue development on the operating system with the following future work:

- We envisage that domain experts would be able to write body sensor network applications using a GUI block programming environment. Thus, we will investigate into integrating such tools.
- We have currently implemented a round-robin thread scheduling policy, however would like to investigate into more sophisticated policies and the associated tradeoffs and benefits.
- The current implementation of the Context Manager exposes functionality to easily define and share context amongst code and applications. We would like to further investigate whether we can integrate commonly used algorithms into the operating system or as system libraries that can be dynamically loaded.

## ACKNOWLEDGMENT

We would like to thank James Patterson for the endless discussions regarding many aspects of the BSN node and support in testing the framework. This work was supported by the ESPRIT project funded by EPSRC.

## REFERENCES

[1] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An operating system for sensor networks," in *in Ambient Intelligence*, 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.129.7716

[2] A. Dunkels, B. Grönvall, and T. Voigt, "Contiki - a lightweight and flexible operating system for tiny networked sensors," *Local Computer Networks, Annual IEEE Conference on*, vol. 0, pp. 455–462, 2004.

[3] J. Ellul and K. Martinez, "Run-time compilation of bytecode in sensor networks," in *Sensor Technologies and Applications (SENSORCOMM), 2010 Fourth International Conference on*, 2010, pp. 133 –138.

[4] D. Simon, C. Cifuentes, D. Cleal, J. Daniels, and D. White, "Java on the bare metal of wireless sensor devices: the squawk java virtual machine," in *Proceedings of the 2nd international conference on Virtual execution environments*, ser. VEE '06. New York, NY, USA: ACM, 2006, pp. 78–88. [Online]. Available: http://doi.acm.org/10.1145/1134760.1134773

[5] F. Aslam, L. Fennell, C. Schindelhauer, P. Thiemann, G. Ernst, E. Haussmann, S. Rührup, and Z. Uzmi, "Optimized Java Binary and Virtual Machine for Tiny Motes," in *Distributed Computing in Sensor Systems*, R. Rajaraman, T. Moscibroda, A. Dunkels, and A. Scaglione, Eds., vol. 6131. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 15–30. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-13651-1_2

[6] N. Brouwers, P. Corke, and K. Langendoen, "Darjeeling, a java compatible virtual machine for microcontrollers," in *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*, ser. Companion '08. New York, NY, USA: ACM, 2008, pp. 18–23. [Online]. Available: http://doi.acm.org/10.1145/1462735.1462740

[7] G.-Z. Yang, *Body Sensor Networks*. Springer-Verlag New York, Inc., 2006.

[8] P. Tyma, "Why are we using java again?" *Commun. ACM*, vol. 41, no. 6, pp. 38–42, 1998.

[9] B. Lo, S. Thiemjarus, R. King, and G. Yang, "Body sensor network - a wireless sensor platform for pervasive healthcare monitoring," in *Proceedings of the 3rd International Conference on Pervasive Computing (Pervasive 2005)*, 2005, pp. 77–80.

Copyright (c) IARIA, 2011. ISBN: 978-1-61208-144-1 