

Beasties: Simple Wireless Sensor Nodes

Asher Hoskins
Imperial College
London SW7 2AZ
United Kingdom
Email: asher@doc.ic.ac.uk

Julie McCann
Imperial College
London SW7 2AZ
United Kingdom
Email: jamm@doc.ic.ac.uk

Abstract—Interest in wireless sensor nodes has escalated over the past 10 years bringing with it a plethora of sensor hardware as well as protocol, algorithmic and application research. Yet there is relatively little diversity in terms of sensor nodes to support this research. This paper presents an alternative sensor node architecture, the Beastie, which aims to improve research and experimentation turnaround time while easing WSN systems programming considerably. We support our claims through quantitative evaluation and describe some applications where the Beasties have been successfully used.

Index Terms—wireless sensor networks, hardware, components

I. INTRODUCTION AND REQUIREMENTS

Beasties were created out of a need to be able to easily and quickly prototype wireless sensor node (WSN) devices. The Autonomic Networked Systems (ANS) project [1], started in mid-2003, called for a network of such devices to be built in order to investigate autonomic (self-configuring, optimising, and healing) network protocols and systems for medical applications. However, we wished to allow our work to be generalisable to other applications, research opportunities and possibly teaching. Our primary hardware requirements therefore were:

- To be able to do rapid one-off development of sensor and actuator peripheral boards. This meant using prototyping stripboard. The node would therefore have to be able to be connected to the low density matrix of a stripboard easily.
- To be able to modify the node as and when necessary without the use of a specialised rework laboratory.
- To produce physically robust prototype systems that could be deployed outside of the laboratory.
- To be able to write our own compilers. This meant that a modern microcontroller design would be desirable.

Several platforms were available when the project started. At one end of the scale, with devices such as Berkeley Motes [2], we found that the existing technologies were too targeted towards “production” environments and so used surface mount components, very fine pitch connectors and lacked physical robustness. This produced a very small, compact, node but one that did not match our requirements. At the other end of the scale were simple low density systems like the Lancaster Smart-ITS [3] and BASIC Stamps [4]. These either lacked built-in radios for networking or used older

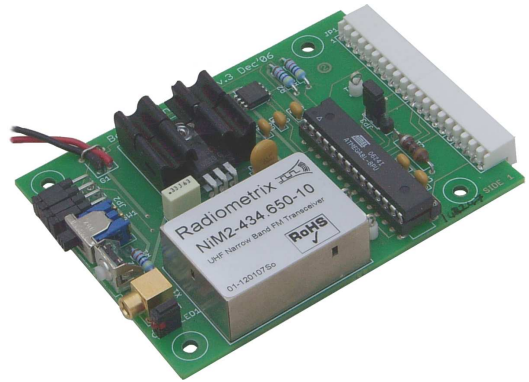


Fig. 1. A Beastie.

microcontroller designs like the Microchip PIC series which are much harder to write compilers for than more modern chips such as the Atmel AVR chips that we ended up using.

None of the then available platforms fitted our needs adequately and so we designed the “Beastie”. In common with other WSNs a Beastie contains a processor, memory and a low power radio for wireless networking. Sensors and actuators to be read and controlled by the Beastie are attached via an expansion bus which also supplies a stabilised power supply to peripherals.

From the early prototypes made on stripboard to the current PCB-based devices the Beastie has shown itself to be a useful research tool, being used in the Bop! [5] and DOORS [6] projects over the past few years as well as the original ANS project and various other side projects investigating solar power and monitoring animals.

II. ARCHITECTURE

A plan view of a Beastie with its key sections highlighted is shown in Fig. 2. The connections between these sections are shown in the system block diagram in Fig. 3.

Where possible the Beastie is built using low density “through-hole” devices, devices that mount to the circuit board using pins that pass through it and which are soldered on the back side of the board rather than more modern, but much harder to modify, surface mount devices.

The use of through-hole parts makes the Beastie bigger than it could be, but simplifies modification and repair and the attachment of extra test points. These were felt to be more

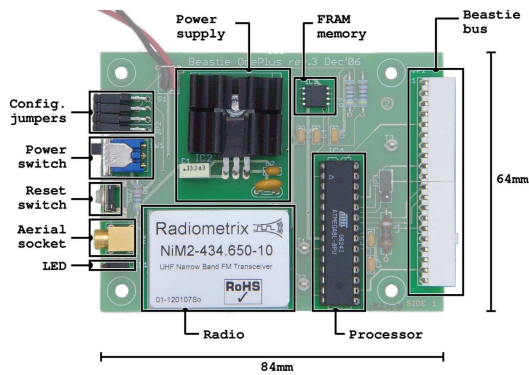


Fig. 2. Beastie Plan View.

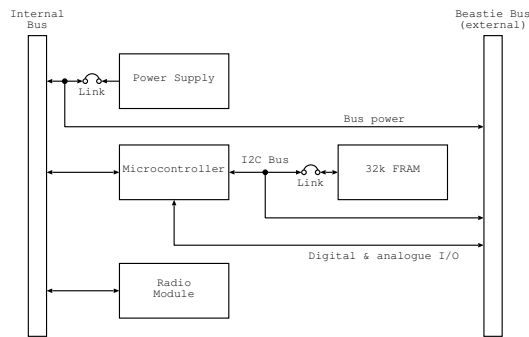


Fig. 3. Beastie Block Diagram.

useful features than extremely small size for the purposes for which the Beastie was designed; something that has been borne out by the applications that the Beasties have been used in (see section IV-A).

A. Atmel ATmega8 Processor

At the core of a Beastie is an Atmel AVR ATmega8 microcontroller. This is a low power 8-bit Harvard architecture RISC microcontroller with 8 kB of program memory, 1 kB of static RAM data memory, and 0.5 kB of non-volatile EEPROM. The chip includes a selection of built-in peripherals, digital and analogue inputs and outputs, and serial interfaces.

The Beastie relies on the processor's internal clock which may be set to 1–8 MHz. Beasties normally run at 4 MHz which gives a good trade-off between power and speed. The Atmel executes most instructions in only one or two clock cycles and so the processor runs at almost 4 MIPS. The clock speed may only be changed using an external programming device and so cannot be varied at runtime (this is a feature of many microcontrollers). The internal clock signal is derived from an RC oscillator and is accurate to $\pm 3\%$. If a more accurate clock is required (as for RS-232 communications) then an external clock signal can be supplied via the Beastie Bus.

The processor is socketed so that it may be easily replaced without soldering should it become damaged. This is an important feature on a tool intended for research. The processor's internal flash and EEPROM memory can only handle a limited number of writes (10,000 for the flash, 100,000 for

the EEPROM) and may be worn out by a runaway program or poor algorithm design. For simplicity and to avoid having to use power to run buffers all of the I/O pins on the Beastie Bus are connected directly to pins on the processor. Wiring faults or strong electrical fields around peripherals may therefore damage the processor. A new processor costs less than £2 at the time of writing, dramatically cheaper than replacing the whole device or paying for surface mount rework as would happen with similar devices that use soldered chips. Socketed devices are used for the same reason on some of the peripheral cards that have been developed such as the RS-232 serial interface (where long serial cables make the RS-232 buffer chip vulnerable to electrical damage, especially if used outside).

B. Radiometrix NiM2 Radio

The wireless networking interface on a Beastie is a Radiometrix NiM2 module [7]. This allows for data rates of up to 10 kbit/s and a maximum stated range (with no obstructions) of 500 m. The range is dramatically reduced indoors due to attenuation of the signal by the structure of the building and internal wiring.

The radio module operates in the unlicensed 433 MHz ISM band. Of the various ISM bands, 433 MHz is one of the best for use in urban spaces since it passes through buildings and people better than the alternatives [8]. It also allows for systems using simple single-hop networking to cover a reasonable area (although it does mean that a larger aerial and lower data rates must be used compared to higher frequency bands such as 2.4 GHz). Beasties currently run the radio modules at half speed and Manchester encode their data to provide a data rate of 2.5 kbit/s. In practice this has been found to be quite adequate for transmitting sensor data.

C. Ramtron FRAM

A Ramtron FM24C256 FRAM memory provides 32 kB of non-volatile storage external to the processor and accessed via a serial bus. "FRAM", ferroelectric memory, is a lower density form of non-volatile memory than the more common flash memory but uses less power and, unlike flash memory, has an essentially unlimited number of write cycles, making it ideal for FIFO buffers and other frequently updated stores.

D. Power Supply

The power supply on the Beastie is a very simple 7805-type linear regulator that accepts a DC input voltage of between 7 V and 20 V and produces a 5 V supply at up to 500 mA to run the Beastie and any peripherals attached via the Beastie Bus. A regulated power supply for peripherals was one of the design aims of the Beasties and makes peripheral devices simpler and thus quicker to prototype since they do not require the construction of their own onboard regulator.

E. Uploading Software

Software is uploaded to the Beastie using a programming device which connects to the Beastie Bus and is able to access

the non-volatile memories of the processor. Note that the RAM of the processor may not be examined using the programming device and so it can not be used to examine the contents of variables and data objects. The processor always starts execution as though it has received a hardware reset when the device disconnects and so this system can not be used as the basis of a debugging system with single-stepping. Such features may be implemented on the processor but they require a debugging kernel to be run which can provide memory probing and stepping control features to an attached PC [9].

The processor could also theoretically be programmed via the radio although this has not been implemented yet. The slow speed and unreliability of the wireless network and the limited memory on a Beastie mean that reprogramming via the network is more difficult than it may first appear. See [10] for a discussion of some of the issues involved.

F. Configuration

Since it was designed as a research tool and is expected to be modified and used in ways that the designer did not envisage during its life a Beastie has a number of hardware and firmware configuration options that change its behaviour.

By removing hardware jumpers the power supply or FRAM subsystems may be disconnected. Disconnecting the power supply allows power to be supplied at a regulated 5 V via the Beastie Bus instead. This allows new power supply designs to be tested such as the DC-DC convertor design mentioned above as well as solar power supplies.

The serial bus that the processor uses to communicate with the FRAM memory takes up two lines on the processor that may otherwise be used as analogue or digital inputs. If the FRAM is not needed for a particular application then it may be disconnected and the serial bus pins used for other purposes.

The radio module used on the Beastie contains an analogue output which gives an indication of the received signal strength. Normally this is ignored since the radio routines use a software method to detect traffic but if desired a jumper can be used to connect the signal strength to one of the analogue inputs on the processor so that it may be monitored in software.

G. Daughterboards

Expansion “daughterboards” may be stacked above the Beastie’s PCB and connected via a backplane that links to the “Beastie Bus” on the edge of the Beastie’s PCB. The bus provides access to most of the processor’s pins that aren’t used to control functions on the Beastie itself as well as a 500 mA, 5 V, power supply (shared with the Beastie’s own internal components).

Daughterboards and Beasties have holes drilled in them so that they may be bolted together into a robust stack using 3 mm metal studding and plastic spacers as shown in Fig. 4. The use of studding to hold a Beastie system together is in marked contrast to other sensor nodes which rely purely on the strength of their connectors. In our experience relying on connector strength leads to problems with reliability, especially when

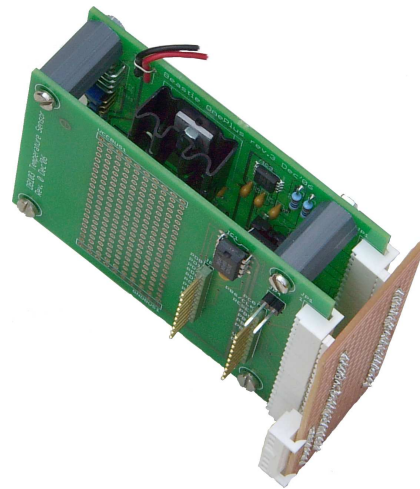


Fig. 4. A Beastie and Daughterboard Connected Via a Backplane.

devices are deployed outside of the laboratory. It does not take much force to partially lever a connector off and cause bad connections, especially on smaller devices with high density connectors.

Beasties were always designed to support easy “one-off” prototyping and so their form factor and bus density was selected to allow daughterboards to be constructed using prototyping stripboard. The mounting holes line up with holes in the stripboard of a daughterboard and match the diameter of common stripboard track cutter tools so that no drill is required when constructing a board.

Daughterboards can also be made as conventional PCBs. This approach has been used whenever more than a few cards of the same type have been required such as the RS-232 interface card and a temperature sensing and prototyping card built for the Bop! project.

Backplanes are currently made using prototyping stripboard (“Veroboard”) and provide two or more rows of pins to link Beasties to one or more daughterboards.

III. SOFTWARE SUPPORT

A. Tesserae

A piece of hardware is nothing without software and software was the real focus of the ANS project. While Beasties can of course be programmed using a simple assembler or C compiler, the intention when they were built was to use component-based software engineering [11] since the component model matches that of embedded systems well (components and interfaces are particularly useful for abstracting sensors and their controls). A “component” is a completely self-contained block of executable code that can only communicate with other components via “interfaces”, lists of possible messages. Components are bound together via matching interfaces in order to build new components and ultimately complete systems. The implementations of components are separate from the descriptions of how they are bound together. Interface definitions are separate from component definitions and so

multiple components can share the same types of interface and thus be connected together.

In order to make self-adaptive systems easier to engineer we wanted a system where it was possible to change the bindings between components at run time and where we could explore the use of different languages when implementing components rather than just building a superset of a single language like C. None of the systems available met these criteria and so we developed “Tesseract”.

The Tesseract system is complex and a full description is beyond the scope of this paper. It is described in full in [12]. Tesseract is an architectural description language that was built to allow the use of high-level abstract component designs. It uniquely features runtime reconfigurable binding between components, generic parameterised components, multiple inheritance, and a fully componentised runtime environment. The implementations of components are separate from their composition (how they are connected together) and the behaviour of components is defined through the use of interfaces. The compiler design allows for easy retargeting to different hardware platforms or to source-level simulators. The executable code that defines the actual behaviour of components is not written in Tesseract, the Tesseract compiler instead farms out compilation of these parts of each source file to separate “plug-in” compilers. At the current time a plug-in compiler for a simple dialect of C is available.

Fig. 5 shows an example of dynamic Tesseract code. This listing defines a dynamic FIFO buffer that buffers data when the system power level is “LOW” or “MEDIUM” and bypasses the buffer while still allowing it to drain when the system power level is “HIGH”. This could be used as part of an application that transmits sensor data via a high speed, high power, network interface as soon as it has taken each measurement if it has battery power to spare; if battery power is more limited then the sensor measurements are buffered and fed out to a slower, lower power, network interface.

B. Component and Interface Library

In general, all the details of a particular sensor or peripheral device will be encapsulated inside a component. Application programmers need never know the details of how to set a bit level on an I/O port or how to send a character to the serial port.

Where possible common interface definitions are used so that hardware can be changed without affecting the application code. For example, all the digital I/O port components provide the same type of interface. Moving an application from controlling one port to controlling a different one is just a case of selecting which port component is bound to the component holding the application logic. The changes can be even larger, when a component was built to control a particular I²C serial bus linked LCD character display it was given the same (simple, character at a time) interface as the RS-232 component and so an application that had previously been tethered to a PC via a serial cable could become mobile by

```
configuration dynbuffer {
    // Input and output interfaces.
    provides data::stream in;
    requires data::stream out;

    // State interface.
    provides ::powerlevel powerlevel;

    // Instantiate a FIFO with a size of 100.
    component clone network::buffer with {SIZE=100} fifo;

    // The output of the FIFO always connects to the
    // component output.
    bind fifo::out, out;

    // Data flows straight through when power is
    // HIGH, otherwise it flows into the FIFO buffer.
    state powerlevel {
        case HIGH {
            bind in, out;
        }
        case MEDIUM, LOW {
            bind in, fifo::in;
        }
    }
}
```

Fig. 5. Tesseract Dynamic Buffer.

just changing the program so that the LCD component was used in place of the serial component.

C. Wireless Networking

The radio module on a Beastie can be driven manually, one bit at a time, by those wishing to experiment with new radio protocols but most of the time the simple “BNET” (Beastie Net) software component is used to drive it. This allows radio packets of up to 128 bytes in length to be either broadcast or unicast to devices within range of the transmitter. Each device has a unique 16-bit address which is stored in the Beastie’s EEPROM.

Multi-hop protocols have been experimented with successfully, but, for the purposes that the Beasties have been used for so far, a simple single hop system has been found to be adequate since the hop size is large enough to cover an entire floor of a typical office block. In most tests a Beastie with an RS-232 interface card connected to a PC and running in promiscuous mode is used as a network sniffer so that a complete record of all network traffic is available to help with experiment analysis.

The ANS protocol [13] is usually run over the basic network layer. This provides a service-based infrastructure where devices may tender to provide information and facilities to other devices on the network, allowing for self-configuration and management. When an ANS device requires a service it broadcasts a request for that service’s name on the network along with parameters specifying how it would like the service provided (for example, how precise a sensor it requires). Devices that support the requested service use a utility function to determine how close they are to the parameters requested and reply with a single integer which the requesting device may use to select the “best” service provider (where “best” means “at least as good as but no more”). Regular retendering

for services allows the system to adapt to new devices entering the network and devices leaving the network as they fail.

The ANS service structure maps well to component-based design. Individual services can be encapsulated inside components so that an application designer need only include components for the services that they wish to use and never needs to control the raw networking layer directly.

IV. EVALUATION

A. Applications, Installations and Robustness

The ease with which Beasties may be connected to sensors and other hardware coupled with their ease of programming has led them to be used for a variety of small one-off experiments both inside and outside of the laboratory. New sensors can be quickly hooked up and evaluated and prototypes built rapidly.

Beasties have also been used as key parts of several projects. The original ANS project [1] set out to investigate how WSNs could be used to monitor medical patients at home. The network devices were to be installed by medical, rather than computing, personnel and so had to configure and maintain themselves automatically. Beasties were used in this project as demonstrators to show various aspects of the ANS protocol [13]. Beasties fitted with potentiometers and knobs, one Beastie playing the part of an accurate sensor and one the part of a crude sensor, were used along with a PC basestation to demonstrate the ANS's self-configuring and self-optimising properties. The basestation automatically switched to use the best available Beastie depending upon which one was powered on and in range. Later on in the project a small network of Beasties fitted with colour indicator lights along with a network sniffer were used to demonstrate more self-* properties as well as its ability to cope with "state flapping" [14].

After the ANS project Beasties were used in partnership with Arup architectural engineers as part of the Bop! installation in Central Saint Martins College of Art Innovation Centre [5]. The Bop! project used WSNs to noninvasively monitor people performing creative work in an office environment and thus provide a way to evaluate the design of their office environment and how it affected creativity. Beasties were used to measure two types of information. One set were fitted with temperature, light, and multiple-echo ultrasound sensors, the output of which was fed into a modified self-organising feature map, running locally on the Beastie, which allowed it to pick out various states of the room (such as "dark", "light", "in use"). A second set of Beasties monitored thermal people counters to count people moving between various parts of the building. Both of these sets of Beasties sent their data back to basestations for aggregation and further processing.

In use, the Beasties proved extremely reliable and survived the entire four week trial without failure, unlike the other types of commercial WSN device that were deployed at the same time. The advantages of their robust bolt together construction showed during the sensor deployment before the trial was started when one of the ultrasonic Beasties survived

TABLE I
BEASTIE POWER USAGE.

| Component | Mode | Consumption | Symbol |
|------------------------------|-------------------------|-------------|-------------|
| Power regulator Processor | 9/12 V input | 3.5 mA | i_{psu} |
| | Running | 10 mA | i_{run} |
| | Sleeping ¹ | 4.5 mA | i_{sleep} |
| | Power save ² | < 2 μ A | |
| | Power save ³ | 13 μ A | |
| Radio | Transmit | 20 mA | i_{tx} |
| | Receive | 15 mA | i_{rx} |
| FRAM | Read/write | 200 μ A | |
| | Standby | 10 μ A | i_{fram} |
| Status LED | On | 10 mA | i_{led} |

¹ In "ADC noise reduction mode". Can be woken up by an internal timer event or various different interrupts.

² Wakeup by external interrupt only.

³ Wakeup from timer driven by external 32.768 kHz crystal. This external crystal is not part of the current Beastie design but may be easily retrofitted.

being trodden on and suffered only a bent connector on the backplane (which was easily unsoldered and replaced). The main problem with the Beasties was poor battery life, with the battery of the ultrasonic Beasties only lasting 40–45 hours. This was due to the high idle currents of both the Beasties (8 mA) and ultrasonic sensors (3 mA) [15]. The high power supply requirements of the people counters meant that the Beasties monitoring these ran off an external mains supply and so battery life was not a problem for them.

Beasties are currently being used in the DOORS project [6] which focuses on using distributed low-cost sensors for civil engineering measurements, initially measuring the sway of the Queen's Tower in the centre of Imperial's campus when the bells are rung. Their role is to function as high accuracy time servers, coordinating measurements throughout the structure. The DOORS project uses several different types of WSN device, notably Sun SPOTs [16], and the Beasties interface with these. Beasties are being used to provide the time service since this requires high accuracy real-time software and Tesseract is easier to adapt to this task than the Java programming environment of the SPOTs.

B. Power Supply

Table I shows the power consumption of the components of a Beastie. Note that some of the figures are in milliAmps and some are in *micro*Amps.

A Beastie will typically be run in one of two modes: transmit-only or as part of an ad-hoc network. In the transmit-only mode it will sleep for as much time as possible, waking only to take a sensor reading and transmit it. In the ad-hoc mode it listens constantly for service requests from other nodes. We can calculate the average power usage and thus approximate battery lifetime for transmit-only mode using the following formula:

$$I_a = I_q + \frac{t_s I_s + t_m I_m + t_{rx} I_{rx} + t_{tx} I_{tx}}{t_{total}}$$

Where t is the length of time spent in a given mode and I is the power usage in that mode. The suffixes 's', 'm', 'rx' and

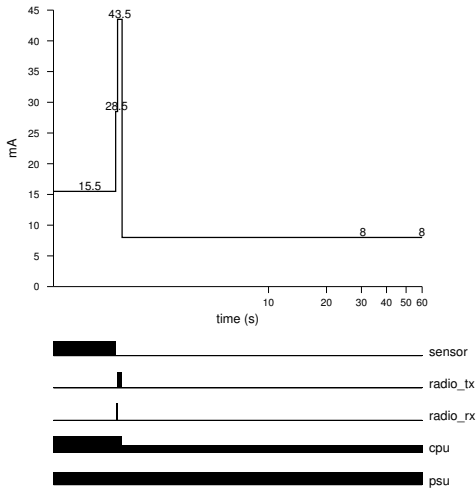


Fig. 6. Transmit-Only Power Consumption.

‘tx’ stand for the sleep, measure, and radio receive and transmit modes respectively. I_q is the quiescent power consumed when the Beastie is doing nothing. Note that even in a transmit-only mode the Beastie must still listen on the radio for a certain amount of time in order to ensure that the airwaves are clear and that no other node is trying to transmit at the same time.

For example, if we assume that a node takes one measurement a minute, using a sensor that takes 2 mA for one second, and then transmits that data straight away in a thirteen byte packet before going back into idle mode we get the power graph shown in Fig. 6. The bars underneath the power graph show when a particular subsystem of the Beastie is turned on. Note that the X-axis is logarithmic.

If we assume that the LED is turned on during radio transmission (this makes debugging easier but increases the transmit power usage by 50%) then the average current consumption of a Beastie operating in this mode is 8.2 mA. A typical alkaline 9V PP3 battery has a capacity of 500–600 mAh and so will last for two and a half to three days. The vast majority of the power is consumed by inefficiencies in the power supply and the processor running in sleep mode when it is doing nothing more than running a timer waiting to take the next measurement. See the future work section (section VI) for a discussion of ways in which power consumption between measurements may be dramatically cut in a modified Beastie so that battery lifetimes can be measured in months rather than days.

In ad-hoc networking mode the Beastie has to listen for packets from other devices constantly. It will be seen from table I that, perhaps unintuitively, it takes almost as much power to listen to the radio as it does to transmit. Therefore, in order to save power the radio routines are designed so that receivers need only have their radios turned on for 25% of the time. The power consumption of a Beastie doing nothing but listening on the network is therefore:

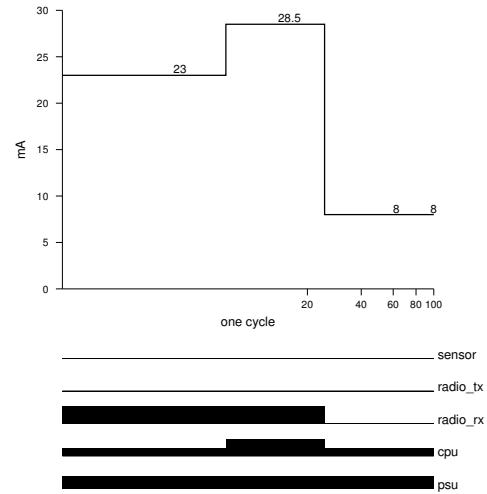


Fig. 7. Ad-hoc Mode Power Consumption.

$$I_a = I_q + T_s I_s + T_{rx} (T_{proc} I_{proc} + T_{radio} I_{radio})$$

T_s and T_{rx} are the proportions of time spent sleeping and receiving (0.75 and 0.25 respectively). T_{proc} is the proportion of the receive time that both the processor and the radio need to be running and T_{radio} is the time when just the radio needs to run while the processor sleeps (it takes a little while for the radio module to stabilise after it is turned on).

The power usage of a listening Beastie is shown in Fig. 7. Its average current consumption is 12.1 mA, giving a PP3 battery lifetime of one and a half to two days. As before, when the Beastie sleep mode power consumption is tamed the lifetime will increase, though not nearly as dramatically as for transmit-only mode since a lot of power will still be used by the radio. This is a fundamental problem of all WSN devices that must listen all the time, not just Beasties, and will only be solved with considerably more complex radio protocols that allow all devices to sleep for a much greater proportion of their run time, ideally more than 99% of the time.

C. Wireless Networking

The NiM2 radio modules on the current Beastie design are easy to use and perform reliably. The decision to use a pre-built radio module in the design of the Beastie makes it more expensive than if the radio system was built onto the main PCB but the simplicity gains were worth it: there was no need to design and evaluate the non-trivial analogue circuitry that accompanies a typical wireless transceiver and no need to produce custom shielding enclosures or use PCBs with more than two layers in order to get adequate shielding.

A range of about 50 m seems to be reliably obtainable in the dense reinforced concrete of a typical office block. Greater ranges are possible in more modern open plan or glass wall partitioned spaces. Outside in good conditions (clear days in the countryside with no obstructions other than a few hedgerows) we have observed ranges of up to 700 m.

Although the radio modules are capable of operating at up to 10kbit/s we run them at 5kbit/s. This is a consequence of the current networking software. In order that one of the analogue inputs of the processor does not have to be permanently allocated to the task of measuring signal strength the detection of whether or not another device is transmitting data is done entirely in software. The radio band is said to be in use when the radio receiver routine is able to decode more than a fixed number of Manchester encoded bits in a continuous row. Unfortunately, a filter built into the output of the radio module means that random network noise appears as a stream of valid bits at 10kbit/s and so this method does not work at that speed.

D. Software Environment

In our experiences, and those of others, it is frequently not processing power that limits sensor node applications but the difficulty of programming them. Programmers are forced to deal with low level details of the devices and sacrifice programming ease or elegance for implementation efficiency [17].

Tesseract has proved to be at least a partial solution to the difficulty of programming these devices. Applications can be developed quickly and easily using pre-existing hardware abstraction components. The component structure allows each subsystem to be tested individually and then assembled into a complete application with a high expectation of reliability.

There are still issues with debugging software since the Tesseract suite does not yet include a source-level debugger. In practice it has been found that the easiest way to develop software is to use a triple row backplane with a Beastie and RS-232 serial interface card bolted together in the bottom two slots and the third slot used for the hardware under development. A software component allows text and numbers to be printed to the serial port (and then to a terminal on the development PC), enabling “debugging by printf” to be used.

A component-based software environment makes another form of software development possible; in “source-level simulation” all of the software components that abstract hardware devices are replaced with ones that instead talk to a simulator (by giving a flag to the compiler that changes which set of system components it uses). The application can then be recompiled to run on a PC instead of a Beastie, allowing for much easier debugging. Once finished, the original hardware components are switched back in and the application is compiled to run on a Beastie again. Development of such a simulator system for Tesseract is about to begin.

E. Construction and Costs

About thirty Beasties have been constructed so far. The design of the Beasties means that when one project finishes they may be reused with different daughterboards in the next project. Having a few Beasties always to hand in the laboratory has proved to be very useful for quick experiments and sensor tests.

The cost of electronic components is changing all the time but at the moment a new Beastie costs about £100 ready assembled, with the radio module making up the bulk of the cost at £40–50. The other electronics components cost about £20 and PCB production and assembly makes up the rest.

V. RELATED WORK

At the time the design of the Beastie was started, in early 2004, the two closest commercially available wireless sensor nodes to our requirements were the Motes developed at Berkeley and the Smart-its from Lancaster University.

The “Mica2” Mote from Berkeley was the closest architecture to a Beastie. It is based around an ATmega128 processor, the same as the Beastie’s but with 128 kB of program memory instead of the Beastie’s 8 kB. It uses a Chipcon CC1000 radio chip to provide a network interface of similar speed to a Beastie in a variety of ISM bands, 433 MHz like a Beastie as well as the 868 MHz/915 MHz bands. Motes are built using surface mount components and connect to peripheral board via a small high density connector.

The DIY Smart-ITS are much closer to the Beasties in philosophy. They use through-hole components which are much lower density than the components used on a Mote but which are much easier to work with without special equipment. This makes the devices far easier to modify and experiment with. Smart-ITS use a “BiM2” radio module, the direct ancestor of the NiM2 radio module used on a Beastie and essentially identical in use apart from having a lower range due to an older receiver design. They use a Microchip PIC16876 microcontroller. These are an older design of microcontroller and have an architecture that is not as easy to write compilers for as the Atmel processors, with few registers and a highly segmented memory map.

The power supply on the current Beasties is one of the more problematic components in terms of power wastage, although we feel that the advantages of easy access to a smooth 5 V supply for all peripherals outweighs the disadvantages in a research tool. Both the Smart-ITS and the Berkeley Motes sidestep this problem by using variable voltage independent components and no regulator at all (although the Smart-ITS do have an onboard regulator for use with an external power supply). The system voltage on these devices starts off at about 3 V when the batteries are fully charged and gradually reduces as the device is used. Any peripherals that require a stable voltage have to incorporate their own voltage regulators.

The Berkeley Motes have evolved since the Beastie was designed. The major changes apparent in the current “Telos” Motes [18] are the use of a different processor (a Texas Instruments MSP430 16-bit microprocessor) and the move to an 802.15.4 compliant radio (operating at 2.4 GHz). While the new radio module is smaller and very low power (and the high frequency allows a smaller aerial which can be incorporated onto the circuit board) the higher radio frequency does not penetrate buildings as well and so multiple nodes and multi-hop radio protocols are required to cover the same distances that Beasties can manage in a single hop when used in an urban

environment. As noted in section IV-D, WSN applications are still limited by software complexity rather than processor speed and so the Beastie's processor remains competitive.

Both Motes and Smart-ITS rely on the strength of their peripheral board connectors to hold the boards onto the main device. This is fine on a workbench but experience has shown that this is an unreliable connection method when devices are deployed since it does not take much of a shock to separate the connectors. Even partial separation can mean greatly increased noise on an analogue input. Beasties are designed to bolt together and into a housing (or a shock-proof mount) and so do not suffer from these problems.

All of the modern WSN designs use surface mount components and emphasize small size over ease of modification. In our experience, the ability to modify a Beastie is worth the extra size.

VI. FUTURE WORK AND DEVELOPMENT OF THE BEASTIE

The Beastie works well as a lab research tool but currently consumes too much power which limits its usefulness for long deployments.

As noted in a previous section, the power supply on the current Beastie is a linear supply and was designed for simplicity and zero chance of radio interference rather than efficiency. This means that it has a high quiescent, no load, current: power that is simply wasted. A move to a switching DC-DC convertor power supply would have two advantages. Firstly, the quiescent power consumption would be dramatically reduced, and secondly, the input power supply voltage could be dropped down to the 2.5–3 V range by using a step-up (“boost”) convertor. This would allow the use of cheaper and more power-dense batteries such as the AA and AAA sizes.

Initial experiments with a Maxim MAX619 step-up convertor running from a 3 V supply show a quiescent current of only 75 μ A, down from the 3.5 mA of the current design and almost four orders of magnitude less power consumption!

The other main drain on the Beastie is the processor's sleep mode which takes 4.5 mA even if all the processor is doing is running an internal clock so that it knows when to wake up. If an external 32.768 kHz crystal (as found inside a typical wristwatch) is added then the processor can use that to run a wakeup timer and shut down into a much lower power “power save” mode (as shown in table I). This modification has been made to an existing Beastie and works well.

With an external crystal and a better power supply a Beastie could potentially run for one to two months off a PP3 battery of the type used at the moment or six to seven months off an AA size battery pair.

Even more power reductions can be made if the processor on the Beastie is replaced with the recently released Atmel ATmega88P processor. This processor, part of Atmel's “picoPower” range, is pin compatible with the current processor and is designed to be much more power efficient. The clock speed can be changed in software and any unused sections of the chip turned off to reduce power consumption. These later designs of processor also allow on-chip debugging via the

proprietary “debugWIRE” interface, which will considerably ease the task of software development. Tests with Beasties fitted with samples of this new processor are currently taking place.

VII. CONCLUSIONS

This paper has introduced the Beastie wireless sensor node which has been used very successfully in a diverse range of research applications. We have observed that our architecture has improved our idea-to-prototype and “what-if” experimentation time considerably for us and for some of our industrial research partners. We critiqued the current Beastie configuration and have indicated how we will proceed to upgrade it in the near future.

REFERENCES

- [1] J. A. McCann, A. Hoskins, and G. Jawaheer, “ANS (autonomic networked system) for ubiquitous computing,” in *UbiComp 2004 Adjunct Proceedings: Posters*. Berlin, Germany: Springer, Sep. 2004, extended abstract for poster.
- [2] *MICA2 Wireless Measurement System*, A ed., Crossbow Technology, Inc.
- [3] M. Strohbach, “The Smart-Its platform for embedded context-aware systems,” in *Proceedings of the First International Workshop on Wearable and Implantable Body Sensor Networks*, Apr. 2004.
- [4] “BASIC Stamp microcontroller,” Web page, Parallax, Inc., 2008, <http://www.parallax.com/>. Accessed 2008-5-16.
- [5] OnOffice, “What workers want,” *OnOffice Magazine*, vol. 2008, no. 3, Mar. 2008.
- [6] “Doors: Gateways and crossroads for pervasive environmental monitoring,” Web page, Centre for Pervasive Sensing, Imperial College London, 2008, <http://www3.imperial.ac.uk/pervasivesensing/projects/doors/>. Accessed 2008-5-16.
- [7] *NiM2 Transceiver Datasheet*, Radiometrix Ltd., Hartcran House, 231 Kenton Lane, Harrow, Middlesex HA3 8RP, United Kingdom, May 2007.
- [8] P. Ali-Rantala, L. Sydänheimo, M. Keskilammi, and M. Kivikoski, “Indoor propagation comparison between 2.45GHz and 433MHz transmissions,” in *Antennas and Propagation Society International Symposium*, vol. 1. IEEE, Jun. 2002, pp. 240–243.
- [9] “The GNU debugger for AVR,” web page, accessed 2007-9-18. [Online]. Available: <http://home.overta.ru/users/denisc/>
- [10] Q. Wang, Y. Zhu, and L. Cheng, “Reprogramming wireless sensor networks: Challenges and approaches,” *IEEE Network*, vol. 20, no. 3, pp. 45–55, May 2006.
- [11] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, 2nd ed. Harlow, UK: Pearson Education, 2002.
- [12] A. Hoskins, *Tessera Reference Manual*, Web page, 2008, <http://www.doc.ic.ac.uk/~asher/ubi/tesserae/>. Accessed 2008-7-23.
- [13] G. Jawaheer and A. Hoskins, “ANS design document,” Imperial College London, ANS Project Deliverable 7, Nov. 2004.
- [14] M. Breza, R. Anthony, and J. McCann, “BIOANS: Bio-inspired ambient intelligence protocol for wireless sensor networks,” *Ubiquitous Computing and Communications Journal*, Oct. 2007, special issue on ubiquitous sensor networks.
- [15] *SRF08 Ultra Sonic Range Finder Technical Specification*, Web page, Devantech Ltd (Robot Electronics), Unit 2B Gilray Road, Diss, Norfolk, IP22 4EU, England, <http://www.robot-electronics.co.uk/html/srf08tech.shtml>. Accessed 2008-7-22.
- [16] “Project Sun SPOT,” Web page, Sun Microsystems, Inc., 2008, <http://www.sunspotworld.com/>. Accessed 2008-5-16.
- [17] R. Sugihara and R. K. Gupta, “Programming models for sensor networks: A survey,” *Transactions on Sensor Networks*, vol. 4, no. 2, pp. 8:1–8:29, Mar. 2008.
- [18] J. Polastre, R. Szewczyk, and D. Culler, “Telos: Enabling ultra-low power wireless research,” in *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*. Piscataway, NJ, USA: IEEE, 2005.