Generalised Interfacing with Microprocessor System Hardware Monitors

Constantine A. Murenin
Department of Computer Science, East Carolina University, Greenville, NC, USA
Faculty of Computing Sciences and Engineering, De Montfort University, Leicester, UK

Abstract

In this paper, we will discuss the possibilities, functionalities and limitations of Microprocessor System Hardware Monitors. We will provide a comparison survey of their main functions, depict some weaknesses and give an overview of what possibilities they provide for the end-user.

We will also discuss OpenBSD's sysctl Hardware Sensors framework, and describe our recent contributions to the addressing scheme and underpinnings of the framework.

1. Background

Power consumption of personal computers grows every year along with their computational capabilities. Recently produced microprocessors consume at times hundreds watts of power, and thus require massive cooling resources. Most often than not, *fans* are used as a low-cost solution that provides adequate cooling in low- and medium-cost personal computers, whilst water-cooling is used in high-end systems. However, fans have several problems associated with them, such as *noise* when run at full speed and, once again, they themselves consume more electricity.

Processor manufacturers have come to realise that power consumption of modern processors is a major drawback from creating high-end personal systems with acceptable power-drawn, so several *power-saving* technologies were deployed, such as deactivating some microprocessor units or modules whilst the processor is in the idle loop. The adoption of these features means that *fans may be turned off while the overall system activity is low, and turned to full speed when the system performs some processor-intensive task*.

It is in the interest of the user to be able to *control the speed at which fans are running* in order to utilise the power-saving features of the processor and *minimise the overall noise* of the system. Practically, this may be proven extremely useful in situations where one is running an OpenBSD-based firewall or fileserver system in the home setting, where the overall processor activity of the server is going to be very minimal, whereas the noise of the system is of a major concern due to the desired 24hr operation of the server within the living room area. Desktop usage of hardware monitoring and controlling software is also of value, since the amount of processor idling time prevails in most desktop applications and minimised noise is of particular value. In the server arena, the monitoring part comes very handy, such that administrators can be sure that their systems are running in accordance with manufacturers' specifications and guidelines.

Vast majority of motherboards in use today already integrate chips that allow temperature, voltage and fan speed monitoring, and many newer motherboards have the circuitry for supplying fans with user-specified voltage so that the user can control fan speed entirely from software. However, hardware manufacturers do not always provide any end-user utilities to do the aforementioned monitoring and controlling, and when it comes to open-source software, the community is expected to come up with a software solution on their own. This works well as long as chip manufacturers release appropriate register documentation for their hardware. In the presence of such documentation, qualified software developers can write drivers for the operating system of their choice, and these drivers are usually shortly integrated into the official distribution of the operating system for which they were written.

2. Hardware review

Hardware monitors usually come either in a distinct chip package or a as a part of a Super I/O solution. Sensor readings and fan-controlling capabilities may be accessed through different busses, mostly I^2C or SMBus, ISA or LPC, or PCI.

I^2C and SMBus are two-wire busses, which are essentially compatible with each other. I^2C is an acronym for Inter Integrated Circuit, a bus developed by Philips Semiconductor with version 1.0 published in 1992 [1], whilst SMBus is an acronym for System Management Bus, a bus defined by Intel in 1995.

LPC stands for Low Pin Count Interface Specification, which is a bus authored by Intel that is hardware and software compatible with the existing ISA devices and applications. [2]

Popular hardware monitoring chips often feature both I^2C and ISA bus access, for example National Semiconductor LM78 [3] or Winbond W83627HF [4]. In such case, accessing sensors over ISA bus is preferred over I^2C, with ISA being the faster of the two. From the practical standpoint, however, unless special care is taken, two instances of the same device may be attached to the system at once — one on I^2C, the other one on ISA. Whilst this usually does not create any resource conflicts, implementations might detach the instance of the driver from the I^2C bus in favour of ISA bus access to avoid any user confusion of two distinct devices being attached with identical sensor readings. (In fact, this is what OpenBSD's

lm(4) driver does — when the hardware monitoring chip is detected on the ISA bus, it is automatically attached on ISA and detached from the I^2C bus.)

It is not less common to find hardware monitors that only have I^2C interface — this is usually the case with simple chips that only have one or two temperature sensors. On the other hand, Super I/O chipsets usually offer ISA/LPC interface as they already use ISA/LPC for the purpose of connecting serial, parallel, infrared and PS/2 ports.

Hardware monitors that are accessible through PCI interface are usually enclosed in South Bridges, such as VIA VT82C686A "Super South" South Bridge. At this time, only VIA is known to have produced this kind of "Super South" South Bridge with system-wide hardware monitoring capabilities.

2.1. Hardware documentation

In order to write an effective driver for a hardware monitoring chip, or any chip for that matter, the manufacturer of the chip has to release appropriate documentation that would summarise main functions of the chip and provide some insightful information on how the chip can be talked-to to perform desired operations.

Unfortunately, in recent years in the computer industry it became very difficult to acquire appropriate documentation that is required for programming the chips. According to Theo de Raadt, the founder and main software-architect of the OpenBSD project, it is the large American companies which usually resist from giving away their documentation that is required for programming the chips. [5] Datasheet unavailability is a major obstacle for open-source software, because it means that driver developers have to reverse-engineer the devices and their interfaces in order to create working drivers.

Fortunately, most popular hardware monitoring chip manufacturers do provide the datasheets for programming their hardware. However, in the next subsection, we will discuss some problems that driver developers may have to face when interpreting provided documentation.

2.2. Datasheet shortcomings

In this section, we will provide a brief survey of common problems that may be encountered by driver developers in writing drivers for most popular hardware monitoring chips. Some problems mentioned in this section can equally apply to the end user as well, for example, what expectations should the user have for hardware monitors in general?

2.2.1. Voltage Input Ambiguity

Manufacturers of most popular hardware monitoring chips only provide recommendations of which leads should be connected to which voltage sensor inputs of the hardware monitoring chip. In other words, they do not enforce any particular connections for Voltage Sensor 'VIN' connectors in their documentation.

From the point of view of the motherboard manufacturer, this is an excellent opportunity to use provided sensors in the most efficient way. For example, the motherboard manufacturer of a server may wish to put several hardware monitoring chips inside their motherboard, and utilise their voltage sensor inputs in such a way as to provide the maximum flexibility in monitoring, say, dual power-supplies that provide uninterrupted power to the motherboard.

However, when one thinks more about this situation, one realises that most power supplies in the desktop market today are standard ATX power supplies, which have the very same connectors from motherboard to motherboard. They also come in standard voltage configurations, for example, they all have a +3.3V lead, +5V regular lead, +5V lead for 'stand-by' use, several +12V leads etc. [6] I.e. the hardware monitoring chip manufacturer should already have a pretty good idea of where their Super I/O chips will be used in the vast majority of cases.

Nevertheless, because chip manufacturers still provide only *recommendations* of which voltages should be connected to which voltage sensor inputs, we have a problem to tackle. The problem cannot be seen until you realise how voltage sensors in the monitoring chip work.

Voltage sensors work as follows: each sensor has a limit of the maximum voltage that it can detect and the maximum voltage it can withstand. Most older chips have the maximum voltage detection level at around 4.0 volts, whilst many newer chips have it at around 2.0 volts. This maximum voltage detection level will only decrease over time, as the semiconductor industry manufactures integrated circuits using a smaller and smaller chip manufacturing process year after year, which in turn makes chips consume less electricity, and moreover makes it impossible to transfer high voltage over such small electrical parts inside the chip.

Therefore, before the voltage can be connected to the monitoring chip, it must be downscaled to no more than the specified maximum detection level. This is done with the help of resistors.

After the voltage is transformed to the appropriate scale and is connected to the sensor, the sensor usually has 8-bit, sometimes 10-bit, scale under which it can report the voltage it senses.

In other words, the detected value from the sensor is usually stored in one byte, and assumptions are made on which resistors were used in order to bring the voltage to the desired level. The resistor factors can be thought of as the conversion table, and because the *chip* manufacturer, essentially, does not require the *motherboard* manufacturer to connect specific leads from the power supply to specific sensor inputs of the monitoring chip using specific resistors — there is no guarantee that the value we think is +12V is indeed connected to the +12V line of the power supply. To illustrate this in a visual example, let us consider that we read an 8-bit value of 0xcb from a voltage sensor on a very common Winbond Super I/O chip — W83627HF:

code	result	description
0xcb	203	raw value
203 * 0.016V	3.24V	voltage on sensor
3.24V * 1.00	3.24V	+3.3V scale
3.24V * 1.68	5.44V	+5V scale
3.24V * 3.80	12.31V	+12V scale

Sensors on the chip in this example only allow for a maximum of around 4.0 volts to be detected. To account for

this, if one wants to measure +12V, one must put some resistors in place that will safely and surely lower the voltage to being on around 3.0 — 3.3 volt scale.

To summarise, it is up to the manufacturer of the end-product (e.g. motherboard) to ensure that appropriate resistors are used and the leads are connected as recommended. In turn, when software reads a voltage data from the chip, it multiplies the value by artificial resistor factor to align the raw value against the scale. As a result of these multiplications, we usually end up with the value that we *already expect* from that particular sensor, i.e. 3.24V, 5.44V or 12.31V reading for a 3.33V, 5V or 12V sensor. Unfortunately, there is no way to verify that these readings are indeed what we think they are — the device driver has no way of knowing for sure that the end-product manufacturer didn't switch 5V and 12V circuitry, so if your 12V reading is off, it's not necessarily that your +12V line of power supply needs some attention, it might be the case that you think that it's a +12V line, whereas in fact it is a +5V or +3.33V line.

2.2.2. LM78 compliance

Unnecessary backwards compliance with LM78-chips was implemented in many Winbond Super I/O chips, which made Winbond chips look quite dodgy from the software implementation standpoint.

Temperature

One obvious example of LM78 compliance is the temperature reading register. In LM78 there is only one temperature reading sensor that can be accessed at register 0x27, and registers 'before' and 'after' are used for voltage and fan sensors respectfully. Winbond wanted to keep the compliance, and thus decided to put the second and third temperature readings in register 0x50 'bank 1' and 'bank 2'.

Fan divisor bits

A better example of the shortcomings of legacy compliance comes with fan divisors. Remember, we usually have only 1 byte for the raw sensor data. With fans varying in speed from 200 rpm to 8000 rpm, we must detect a broad range of values. To accomplish this, LM78 stores two-bit divisors for two out of three fans, with the third fan having a constant divisor.

However, engineers of W83627HF decided that they wanted to have 3 bits for divisors and adjustable divisors for all three fans (not just for two as in LM78), but still keep the compatibility with LM78 implementations. Therefore, in W83627HF, the most significant bit for each divisor is stored in a register entirely different from the remaining two least significant bits for the two legacy divisors. This, in turn, creates additional implementation challenges by requiring the device driver to scan several entirely different registers for divisor bits, and then to assemble these divisor bits into a single divisor for each fan.

The problem with fan divisors splattered across multiple registers is amplified further by the fact that hardware monitoring chips do not automatically modify fan divisors to accommodate slower fans, i.e. unless divisor bits are modified by software, no fans with speeds slower than 2657 RPM may be detected with the default divisor — 2^1.

Voltage

It must be noted that this backwards compatibility with LM78 is rendered even more useless due to Winbond's recommendation to use different resistors with W83627HF than the ones that were recommended by National Semiconductor for use with LM78. (E.g.: +12V line suggested resistors in LM78 are 30 and 10 kilo ohm, which make up a (30+10)/10 = 4.0 factor over the input, whereas W83627HF recommended resistors are 28 and 10 kilo ohm, which make up a (28+10)/10 = 3.8 factor over input.) Therefore, whilst the logical meaning of the registers is compatible, i.e. register 0x24 provides the reading for +12V in both LM78 and W83627HF, their actual values unnecessarily differ due to the different resistors being proposed and utilised.

2.2.3. 0—4.096V paradox

Many Winbond datasheets talk about 4.096 V being the maximum detectable voltage, with the excess to be removed by resistors. Quote:

> *The maximum input voltage of the analog pin is 4.096V because the 8-bit ADC has a 16mv LSB. Really, the application of the PC monitoring would most often be connected to power suppliers.*
>
> Winbond datasheets, original spelling preserved

Specifically, they say that sensors can measure from 0V to 4.096V with 8-bit raw data being encoded in 0.016V units. However, it remains to be seen how 4.096V could be measured in one byte using 16mV LSB:

```
  0 * 0.016 V = 0.000 V
...
255 * 0.016 V = 4.080 V
```

2.3. Light at the end of the tunnel

It must be noted that not every chip is accompanied by datasheets and recommendations that are such unaccountable for development of generalised software with hardware monitoring support that can be trusted. One exemplar is Standard Microsystems Corporation (SMSC) SCH5017 Super I/O chip, which is described to have internal resistors, such that it automatically scales sensed voltage in such a way that the correct value refers to $3/4^{th}$ of the scale or 192 decimal in 8-bit reading (SCH5017, page 200 out of 362). [7] This allows the device driver to have a simple register-voltage relationship table for conversion purposes, and the very same table can also be used for descriptive purposes, leaving very little room for any kind of error.

3. Software review

In this section, we will give a brief overview of software packages that exist for the purpose of monitoring and/or controlling various environmental characteristics of a computer. Where possible, we will compare the effectiveness of the software against the OpenBSD's sysctl(8) approach at hardware monitoring.

3.1. SpeedFan

SpeedFan is a closed-source freeware win32 software developed by one individual in Italy. It provides support for most hardware monitoring devices, and it can access them

on PCI, ISA and SMBus buses. (As mentioned previously, LCP-devices are software compatible with ISA, and I^2C and SMBus are also essentially compatible with each other.)

Other than providing access to the standard hardware monitors, SpeedFan also allows one to access the information about Hard Disc Drives that is available via S.M.A.R.T., the Self-Monitoring, Analysis and Reporting Technology.

Of particular interest about this software is its homepage, which includes a compiled list of various hardware monitoring chips that the software supports, and provides a visual comparison of their major functions. It also includes specific descriptions of most chips, which may include comments in regards to the features and documentation availability. [8]

3.2. lm_sensors

lm_sensors is a popular GPL-licensed package that could only be used with the Linux kernel, such as in the GNU/Linux (GNU's Not Unix) environment. The package has a fair amount of modularisation, and some parts of the suite are standardised between drivers to allow for an efficient functioning of several generalised end-user utilities.

3.3. healthd

healthd, an open-source BSD-licensed package for FreeBSD, features a very complex all-in-one solution that consists of two utilities: a *daemon healthd* and a *client healthdc*.

In turn, healthd performs three entirely distinct functions: it talks directly with the ISA or SMB buses to detect and query hardware monitors; it performs comparison of the previously read values from the sensors with the required range as specified in the configuration file and reports any abnormalities to syslogd(8); and it listens on the IANA-assigned port number 1281 for requests via the healthd protocol.

The healthdc client allows one to remotely connect to the healthd daemon, and query the daemon for information about the sensors.

As it is clear from the description, this healthd daemon is overwhelmed with complexity, and does not follow the UNIX approach of making things as simple, flexible and abstract as they could possibly be made. Specifically, it does not seem to provide any abstractions for writing drivers for hardware monitors, and all definitions of various chipsets are located in a single set of files.

healthd package has no support for any kind of fan controlling, and it also does not modify the divisor bits, thus on many systems with default fan divisors the minimum detectable fan-speed would be 2657 RPM.

3.4. xmbmon

xmbmon is another package that provides interfacing with hardware monitoring chips. The package consists of a command-line utility mbmon and an X-client xmbmon. This package is noteworthy because it supports many operating systems at once — the webpage of the utility claims that FreeBSD, NetBSD, OpenBSD and Linux are all supported. Other noteworthy characteristic of the package is its support for numerous hardware monitoring chips, and a certain level of abstraction between drivers. Although this package does not support fan-controlling, it is quite ahead of other similar utilities for FreeBSD as far as monitoring features are concerned. E.g. unlike healthd, mbmon modifies fan divisors on many chips, and thus mbmon can detect slower fans that go undetected with healthd.

3.5. NetBSD's sysmon(4)

NetBSD is the most portable operating system in the world that runs on a variety of devices, even on toasters. From production servers in NASA to toasting fresh bread in your personal kitchen, NetBSD provides a flexible approach in developing an operating system for any kind of environment, embedded or not. [9]

Speaking of toasters, there are reports that the TS-7200 based toaster uses sysctl(8), and not toastctl(8) as one might imagine, in order to access various information about readiness of a toast, and in order to control toasting preferences, such as the toast burn level. [10]

Nonetheless, in spite of the flexibility and the possible simplicity that is clearly associated with the sysctl(8) interfacing, the general system hardware monitoring interface in NetBSD was not done with sysctl(8), but a distinct API infrastructure was created for monitoring and controlling hardware monitoring chips. The infrastructure itself consists of a device driver, /dev/sysmon, which provides an abstraction layer between the end-user utilities and the actual hardware monitoring device drivers, such as lm(4).

NetBSD already contains many hardware monitoring device drivers that interface with sysmon(4)/envsys(4), most notable examples being lm(4) and viaenv(4). An utility called envstat(8) is provided in /usr/sbin/envstat for querying /dev/sysmon.

It must be noted, however, that the envsys(4) API is considered experimental, and the entire API shall be replaced by a sysctl(8) interface, should one be developed. [11]

3.6. OpenBSD's sensors.h

OpenBSD's sensors.h is the infrastructure that we are going to describe in greater detail; moreover, we will provide some information on how it was improved and what new functionality was added as a part of this project.

Originally, the framework was ported from the aforementioned NetBSD framework by Alexander Yurchenko, but instead of porting the sysmon(4) / envsys(4) interfacing, a much simpler infrastructure was created that 'just works'.

3.6.1. sensors.h

sys/sensors.h part of the framework defines the data-structures that are used by sensors, and provides function prototypes which are used by device drivers for adding and deleting sensors, and by sysctl(3) in order to provide requested sensors to the userland. (Prototypes of functions

that facilitate task scheduling for refreshment of the sensed data by the device driver are also available in sensors.h; however, we will not discuss task scheduling in this paper.)

3.6.2. kern_sensors.c

At the start of this project, drivers were adding sensors using some simple macros that were defined in the header file itself; however, this has since changed, and the definitions of functions that are used in adding and removing sensors are now located in kern/kern_sensors.c.

3.6.3. sysctl(3)

kern/kern_sysctl.c implements the kernel part of the userspace API that is used by programmes like sensorsd(8) and sysctl(8). A tree called 'hw.sensors' is dedicated entirely for accessing the information from the sensors.

3.6.4. sysctl(8)

sbin/sysctl/sysctl.c provides the end-user interface for general system controlling. It is also the preferred utility for displaying the information about sensors, although with sysctl(3) API anyone can write their own utility for accessing this information directly from the kernel, and displaying it in the way they please.

Before the results of the research accompanying this paper were integrated into OpenBSD, the information about sensors was displayed in a way similar to the following:

```
tvc:constant {3205} sysctl hw.sensors.0
hw.sensors.0=lm0, CPU VCore, 1.68 V DC
```

In this example, one can see that we ask the system for the sensor numbered zero to be presented. In the output, the sensor is identified as being attached to the lm0 device, the description of the sensor is 'CPU VCore', and the last updated output of the sensor is 1.68 volts DC.

3.6.5. sensorsd(8)

usr.sbin/sensorsd/sensorsd.c allows the user to specify limits for particular sensors in the configuration file /etc/sensorsd.conf, and when sensorsd is run, it queries sensors using sysctl(3), and compares gathered values with the limits specified by the user. If the values go out of range, then sensorsd issues a warning into syslog(3), and may also execute a user-specified command (i.e. it can execute a command that may email, page, provide a visual or audible alert to the user).

4. Design

In this section, we will try to analyse the possibilities for updating OpenBSD's sensors framework. The primary aim of the discussion is focused on making it easier for the end-user to query specific sensors on specific sensor chips. The laid foundation has the purpose of building sound grounds for later additions of functions such as controlling of fan speeds.

In modifying the framework, we must take into account the massive number of devices in OpenBSD that are capable of attaching sensors: they range from standard temperature, voltage and fan monitoring chips, to complex interfaces, such as ipmi(4), to SCSI enclosures.

4.1. Improving in steps

When we have first contacted Theo de Raadt (the lead architect and the founder of the OpenBSD project) with the ideas about improving the infrastructure, his reaction was to suggest that our proposed ideas are at first implemented without adding any fan-controlling functionality. For example, we have proposed to effectively change the way in which sensors of a device are accessed from 'hw.sensors.0' style of addressing to 'hw.sensors.lm0.volt0'.

In other words, it was agreed that changing the speed on the third fan by a command similar to 'sysctl hw.sensors.12=50%', and then monitoring the temperature it affects by 'sysctl hw.sensors.9' is not going to be very effective and flexible, so a change had to be made to the way sensors are addressed. Therefore, the first step was to implement the new addressing, and only then think about implementing the functionality for allowing any kind of user-specified control of the chip to take place.

4.2. How sysctl(8) works

Let us briefly explain of what happens you run sysctl(8) to access the hw.sensors tree.

When you run 'sysctl hw.sensors', the sysctl(8) converts the 'hw' and 'sensors' strings into their numerical representation, for example, 'hw' turns out to be number 6 (as defined by the CTL_HW pre-processor constant), and 'sensors' turns out to be number 11 (HW_SENSORS constant). These numbers form what is known as Management Information Base (MIB), an array of integer values used for making various sysctl(3) calls.

After sysctl(8) determines that a call for hardware sensors is being made, it transfers control to a local function called sysctl_sensors(), which then handles some further processing of the string into the MIB elements.

When the turn finally comes to the final leaf, a sysctl(3) kernel call is made to gather the latest snapshot of sensor reading, and the results are interpreted and printed to the user.

4.3. What we have proposed and implemented

We have proposed to have a different tree structure under 'sysctl hw.sensors'. For example, the one that would allow the end-user to access the temperature of the CPU on the lm0 chip by typing 'sysctl hw.sensors.lm0.temp1', instead of 'sysctl hw.sensors.8' as it was before.

In turn, sysctl(3) MIB array will look as follows for providing this information:

```
CTL_HW
HW_SENSORS
(device number)
(sensor type)
(sensor number of this type on this
instance of the device)
```

As a comparison, before this change was made, the MIB array for accessing hardware sensors looked as follows:

```
CTL_HW
HW_SENSORS
(sensor number)
```

Obviously, device identifiers in our sysctl(8) examples are literals, but sysctl(3) may only operate on integer nodes, thus we must have a way to map literal names (i.e. strings such as 'lm0') onto integer numbers. For this purpose, we have made it possible to access generic information about hardware monitoring *sensor devices*, by omitting the sensor type and sensor number arguments from the MIB array. This will allow userland programmes, such as sysctl(8) and sensorsd(8), to map literal names of sensor devices to integer numbers.

Now that we know how we want to access these sensors, we must think of the way they will be stored and registered from the device-driver point-of-view. Here, we have many different options, which all vary by their performance, flexibility and complexity. The obvious thing is to have every sensor attached specifically to some driver, and not to the common linked list as it was done previously. For the list of drivers themselves, a linked list could be used in the same way as it was previously used for the individual sensors.

Due to length limitations of these conference proceedings, we will omit the comparison between various ways of handling sensor management in kernelspace, which we have discussed extensively in our B. Sc. (Honours) Final Year Project's thesis at De Montfort University [12]. For more information on the actual implementation that we have realised and which was accepted by the OpenBSD Project, the reader is advised to refer to the sensor_attach(9) manual page [13], an OpenBSD Journal article about the new two-level sensors framework [14] and OpenBSD's source code starting from 2006-12-23.

5. Conclusion and future plans

In this paper, we have attempted to present detailed information on how microprocessor system hardware monitors work from the programmers' standpoint. We have researched into the problems that the device-driver developers may have to face with in implementing support for such monitors, along with the expectation that the user should have when using such drivers.

We have also provided an overview of major programmes that can be used to monitor sensor activity on a personal workstation. The situation with programmes on the *BSD arena was explored throughout and OpenBSD's sysctl(8) approach was studied and explained in detail.

In tandem with this paper, a patch implementing the two-level addressing interface for OpenBSD's sysctl hardware sensors was released and integrated into OpenBSD's CVS tree on 2006-12-23. As a part of this patch, 44 existing in-kernel hardware monitoring device drivers were converted from the old one-level API to the new two-level API. Most importantly, the patch converted user utilities sysctl(8), sensorsd(8) and ntpd(8) to the new two-level addressing, and a patch for converting symon system monitoring utility was also released and appropriately integrated.

A prototype of fan controlling functionality for hardware monitoring chips was developed and successfully presented on the B.Sc. (Hons) Final Year Project Viva at De Montfort University, but not yet made available to the general public. Future work will include developing a generalised scheme for utilising existing sensors API to accomplish an effective fan-speed controlling interface via the sysctl(8) mechanism.

Acknowledgements

Most parts of this paper were written as a part of the B. Sc. (Hons) Final Year Computing Project supervised by *Dr. Jordan Dimitrov* and *Prof. Mikhail Goman* at De Montfort University, where the author was engaged in a one-year exchange from East Carolina University, but was awarded an unplanned Bachelor of Science (Honours) in Computer Science diploma nonetheless.

We would like to thank everyone who made this paper and our experiences in England possible and enjoyable, and specifically *Thomas W. Rivers Foreign Exchange Endowment Fund* for providing us with a scholarship award that in part helped fund this study abroad experience.

Special thanks also go to *Theo de Raadt*, *Alexander Yurchenko* and *David Gwynne* of the OpenBSD project.

References

[1] The I²C-bus specification, Version 2.1, January 2000, *Philips Semiconductor*, http://www.semiconductors.philips.com/acrobat_download/literature/9398/39340011.pdf

[2] Low Pin Count (LPC) Interface Specification, *Intel Industry Specification*, http://www.intel.com/design/chipsets/industry/lpc.htm

[3] LM78 Microprocessor System Hardware Monitor, February 2002, *National Semiconductor*, http://cache.national.com/ds/LM/LM78.pdf

[4] W83627HF/F Winbond I/O, November 2002, *Winbond Electronics*, http://www.winbond.com/PDF/sheet/w83627hf.pdf

[5] Interview with Theo de Raadt, 2 May 2006, *KernelTrap*, http://kerneltrap.org/node/6550

[6] ATX Specification 2.2, 2004, *Intel Corporation*, http://www.formfactors.org/developer/specs/atx2_2.pdf

[7] Datasheet for SCH5017, 2005-10-17, *Standard Microsystems Corporation (SMSC)*, http://www.smsc.com/main/datasheets/5017.pdf

[8] SpeedFan — access temperature sensors in your computer, 2000/2006, *Alfredo Milani Comparetti*, http://www.almico.com/speedfan.php

[9] NetBSD Controlled Toaster, 2005-10, *Technologic Systems*, http://www.embeddedarm.com/news/netbsd_toaster.htm

[10] Christian von Kleist, "The NetBSD Toaster", *Slashdot*, 2005-08-11, http://hardware.slashdot.org/comments.pl?sid=158747&cid=13298702

[11] Tim Rightnour and Bill Squier, "envsys(4) — environmental systems API manual page", *The NetBSD Foundation*, 2000/2006, http://man.netbsd.org/cgi-bin/man-cgi?envsys+4

[12] Constantine A. Murenin, B. Sc. (Hons) Final Year Project Main Report: "Microprocessor system hardware monitors. Interfacing on OpenBSD with sysctl(8).", *Faculty of Computing Sciences and Engineering, De Montfort University, Leicester, UK*, May 2006.

[13] Constantine A. Murenin and Michael Knudsen, "sensor_attach(9) — sensors framework API manual page", *The OpenBSD Project*, December 2006, http://www.openbsd.org/cgi-bin/man.cgi?query=sensor_attach&sektion=9&manpath=OpenBSD+4.1

[14] Constantine A. Murenin, "New two-level sensor API", *The OpenBSD Journal*, December 2006, http://undeadly.org/cgi?action=article&sid=20061230235005