School of Electronics and Computer Science
Faculty of Engineering, Science and Mathematics
UNIVERSITY OF SOUTHAMPTON

Robert D. Spanton
May 10, 2007

Wireless Sensor Network Platform for Autonomous Experimentation

Project supervisor: Klaus-Peter Zauner
Second examiner: Prof. James S. Wilkinson

A project report submitted for the award of MEng Electronic Engineering

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

by Robert D. Spanton

Developments in wireless sensor networks (WSNs) are strongly focused on minimizing the power requirements of sensor nodes. As sensors with significantly large resource requirements are introduced into WSN systems, algorithms must be developed to maximise the information returned per sensor reading. The integration of "lab on a chip" sensors into WSNs widens the focus from power reduction to the reduction of sensor usage, since the limited reagent resources available to such nodes will limit node lifetime.

This project builds upon the prototype Gumsense sensor node to create a platform that is suitable for the development and deployment of WSNs that incorporate "autonomous experimentation" algorithms. Substantial changes are made to the architecture of the Gumsense hardware platform in pursuit of this new philosophy. The platform is upgraded so that data can be sampled whilst remaining in a low power sleep state to minimise resource usage. C and Tcl libraries are developed to enable the rapid development of algorithms for the new Gumsense.

Contents

Acknowledgements

I would like to thank my family, who have supported me throughout all this insanity, and Klaus-Peter Zauner for being a fantastic project supervisor!

Chapter 1

Introduction

A wireless sensor network (WSN) is a distributed array of electronic devices, each fitted with one or more sensors and a wireless transceiver [1, 2]. Most WSNs contain an *information sink* that forms a bridge between the network and the Internet, allowing near real-time access to sensor information.

WSN systems can be used to generate data for scientific research purposes, whilst others can generate alerts in response to various events. Applications of WSNs include seismic activity [3], animal habitat [4], traffic [5] and pollutant monitoring. As WSN technology develops, an increasing number of applications become feasible.

Typically, each device, or *node*, is powered by battery. However, sourcing energy from the surrounding environment, perhaps through solar power [6] or vibrations [7], is an active area of research.

The design of wireless sensor nodes and their interaction as a network is primarily constrained by the tradeoff between power consumption and node lifetime. Therefore, research focuses upon the part of the design that uses the most energy. The energy used by this part is reduced either by replacing or redesigning the part itself or by changing the way that the rest of the system interacts with the part (e.g. by reducing the duration for which the part is powered).

For this reason, the primary focus of WSN research has been reducing the power consumption and temporal usage characteristics of node radios, since the energy required to transmit, receive and listen [8] for data on the radio link has been the most significant power drain. Clustering [9, 10, 11], data aggregation [12, 13] and informed routing protocol design [14] can all contribute to reductions in the energy used by WSN radio devices.

Much of this work regarding the reduction of radio usage uses the assumption that the nodes of the WSN are densely packed within the monitored environment [12]. Without significant technological improvement that reduce the size and power requirements of radio, computational and sensor apparatus, dense packing is difficult to achieve. Furthermore, the calibration of all of the sensing devices of a densely packed WSN presents a significant challenge [3]. Also, the task of removing hundreds or thousands of nodes from an area when a network is no longer needed brings environmental concerns. For these reasons only sparsely distributed WSN networks have been created in practice.

As the applications of WSNs become increasingly complex, the energy required to power sensors and process sensor data upon the WSN node becomes increasingly significant. Some WSNs are beginning to use CCD and CMOS imaging devices to measure the properties of their environment [15, 16], for example in order to count the number of eggs in a bird nestbox [3]. The energy required by such imaging devices is significantly larger than required by previously used sensors [16] as is the energy required to perform the image processing tasks necessary to extract measurements from

these images. These images must be processed within the nodes themselves, since the amount of energy to transmit such large quantities of data is prohibitively large.

Wireless sensor nodes must be able to perform the processing of data from image sensors at the same speed that it is sampled. The low power microcontrollers that have previously been deployed within WSNs, such as on the Mica Mote [17], must therefore be replaced, or accompanied by a microprocessor with greater computational ability.

This increase in the energy per sensing action logically leads to the development of algorithms that maximise the information returned by the energy invested in a sensor sample. However, it is important that reading accuracy remains above a particular level. Using the distributed nature of the WSN, it should be possible to use information from neighbouring node sensors to reduce the number of sensor readings necessary to achieve a specified level of accuracy about the sensor environment.

Furthermore, the development of microfluidic "lab on a chip" devices introduces the possibility of integrating such components into wireless sensor nodes. Many of these sensors would only have limited supplies of chemical reagents available to them. This scarcity of reagents will need to be incorporated into the algorithms implemented on WSN nodes.

The prediction of the information return from a sensor reading shares properties with the scientific research process. Measurements are performed when values cannot be predicted to the desired level of accuracy. A system of continuous hypothesis generation and testing can be created [18]. The application of this concept to WSNs with high power sensing capabilities should be fruitful. The term *autonomous experimentation* refers to the behaviour of an algorithm that intelligently takes into account previous sensor readings, their predicted values and the uncertainty in future readings [19].

In order to research new applications for WSNs it is useful to have a flexible development platform that can be used in a number of applications. There have been many development platforms created by a number of research groups, including LEAP [16], PASTA [5], the Mica platform [17] and MoteLab [20].

A prototype of a sensor node that has a peak computing capability comparable to a high end mobile phone, but adheres to stringent power requirements in sleep mode has been developed and built within ECS. This prototype, developed by Dr. Royan Ong, has been given the name "Gumsense". This prototype served as a starting point for this project and fundamental changes to its architecture were made. Software libraries were developed to enable the rapid development of algorithms for this new platform.

WSNs are often deployed in extreme environments, such as Glaciers [21], and so the installation and maintenance of sensor nodes forms a significant

challenge. It is therefore important that nodes do not suffer from software errors whilst they are deployed. Throughout this project, great care is taken to ensure that the software provides a robust and reliable interface to the Gumsense hardware.

1.1 Previous Work

A prototype wireless sensor platform was previously designed and constructed by Dr Royan Ong. This device connects to a Gumstix embedded computer board, a product of Gumstix Inc, Portola Valley, California [22]. This extension card was given the name "Gumsense", and is shown in figure 1.1.

The Gumstix is an embedded computer that runs the Linux kernel. The Gumstix has relatively significant computational capability when compared with the micro-controllers that the majority of previously constructed WSN nodes have - such as the ATmega103 microcontroller employed on the Mica platform [17], which is compared against the specification of the Gumstix used in this project in table 1.1.

The Gumsense provides the following functionality:

- Two analogue inputs connected through inverting amplifiers with an adjustable gain range from zero to -10.
- Two differential analogue inputs configured for connection to a resistive sensor in a Wheatstone bridge.
- Current limited supply rails which can be enabled/disabled by the Gumstix.
- Access to some of the Gumstix IO, including the serial ports.

All of the inputs and outputs of the Gumsense are exposed on 2.5mm pitch pin headers (with the exception of the Gumstix USB) for easy access.

1.1.1 Architecture

The original Gumsense prototype has the architecture shown in figure 1.2.

TABLE 1.1: A comparison of the hardware of the ATmega103 microcontroller [23], found on the Mica platform [17], against the Gumstix Basic-XM [24].

Specification	**Gumstix**	**ATmega103**
Word Length	32-bit	8-bit
Clock Frequency	200 Mhz	6 Mhz
RAM	64 MB DRAM	4 kB SRAM
Non-volatile Storage	16 MB flash	128 kB flash

The Gumstix power is controlled using a DS1337 RTC, which is configured using the I^2C bus connected to the Gumstix. Similarly, the RTC also controls the power delivered to the PIC microcontroller. By design, the PIC can only be powered at the same time as the Gumstix.

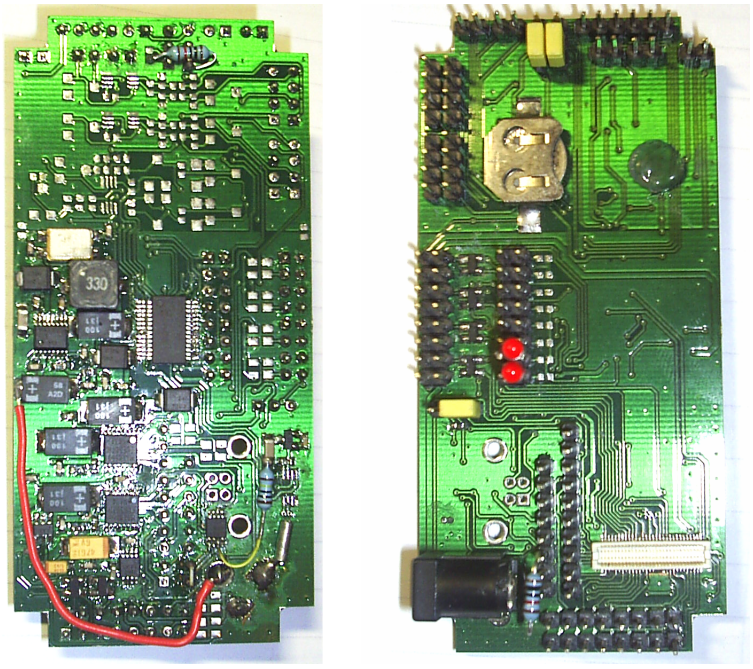

FIGURE 1.1: The Gumsense prototype. Left: bottom view. Right: top view.

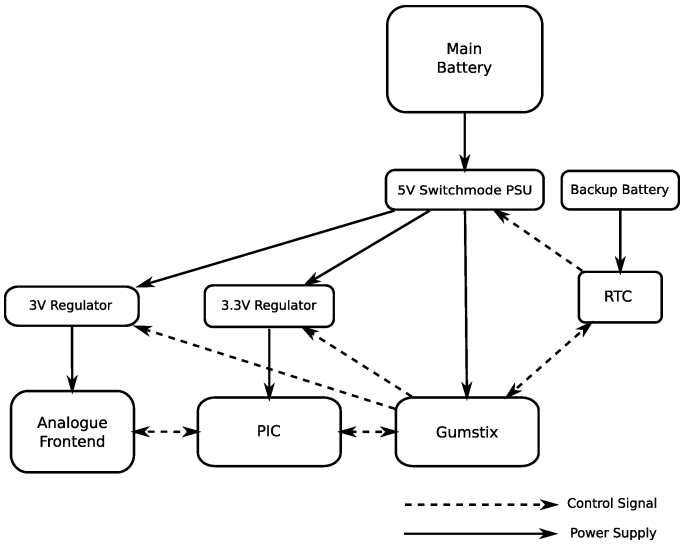

FIGURE 1.2: Original Gumsense prototype architecture.

1.1.2 Analysis

This prototype functioned as desired, with a few minor exceptions. However, there are several things that can be improved:

- The device was initially created with the intention of using it with a wireless network card attached to the Gumstix. However, the dimensions of the Gumstix with wireless card attached had not been taken into account when designing the Gumsense PCB, and so it was not possible to use the wireless card.
- Since the analogue-to-digital converters are in the PIC microcontroller, and the PIC shares a power supply with the Gumstix, the current design requires that both the Gumstix and PIC are powered when an analogue input is sampled. This is quite wasteful of energy if performing a single reading.

1.2 Goals

The aim of this project was to develop the hardware and software for a wireless sensor node with significant on-board computing capability, implement a network of nodes and demonstrate its operation in a test scenario.

To achieve this aim the following goals were created:

- Improve the performance of the existing prototype.
- Develop software libraries and drivers to be used on the Gumstix to interface with the Gumsense.
- Create scripting language bindings of the software library, to allow for fast prototyping of algorithms.
- Develop autonomous experimentation algorithms for use upon the Gumsense platform and demonstrate them on the platform.
- Document the hardware and software platform so that it can be used for a variety of research purposes.

Chapter 2

Technical Progress

2.1 Gumstix Sleep

A method of reducing the power consumption of the Gumsense module during periods of computational inactivity is for the microprocessor on the Gumstix to enter sleep mode. However, no previous record of getting a Gumstix to resume operation from sleep mode could be found.

2.1.1 Linux and Sleep Mode

The state of the software associated with the Gumstix sleep was examined.

It is necessary to compile the Linux kernel with the `CONFIG_PM` configuration option enabled (as stated in `/kernel/power/Kconfig` in the kernel source tree [25]) in order for the sleep functionality to be present.

Upon a request for sleep mode, issued by writing the string "`mem`" to `/sys/power/mem`, the kernel suspends execution of user-space processes, puts the external memory into self-refresh mode and puts the processor into sleep mode. Sleep mode is entered in `/arch/arm/mach-pxa/sleep.S`, which is called by `pxa_cpu_pm_enter()` (of `/arch/arm/mach-pxa/pxa25x.c`).

The Gumstix microprocessor, the PXA255 resumes execution upon a wake-up event interrupt [26, p. 3-18]. This interrupt can be triggered by either a GPIO pin or the internal RTC. The GPIOs that can be used, `GPIO[15:0]` [26, p. 4-1], are not connected through to the Gumstix header [27]. Therefore, the only method of waking the Gumstix from sleep is by using the internal RTC of the processor.

2.1.2 Power Consumption

The performance benefits of using the Gumstix sleep mode were considered. The PXA255 electrical specifications datasheet specifies that the supply current in sleep mode is a maximum of $75\mu A$ [28, p. 25].

However, another part of the Gumstix is the SDRAM external to the PXA255 processor. Through examination of the chip top-mark coding, the full part number of the RAM ICs used on the Gumstix was found to be the Micron Technology `MT48LC16M16A2FG-75:D`. The datasheet for this device specified that the self-refresh current (the lowest power mode) is a maximum of 2.5mA per device [29], thus rendering the PXA255 sleep current insignificant.

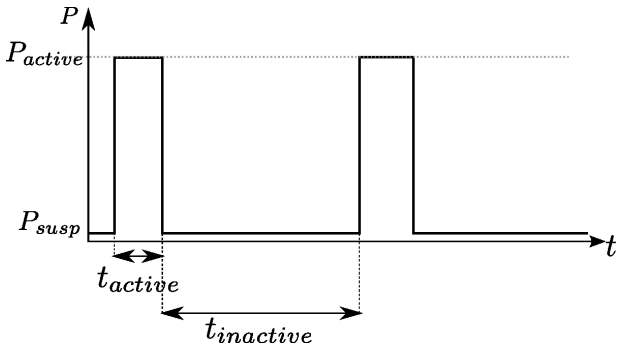

FIGURE 2.1: Graph showing the power use of a typical sensor node.

2.1.3 Analysis

In order to obtain a figure to compare the sleep current to, the approximate current consumption of the Gumstix whilst in active mode was measured to be a maximum of 110mA, to an accuracy of $\pm 5mA$ using a bench power supply (a GW Instek GPC-1850D).

The relatively high power of the sleep mode of the Gumstix poses a problem if the device is to have a reasonably long battery-powered lifetime. If one considers an application of the Gumsense in which the Gumstix is subject to a periodic sleep-resume cycle, as shown in figure 2.1, then the amount of energy wasted in powering the Gumstix during sleep mode is:

$$E_{susp} = NP_{susp}t_{inactive} \tag{2.1}$$

Where N is the number of sleep-resume cycles that the device performs, P_{susp} is the power usage whilst in sleep mode and $t_{inactive}$ is the duration of the inactive period. The time required to enter and exit sleep has been regarded as insignificant.

If the Gumstix is turned off during periods of processor inactivity, then the wasted energy is the energy required to boot[1] the Gumstix software:

$$E_{boot} = NP_{active}t_{boot} \tag{2.2}$$

Where P_{active} is the power usage whilst the CPU is active, and t_{boot} is the combined power-up and power-down time.

The condition of interest is when the energy wasted due to the sleep power of the Gumstix is greater than the energy wasted booting the Gumstix:

$$E_{susp} > E_{boot}$$

Using equations 2.1 and 2.2:

$$P_{susp}t_{inactive} > P_{active}t_{boot}$$

[1]In this context, the term *boot* refers to the power-up and power-down processes.

Since the voltage rail supplied to the Gumstix is constant:

$$\frac{t_{inactive}}{t_{boot}} > \frac{I_{active}}{I_{susp}}$$

The power-up time of the default Gumstix environment is 21 seconds [30], but there have been reports of drastic reductions in this time to only 5 seconds [31]. The power-down time is of the order of 5 seconds. By taking a boot time of 26 seconds, a sleep current of 5 mA and an active current of 110 mA:

$$t_{inactive} > 22t_{boot}$$

$$\Rightarrow t_{inactive} > 572$$

Therefore, if the period of inactivity of the Gumstix is greater than 572 seconds (9 minutes 32 seconds), energy will be saved by turning the Gumstix off instead of sleeping during these periods. If, as discussed above, the Gumstix power-up time is optimised to 5 seconds and the power-down time is also 5 seconds, then the threshold inactivity period drops further to 220 seconds.

Both of these time periods indicate that switching the Gumstix power rail off is beneficial in applications with a low frequency suspend-resume cycle period.

In a situation in which the suspend-resume cycle period is variable, the above analysis can be performed by replacing $t_{inactive}$ with its mean value to achieve similar results.

Using these results, it was decided that the Gumsense must be able to control the power rail of the Gumstix. This way, the Gumstix can use sleep mode (if one were to use the inbuilt RTC as a trigger for a resume event) in situations where the average period of inactivity is less than $22t_{boot}$ and power down during longer periods.

However, this inactivity period threshold may be further reduced if more than one set of sampled data can be processed by the Gumstix during each active period. If K samples can be sampled by a significantly lower power device than the Gumstix, whilst the Gumstix is inactive, the inactivity threshold becomes:

$$\frac{22t_{boot}}{K}$$

Evidently, the number of samples that it is feasible to buffer is application dependent. For example, the refresh currents of the memory required to store images obtained from a CCD would play a significant part in determining the optimum number.

To allow for sample buffering, it is necessary to have a device upon the Gumsense that performs the sampling and buffering independently of the power state of the Gumstix. Therefore the task of triggering sampling

events should be moved from the Gumstix to the Gumsense microcontroller in this new Gumsense revision. Similarly, the Gumsense microcontroller should control the power rails delivered to off board sensing devices.

TABLE 2.1: The sleep currents of various microcontrollers. The "timer sleep current" is the current required for all peripherals except for the timer peripheral and crystal oscillator operating at 32768 Hz. All values are for room temperature operation (25°C) with a 3.3 V supply rail. Compiled from [32, 33, 34, 35].

Device	Sleep Current	Timer Sleep Current
PIC16LF876A	6.3 μA	11.3 μA
tinyAVR ATtiny28L	<1 μA	9 μA
MSP430F169	0.2 μA	~3.3 μA
LPC2148	40 μA	unknown (>40 μA)

2.2 New Gumsense Design

2.2.1 Microcontroller Investigation

Since the new design of the Gumsense would place different demands upon the microcontroller than on the prototype, alternatives to the PIC16F876A were considered.

The main factors affecting the choice of microcontroller were the sleep current and the presence of I^2C and ADC peripherals. Table 2.1 shows a comparison of the sleep currents of a selection of microcontrollers that were considered for the project. It is evident that the MSP430 uses significantly less power than the other devices.

The time-keeping operation of the DS1337 RTC can be moved into the software of the MSP430F169. This involves an increase in the operating current of the system, since the time-keeping current of the DS1337 is 1.5 μA [36], whilst the MSP430F169 consumes 3.3 μA. However, this decision is well justified because the architecture of the system is simplified. The MSP430 does not have to interface with the DS1337 to schedule wake-ups. Also the MSP430 will not need to store readings, which are pending transfer to the Gumstix, in non-volatile memory (a process which consumes energy).

The architecture for the new Gumsense design is shown in figure 2.2.

2.2.2 Schematic Design

Using the new Gumsense architecture design, the schematic for the new PCB was put together in the EAGLE Layout Editor from Cadsoft, Delray Beach, Florida [37]. The schematics are shown in appendix A, and the bill of materials is shown in table A.1.

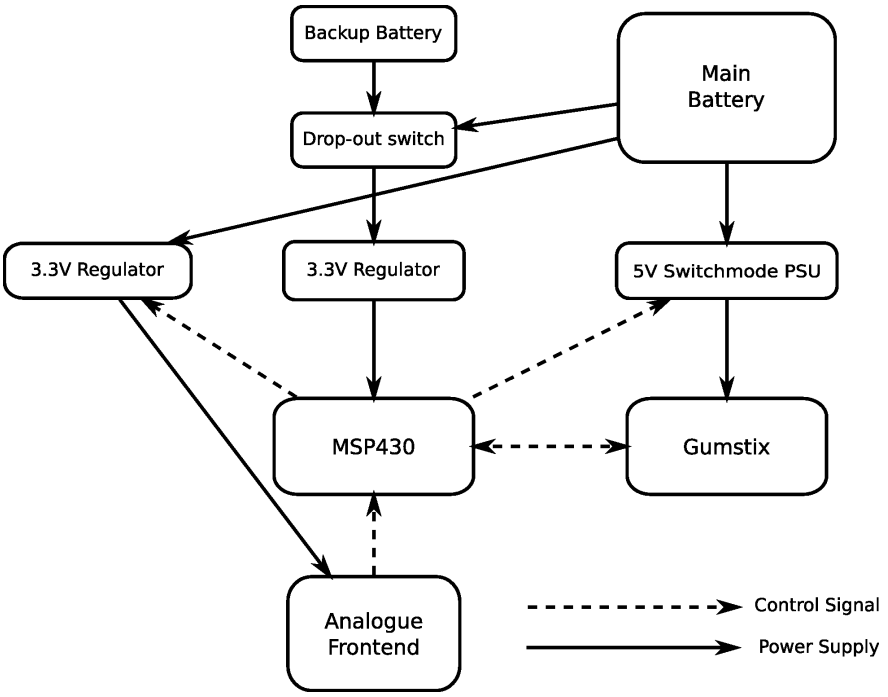

FIGURE 2.2: Enhanced Gumsense architecture.

A list of the headers on the Gumsense can be found in table 2.2. A number of these headers, including JP4, JP7 and JP13, have been implemented to allow for future expansion of the MSP430 software. In particular, JP13 provides an interface to the "spare" serial interface of the MSP430. In the future, this could be used to interface the MSP430 directly with a PC, and thus allow for "standalone" operation of the Gumsense without an attached Gumstix. This would allow for heterogeneous WSN configurations to be developed.

A set of additional IO connections between the Gumstix and MSP430 were created, namely `MSP430_0` and `MSP430_3`. Originally there were 4 connections. Two of these had to be removed in order to simplify PCB routing. These connections were installed to allow for future development.

The design of the switch-mode power supply for the Gumstix was reviewed and modified according to requirements. The output voltage of the switch-mode supply on the original Gumsense had been 5.8 V. With the change in Gumsense architecture, this needed changing to 5 V. Therefore, the feedback resistor values of the MAX1685 were modified accordingly. With this new output voltage a smaller output capacitor and inductor could be used.

The differential amplifier design was reviewed with reference to the AD623 datasheet [38]. A fault in the use of the AD623 in the original Gumsense was found; the "REF" pin of the AD623 was connected to ground. The AD623 datasheet states that [38, p. 11]:

TABLE 2.2: The headers on the Gumsense.

Designator	Description
JP1	MSP430 4-wire JTAG
JP2	Differential Amplifier 1 Input
JP3	Differential Amplifier 2 Input
JP4	Gumstix IO
JP5	Inverting Amplifier 1 Input
JP6	Inverting Amplifier 2 Input
JP7	MSP430 IO
JP8	Gumstix FF Asynchronous Serial Port (Default for serial console)
JP9	Controlled 3.3V Power Rails
JP10	Controlled 5V Power Rails
JP11	Gumstix HW Asynchronous Serial Port
JP12	I^2C and Gumstix PWM Output
JP13	MSP430 Asynchronous Serial
JP14	Gumstix serial interface for audio
JP15	Analogue inputs without interface amplifiers

> *The output signal appears as the voltage difference between the Output pin and the externally applied voltage on the REF input. For a ground referenced output, REF should be grounded.*

In the original Gumsense, the differential amplifier output was "ground referenced", meaning that the output voltage was configured to swing about ground. Since the AD623 was not supplied with a negative rail, this would prevent the amplifier from detecting input voltage swings in one direction. To correct this problem, the REF pin was connected to a potential divider providing half the supply rails.

A distributor of the MAX4477 dual op-amp could not be found, and so the OPA2335D dual op-amp was used as a replacement. The LDO regulator chosen to regulate the analogue supply rail is the MAX8881. This regulator has a shutdown pin, which has been connected to an MSP430 general-purpose IO pin. The MAX8881 comes in several variants [39] with different preset output voltages. A 5V version was also added to the Gumsense so that 5V off-board devices could be powered. The shutdown pin of this regulator was also connected to an MSP430 pin.

2.2.3 PCB Design

The PCB was routed using the EAGLE Layout editor [37] autorouter. An initial attempt at manual routing was made, but this would have taken

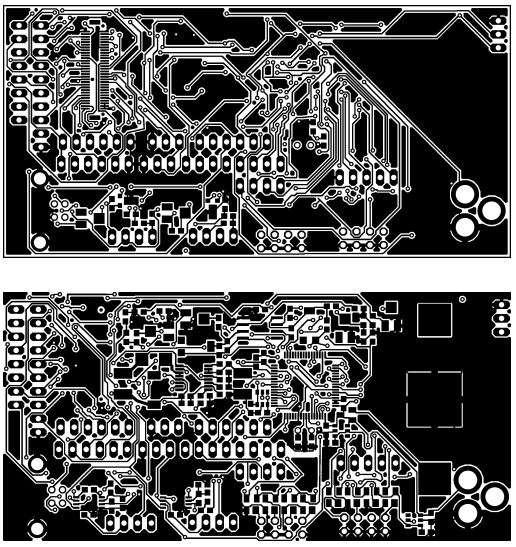

FIGURE 2.3: The top and bottom layer masks of the Gumsense PCB. Actual dimensions are 95 by 47 mm.

much more time than the autorouting procedure (the autorouting process took approximately an hour, whilst about 75% of the airwires were routed in around ten hours). The resultant board masks are shown in figure 2.3 and diagrams of component placement are shown in figure 2.4.

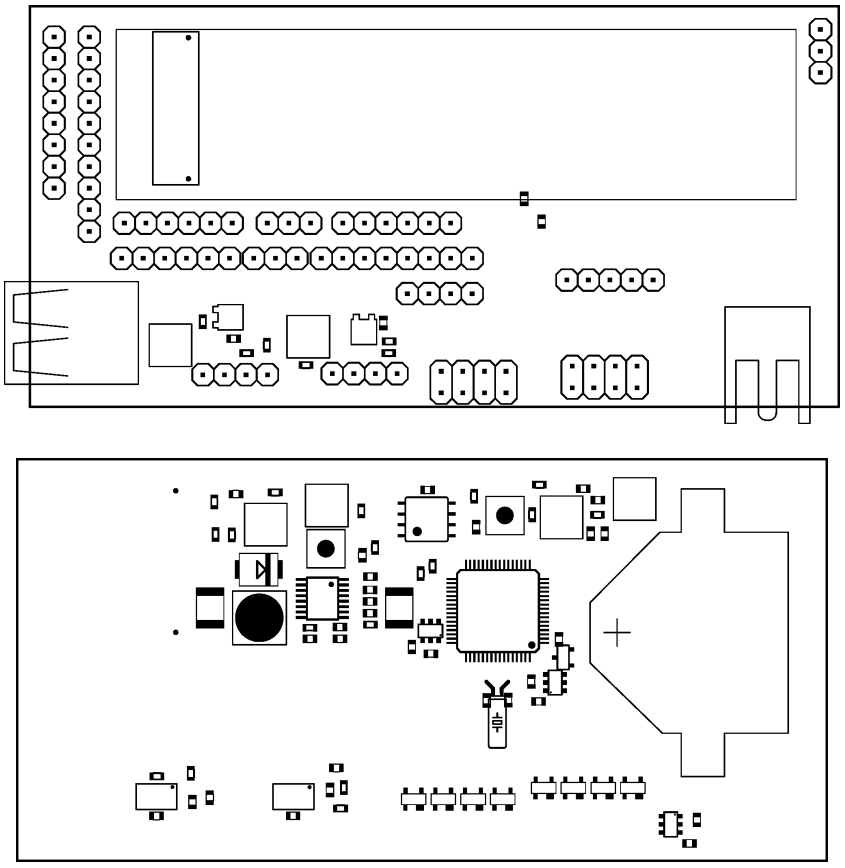

FIGURE 2.4: The component placements on the top and bottom of the Gumsense PCB (95 by 47 mm).

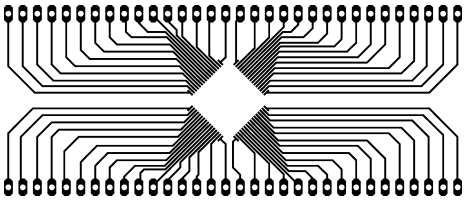

FIGURE 2.5: PCB layout for QFP64 to DIP64 adapter.

2.3 MSP430 Development

2.3.1 MSP430 Selection

The MSP430 micro-controller selected to be used on the Gumsense is the MSP430F169. It was chosen because it provides a large number of GPIO pins, as well as an I^2C slave device and a 12-bit, 8-channel ADC peripheral.

The MSP430F169 is in a 64 pin Quad Flat Pack (QFP) package. In order to prototype the Gumsense design, an adapter board was constructed to allow a MSP430F169 to be plugged into breadboard. This PCB was designed using the EAGLE Layout Editor from CadSoft, Delray Beach, Florida [37]. Figure 2.5 shows the PCB design and figure 2.6 shows a picture of the soldered device. After manufacturing the PCB, it was discovered that the spacing of the breadboard holes across two attached breadboards was a multiple of 50 mil, rather than 100 mil. This meant that the wires on either side of the adapter were at an angle, and the module was relatively challenging to remove from the breadboard.

2.3.2 Toolchain

The mspgcc [40] toolchain was used to compile the software for the MSP430. mspgcc is still under development, and so some experimentation was necessary in order to get a working version. Binutils version 2.16 was used with GCC 3.2.3. Programming of the MSP430 was performed through an MSP430 JTAG adapter.

2.3.3 Software Specification

The specification for the MSP430 software is as follows:

- Sample analogue inputs at times specified by the Gumstix.
- Enable and disable the Gumstix power rail at times given by the Gumstix.

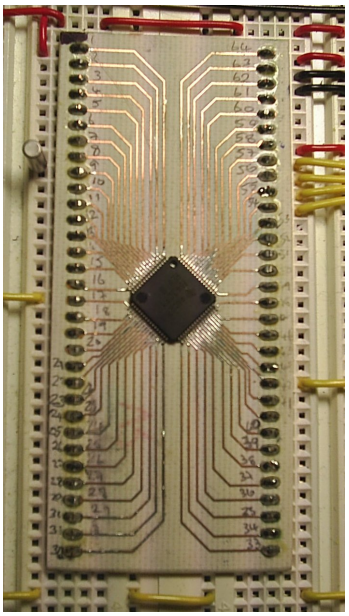

FIGURE 2.6: The MSP430F169 mounted on the PCB for prototyping. The PCB is pictured plugged into protoboard.

- Enable and disable peripheral power rails at times specified by the Gumstix.
- Provide access, through an I^2C bus, to parameters that configure the operation of the software and to data that has been collected by sampling inputs.

The MSP430 has several power modes. Two of these modes have been used:

LPM3 (Low Power Mode 3), in which the CPU core executes no instructions. Various device peripherals remain active - including the timer and USART (used for I^2C) peripherals.

AM (Active Mode) The core is clocked and executes instructions.

In the event of an interrupt, the MSP430 interrupt mechanism allows the device to transition from LPM3 to AM. After this transition, the device executes the relevant interrupt service routine (ISR). Upon completion of ISR execution, the device enters the power state it was in prior to ISR execution.

This ability of the MSP430 to automatically transition between power states combined with the event-driven nature of the Gumsense MSP430 software lends itself to placing the main functional software in the interrupt service routines.

The current revision of the MSP430 software is included in appendix B.

2.3.4 Time

The software keeps track of the time using a 32.768 KHz crystal (chosen because a division of the frequency by 2^{15} gives a signal of 1 Hz) as the clock for the MSP430 'timer A' internal peripheral. The timer A peripheral has a 16-bit counter register. Therefore the timer A peripheral can time a maximum of 2 seconds without CPU core activity. The MSP430 timer peripheral can toggle the value of an output pin when a value is reached. Timer A was configured so an output pin would toggle at a rate of 1 Hz. This output was connected to the clock input of the 'timer B' peripheral, which also contained a 16-bit counter register. This increased the period of time that could be measured without waking the microprocessor core to 2^{16} seconds.

This period can be extended by using the timer peripheral clock dividers. However, sleeping for such large times is incredibly unlikely to happen.

2.3.5 Scheduling

A system has been created in which the MSP430 maintains a list of recurring jobs, each of which has the following properties:

- A set of analogue inputs to sample, in the form of the bit mask `channels`.
- A set of power rails which must be powered during sampling, `power_mask`.
- The sampling period in seconds, `interval`.
- The time between power-up and analogue-to-digital conversion, `power_up_time`.

This information is stored in a 16-entry array of C structures with the format shown in figure 2.7. 16 entries were created so that the logging of data was simpler, and this number was deemed to be sufficient, since it would allow 2 jobs per ADC channel. This job table is modified via the I^2C bus by the Gumstix (see section 2.5).

The job scheduling software is mainly implemented in the timer B interrupt routine (in `timer-b.c`). This routine invokes two scheduling operations: `schedule_do_jobs()` and `schedule_next_wakeup()`. `schedule_do_jobs()` executes the jobs which need to be executed at the current time. `schedule_next_wakeup()` determines the next time at which the CPU must enter active mode.

Whilst implementing the job scheduling routines, a test job with a period of one second was hard-coded into the firmware. For operational verification, the code was modified so that upon execution of a job, the value of an output pin would be toggled. Using this method of debugging, an error

```
typedef struct
{
  time_t interval, next_time;

  uint8_t power_up_time;
  uint8_t power_mask;

  uint8_t channels;

  uint8_t powered:1;
  uint8_t adc_wait:1;
} job_t;
```

FIGURE 2.7: The job information table structure

was discovered whereby the job execution period was one second longer than desired. It was later discovered that the timer B interrupt occurs one timer clock cycle *after* the specified timer value has been reached. The frequency of the clock generated by timer A was doubled to 2 Hz. The target timer B value was in turn modified so that it would be reached half a second prior to the time of the desired interrupt. The timer B interrupt thus occurred at the desired time.

2.3.6 I^{2}C Routines

To multiplex the necessary functions through same I^{2}C bus address, a command/register system similar to that specified by the SMBus specification [41] was used. The Linux kernel provides an API to SMBus compatible devices, and so it was possible to use this to interface with the MSP430. The register read and write command sequences are shown in figure .

The MSP430 I^{2}C peripheral driver is implemented in `i2c.c` ,which can be found in appendix B. The specifics of this peripheral were hidden from the routines that produce and process register values through the creation of an abstract interface to read, write and size functions for each register, in `i2c_reg.c`. Pointers to the register access functions are stored in the array `dev_regs`. This greatly simplified the implementation and debugging of the I^{2}C code.

A description follows of the functionality of each of the I^{2}C registers that were implemented:

0: Identity This is a read-only register that always reads as 0x0100AB. This is useful for automatically identifying the Gumsense from the Gumstix software.

1: Time This is 4-byte read-write register. The MSP430 increments its value every second. Perhaps the most sensible value to be stored here is the Unix timestamp (the number of seconds since the Unix epoch). This time is used to control all other operations within the MSP430.

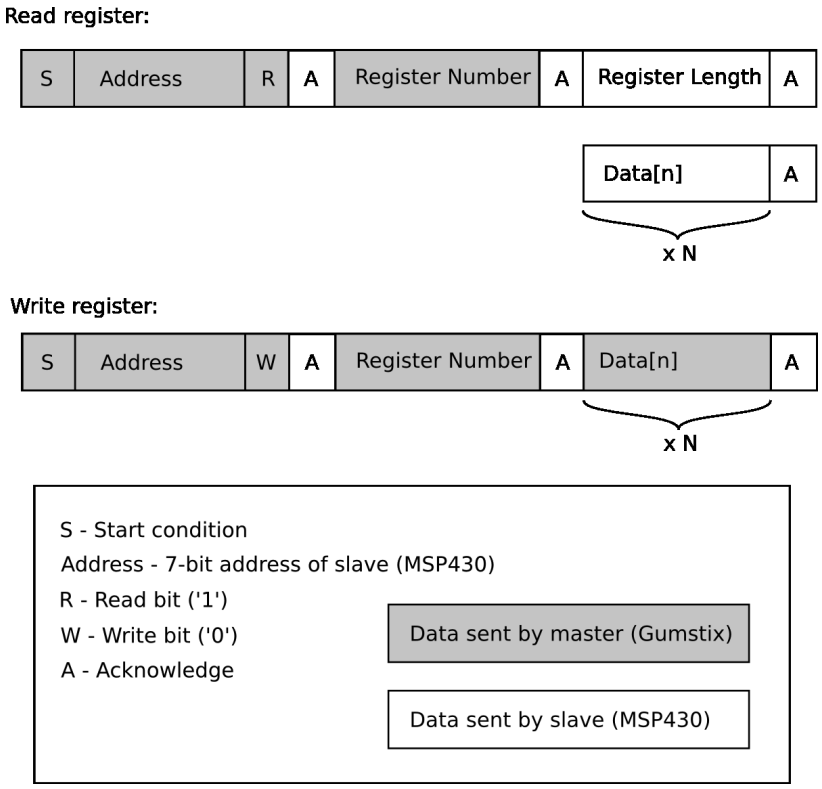

FIGURE 2.8: The sequence of I^2C events for register read and write operations.

2: Job Reading from this register gives all 16 entries of the current job table. The format of this data is described in lines 114-120 of `i2c_reg.c`, which can be found in appendix B. Writing to the register overwrites an existing entry in the table. The data written to the register is prefixed with the number of the job entry to overwrite. The format of the write data is show in lines 143-150 of `i2c_reg.c`.

3: Job Data Size This is a read-only register that gives the number of samples available.

4: Job Data The job data can be read in blocks of 32 bytes from this register. The block number should be set by writing to the job_read_cursor register. The read cursor is automatically-advanced when a block is read from this register.

Writing the "magic" number 0x203E to this register causes the data that has been read from this register to be cleared from the internal buffers.

5: Gumstix Power Writing to this register causes the Gumstix power to be disabled. A "magic" value of 0x78B5 must be written to this register for the power to be disabled. The "magic" value has been implemented so that it is more unlikely that the power will be disabled accidentally.

6: Gumstix Wake Time This stores a 4-byte time value that is both readable and writable. The Gumstix power will be enabled when the time stored by the MSP430 is equal to the value stored in this register.

7: Job Execution Mask This is a single byte read-write register. If the value of this register is non-zero then jobs are executed. If it is zero then no jobs are executed (with the exception of powering up the Gumstix).

8: Job Data CRC This register is currently partially implemented. It was envisioned that the sampled data would be CRC check-summed on the MSP430 and that it would be possible to read this value through the register.

9: Job Read Cursor This is a 2-byte read-write register that contains the number of the data block to be read. In normal operation, it is only necessary to zero this register before reading sample data from the job data register.

10: Job Data Size/Time This is a read-only register that contains both the number of samples available and the time of the last sample.

The MSP430 I^2C lines were connected to a Gumstix via a Gumstix "console-hw" expansion board. Section 2.5 discusses the Gumstix I^2C software in detail.

Whilst debugging the I^2C routines, a bug in the MSP430 compiler being used was found. The machine code generated by the compiler when it processed a `switch` on a volatile variable incorrectly accessed that variable twice. Figure 2.9 shows the smallest amount of code required to replicate this bug. A disassembly of the relevant section of machine code is shown in figure 2.10. The listing clearly shows that the volatile value is accessed twice, yet assumes that the value stays constant. This was a problem, since the I^2C interrupt vector register, I2CIV, (which stores the type of the highest priority I^2C interrupt) value changes upon access [42, p. 15-19]. Therefore, a workaround was used where the value of the register was copied into a local variable, which was then used as the parameter for the switch statement.

2.3.7 Data Logging

The MSP430 software samples data and stores it in an array in RAM. Two methods of formatting this data were considered. In order to maintain a time-value mapping for each ADC channel, the absolute time (4 bytes), the channel number (3 bits) and the sample value (12 bits) could be stored for each sample. This would be quite inefficient, as it would contain a large

```
volatile int test;

int main( void )
{
        int i;

        switch( test )
        {
        case 0: i = 1; break;
        case 1: i = 2; break;
        case 2: i = 3; break;
        case 3: i = 4; break;
        case 4: i = 5; break;
        case 5: i = 6; break;
        case 6: i = 7; break;
        case 7: i = 8; break;
        }
}
```

FIGURE 2.9: Smallest amount of code that demonstrates the "switch on volatile" compiler bug.

```
1  0000fc40 <main>:
2      fc40:        31 40 7e 02     mov     #638,   r1      ;#0x027e
3      fc44:        04 41           mov     r1,     r4      ;
4      fc46:        b2 92 00 02     cmp     #8,     &0x0200 ;r2 As==11
5      fc4a:        2a 2c           jc      $+86            ;abs 0xfca0
6      fc4c:        1f 42 00 02     mov     &0x0200,r15     ;0x0200
7      fc50:        0f 5f           rla     r15             ;
8      fc52:        3f 50 5a fc     add     #-934,  r15     ;#0xfc5a
9      fc56:        2f 4f           mov     @r15,   r15     ;
10     fc58:        00 4f           br      r15             ;
```

FIGURE 2.10: Portion of the disassembly of the code in listing 2.9. Memory address location 0x200 is accessed twice (lines 4 and 6).

amount of redundant information, since the samples are periodic and their period is stored in the job table. The array of samples was therefore implemented so that it would only contain the sample values and the channels that those samples represent. When the Gumstix reads the sample table out of the MSP430, it recreates the mapping of samples to absolute time using the time of the last sample recorded provided by the MSP430.

Initially, job execution (i.e. logging) had to be suspended whilst sample data was read out of the MSP430. However, this proved to be inconvenient, as samples could be missed if it took too long to read the data from the MSP430. Therefore, the sample logging and access system was modified to treat the sample buffer as circular. The routines responsible for logging data would add data to the current "tail" of the buffer, which would wrap round from the end to the beginning when necessary. This meant that space in the buffer could be freed by the read routines whilst the logging routines could continue to write to the buffer whilst the data was still being read. The I^2C operation that would clear all data was adapted so that it would clear only the data that had been read from the buffer.

2.3.8 Continuous Time Measurement

The time stored in the MSP430 is updated every time a scheduled job executes, in the timer B interrupt service routine. If the current time was read from the Gumsense via I^2C at some point between these time updates, only the time at which the last job executed would be read.

By modifying the I^2C time register read routine (`i2cr_time`) to use the current value stored in the timer B counter combined with the time stored by the last interrupt, it was hoped that this would allow reading of the correct time at any time. With the use of a test program that read the time from the Gumsense every tenth of a second, it was discovered that this generated some erroneous values. The output of this program with a single job with a 10 second period was ... 259, 259, 260, 260, 260, 260, 270, 270, 270, 270, 270, 261, 261 .... A job was executed when the time transitioned from 260 to 261, which is when the value read from the Gumsense moved from 260 to 270. It is evident from this data that there is a period of at least 0.5 seconds in which the value read from the MSP430 is out by 10 seconds.

After further examination of the MSP430x1xx User Guide [42, p. 230], it was discovered that the service routine of the timer B interrupt `TBCCR0` is executed immediately when the timer reaches the requested value. Therefore during the 0.5 seconds following this interrupt, the addition of the timer register to the time variable will yield the observed erroneous value.

The timer B peripheral has a second interrupt that is triggered one timer B clock cycle later than `TBCCR0`, called `TBIV`. By switching to use this interrupt the erroneous time readings were eliminated.

2.4 Assembly

A set of new Gumsense PCBs was kindly manufactured by the Cambridge Circuit Company Ltd, Cambridge, UK. The components were subsequently soldered onto one of them using a hot-air reflow soldering tool. The different parts of the PCB were tested as the device was assembled.

During assembly, it was discovered that an error had been made in purchasing the 1 μF 0603 capacitors. So, 4.7 μF capacitors were used in their place. This meant that the maximum supply voltage that could be applied to the Gumsense was 6 V, because the substitute capacitors were not rated to the 16 V that the intended ones were.

There were two errors that required PCB net modification. The ground connection to the diode and capacitor on the output of the switch-mode supply was not present. This was fixed by placing a short piece of wire between a nearby grounded via and the terminal of the capacitor. The second fault was that the output of the MSP430 supply 3.3 V regulator was connected straight to the MSP430, rather than going through the dropout diode. This was fixed by cutting some tracks and implementing the correct connectivity using three short pieces of wire. The two nets were connected in the netlist, yet the schematic editor did not provide any visual indication of this - and so the error was not discovered until testing took place.

The USB connection to the Gumstix was found not to function. This was because the USB had been assumed to work on the original Gumsense. However, this was not the case. The Gumstix website [22] provided no specific documentation on how to use the USB connections of the Gumstix. By examining the schematic of the conshole-hw board [43], it was discovered that an additional chip and Gumstix IO line were required to implement the software controlled 5V pull-up that the USB specification requires [44, p. 120]. There was not sufficient time remaining to produce a second PCB revision with these modifications, and so this is left for future work.

It was not possible to source any MAX4794 current switches, and so these were not implemented upon the PCB. However, it was realised that the quiescent current of these devices would drastically increase the sleep current of the device. If a p-channel MOSFET was placed between the MAX4794 supply pins and the supply rail, then the MSP430 would be able to completely remove this standby current (when none of the rails were being used).

Figure 2.11 shows the new assembled Gumsense PCB. Figure 2.12 shows the Gumsense with the Gumstix attached and also with the Compact Flash WiFi card attached.

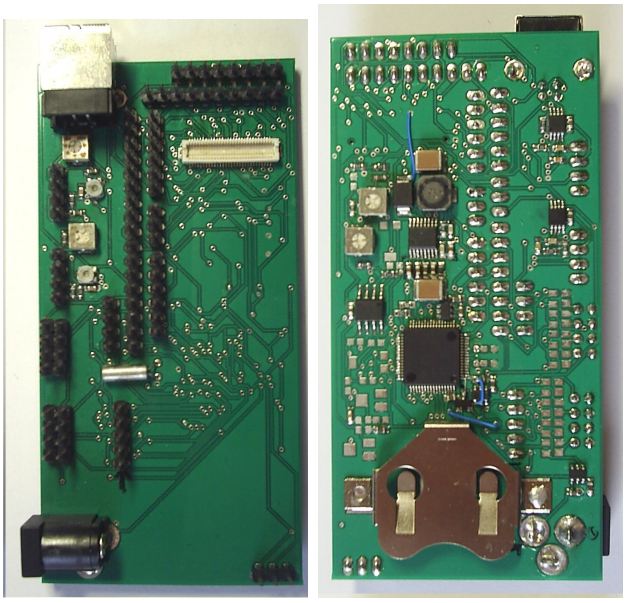

FIGURE 2.11: The new Gumsense PCB top (left) and bottom (right). Actual dimensions are 95 by 47 mm.

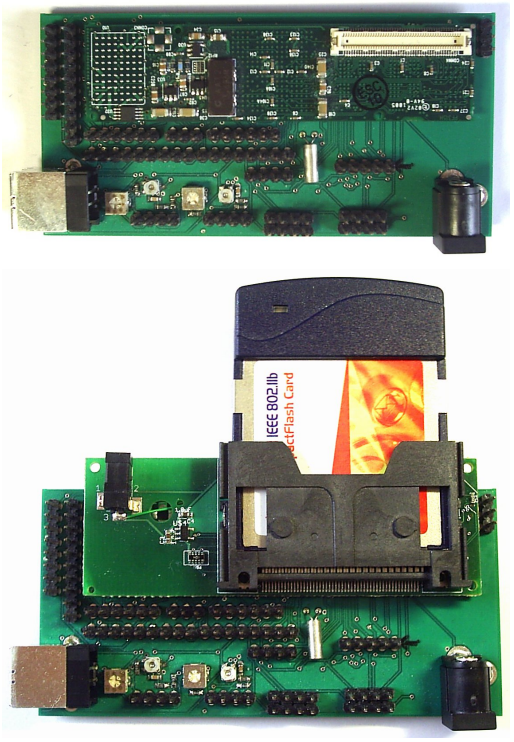

FIGURE 2.12: The assembled Gumsense with Gumstix attached (top) and with Gumstix and wireless card attached (bottom).

2.5 Gumstix Software

There are three main components of the Gumstix software stack:

Linux [25] The Gumstix uses the Linux kernel compiled for ARM microprocessors. Version 2.6.18 was used initially and 2.6.20 was used when the Gumstix Buildroot changed to use it.

uClibc 0.9.28 [45] This is a C library designed for use in embedded Linux systems. It is designed for systems with limited memory.

BusyBox 1.1.2 This provides a set of common shell commands with a small memory footprint.

The Gumstix project (`www.gumstix.org`) provides a set of scripts, collectively called Buildroot, which automate the process of configuring and building an image of the Gumstix file system. For this project, the Gumstix Buildroot subversion repository was cloned into a Git (`http://git.or.cz/`) repository[2]. By cloning into a Git repository, changes made "upstream" by the Gumstix project could be integrated with the Gumsense specific changes with ease. Changes that were made to Buildroot that could be useful to the rest of the community were submitted upstream; however little response was observed.

2.5.1 C Library

A C library was written for use on the Gumstix. This would provide a layer of abstraction between the Gumsense and application software.

It was necessary for the C library to interface with the Gumsense from userspace. Enabling the Linux `CONFIG_I2C_CHARDEV` configuration option creates the device node `/dev/i2c-0`. Using the `i2c-dev.h` header file, from the lm-sensors hardware monitoring project (`http://www.lm-sensors.org/`).

The library uses an opaque structure, `struct gum_t`, to represent the connection to the Gumsense from the application that is using the library. The `gumsense_open()` function should be called by the client application to initialise a new Gumsense connection. The C library provides a number of functions for performing the various operations that the Gumsense provides. These functions can only be used once the connection has been established. A set of functions is provided for each of the I^2C registers. The mapping is shown in table 2.3.

[2]The Gumsense Buildroot Git repository is available at `http://users.ecs.soton.ac.uk/~rds204/gumsense` and on the project CD.

TABLE 2.3: Mapping of C functions to I^2C registers.

Register	**Read Function**	**Write Function**
Identity	`gumsense_check_identity()`	-
Time	`gumsense_time_get()`	`gumsense_time_set()`
Job Table	`gumsense_read_job_table()`	`gumsense_job_write()`
Job Data Size	`gumsense_num_readings()`	-
Job Data	`gumsense_read_data_block()` and `gumsense_read_data()`	-
Gumstix Power	-	Not implemented[a]
Gumstix Wake Time	`gumsense_wake_get()`	`gumsense_wake_set()`
Job Execution Mask	`gumsense_exec_mask_get()`	`gumsense_exec_mask_set()`
Job Read Cursor	`gumsense_job_data_cursor_get()`	`gumsense_job_cursor_seek()`
Job Data Size/Time	`gumsense_num_readings_time()`	-

[a]The C library must never disable the Gumstix power; this is done from the shutdown routines of the Gumsense kernel driver.

2.5.1.1 Callback Mechanism

As indicated in table 2.3, two functions are implemented for reading sampled data from the Gumsense. The first of these, `gumsense_read_data_block()`, reads a single block of sample data from the Gumsense. Whilst writing utilities that made use of this function, it became clear that this interface was cumbersome to use due to the number of loops that had to be implemented, and the lack of translation of the Gumsense response from an array of bytes into an array of samples.

For these reasons the second method was implemented, namely `gumsense_read_data()`. This function would read all of the available data from the Gumsense and call a callback function once per sample. The callback function is provided with a `gum_reading_t` structure, containing the channel number and sample value. A second callback function would be called once with the number of samples and a pointer that would be shared with all the sample callback calls. This method greatly reduced the memory allocation overhead, avoided the need to append data blocks together to process data, and moved code that would almost certainly be duplicated in applications that use the library into the library itself.

2.5.1.2 Testing of the C library

Several simple programs were written to test the interaction of the C library with the Gumsense. These are:

gum-dump This reads and displays all of sampled data from Gumsense. It uses the callback method of retrieving the data.

gum-jobs This reads and displays the contents of the job table.

gum-settime This reads the current time from the Gumsense and sets the system time to it. This is useful for loading the time into the Gumstix on boot.

All of these programs can be found in the project subversion repository.

An error in the MSP430 firmware was discovered using the `gum-dump` program. The MSP430 would report that it had a certain number of samples available for transfer, yet would only provide half that number to the `gum-dump` utility. After analysing the `i2cr_job_data` function, it was eventually realised that the fault was because the read cursor was being advanced by twice the amount it should be at the end of every data block. The constant `MAX_BLOCK_SIZE` had been added to the read cursor every time. This constant was the maximum desired I^2C transaction size in *bytes*. The variable `read_cursor` stores the offset within the sample buffer of the first

sample of the next block to be read. This difference in units between the read cursor and the block size constant was the reason for this error not being noticed whilst writing the MSP430 firmware.

2.5.2 Tcl Library

A Gumsense Tcl library was written that made use of the Gumsense C library. Although Tcl, an interpreted scripting language, may not be the most energy efficient programming language that could be used, it certainly presents a clear, simple and fast method of developing and researching WSN algorithms. Furthermore, it could potentially ease the process of migrating algorithm platform - for example, to the Gumsense from a simulated environment on a PC.

When the compilation of Tcl was enabled in the Buildroot configuration file, Tcl failed to build. This was fixed by modifying Buildroot to build Tcl 8.4.14 rather than 8.4.12 (by altering the values of `TCL_EXTRACTED` and `TCL_TARBALL` appropriately in `package/tcl/tcl.mk` of Buildroot). Furthermore, the Tcl build scripts were modified so that the Tcl install scripts would only run when required (previously they ran every time make was invoked in Buildroot). The patch for these changes can be found in `tcl-build-fix.patch` in appendix F.

The Tcl library was written in C, compiled into a shared library for loading by the Tcl interpreter. Appendix E contains the source code of this library. Similar functionality to the C library was created, however a procedure name change was made so that it would be easier to distinguish whether one was referring to a function from the C or Tcl library.

Figure 2.13 shows an example of the usage of the Gumsense Tcl library.

It was necessary to translate the C structure that represents a sampling job into some form of Tcl structure. Ideally, the Tcl procedure to read the table of jobs would return an associative array containing the various settings that a job has. However, Tcl provides no methods for creating arrays from C extensions. Therefore the C routines were written to return a list containing the job attributes. The offset of the data within the list related to its value. Table 2.4 lists the indices of these values.

This interface was evidently unintuitive to use, since it involves the use of arbitrary indices. A solution, suggested by by Klaus-Peter Zauner, was that the Tcl C extension function could return a list interleaved with property names and values. This list could then be used by the Tcl `array get` procedure to create an associative array with an intuitive name-value mapping. This solution was not implemented from within the Tcl C extension, as time was of the essence. The simple Tcl routine shown in figure 2.14 can be used to translate a job represented as a list into an array.

```
#!/usr/bin/tclsh8.4
lappend auto_path .
package require gumsense

gumsense::open

set data [gumsense::read_readings]
gumsense::clear_readings
set t [lindex $data 0]
set readings [lindex $data 1]
set num_readings [llength $readings]
set time_off [expr {$t - $num_readings}]
set n 0

puts "$num_readings readings read"

set values [list]
set channels [list]
set times [list]

foreach i $readings {
    lappend values [lindex $i 1]
    lappend channels [lindex $i 0]
    lappend times [expr {$time_off + $n}]

    set n [expr {$n + 1}]
}

set valstr [regsub -all " " $values +]
set chanstr [regsub -all " " $channels +]
set timestr [regsub -all " " $times +]
set getstr data=$valstr&channels=$chanstr&times=$timestr

puts $getstr
#Log data to remote host
exec wget -q -O /dev/null http://192.168.3.1/~rob/gum/log.php?$getstr

gumsense::set_wake_time [expr {[gumsense::read_time] + 300}]
exec /sbin/poweroff
```

FIGURE 2.13: An example showing the use of the Tcl library.

TABLE 2.4: The indices of the job attributes within a Tcl list representing a job.

Index	Description
0	Interval
1	Next Time
2	Power-up Time
3	Power Mask
4	Channels

```
proc job-to-array { job } {
    set a [linsert $job 4 channels]
    set a [linsert $a 3 power_mask]
    set a [linsert $a 2 power_up_time]
    set a [linsert $a 1 next_time]
    set a [linsert $a 0 interval]

    array set jobarr $a
}
```

FIGURE 2.14: Translating a list of job properties into the array `jobarr`.

In order to integrate the Gumsense Tcl library into the Tcl package system, it was necessary to alter the Buildroot scripts that configure and install Tcl so that the package management scripts would be installed correctly. Also, the way that the Tcl Buildroot makefile invoked the Tcl `configure` script was altered so that the paths in which the Tcl package system looked for package files was correct. These changes can be found in `tcl-package-build-fix.patch`.

TABLE 2.5: Signals that can be sent to the init process.

Signal	Function
SIGUSR1	Halt
SIGUSR2	Power Off
SIGTERM	Reboot

2.6 Powerdown

The 5V rail delivered to the Gumstix must be switched off when the Gumstix has finished all of its processing. Of course, it is necessary for the programs running on the Gumstix to finalise their operations correctly prior to the rail being removed. The best place to hook into this procedure is where the power would normally be disabled on any other Linux system. A short investigation was conducted to discover how to integrate this into the software stack.

It was known that by running `/sbin/poweroff` the procedure to power down the system would be initiated. The `poweroff` program is part of the BusyBox set of tools, and causes the `halt_main()` function found in `init/halt.c` of the BusyBox code to be executed. This function is also executed when `/sbin/reboot` or `/sbin/halt` is invoked. The `halt_main()` function performs two different functions, depending on whether BusyBox is configured to provide the `init` process or not. In the case of the Gumstix, BusyBox provides `init`. `halt_main()` sends a signal to the `init` process. The signal that is sent depends on the requested operation. Table 2.5 shows the mapping of functionality to different signals.

When init receives the `SIGUSR2` signal, it enters the `halt_signal()` routine found in `init/init.c` of the BusyBox source. This function sleeps for 2 seconds before continuing to call `init_reboot()`. The purpose of the 2 second pause is to allow for any data resident in the serial console output buffer to reach its destination before the power is switched off. The pause was removed in the Gumsense buildroot tree in order to reduce the amount of energy consumed during Gumsense shutdown.

The `init_reboot()` function forks the init process (for reasons that are unimportant), and one of the resultant processes calls `reboot()`, which through the C library (uClibc) performs the reboot syscall. Linux's `sys_reboot()` function is invoked by this. If the `pm_power_off` function pointer is not NULL, then it is eventually called. If it is NULL, then the poweroff sequence is converted into a halt sequence. A halt results in the system entering an infinite loop rather than powering down. An overview of this sequence of calls is provided in figure 2.15.

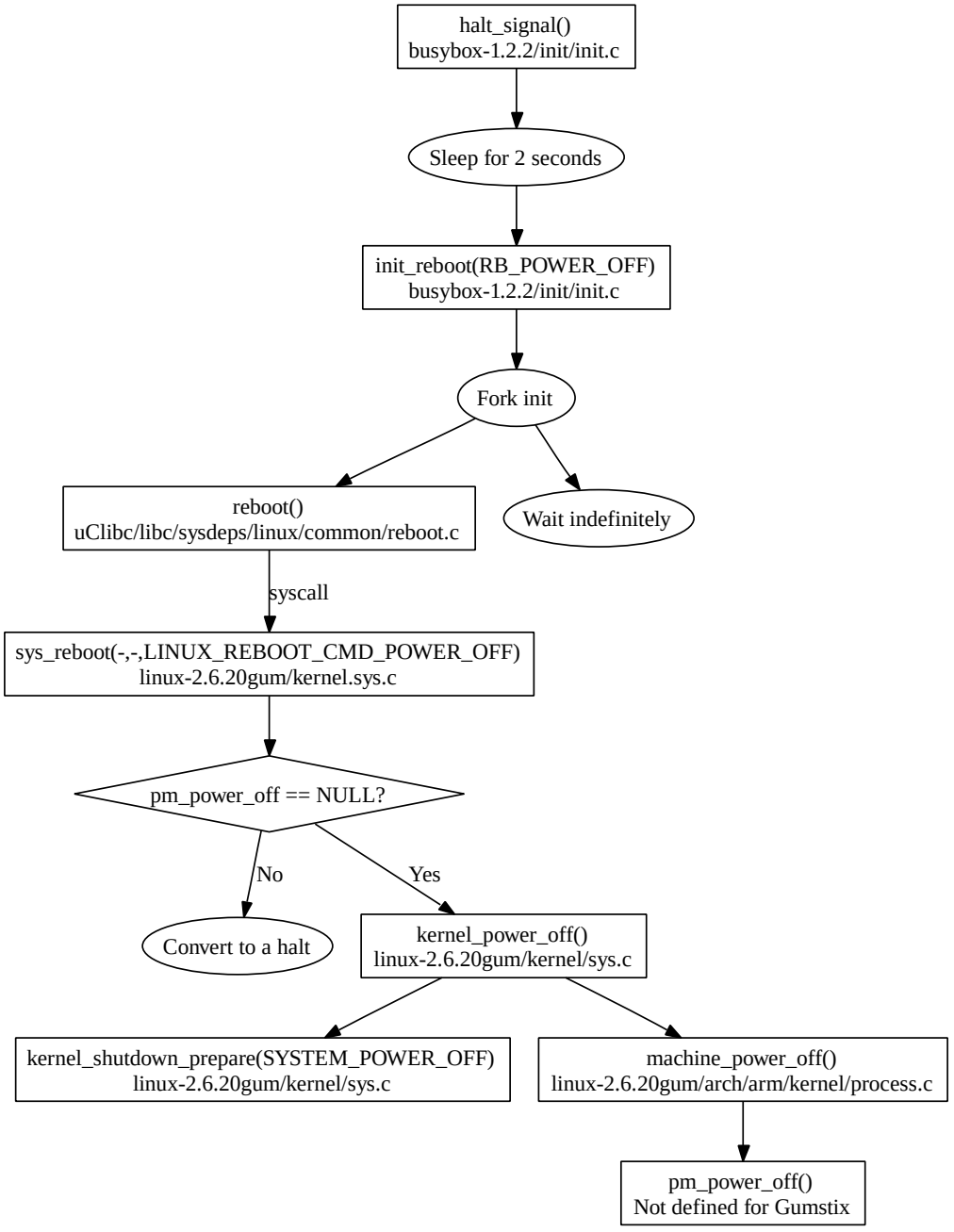

FIGURE 2.15: The sequence of function calls that are made by BusyBox init when approaching power-down. Functions are shown in rectangles and actions are shown in ellipses.

As a result of this investigation, it was decided that the best method of hooking into the poweroff code was to set the `pm_power_off` function pointer to refer to a function that sends the command to the Gumsense to turn off the Gumstix power. If `pm_power_off` is set to be non-NULL then, as per figure 2.15, it is called after driver finalisation.

In order to send the power-down command to the Gumsense, a pointer to an `i2c_client` structure is required. The only practical method of getting a pointer to an `i2c_client` structure is by implementing an I^{2}C chip kernel driver for the Gumsense.

2.6.1 Kernel Driver

Using the information regarding writing drivers that are part of the I^{2}C subsystem in `Documentation/i2c` of the Linux sources, and chapter 2 of [46] for information on how to write a kernel module, a kernel driver was written.

Appendix C contains the source code of the kernel module. This file was placed in the `drivers/i2c/chips` directory of the Linux source code. Modifications were made to the makefile and configuration file of the same directory. The patches `makefile.patch` and `kconfig.patch`, which can be found in appendix C, contain these changes.

The `gumsense` kernel module was written as an I^{2}C "chip" driver. The function `gumsense_init()` gets called when the module is loaded. On line 163 of `gumsense.c`, this registers the Gumsense I^{2}C driver and, on line 168, registers the local function `gumsense_power_off` to be called when the power needs to be terminated.

The `gumsense_driver` structure, defined on line 67 of `gumsense.c`, describes the "attach" and "detach" functions of the I^{2}C chip driver. `gumsense_attach_adapter()` initiates a scan of the bus for I^{2}C clients that may be the Gumsense. When a candidate is found, the Gumsense I^{2}C identity command is issued by `gumsense_power_off()`. If the correct identity bytes are received, then a new `gumsense_data` structure is created. This contains a pointer to the `i2c_client` structure, that will later be required to power down the device, and a `list_head` structure, which allows a list of Gumsense boards to be stored. This list of clients allows for more than one Gumsense board to be on the same I^{2}C bus (however, changes to addresses would need to be made).

Chapter 3

Conclusions and Future Work

3.1 Evaluation

This project achieved most of what it set out to do. The performance of the Gumsense development platform was enhanced and a programming environment suitable for the rapid implementation of algorithms was created.

The time allocated to the project was not sufficient to develop such algorithms, as the change in Gumsense architecture took longer than initially anticipated. In particular, the time required to redesign the PCB was far greater than initially expected.

During the project, the project goals were re-evaluated because of an initially unknown factor; the Gumstix sleep current. After an analysis of this factor, it was decided that the design should be changed to switch the Gumstix fully off during inactive periods. This change allows energy to be saved in WSN applications, where long periods of processor activity are common.

3.2 Future Work

3.2.1 Hardware Development

Aside from the small set of PCB corrections that were detailed earlier, there are improvements that could be made to the design to enhance functionality. The addition of a method of measuring and logging the device power usage (a feature of the LEAP platform [16]) may be useful for developing "power-aware" algorithms. Furthermore, the addition of a set of jumpers, or perhaps a DIP switch to the design would allow the selection of device mode in the field.

The standby current of the Gumstix is mainly due to the DRAM refresh current. If the DRAM was to be replaced with MRAM then the Gumstix standby current would potentially be much less.

3.2.2 Software Development

Great energy savings could be achieved by reducing the Gumstix boot time. A large segment of the boot time is due to the scanning of the JFFS2 volume, all of which must be read before it is mounted. It should be possible to develop a file-system that stores metadata on a separate non-volatile memory that is capable of many more read-write cycles than the Gumstix flash. This memory could be placed on the Gumsense. This would completely eliminate the need to read the entire contents of the flash memory at boot time, and therefore substantially reduce boot time.

Further reductions in boot time could potentially by achieved by modifying the system boot process. Of course, the shutdown time should also be optimised.

Bibliography

[1] GJ Pottie and WJ Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[2] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survery. *Computer Networks*, pages 393–422, December 2001.

[3] D. Estrin. Wireless sensing systems: from eco-systems to human-systems. A Campbell lecture given at the University of Southampton, 1st May 2007.

[4] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a sensor network expedition. *First European Workshop on Wireless Sensor Networks (EWSN04), January*, 2004.

[5] B. Schott, M. Bajura, J. Czarnaski, J. Flidr, T. Tho, and L. Wang. A modular power-aware microsensor with ¿1000X dynamic power range. *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 469–474, 2005.

[6] B.A. Warneke, M.D. Scott, B.S. Leibowitz, L. Zhou, C.L. Bellew, J.A. Chediak, J.M. Kahn, B.E. Boser, and K.S.J. Pister. An Autonomous 16mm 3 Solar-Powered Node for Distributed Wireless Sensor Networks. *Proceedings of Sensors 02*, 2002.

[7] S. Roundy, P.K. Wright, and J. Rabaey. A study of low level vibrations as a power source for wireless sensor nodes. *Computer Communications*, 26(11):1131–1144, 2003.

[8] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient MAC protocol for wireless sensor networks. *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, 3, 2002.

[9] S. Goel, A. Passarella, and T. Imielinski. Using buddies to live longer in a boring world. *Proceedings of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops*, 2006.

[10] A. Gupta, C. Gui, and P. Mohapatra. Exploiting Multi-Channel Clustering for Power Efficiency in Sensor Networks. *Communication System Software and Middleware, 2006. Comsware 2006. First International Conference on*, pages 1–10, 2006.

[11] N.D. Georganas. A coverage-preserving node scheduling scheme for large wireless sensor networks. *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 32–41, 2002.

[12] L. Krishnamachari, D. Estrin, and S. Wicker. The impact of data aggregation in wireless sensor networks. *Distributed Computing Systems Workshops, 2002. Proceedings. 22nd International Conference on*, pages 575–578, 2002.

[13] D. Petrovic, RC Shah, K. Ramchandran, and J. Rabaey. Data funneling: routing with aggregation and compression for wireless sensor networks. *Sensor Network Protocols and Applications, 2003. Proceedings of the First IEEE. 2003 IEEE International Workshop on*, pages 156–162, 2003.

[14] K. Sohrabi, J. Gao, V. Ailawadhi, and GJ Pottie. Protocols for self-organization of a wireless sensor network. *Personal Communications, IEEE [see also IEEE Wireless Communications]*, 7(5):16–27, 2000.

[15] M. Rahimi, R. Baer, O.I. Iroezi, J.C. Garcia, J. Warrior, and M. Srivastava. Cyclops: in situ image sensing and interpretation in wireless sensor networks. *Proceedings of the 3rd international conference on Embedded networked sensor systems*, pages 192–204, 2005.

[16] D. McIntire, K. Ho, B. Yip, A. Singh, W. Wu, and W.J. Kaiser. The low power energy aware processing (LEAP) embedded networked sensor system. *Proceedings of the fifth international conference on Information processing in sensor networks*, pages 449–457, 2006.

[17] JL Hill and DE Culler. Mica: a wireless platform for deeply embedded networks. *Micro, IEEE*, 22(6):12–24, 2002.

[18] R.D. King, K.E. Whelan, F.M. Jones, P.G.K. Reiser, C.H. Bryant, S.H. Muggleton, and D.B. Kell. Functional genomic hypothesis generation and experimentation by a robot scientist. *Nature*, 427:247–252, 2004.

[19] N. Matsumaru, F. Centler, K.P. Zauner, and P. Dittrich. Self-adaptive Scouting—Autonomous Experimentation for Systems Biology. *Applications of Evolutionary Computing*, 3005:52–62.

[20] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: a wireless sensor network testbed. *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 483–488, 2005.

[21] K. Martinez, R. Ong, and J. Hart. Glacsweb: a sensor network for hostile environments. *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pages 81–87, 2004.

[22] Gumstix - way small computing. `http://www.gumstix.com/`, 2007. As read on 9th May 2007.

[23] Atmel Corporation. *8-bit AVR Microcontroller with 128K Bytes In-Sytem Programmable Flash - ATmega103*, February 2007.

[24] Gumstix Inc. Features of the gumstix basic motherboard. `http://docwiki.gumstix.org/Basix_and_connex#Features_of_the_gumstix_basix_motherboard`. As read on 9th May 2007.

[25] Linux v2.6.20 source code. `http://www.kernel.org/`. As read on 9th May 2007.

[26] Intel. *Intel PXA255 Processor Developer's Manual*, January 2004.

[27] Gumstix 60 pin connector layout. `http://docwiki.gumstix.org/Gumstix_motherboard_I/O`. As read on 8th May 2007.

[28] Intel. *Intel PXA255 Processor Electrical, Mechanical, and Thermal Specification*, February 2004.

[29] Micron Technology Inc. *MT48LC64M4A2 Synchronous DRAM Datasheet*, 1999.

[30] Craig Hughes. `http://article.gmane.org/gmane.linux.distributions.gumstix.general/1569`, February 2005.

[31] Bernard Blackham. `http://article.gmane.org/gmane.linux.distributions.gumstix.general/15119/match=boot+time`, August 2006.

[32] Microchip Technology Inc. *PIC16F87XA Data Sheet*, 2003.

[33] Atmel Corporation. *ATtiny28L/V Datasheet*, 2006.

[34] Texas Instruments Inc. *MSP430x15x, MSP430x16x, MSP430x161x Mixed Signal Microcontroller*, October 2002.

[35] Philips Electronics. *LPC2141/42/44/46/48 Product Data Sheet*, August 2006.

[36] Maxim Integrated Products. *DS1337 I^2C Serial Real-Time Clock*, 2006.

[37] Cadsoft online: Home of the eagle layout editor. `http://cadsoftusa.com/`, 2007. As read on 11th January 2007.

[38] Analog Devices Inc. *AD623 Datasheet. Single Supply, Rail-to-Rail, Low Cost Instrumentation Amplifier*, 1999.

[39] Maxim Integrated Products. *MAX8881 Datasheet. 12V, Ultra-Low-I_Q, Low-Dropout Linear Regulators with POK*, March 2004.

[40] mspgcc - gcc toolchain for msp430. `http://mspgcc.sf.net/`, 2007. As read on 11th January 2007.

[41] SBS Implementers Forum. *System Management Bus (SMBus) Specification*, August 2000.

[42] Texas Instruments Inc. *MSP430x1xx Family User's Guide*, 2006.

[43] Gumstix Inc. Gumstix technical publications: Thumbstix schematics. `http://pubs.gumstix.org/boards/THUMBSTIX/PCB00021-R633/`. As viewed on 9th May 2007.

[44] USB Implementers Forum, Inc. *Universal Serial Bus Specification*, 2.0 edition, April 2007.

[45] A c library for embedded linux. `http://www.uclibc.org/about.html`. As read on 9th May 2007.

[46] *Linux Device Drivers, Third Edition.* O'Reilly Media Inc., 2005.

Appendix A

Gumsense Schematics

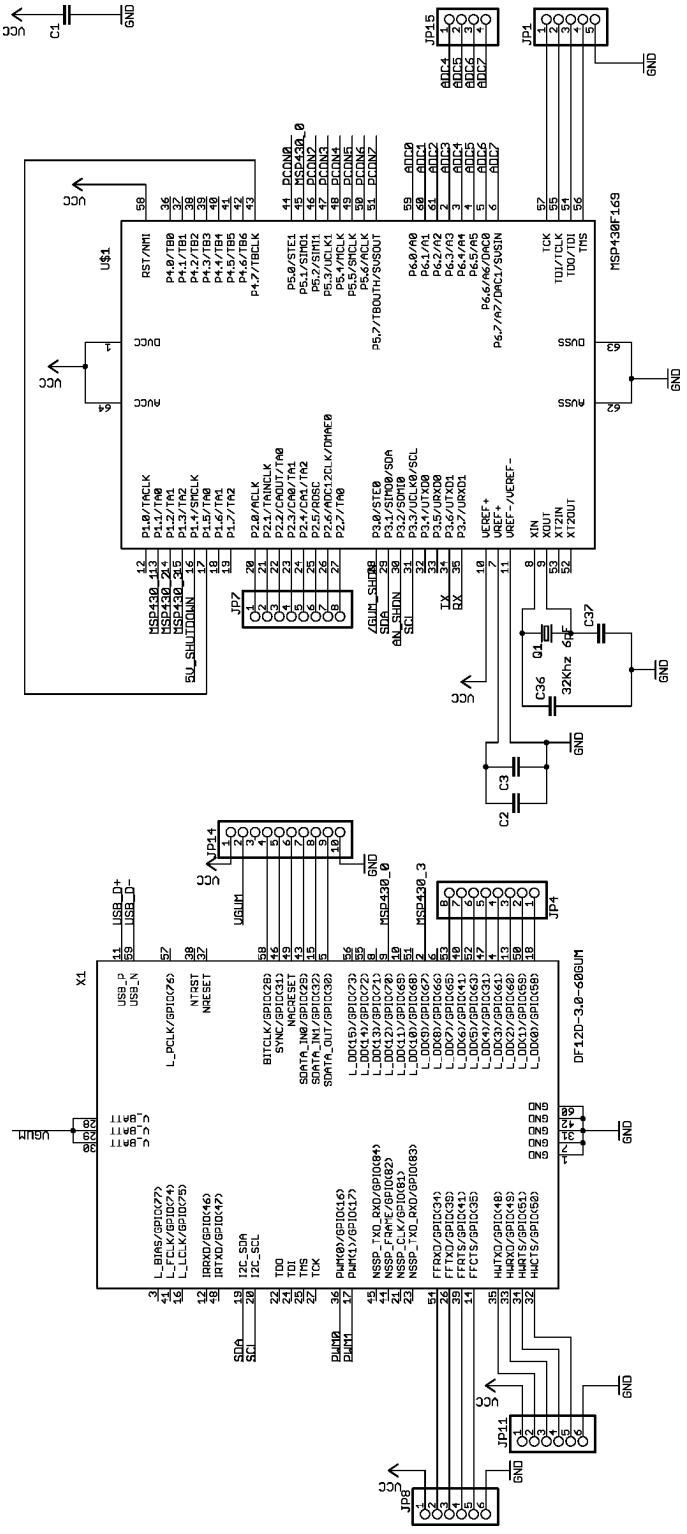

FIGURE A.1:

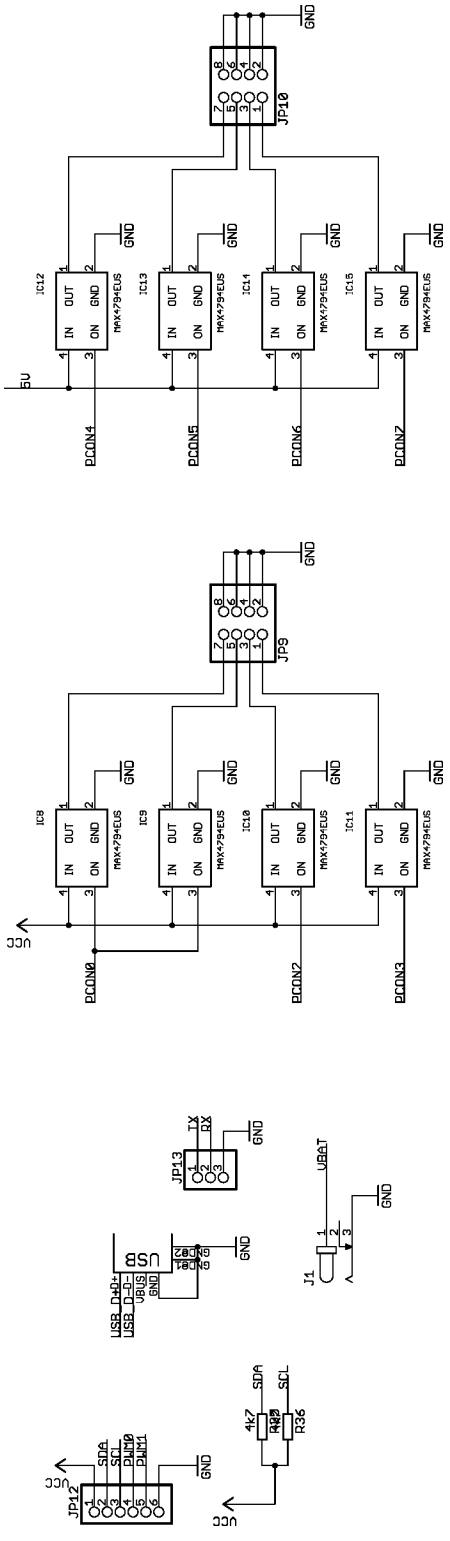
JP10
JP9
JP13
JP12
IC8
IC9
IC10
IC11
IC12
IC13
IC14
IC15
MAX4794EUS
IN
OUT
ON
GND
5V
VCC
VBAT
PCON0
PCON2
PCON3
PCON4
PCON5
PCON6
PCON7
TX
RX
SDA
SCL
PWM10
PWM11
USB D+D+
USB D-D-
VBUS
4k7
R35
R36
J1

FIGURE A.2:

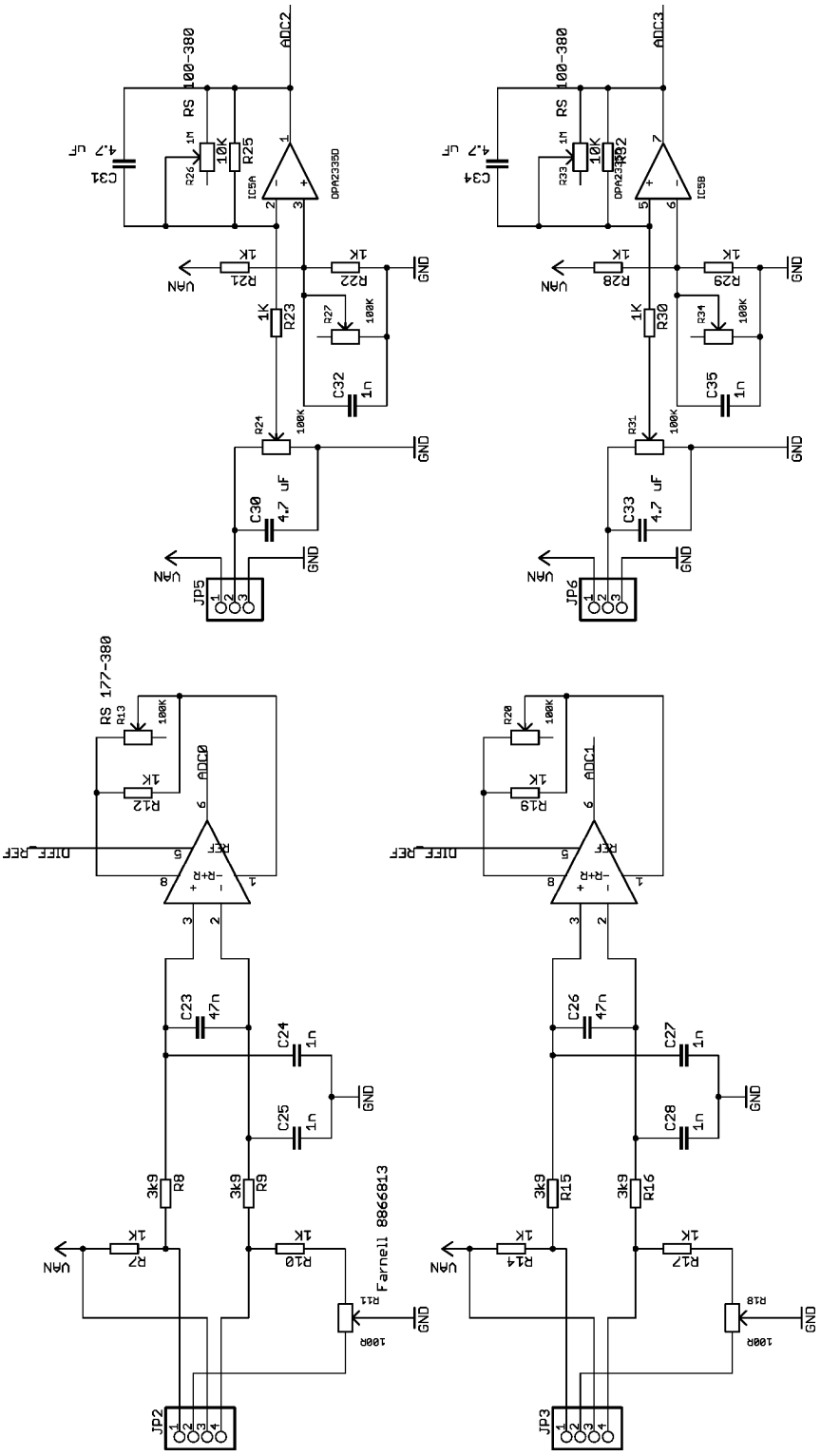
ADC2
ADC3
ADC0
ADC1
RS 100-380
RS 177-380
DIFF_REF
GND
VAN
JP2
JP3
JP5
JP6
Farnell 8866813

FIGURE A.3:

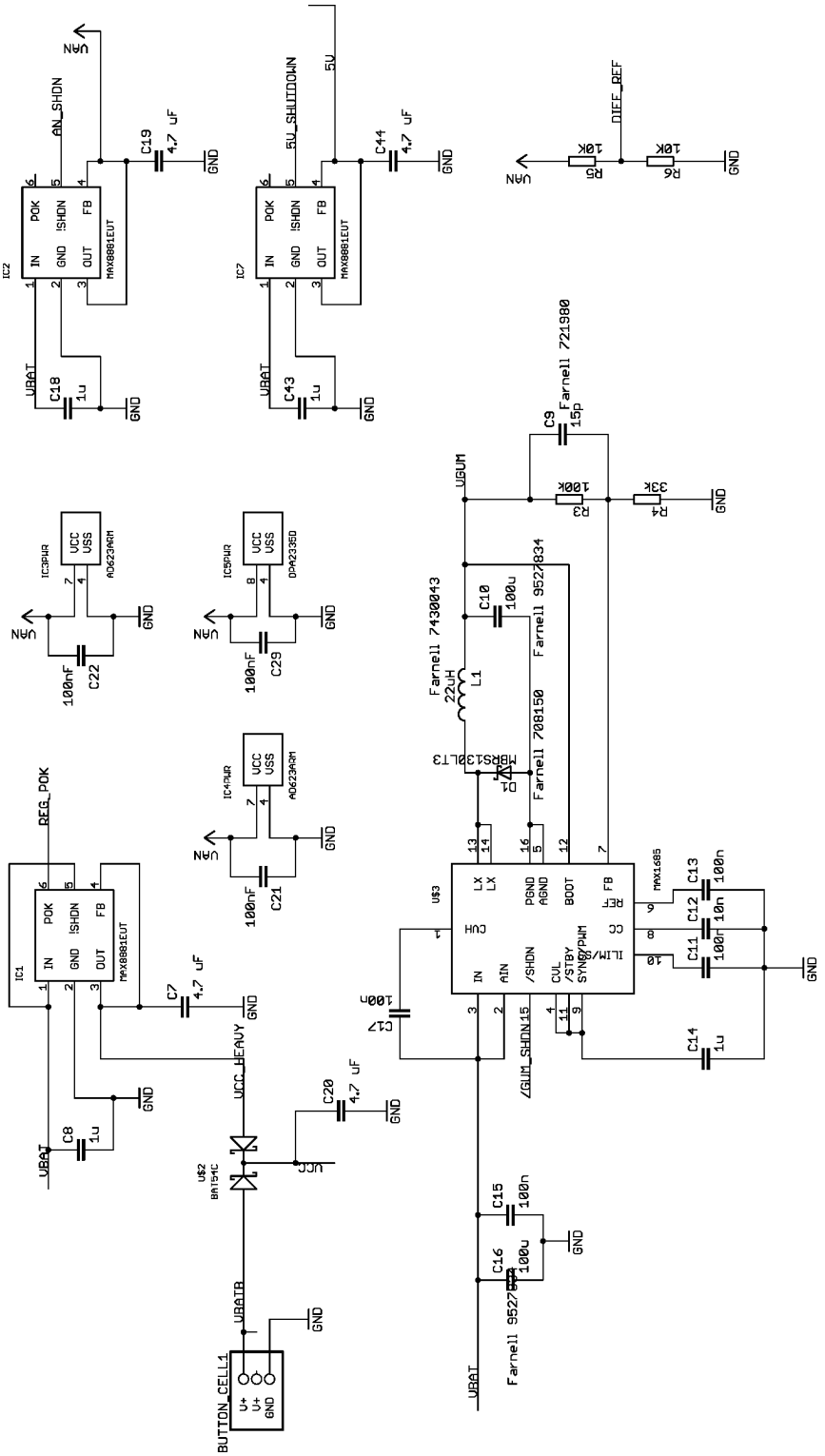
5VN
AN_SHDN
C19
4.7 uF
GND
IC2
MAX8881EUT
POK
!SHDN
FB
IN
GND
OUT
VBAT
C18
1u
5V
5V_SHUTDOWN
C44
4.7 uF
IC7
C43
DIFF_REF
R5
10K
R6
IC3PWR
VCC
VSS
AD623ARM
C22
100nF
IC5PWR
DPA2336D
C29
IC4PWR
C21
REG_POK
IC1
C7
VCC_HEAVY
C8
C20
US2
BAT54C
VBATB
BUTTON_CELL1
U+
U-
VBUM
C9
15p
Farnell 721980
R3
100K
R4
33K
C10
100u
Farnell 9527834
Farnell 7430043
22uH
L1
Farnell 708150
MBRS130LT3
D1
US3
LX
PGND
AGND
BOOT
FB
CUR
CC
ILIM/SS/PWM
AIN
/SHDN
CUR
/STBY
SYNC
MAX1685
C13
100n
C12
10n
C11
100n
C17
100n
C14
1u
/GUM_SHDN15
C15
100n
C16
100u
Farnell 9527834

FIGURE A.4:

TABLE A.1: Bill of materials for the new Gumsense design.

Part Number	Description	Quantity
DF12(3.0)-60DP-0.5V(86)	Gumstix Connector	1
MSP430F169	Microcontroller	1
AD623	Instrumentation Amplifier	2
OPA2335D	Dual Operational Amplifier	1
	100 Ohm SM Trimmer	2
	100K SM Trimmer	6
	1M SM Trimmer	2
MAX4794EUS	Current Limiting Analogue Switch	8
MAX8881EUT50-T	Voltage Regulator	1
MAX8881EUT33-T	Voltage Regulator	2
MAX1685EEE	Switchmode Step-down converter	1
	USB Connector	1
	Power Jack	1
	32 Khz Watch Crystal	1
	Button Cell	1
	Button Cell Holder	1
	22uH Inductor	1
MBRS130LT3	Diode	1
BAT54C	Dual Diode	1
	2x4 Pin Header	2
	10 Pin Header	1
	8 Pin Header	2
	6 Pin Header	3
	4 Pin Header	3
	5 Pin Header	1
	3 Pin Header	3
	100k 0603 Resistor	1
	33k 0603 Resistor	1
	1K 0603 Resistor	12
	3k9 0603 Resistor	4
	10K 0603 Resistor	4
	4k7 0603 Resistor	2
	1nF 0603 Capacitor	6
	47nF 0603 Capacitor	2
	4.7uF 0603 Capacitor	8
	100n 0603 Capacitor	8
	15p 0603 Capacitor	1
	10n 0603 Capacitor	1
	100u 1812 Capacitor	2
	1uF 0603 Capacitor	4

Test