Development of an Internet Gateway for a Wireless Sensor Network

Gary de Villiers

Development of an Internet Gateway for a Wireless Sensor Network

Gary de Villiers

Centre for Instrumentation Research

Cape Peninsula University of Technology

A thesis submitted for the degree of

Magister Technologiae: Electrical Engineering

September 2012

Declaration

I, Gary de Villiers, declare that the content of this thesis represents my own work, and that this thesis has not been previously submitted for examination towards any academic qualification. Furthermore, it represents my own opinions and not necessarily those of the Centre for Instrumentation Research or Cape Peninsula University of Technology.

Signature of author .....

Cape Town
September 2012

I would like to dedicate this thesis to my wife Melissa who has supported and encouraged me throughout this study.

Acknowledgements

I would like to thank my supervisor Professor Richardt Wilkinson of the CIR for his support during the study. My sincerest thanks goes to my co-supervisor Andrew van der Byl for his encouragement, assistance and time given to me so generously throughout this study. My thanks to Professor Gerhard De Jager of UCT for proofreading the final draft of this thesis.

The financial assistance of the National Research Foundation, Cape Peninsula University of Technology and Eskom towards this research is acknowledged and greatly appreciated. The opinions expressed in this thesis and the conclusions arrived at are those of the author and are not necessarily to be attributed to the respective establishments mentioned.

And finally to my wife, Melissa, thank you for your unfailing patience and outstanding encouragement throughout my studies!

Abstract

Wireless Sensor Networks (WSNs) are being employed frequently to gather data from an ever-increasing variety of environments and phenomena. WSNs offers users in the industrial and academic community a low power, unobtrusive, adaptable and wireless alternative to traditional sensing equipment. This study focuses on the development of a configurable, low power and cost-effective gateway that will link a WSN to the Internet. In addition, the gateway provides onsite storage for the data collected by the WSN which may be uploaded at a scheduled time to a computer at a specified Internet address through a GSM or Ethernet link. Furthermore, the gateway provides a suitable platform capable of executing a variety of multitasking operating systems such as Windows Embedded CE or Linux. This multi-platform support permits the gateway the flexibility, through custom user applications, to adapt its functionality to the requirements of a WSN and the environments into which these networks may be deployed. This study has been successful in the design and development of a low-cost gateway solution, and has produced a functional prototype system.

List of abbreviations

ADC	Analogue to Digital Converter
APN	Access Point Name
BGA	Ball Grid Array
BSP	Board Support Package
CPU	Central Processing Unit
CS	Chip Select
CTS	Clear To Send
DLL	Dynamic Link Library
DNS	Domain Name Server
ECC	Error Correcting Code
EK	Evaluation Kit
EMI	Electromagnetic Interference
ENIG	Electroless Nickel Immersion Gold
ESL	Equivalent Series Inductance
ESR	Equivalent Series Resistance
FTP	File Transfer Protocol
FTDI	Future Technology Devices International
GPIO	General Purpose Input Output
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HTTP	Hyper Text Transfer Protocol
IC	Integrated Circuit
IDE	Integrated Development Environment
IO	Input Output
KITL	Kernel Independent Transport Layer
LED	Light Emitting Diode
LDO	Low Dropout regulator
MAC	Media Access Control
MISO	Master In Slave Out
MLCC	Multi Layer Ceramic Capacitor
MMC	Multi Media Card
MMU	Memory Management Unit
MOSI	Master Out Slave In
.NETCF	Dot Net Compact Framework
OS	Operating System
OEM	Original Equipment Manufacturer
PB	Platform Builder
PCB	Printed Circuit Board
PDP	Packet Data Protocol
PHY	Physical
PIFA	Planar Inverted F Antenna
PQFP	Plastic Quad Flat Pack
QFP	Quad Flat Pack
RAM	Random Access Memory
RF	Radio Frequency
RISC	Reduced Instruction Set Computing
RMII	Reduced Media Independent Interface
RTS	Request To Send
SBC	Single Board Computer
SCLK	Serial Clock
SD	Secure Digital
SDK	Software Development Kit
SDRAM	Synchronous Dynamic Random-Access Memory
SMD	Surface Mount Device
SPI	Serial Peripheral Interface
TCP/IP	Transmission Control Protocol/Internet Protocol
TFTP	Trivial File Transfer Protocol
UART	Universal Asynchronous Receiver/Transmitter
UI	User Interface
USB	Universal Serial Bus
VCP	Virtual Comm Port
VOIP	Voice Over Internet Protocol
VS	Visual Studio
WSN	Wireless Sensor Network
WLAN	Wireless Local Area Network

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A trend of the modern era is the miniaturisation of technology. Nearly every facet of modern-day life has been affected by this drive for smaller, lighter, faster and smarter devices. Vast technical improvements have been made in the development of many devices such as microprocessors and radio transceivers where their size and power consumption have been greatly reduced. The increase in the processing capability of low power microprocessors and the integration of a radio transceiver has lead to a new paradigm in sensing capability. Environments, equipment, structural and even biological platforms, which were once monitored by single sensors and vast quantities of cable to transmit power and captured data, can now be monitored by small, unobtrusive wireless sensors providing unparalleled spatial and parametric sensing.

A vast variety of sensing devices used in such applications is available. Typical devices incorporate a microcontroller, radio transceiver, data memory, analogue to digital converter (ADC) and electronic sensors. Sensing devices such as the TelosB, FireFly and Mica2 incorporate a small 8-bit microcontroller which offer a moderate data processing capability and are adequate for the majority of applications. Where high performance data processing or larger data throughput is required, devices such as the IMote 2.0 (Crossbow Technologies, 2012), which incorporates a 32-bit ARM processor and MMX DSP coprocessor, are available.

equates to a theoretical limit of 2^{64} devices. Each sensor has a unique Media Access Control (MAC) address permitting system diagnostic tests and absolute sensor identification on the network.

The network may be arranged in one or more combinations of the following topologies: star, peer to peer or mesh. A star network consists of a central control sensor which coordinates the network and routes data from outlying sensors to adjacent sensors. The peer to peer topology forms the basis for the mesh network where any sensor is able to communicate with any other sensor within range of its radio transceiver. These are termed *ad hoc* networks as the sensors determine their own data routing paths and have the ability to self heal (a self healing network is able to maintain its operational integrity by routing the data around any sensors that may have failed).

A considerable amount of research has been done to date on many aspects of the design, implementation and operation of sensor networks. Research focus areas include:

- small footprint Operating Systems (OS) such as *TinyOS* (TinyOS, 2010) tailored to be executed on sensor devices with constrained resources.
- specifications for the physical and MAC layers and networking topologies as defined by the IEEE 802.15.4 standard
- routing protocols to improve the reliability and efficiency of data transfer between sensors over low power radio links (Woo, Tong and Culler, 2003)
- sensor network data security

In addition to the research relating to aspects of the network operation, considerable work has been done to develop and improve the hardware of the sensor devices. Areas of interest include the reduction of power consumption to extend a sensor's battery life, the reduction of the size and mass of the sensors to permit their deployment in environments not easily monitored previously and the most efficient design of the transceiver's antenna to ensure maximum reliability of the transfer of data for the least amount of input power.

Typical sensors incorporate a microcontroller, memory and a low powered radio transceiver. Sensors such as the TelosB incorporate several onboard electronic sensors capable sampling both visible and infra red light, temperature and humidity in the immediate environment. Additional sensing capabilities resulting in increased versatility in varied deployment applications within, but not limited to, the automotive, chemical, environmental and electrical disciplines may be extended by the addition of appropriate sensors.

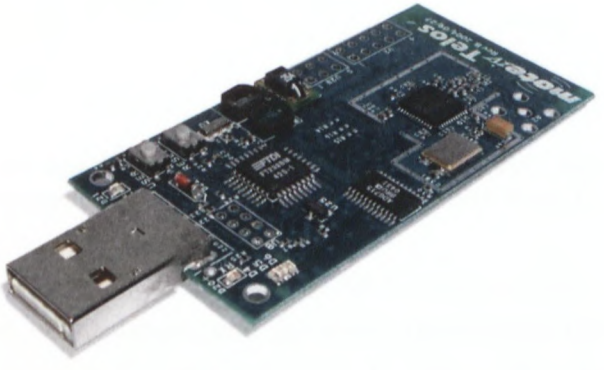


Figure 1.1: The TelosB sensor node shown (Polastre et al., 2005) incorporates three onboard sensors and an IEEE 802.15.4 compliant radio transceiver driving a PIF antenna etched onto the PCB. The transceiver transfers data at a rate of up to 250 kbits/s over a distance of 100m in an outdoor environment. Programming and configuring the node is done via the USB plug at the top of the PCB. A 16-pin expansion port permits analogue, digital, control and serial communication signals to be accessible to external equipment. The sensor may be powered by batteries which are housed in a casing located underneath the sensor (not shown) when necessary. Three programmable LEDs, a reset and general pushbutton switch are available to the user. The TelosB is built into a compact form factor that measures approximately 65mm x 31mm x 6mm, excluding the battery casing (Willow Technologies Ltd, 2012).

WSNs are deployed into environments to perform monitoring tasks. At some point the data captured by the sensors in the network will be required for use outside the deployment environment. All the data collected by the sensors will need to be transferred to a single point and to be made available for external use. Any one of the sensor nodes within the network may be nominated as the collection point or *sink node* and is usually connected to a device with extended

functionality such as providing Internet access. The device that performs a go-between the sensor network and the outside world is termed a *gateway*.

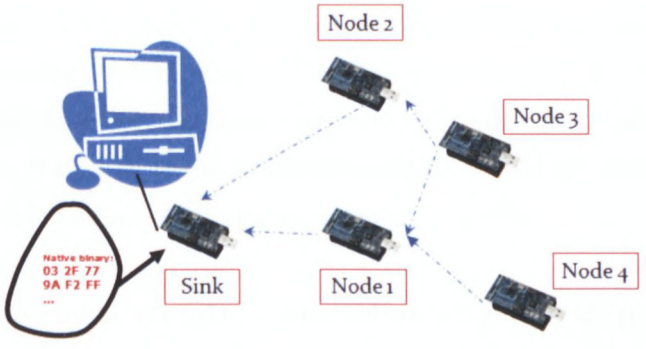


Figure 1.2: This figure shows how a sink node is an integral part of a typical WSN deployment and links the entire network to a gateway device (Shanghai Jiao Tong University, 2011).

The gateway is a fundamental component of a sensor network and is often neglected in terms of performance and practical implementation. The gateway's fundamental function is to provide access to the data collected by the sensor network. Additional functionality such as GPS capability, persistent data storage, network re-programming and configuration may also be provided. Remote access to the gateway may be achieved through the Internet by multiple access media such as GSM, WiFi or Ethernet.

The research discussed in section 1.1 provides details of the gateways implemented in actual WSN deployments and highlights the limitations and disadvantages of each of them. A common theme found throughout the case studies is the need for a generic, feature rich, low power, low cost and robust gateway to meet the requirements of typical WSN deployments. It is the focus of this study to determine the specification required for a gateway based on real deployments and to design and a construct a generic gateway to fulfill this need.

1.1 Previous gateway implementations

The following case studies examine the gateways implemented for WSN deployments in various environments. An assessment of the implementation, hardware platform, operating system, any additional features and the likely cost was made of each gateway and the research outputs summarised in table 1.1. Any limitations that were found in these implementations would be avoided where possible during the development of the gateway in this study.

The WSN deployment on GDI (Mainwaring et al., 2002) presented researchers with several challenges due to the remote location of the island and the vast area of the habitats to be studied. The deployment architecture resulted in patches of Mica2 network sensors connected to the gateway via a transit network consisting of either a series of sensor nodes or a single-hop link.

The gateway consisted of a laptop computer executing PostgreSQL to provide time-stamped storage for data collected from the WSN and to perform regular data forwarding to a PostgreSQL database off-site. Internet access and remote connectivity was provided by a Hughes two-way satellite communication system for data retrieval and network monitoring. A further example of the use of a laptop computer as a gateway was for the deployment of a network consisting of TMote Sky sensors on the active Volcán Tungurahua volcano in central Ecuador which collected seismic and infrasonic data (Lorincz et al., 2006). The gateway consisted of a laptop to provide persistent data logging which was connected to the sensor network via a long range radio link. No mention is made of Internet connectivity in this deployment.

Laptop computers are able to execute powerful operating systems such as Windows or Linux, are moderately cost effective and offer several means of Internet connectivity, thus making them suitable for use as gateways. However, laptops are not easily serviceable in remote or isolated environments, are fragile and sensitive to exterior climatic conditions, have an excess of peripheral devices not required by a gateway and require large amounts of power to operate.

These attributes make laptop computers unsuitable as gateways for long term, remote WSN deployments.

A gateway for the PipeNet study (Stoianov et al., 2007) in the US and the study of power-efficient gateway design by Musaloiu-Elefteri, et al. (Musaloiu-Elefteri, Musaloiu-Elefteri and Terzis, 2008) were both based on the Stargate SPB400 Single Board Computer (SBC) by Crossbow Technology (Crossbow Technologies, 2010*b*).

The Stargate is based on an Intel PXA255 400MHz processor with 64MBytes of SDRAM, PCMCIA and Compact Flash slots; a daughter board that interfaces with the processor board provides 10-BaseT Ethernet, an RS232 communications port and a USB host. An additional board is required to interface the Stargate to Crossbows IRIS, MicaZ and Mica2 sensor networks. The Stargate is supplied preloaded with a distribution of Linux less than 10 MByte in size and includes basic drivers for the board.

Internet connectivity for the PipeNet deployment was provided by a GPRS modem and an 802.11 WiFi radio transceiver connected to the Stargate permitting either remote or local drive-by network debugging and configuration. Musaloiu-Elefteri employed an AmbiCom 802.11b WL1100 CF WiFi and a Sierra Wireless AirCard 860 HSDPA PCMCIA card to connect their gateway to the Internet. All data received by the gateway from the network was stored in a persistent memory ensuring zero data loss in the event of a system power failure or data link interruption during a periodic upload.

Data storage and transfer integrity relies on the gateway power supply which requires careful consideration before the gateway is deployed. In addition, the type of power supply selected may impact the deployment period of a sensor network. Power for the PipeNet gateway was obtained from a nearby lamppost thus guaranteeing an unlimited power source which was insensitive to the power requirements of the gateway. In contrast, the gateway implemented by Musaloiu-Elefteri was battery powered and required an external power controller to reduce

the consumption of their system in order to achieve the study's deployment period.

The power consumption of any system is affected by the number of components within the system. Many daughter boards and add-on modules were required to extend the functionality of the basic Stargate SBC to implement the gateways presented in these studies. Over and above increased power consumption, these additional components result in reduced system reliability, increased complexity and cost. Increased complexity results in additional potential points of failure in the gateway, thus increasing the degree of difficulty of an on-site repair. In addition, the manufacture of the Stargate SPB400 has been discontinued (Crossbow Technologies, 2009) thus making the servicing or replacement of an existing gateway based on this platform more difficult.

The Permasense II study (Tschudin, VonderMuehll and Gruber, 2009) researches phenomena such as rock conductivity, crack deformation and rock temperatures at different depths using a WSN distributed over several mountain rock faces in the Swiss Alps. The hostile environment required a demanding specification for the equipment used throughout the deployment. The sensor network would have to function for at least twelve months without intervention as the mountain slopes are normally unreachable for up to eight months at a time due to poor weather conditions with temperatures falling to -30 °C.

Researchers wanted to be able to manage and record context sensitive data from a distance and it was decided that a GPRS link would be employed to achieve this aim. TinyNode sensors were used in the deployment and one of these was incorporated as a sink node in the gateway design. A Gumstix Verdex ARM Computer-on-module executing Linux was used as a central processing unit interfaced with a MikroTik 433 Series WLAN server and GPRS modem to provide Internet connectivity (Keller and Beutel, 2009). Power for the gateway was provided by a battery that was charged by a photovoltaic array and associated control equipment.

Gumstix modules are available for online purchase directly from Gumstix as either stand-alone boards or in a starter pack tailored to a specific application. An example of a suitable starter pack is the Gumstix Overo Network pack (Crossbow Technologies, 2011) which would provide a very competent base for a WSN gateway. The pack includes an Overo Water Computer On Module (COM) board, a Tobi communications, USB and display expansion board, 8GB Micro SD card, power supply and miscellaneous connecting cables. The Water COM processor board incorporates a TI OMAP3530 applications processor based on an ARM Cortex-A8 CPU running at 720MHz, 512MB RAM, 512MB Flash and a micro SD card slot; the Tobi comprises a 10/100baseT Ethernet interface, high speed USB Host, HDMI over DVI for connecting a display monitor and general purpose digital and audio IO.

The Network Starter pack is advertised for sale on the Gumstix website for 327USD, as at February 2012. This price makes the use of the Gumstix system a potentially expensive option in terms of a South African context.

There are similarities between the Crossbow Stargate and Gumstix systems which tend to make these products less attractive for use as a gateway. Excluding availability and cost price, both these systems rely on the use of multiple daughter boards interfacing with a main processor board to form a functional gateway. This leads to redundancy of components such as audio and display capabilities which are not required by a gateway. These extraneous components add to the footprint size, cost of the system and increase the overall power consumption. In addition, the large arrays of finely-pitched contacts on the board interfaces are all potential points of failure in a system that is required to be robust, especially for external deployments where extremes of temperature, humidity, atmospheric pollutants and vibrations are abundant. The minute size of the Gumstix platform would decrease the potential for any on-site repair of a gateway should a failure occur in a remote deployment.

In 2009, researchers at the Cape Peninsula University of Technology developed a proof-of-concept gateway (Steenkamp, Kaplan and Wilkinson, 2009) based on an

Atmel AT91RM9200 evaluation kit. A Tmote Sky sensor was used as a sink node and a Sony Ericsson GSM modem to connect the gateway to the Internet. Linux was the selected operating system due to its open-source nature, freely available documentation and compilation tools and its flexibility in permitting a user to execute application-specific software on the platform. This study utilised the applications *serial forward*, `sf`, supplied in the *TinyOS* software development kit (SDK) to demonstrate the functionality of the gateway by forwarding all packets received from a WSN sink node connected to a USB-enabled serial port to an open TCP port on the gateway. The *serial listen* application, `sflisten`, also found in the *TinyOS* SDK, was modified and cross-compiled for the AT91RM9200 to timestamp and write packets of data received from the sink node by the `sf` application to an SD card for persistent storage and future transfer to a remote computer.

No hardware development was necessary for this study as a development kit was employed and is suitable only to demonstrate a proof-of-concept design, for software development and debugging. Similarly to the Gumstix solution, an evaluation kit for a particular processor usually implements all available peripheral functions in order to demonstrate the capabilities of the processor which are not necessarily required by a gateway.

Many commercial firms are now offering gateway equipment designed to operate with their own proprietary WSN and sensors. Crossbow Technology offers the eKo Pro Series system (Crossbow Technologies, 2010*a*), a complete WSN solution suitable for environmental monitoring. A typical eKo Pro system may consist of rugged eKo node wireless sensor nodes, an eB2110 eKo Base Radio and the eG2100 eKo Gateway.

The base radio implements the IEEE 802.15.4 radio standard and acts as the sink node for the WSN is linked to the gateway via a USB cable. The gateway executes the Debian Linux operating system (OS) and in addition is preloaded with eKoServe and Xserve which permits a user to view real-time data, run reports and set network alarm levels.

The gateway has USB and Ethernet connections and requires a 5V DC supply capable of delivering 4W.

Similarly, National Instruments (NI) produce a WSN system that permits seamless integration into the LabVIEW software suite (National Instruments Corporation, 2010*b*) by means of either the NI-9791 or NI-9792 (National Instruments Corporation, 2010*a*,*c*) ethernet gateways. The gateways incorporate a built in radio transceiver which implements the IEEE 802.15.4 specification co-ordinating communications with up to a maximum of thirty-six National Instruments WSN-32xx sensor nodes linking a distributed sensor network to a PC via ethernet or serial link. The NI-9792 incorporates a processor which operates at 533MHz, has 256MB of DDR2 SDRAM and 2GB of persistent data storage. The OS incorporates a web server permitting remote user access to captured data and system configuration settings via a web browser. The NI-9792 requires an external power supply capable of delivering a voltage between 9 and 30V at 9,5W.

Both the commercial gateways discussed rely on a host service or auxiliary equipment to provide access to the Internet and a power supply which may limit their suitability for remote exterior deployments.

In March 2011 Libelium launched its newest gateway product, the Meshlium Xtreme (Libelium Comunicaciones Distribuidas, 2011*a*,*b*). This gateway executes Debian Linux and provides support for 5 wireless standards WiFi, ZigBee, GPRS, Bluetooth and GPS and wired Ethernet giving a selection of methods for connecting a WSN to the Internet. It also stores sensor data locally in an internal database as well as on external Internet servers. This gateway may only be used for WSN based on the 802.15.4 ZigBee protocol which may limit its use as a generic gateway, however, it is housed in a robust aluminium enclosure suitable for exterior deployments.

Table 1.1 summarises the salient features and technologies implemented in the gateways discussed. It has been shown that a variety of differing platforms and technologies have been used to implement gateways for WSN deployments.

Study	Sensor type	Platform & Internet	OS	Power supply	Construction	Features	Cost
Wireless sensor networks for habitat monitoring (Mainwaring et al., 2002)	Mica2	Laptop Hughes satellite comms link.	Not known	Mains supply and UPS required Unsuited to battery operation High power consumption	Fragile construction. Unsuitable for exterior deployments	PostgreSQL Persistent data storage Periodic data upload	Moderate to high initial cost High monthly satellite Internet cost. Approx. 530USD (TS2 Technologie Satelitarne, 2012)
Deploying a Wireless Sensor Network on an Active Volcano (Lorincz et al., 2006)	TMote Sky	Laptop Link not specified	Not known	Mains supply and UPS required Unsuited to battery operation High power consumption	Fragile construction. Unsuitable for exterior deployments	Persistent data storage	Moderate to high initial cost
PIPENET: A Wireless Sensor Network for Pipeline Monitoring (Stoianov et al., 2007)	Intel Mote	Crossbow Stargate SPB400 (discontinued) GPRS modem 802.11 WiFi transceiver	Linux	Local mains supply	Several boards required for base system Board interface connectors are potential points of failure Redundant functionality	Persistent data storage Periodic data upload	Moderate initial cost
Gateway Design for Data Gathering Sensor Networks (Musaloiu-Elefteri et al., 2008)	MicaZ Mica2	Crossbow Stargate SPB400 (discontinued) HSDPA PCMCIA modem AmbiCom WL1100 CF 802.11 transceiver	Linux	Battery operation	Several boards required for basic system Board interface connectors are potential points of failure Redundant functionality	External power controller to extend battery life	Moderate initial cost
Technology for Permasense (Keller and Beutel, 2009)	TinyNode	Gumstix Verdex ARM Computer-on-module GPRS modem MikroTik 433 Series WLAN server	Linux	Battery operation with photovoltaic backup	Several boards required for basic system Board interface connectors are potential points of failure Redundant functionality	Operational over large temperature range	Moderate to high initial cost Gumstix Network starter pack 327USD (Crossbow Technologies, 2011)
Wireless Sensor Network Gateway (Steenkamp et al., 2009)	Tmote Sky	Atmel AT91RM9200 Evaluation kit (discontinued) Sony Ericsson GPRS modem	Linux	Battery operation with photovoltaic backup Mains supply	Robust single board construction Minimal interface connectors	Modified *TinyOS* functions `sf` and `sflisten` Persistent data storage TFTP server	High initial cost Approx. 1316USD (*Mouser Electronics*, 2012)
Commercial Company	**Sensor type**	**Platform & Internet**	**OS**	**Power supply**	**Construction**	**Features**	**Cost**
Crossbow Technology	Proprietary	Crossbow eKo Gateway eG2100 eB2110 Base radio Ethernet and USB Host required to connect to Internet	Linux	Mains powered 4W DC supply	Robust construction Unsuitable for exterior deployment	eKoServe and Xserve software Embedded HTTP server permitting real-time data viewing, reporting, network configuration	High initial costs Price on application
National Instruments	Proprietary	NI-9791 gateway NI-9792 gateway Ethernet Host required to connect to Internet	Not known	Mains powered 9.5W DC supply	Robust construction Unsuitable for exterior deployment	Embedded FTP and HTTP servers 2G persistent data storage	High initial costs Approx. 1600USD (Test & Measurement World, 2011)
Libelium	Proprietary	Meshlium Xtreme Ethernet GPRS 802.11 WiFi Bluetooth (optional)	Linux	Power over Ethernet Battery operation with photovoltaic backup Mains powered 5W DC supply	Robust construction Suitable for exterior deployments	Embedded FTP and HTTP servers MySQL Persistent data storage	Initial costs likely to be high Price on application

Table 1.1: Internet gateway technologies

Very few of the gateway examples cited are constructed in a robust manner suitable for an external environmental deployment. Imminent failure or compromised reliability of the WSN would certainly result if they were deployed.

In several cases, the initial procurement costs and recurring monthly costs of a functional gateway are high. In terms of a South African context, Internet connectivity via Satellite is very expensive. The reasons for a WSN deployment requiring such a link would probably need to be exceptional in order to justify a large recurring expense. South Africa has widespread GPRS coverage and provides an affordable connection to the Internet for the transfer of moderate amounts of data.

Linux was found to be the most common OS used in all the gateways discussed. This is probably due to Linux being freely available, adaptable to different hardware platforms and is supported by a large community of software developers. Linux provides a platform capable of supporting database applications such as PostgeSQL and MySQL which were used to store and manage data received from the WSN.

In addition, the majority of gateways analysed provided persistent storage for the data captured by the WSN. This is an important feature that is required to ensure the integrity of the WSN as a monitoring tool as the loss of any data would be deemed unacceptable. Scheduled, periodic uploading of the stored data to remote file or database servers is permitted by many of the gateways studied.

An important consideration for the deployment of any WSN and gateway are their power supplies. Typical sensor nodes run on batteries and are optimised to be as energy efficient as possible. A gateway typically consumes orders of magnitude more power than a sensor due to its increased functionality, computational power and data transfer to a remote site. Batteries are a viable proposition to power a gateway provided that its power consumption is kept within reasonable limits. A photovoltaic backup system would permit the charging of the supply batteries during the daylight hours thus permitting a gateway deployment in environments where mains power was unavailable.

The results of the research show that a generic, low cost and robust gateway solution does not seem to be available at present. The objective of this study seeks to address this problem by developing a functional prototype gateway based on the outcomes of the research in order to answer the research questions posed in the following section.

1.2 Research questions

The following research questions relating to the stated objective were answered by the outcomes of this study:

1. is it possible to design a gateway generic in nature to suit a variety of WSNs?

2. is it possible to design a low-power gateway with sufficient resources to execute a multi-tasking operating system, provide persistent data storage and scheduled transfers of the data to remote locations via the Internet?

3. is it possible to design a gateway that provides several Internet communication interfaces?

4. is it possible to design a gateway that included an interface for debugging and configuring a WSN and the gateway via the Internet?

5. is it possible to design a gateway that was cost effective and constructed with components obtainable from South African electronics suppliers?

1.3 Research methodology

A literature search was undertaken to investigate the designs and attributes of gateway implementations in several deployments and those offered by commercial firms. An analysis of the findings resulted in a defined set of requirements for the specification of the gateway proposed in this study. The specification guided the selection of a suitable hardware evaluation platform for the development of the OS and gateway test application. This was followed by the design, construction

and evaluation of the new hardware by executing the gateway test application. The results obtained from several tests performed on the gateway are analysed in order to answer the questions posed in section 1.2.

1.4 A generic gateway proposal

The popularity of employing WSN for the monitoring of varied environments is ever-increasing which has lead to the development of a broad spectrum of sensor nodes. The furthering of gateway equipment has lagged this development considerably.

Therefore, a gateway is required that:
- is generic in nature and indifferent to the type of WSN connected to it
- permits various forms of connectivity to the Internet
- permits bidirectional Internet access with the WSN
- has a low power consumption suitable for battery operation
- is cost-effective
- is constructed in a robust manner suitable for exterior deployments

The following sections describe the software and hardware requirements for the proposed generic Internet gateway.

1.4.1 Software requirements

The functionality of the gateway is determined mostly by the software being executed on it. The majority of the gateway features are provided by the OS, however, additional functionality may be added by applications running on the OS, such as the database programme PostgreSQL. Therefore, a multitasking OS is required.

The following functionality is required and may be provided by either the OS or a user application:
- persistent storage for all *TinyOS* data packets received from the sink node

- a file system suitable for the storage and retrieval of captured data (a transaction-safe filesystem may be advantageous)
- the compression of data files to limit the amount of data to be uploaded
- periodic uploading of compressed data to a remote server with online access
- a user interface to permit configuration of the gateway application
- utilities to assist with remote troubleshooting of the gateway
- FTP or TFTP and Telnet servers and an optional HTTP server

The OS should permit the connection of a sink sensor for the transfer of data to and from the WSN. Most sensors transfer data via a serial port and therefore the OS is required to accommodate serial communications up to 115200 bits per second.

In a typical WSN, data is transferred in the form of packets, the size of which is determined by the WSN system developer. The OS or user application is required to be configurable to accommodate data packets of differing lengths. The maximum permissible length of any packet to be buffered by the gateway at a time is 10 kilobytes.

The rate of the packet data throughput will be determined by OS and/or application latencies in response to the arrival of the new packet. In addition, the rate at which the data is written to the persistent memory or redirected to a remote server on the Internet may also impact overall rate of data throughput. Due to these constraints, the gateway developed in this study will not be suitable for use in deployments requiring exceptionally high rates of data transfer such as those required for live audio or video streaming.

The gateway OS is required to recover and resume normal operation following a power interruption.

The gateway configuration is to be stored in a persistent memory.

1.4.2 Hardware requirements

The functionality of the gateway is mostly determined by its software. Many OSs permit user interaction via a display or graphical user interface, however such will not be incorporated into this specification in order to maintain the low cost, low power and robust design requirements of the study. Interactions with the gateway will occur by means of a terminal emulator via a UART, Telnet client or Web page interface.

The following is the proposed specification:

Processor and memory

- a microprocessor with sufficient speed, processing and peripheral capabilities to support a multitasking OS such as Windows Embedded CE or an embedded distribution of Linux
- Flash memory suitable for the persistent storage of the OS image, filesystem and OS configuration
- sufficient RAM in order to support the OS and any applications required to meet the operational requirements of the gateway
- SD/MMC card support to enable the persistent storage of data captured from the WSN

Communication ports

- USB host port for the connection of a WSN sink node to the gateway
- 802.3 auto-negotiated 10BaseT or 100BaseTX wired Ethernet connection
- full modem UART terminated in a header with a pitch of 2,54mm
- UART for connection to a local PC

Power supply

- power consumption not to exceed 3,5W during normal use to permit stand-alone operation from a 12Ah battery up to a maximum of 18 hours
- polarity protected DC input ranging from 8V to 15V

Miscellaneous

- general purpose processor IO port pins terminated at a header
- LED system status indicators
- low profile and compact design
- robust electrical connectors and headers
- the gateway printed circuit board is to fit within an enclosure with a minimum of an IP54 environmental rating
- ensure the use of components with an *industrial* temperature rating of -40°C to 85°C where possible
- large heatsinks or forced cooling by fans of the gateway is not permitted

The following considerations shall be taken during the design of the gateway:

Modular design

- the proposed gateway is required to be generic in nature and its Internet connectivity adaptable to suit the deployment environment
- separate modules, based on different Internet connectivity technologies, will be constructed to fit within a defined footprint and incorporate a header for serial communications and module control compatible with that of the gateway

Cost effective design

- no graphical user interface (GUI) is required
- attempt to source all components required from companies that are well represented and supported in South Africa to reduce costs
- design a PCB with the minimum of copper layers required in order to reduce manufacturing costs
- avoid incorporating components in packages such as ball-grid-arrays (BGA) that require specialist equipment to assemble or repair the gateway PCB

The following considerations shall, in addition to those listed, be applied to the design of any communication module for the gateway even though a specific Internet communication technology is not specified here:

UART and control interface

- UART and power control lines to be buffered or level shifted
- UART and power control lines to be terminated in a header that is compatible with the communications header on the gateway PCB

1.4.3 Delineation of study

The emphasis of this study was on the design and development of a suitable gateway hardware platform capable of supporting a multi-tasking operating system. In addition, a custom software application to exercise and evaluate the gateway hardware design was written and executed on the installed operating system. The software application was only required to demonstrate the functionality of the the gateway and did not focus on aspects such as system and data security or data manipulation, mining or databasing.

Power supply design issues relating to photovoltaic usage or battery maintenance were not addressed.

A single TelosB wireless node was used to represent a WSN deployment by generating packet data typical of network traffic that may be encountered by the gateway.

1.5 Summary

This chapter introduces the topic of Wireless Sensor Networks and cites typical applications and deployment environments. The attributes of a typical sensor are discussed and why these are important. It is shown that in many situations WSNs require Internet access for data transfer and for remote configuration. The data captured by a WSN is usually routed to a sink node which provides a bidirectional link to the WSN through which all data is transferred and network configuration is permitted. The device known as a gateway is introduced and its functionality and importance defined.

An in depth study of several existing gateway implementation is given and the findings summarised in Table 1.1. The findings suggest that a need for the development of a generic, cost-effective and robust gateway for a WSN exists thus resulting in a defined study objective and specific research questions.

A specification of a proposed generic gateway is given including specific details and considerations relating to the hardware design and software requirements.

The chapter concludes with a Delineation of Study.

Chapter 2

The Gateway Hardware Platform

2.1 Overview

A hardware platform may be defined as the type of processor, controller or other hardware on which a given OS or application is executed. Choosing an appropriate platform may become an overwhelming task as the variety of processor architectures, processor peripheral functionality, internal complexity and modern packaging formats offered is extremely varied. Most of the large integrated circuit (IC) manufacturers such as Texas Instruments (TI), Atmel, NXP, Infineon, Freescale, Toshiba and ST Microelectronics each offer their own ranges of microprocessors, all of them having characteristics suited towards specific end-use applications such as 3D graphics rendering systems, Smartphones or personal navigation devices. The result is a vast selection of available processors. This chapter explores several processor offerings for embedded applications and refines the selection to enable a suitable device to be employed for this study.

2.2 Platform selection

2.2.1 System architecture selection

The gateway proposed is required to support a multi-tasking OS such as an embedded distribution of Linux or Windows Embedded CE (WinCE). These two OS are amongst the most popular embedded operating systems because of the many attributes and scalability they offer to embedded system developers. Linux is able to be compiled to permit its execution on several embedded processor architectures such as x86, Microprocessor without Interlocked Pipeline Stages (MIPS), PowerPC, ARM, Motorola 68K, CRIS, V850 and SuperH (McCarty, 1999; Raghavan, Lad and Neelakandan, 2006). Linux is therefore considered to be a versatile *cross-platform* OS. In contrast, the number of processor core architectures that are able to execute WinCE is a subset of those that support Linux, namely ARM, MIPS, x86, and SuperH, (Pavlov and Belevsky, 2008*a*) and it was decided to limit the choice of core technologies to these.

In order to select an appropriate architecture from those listed for use in this study, it was decided to investigate which of the IC manufacturers that produce processors and development tools based on these architectures is represented in South Africa, as required by the gateway specification. In February 2011 a Web-based search of three prominent microprocessor suppliers in South Africa namely, Arrow Altech, Avnet Kopp and EBV Elektronik was conducted and it was found that the vast majority of manufacturers they represent produce processors based on the ARM architecture. It was found that the remaining architectures had little to no representation at all. This finding narrowed the search of a suitable architecture to ARM-based microprocessors only.

An ARM is a 32-bit reduced instruction set computer (RISC) first developed by Advanced RISC Machines, which later became ARM Holdings in 1998 (ARM holdings, 2010*a*). ARM Holdings do not physically manufacture and distribute ICs but rather design and license intellectual property to Partners which include many prominent IC manufacturers and

systems design companies (ARM holdings, 2010*b*). ARM currently offer three processor groupings depending on performance, functionality and capability. They are the Classic ARM processors, Embedded Cortex processors and Application Cortex processors.

Several of the processors in these families are able to execute full OSs such as Linux and WinCE. A processor selection tool is available on the ARM Web site (ARM holdings, 2012) which permits a user to input criteria to search for the closest suitable processor. The following criteria was applied to the processor search tool: No multi-core processors; ARM, Jazelle and Thumb instruction sets supported; cache memory required; a memory management unit (MMU) - an MMU is a mandatory requirement for the execution of Linux and WinCE. Table 2.1 summarises the results of the processor search and compares common attributes.

Family	**Cortex-A**	**ARM11**		**ARM9**
Architecture	ARMv7-A	ARMv6		ARMv5TEJ
Processor	Cortex-A8	ARM1136J(F)-S	ARM1176JZ(F)-S	ARM926EJ-S
ARM	✓	✓	✓	✓
Floating point	✓	✓	✓	✓
Jazelle	✓	✓	✓	✓
Thumb	✓	✓	✓	✓
L1 cache	32kB	64kB	64kB	8kB
MMU	✓	✓	✓	✓
Preformance DMIPS	1200	768	965	402
Performance DMIPS/MHz	2	1,26	1,25	1,1
Max frequency MHz	600	610	772	366
Power with cache mW/MHz	0,75	0,36	0,208	0,244
Power at max frequency mW	450	219,6	160,576	89,304

Table 2.1: ARM processor attributes

Table 2.1 shows four ARM processor architectures, which cover a spectrum of performance and power consumption levels, are available and able to execute Linux or WinCE.[1] The Cortex-A8 has the highest level of DMIPS performance but is also consumes the most power at its maximum frequency. The ARM11xx processors may operate at higher frequencies but they do not guarantee a significant increase in performance for the amount of power they consume. The ARM926EJ-S has the lowest Dhrystone performance level and consumes marginally more power per megahertz than the ARM11, but at its maximum frequency consumes the least power of all the processors.

The specification of the proposed gateway requires conservative power consumption so that the system may be powered by a battery. In addition to the consideration of the overall power consumption, the highest level of computing performance is not required for this study as there is no display, requiring complex graphics capabilities, nor is there a requirement for real-time data processing nor high-speed data transfer. Therefore, the ARM926EJ-S is an ideal processor core to consider for this study. The core is highly popular and has been incorporated into a broad spectrum of embedded applications by many IC manufacturers. The resulting selection of microprocessors, each with its own unique capabilities and peripherals, is vast. The following section deals with the selection of a microprocessor suitable for this study.

2.2.2 Microprocessor selection

TI, Atmel, NXP and Infineon are companies that manufacture microprocessors based on the ARM926EJ-S core with varied peripherals and are represented by agents in South Africa, as mentioned previously. The web sites of these manufacturers were investigated to ascertain the particular device number for any microprocessor based on the ARM926EJ-S core. Processors not meeting the minimum specification were not considered for inclusion in the investigation.

[1]The results shown in table 2.1 are for information purposes only and may vary due to the process employed to either manufacture or synthesise a particular processor core.

The following minimum characteristics and peripherals are required on the processor in order for it to be considered a viable option for this study:

- ARM926EJ-S core
- a 180MHz to 400MHz oscillator frequency
- SDRAM memory access
- SD/MMC card interface
- a minimum of two UARTs
- USB host port
- 10BaseT / 100BaseTX RMII Ethernet interface
- SPI, I^2C peripheral interfaces

Table 2.2 summarises the attributes of potentially viable microprocessors found on various manufacturers web sites. The listed devices all meet the minimum requirements. Extraneous peripherals and features have been omitted in order to promote the simplicity of the selection.

The results of the investigation show that several microprocessors would be suitable for use in this study. The performance between these processors is likely to vary based on the maximum clock speed alone as the ARM926EJ-S core is common to all. An important requirement of the proposed gateway is its ability to be able to be battery powered. This constraint leads to a trade-off between the power consumption of a microprocessor and its performance. It was decided to select a processor with a lower operating frequency and to sacrifice some computing performance in order to promote low power consumption of the gateway.

In addition to the power constraint imposed by the hardware specification, the specification also requires manufacturing costs to be kept to a minimum where possible. The package style of any device has a cost implication for the manufacturing process of the system. The investigation revealed that the majority of the suitable microprocessors found are supplied in a Ball Grid Array (BGA) package.

Manufacturer	Device	Max clock MHz	Memory	Peripherals	Serial ports	Package
Texas Inst.	AM1705-375 AM1705-456	375 456	NAND, NOR, SDRAM	USB, MMC/SD, RMII	I^2C, SPI, UART	HLQFP 176
Texas Inst.	AM1707-375 AM1707-456	375 456	NAND, NOR, SDRAM	USB, MMC/SD, RMII,LCD	I^2C, SPI, UART	BGA 256
NXP	LPC3240FET296 LPC3250FET296	266	NAND, SDRAM	USB, MMC/SD, RMII USB, MMC/SD, RMII, LCD	I^2C, SPI, UART	LFBGA 296
Freescale	MCIMX253 MCIMX258	400	NAND, SDRAM, DDR2	USB, MMC/SD, RMII, LCD USB, MMC/SD, RMII, Touch	I^2C, SPI, UART	MAPBGA 400
Atmel	AT91SAM9XE128 AT91SAM9XE256 AT91SAM9XE512	180	NAND, SDRAM, CF	USB, MMC/SD, RMII	I^2C, SPI, UART	PQFP 208 LFBGA 217
Atmel	AT91SAM9260	180	NAND, SDRAM, CF	USB, MMC/SD, RMII	I^2C, SPI, UART	PQFP 208 LFBGA 217

Table 2.2: ARM926EJ-S based microprocessor attributes

The BGA package style differs from the traditional leaded device by bringing the internal connections out to an array of tiny solder balls arranged in a matrix underneath the device carrier.

This style of package has several advantages: it saves area on a PCB which promotes a compact system design, reduces soldering shorts during mass production, improves the transfer of heat from the IC to the PCB and offers low inductance signal paths.

The BGA package style has several disadvantages that, in terms of this study, require consideration. BGA-packaged devices that have several hundred connections concentrated beneath the IC carrier, such as the microprocessors listed in Table 2.2, require PCBs consisting of many internal signal layers to permit the routing of the signals from each ball. PCB designs incorporating six or more layers designs are very common when employing large BGA devices which tends to escalate the costs of the PCB manufacture significantly (Pfeil, 2007). In addition to elevated manufacturing costs, designs incorporating BGA are not easily repaired and require specialist equipment and expertise (Cirimele, 2004).

It was decided to select a microprocessor in a leaded package as apposed to a BGA. The Quad Flat Pack (QFP) is a leaded package that presents several attributes which meet the specification of this study. The advantages of the QFP package are as follows:

- QFP devices may consume more surface area on a PCB but they promote a PCB design with fewer internal layers than an equivalent BGA device, thus reducing design, manufacturing and assembly costs

- the solder joints of a QFP device on a PCB may be easily inspected for defect using a microscope or magnifier, thus reducing requirements for specialist repair equipment

- PCB incorporating QFP devices may be assembled or repaired easily without the need for costly component placing equipment and solder masks as the hardware for this study is to be assembled by hand

The results of the investigation in Table 2.2 show that a few microcontrollers that have the required peripherals are manufactured in a QFP package. The package type and the requirement for low power consumption, as discussed previously, place a limit on the number of possible microprocessors that may be employed in this study. Therefore the suitable microprocessors were the AT91SAM9XExxx and AT91SAM9260 and are manufactured by Atmel.

At the time of this investigation, it was found that the AT91SAM9XE devices were being distributed for sampling proposes only and it is unknown if an evaluation kit for these devices was available for purchase. The AT91SAM9260 has been in production for some time and employed in a wide spectrum of applications.

2.2.3 The AT91SAM9260 evaluation kit

An Evaluation Kit (EK) is a PCB that incorporates a particular IC and support components to form a functional circuit permitting the evaluation and familiarisation of the IC. These kits are usually prebuilt and tested by the manufacturer and are usually fully functional without modification. Advantages of using an evaluation kit are that it provides a functional reference design that may be used as a basis for new hardware development and for the development of software that, with minor modifications, will run on the new development. The Atmel AT91SAM9260-EK evaluation kit was acquired for this study because as it is based on the AT91SAM9260 microprocessor.

The AT91SAM9260-EK has the following specification:

- AT91SAM9260 microprocessor
- 64 Mbytes of SDRAM memory
- 256 Mbytes of NAND flash memory
- 8 Mbytes of dataflash
- one USB device port interface
- two USB V2.0 full-speed host port interfaces
- SD/MMC card slot
- processor debug UART
- two UARTs, one of which incorporates modem handshaking signals
- 10/100BaseT Ethernet interface with status LEDs

The AT91SAM9260-EK had the necessary peripheral functionality to satisfy the specification for the proposed gateway. It was used to investigate and develop the gateway's OS, for the development of a prototype gateway application and it formed the basis of the gateway hardware design.

2.3 GSM/GPRS module selection

A GSM modem was required to satisfy the requirement of being able to deploy the gateway to a remote location without access to a wired Ethernet connection. Many GSM modems that are available have only an RS232 serial communications port interface and very limited means to control their power consumption. In addition to these limitations, these modems were usually expensive due to their robust and physically larger enclosures and the extra power and data connectors. It was decided to search for a GSM module instead to significantly reduce the footprint size, increase the versatility and reduce the cost of the communication module for the gateway.

The communications vendor Telit produce a wide range of GSM modules and are represented by local agents. This meant that obtaining a module and technical support, if required, would be greatly simplified satisfying the requirement of us-

ing locally available components. It was decided to investigate the GC864 and GM862 modules to determine if they would be suitable for use in this study.

After an investigation and comparison of the specifications of the GC864 and GM862 families, it was found that the GM862-GPS was a suitable choice as it features embedded TCP/IP, FTP and SMTP stacks, extended RF sensitivity and Quad band, class 10 GPRS functionality (Telit Communications, 2011*b*). In addition, the SIM card holder is integrated into the module's case thus simplifying the system design and reducing the component count and work required during assembly of the gateway PCBs. A 50-way connector permits comprehensive connectivity to the module providing the ability to control important aspects of its operation such as the amount of power it consumes and the monitoring of its connectivity status. The module also incorporates an integrated 20 channel GPS engine which may provide the gateway with additional functionality such as an accurate time base or the ability to perform location-based data logging.

A GM862-GPS evaluation kit and prepaid SIM card were obtained in order to develop and debug the GSM control and communications software module of gateway application.

2.4 Summary

This section discusses the hardware platform required to support multi-tasking operating systems such as Linux and Windows Embedded CE. Several microprocessor architectures suitable for the execution of these operating systems were found, however packaging and availability constraints limited the selection considerably. The ARM926EJ-S processor core was well suited to the gateway specification and widely available from several manufacturers represented by local agencies.

Several factors such as the performance, power consumption, cost of system production, peripheral set and packaging were considered during the selection of the final choice of microprocessor. The AT91SAM9260 microprocessor by Atmel met all the requirements for the gateway specification and was the final choice for the design. An AT91SAM9260-EK evaluation kit was obtained in order to develop the gateway operating system, evaluation application and it formed the basis of the gateway's hardware design.

A Telit GM862-GPS module was selected to provide the gateway with remote Internet connectivity as it is a single-component, compact GSM solution offering maximum control over its operation via a 50way interface. The GM862-GPS incorporates embedded FTP, GPRS and SMTP stacks as well as a GPS receiver which may be used for system time setting and location-based data logging if required.

Chapter 3

Software Development

3.1 Overview

The previous chapter discussed the functionality, specification and selection of a suitable hardware platform required for the gateway development. This chapter provides details of software required to satisfy this study's specification in section 1.4. The selection of a suitable multi-tasking OS, its customisation, deployment and execution on the evaluation hardware platform is discussed. This is followed by detailed explanations of the the development of a gateway application to be executed by the deployed OS.

3.2 OS Investigation

The choice of OS has several implications for the development of a system. These include factors such as the quality and costs associated with the OS and application development tools, the ease of application development, debugging and future functionality extensions. The OS selected should be scalable in order to match it to the platform it is to be executed on. Careful consideration is required if the platform's resources such as the amount of system memory, processor operational speed and variety peripheral devices are limited or constrained. The requirement for embedded functionality such as web, file and remote communications servers within the OS may also be a consideration when choosing an OS for a development. There were several OS such as VxWorks, eCOS, FreeRTOS, Windows Embedded CE, μC/OS-II, OpenEmbedded and Angstrom Linux distributions that were available and offered the functionality and attributes required by specification for this development.

It was decided to limit the investigation of a suitable OS to Windows Embedded CE as considerable documentation, knowledge bases and support forums were readily available on the Internet. In addition, a demonstration version of Windows Embedded CE and its Board Support Package (BSP) were available for the EK selected in the previous chapter on Atmel's Windows4SAM website (Atmel Corporation, 2010*c*).

3.2.1 Windows Embedded CE

Windows Embedded CE (WinCE) is a small-footprint, scalable or modular 32-bit OS developed by Microsoft to run on non-PC embedded devices with limited hardware resources. WinCE1.0 was introduced by Microsoft in 1996 which provided programmers with PC software development knowledge an opportunity to build applications for new devices using a common programming interface (Pavlov and Belevsky, 2008*b*). The OS has been modified and expanded several times since then and to date includes support for technologies such as IPv6, VoIP, the .NET Compact Framework and USB 2.0.

Windows Embedded CE 6.0 was released in September 2009 (Microsoft Corporation, 2009) and was selected for this evaluation.

WinCE is a flexible OS as its footprint and functionality may be scaled to meet the software specification and hardware peripherals of a particular device. This is achieved by configuring the OS compiler to add modules of functionality to a basic OS core with limited peripheral functionality during a *build* or compilation session. Each module of functionality may be selected from a catalogue of about 600 software components which is subdivided into broad categories such as Communication Services and Networking, File Systems and Data Store and Internet Client Services.

WinCE is intended for use in embedded systems which incorporate peripheral devices such as USB ports, memories with Serial Peripheral Interfaces (SPI) and General Purpose IO (GPIO) processor port pins. These devices require low-level software routines or drivers which provide an interface, or Original Equipment Manufacturer (OEM) adaption layer, between the physical hardware peripheral and the OS. Drivers are written to be specific to the hardware and are responsible for the initialising, maintenance and data transfer between the OS and the peripheral when required.

Groups of drivers and a *bootloader* programme, which is responsible for the initialisation of the target platform and the launching the OS image (Microsoft Corporation, 2002*a*), specific to a particular hardware platform are usually combined into a single collective known as a Board Support Package (BSP). BSPs are complex and are generally written by third-party specialists as the integrity of the entire OS relies on the quality and functionality of the underlying drivers. A BSP may be supplied as a collection of precompiled binary files specific to a particular target platform or as files of source code which are compiled by the end-user during an OS build. The latter option permits an end-user the ability to modify the operation of the drivers to suit specific requirements of their target hardware platform.

During the configuration phase of OS development the drivers within a BSP may be added to the catalogue of selectable OS software modules. This permits the selection of only the drivers required for the target hardware in a similar manner to the software modules supplied with the OS development environment. A runtime image of the OS is built by a *toolchain* installed onto the development PC once the entire OS has been configured within the development environment. The installation and verification of an Integrated Development Environment (IDE) and toolchain to build a WinCE image is discussed in the following section.

3.2.1.1 Windows CE 6.0 configuration

An IDE and toolchain were assembled and installed onto a PC to gain an understanding of the process required to configure and build a functional OS image for evaluation. The resulting OS image would be deployed onto the EK and its functionality verified. If the the OS was found to be functional, a small sample application would be written and deployed to the EK and its functionality also verified. The results of the OS and sample application evaluations would prove that the entire software development process was functional and the development of the gateway application could begin.

WinCE and its development tools are not open-source software and they may not be available for use without charge. This could have cost and licensing implications for the development of commercial systems employing this OS, however for the purposes of this study, it was found that trial versions of the development tools and IDE, which operate for a period of 180 days, are freely available for download from the Microsoft website (Microsoft Corporation, 2010*a*).

The following packages were downloaded from Microsoft and installed onto the development PC:

Visual Studio 2005, with Smart Device functionality included
Visual Studio 2005 Service Pack 1
WinCE 6.0 180 Evaluation version (Platform Builder)

WinCE 6.0 Platform Builder Service Pack 1
WinCE 6.0 R2 and R3
WinCE 6.0 Product update packages

The central core of the entire WinCE OS development toolchain is Platform Builder (PB) which is a software plug-in module for the Visual Studio 2005 (VS) IDE. PB provides the compiler and linker used to build the image as well as the catalogue of available software components for inclusion during the configuration of an OS. The tools to develop and modify a BSP are also included with PB (Phung, 2009*a*). In addition, the PB plug-in provides debugging and monitoring utilities to troubleshoot or modify the operation of the OS or any user applications being executed on a target hardware platform.

A BSP for the AT91SAM9260-EK evaluation kit was available for download from the Atmel AT91SAM Community website (Atmel Corporation, 2010*b*). The BSP's source code was available for viewing and modification in VS after it was installed onto the development PC. It was suggested by the BSP developers and deemed good practice to clone the BSP to preserve its original source code as any modifications made would be incorporated into subsequent OS builds. The installation of the BSP completed the toolchain installation required to build an OS image.

The configuration of an OS began by creating a new OSDesign project in VS. A Design Wizard was launched automatically which assisted with the setting up of the working framework for the resulting image build. The AT91SAM9260EK BSP was amongst a list of BSPs for other platforms supplied with PB and selected by means of a check box. This was followed by the selection of a Design Template which configured a set of predefined catalogue items that most closely matched the requirements of the likely target hardware. There were several templates such as Custom Device, Industrial Device or Gateway device available for selection; the *Custom Device* type was chosen which left all the catalogue components unselected.

The *Catalogues Items View* pane in VS, as shown in Fig 3.1, permitted the modification of the OS configuration once the initial setup wizard was complete.

Figure 3.1: The Catalog Items View pane in VS permits the configuration of Windows CE. The BSP for the AT91SAM9260EK was cloned and renamed to GATEWAY:ARMV4I. Shown here is the selection of the Ethernet driver for inclusion in the OS build.

The following listing is a selection of software modules in the catalogue that were added to the basic OS core:

Applications and Services development:

.NET Compact Framework 3.5

Networking:

Wired LAN 802.3

TCP/IP support

Network utilities: IPconfig, Ping

Telnet, Web and FTP servers

Core OS services:

Device manager

USB host support

File systems:

Hive-based registry storage

FAT file system

Table 3.1 lists the required drivers selected for inclusion into the OS image:

UART	SPI
USB host	Ethernet
Bootloader on CS1	Registry in NAND flash
SD memory card	I^2C EEPROM
GPIO	-

Table 3.1: Only the drivers required for the peripheral devices in the final hardware design were enabled for inclusion in the OS build.

The OS project properties may be configured as either of two types, *Debug* or *Release*, permitting a user to configure an image build to include debugging functionality or not respectively. It is important to note that this selection does not alter any of the chosen OS catalogue items but only the *build* options. It was decided to configure a *Debug* OS build to include the debugging capabilities in the output image for the OS evaluation.

A WinCE *Debug* image incorporates a kernel debugger, Kernel Independent Transport Layer (KITL). KITL connects to PB via the onboard Ethernet connection which transfers debug messages generated by the OS kernel to an output window in VS where several debugging tools such as the Registry, File and Process viewers are located. These tools provide a user with real-time debugging messages relating to the system operation and performance. The user is also permitted to modify the execution of the system, for example by modifying the values in the Windows registry or terminating a programme being executed by the OS. Enabling the KITL debugger disables the third party Ethernet driver selected in the BSP and substitutes it with an in-built system driver. Changing the project properties to *Release* before a build disables KITL and reinstates the

Ethernet driver selected in the BSP.

After the project properties were adjusted and the OS configured, the build process was started by selecting *Build Solution* in the Build menu in VS. Messages generated by the compiler and linker were displayed in the Output window of the IDE where the build process was monitored. Many warning messages were generated and ignored, however the build process would be terminated if a fatal error was encountered by the compiler or linker. A typical OS build required an average of about 30 minutes to complete and was dependent on the processing capabilities of the development PC.

The build process completed without any fatal errors and the image file *nk.bin* was generated along with a message of success. Several numeric metrics relating to the OS image such as *starting ip*, *image start address* and *total ROM size* were displayed and were required during the deployment of the OS to the EK. It was found that these metrics were also written to the file *makeimg.out* which was located in the build output directory.

3.2.1.2 Bootloader deployment

The EK was prepared for OS deployment by copying two bootloader programmes *FIRSTBOOT* and *EBOOT* generated during the OS build to the dataflash memory on the EK. The purpose of the bootloaders is to initialise the peripheral devices on a hardware platform and to prepare the platform memories and processor to launch and execute the OS (Microsoft Corporation, 2002*a*). They may also be used to deploy an OS image to a persistent memory on the EK.

The AT91SAM9260 processor on the EK incorporates a ROM which is preloaded by the manufacturer with the bootloader *ROMboot* which is executed first after the processor is released from a reset condition. *ROMboot* attempts to locate a valid signature *FIRSTBOOT* at address 0x0 in the dataflash and NAND flash memories on the EK. If *FIRSTBOOT* is not found, *ROMboot* attempts to connect to SAM-BA (Atmel Corporation, 2010*a*) via the USB port on the EK. SAM-BA is

a free, PC based in-system programming utility developed and supplied by Atmel and performs low level functions such as memory verification sequences, erasing and programming of platforms incorporating Atmel's ARM microprocessors.

SAM-BA was launched on the development PC and a USB connection established between it and the EK to copy the files *FIRSTBOOT.nb0* and *EBOOT.nb0* to addresses 0x0 and 0x4000 of the dataflash memory respectively. SAM-BA was terminated and the EK reset after the bootloaders in the dataflash memory were verified as being identical to the original files on the PC. *HyperTerminal*, a Windows-based terminal emulator, was launched on the development PC and a serial connection established between it and the EK's debug UART to permit the configuration of *EBOOT* after a reset of the EK.

A three stage boot process followed the reset: *ROMboot* was launched and a simple text message was displayed on the terminal confirming its operation and that of the debug UART.

A valid signature for *FIRSTBOOT* was found at address 0x0 in the dataflash which started its execution. *FIRSTBOOT* checked location 0x4000 in the dataflash for *EBOOT*'s signature, which was found and *EBOOT* launched.

EBOOT provides a simple menu based user interface through the terminal permitting the configuration of the startup behaviour of the EK. *EBOOT* may be configured to launch an existing OS image stored in the EK's NAND flash memory or to download a new OS image from PB on the development PC via Ethernet to either the NAND flash or RAM; *EBOOT* was configured to perform the latter at startup and to write the image to the NAND flash memory. The numeric metrics that were generated and saved in *makeimg.out* during the OS build process were written into the *Image flash configuration* menu and the EK reset to permit the modified configuration to take effect.

3.2.1.3 OS image deployment

The OS image built by the PB toolchain on the development PC was downloaded by *EBOOT* via the Ethernet connection and written to the NAND flash memory. The EK was reset in order to test the booting process and OS launch from startup. The messages displayed by the booting process, *EBOOT* and the OS image transfer from the NAND flash to RAM were monitored on *HyperTerminal* and the process completed successfully without any errors. The OS image, now resident in the RAM, was launched by *EBOOT* and successfully initialised.

In order to verify that the OS was operating correctly, a connection via the Ethernet to the EK was attempted from the *Attach Device* tool built into PB. The initial attempts failed and messages stating that "Kernel debugging had been successfully disconnected" were displayed in VS. It was found that the default KITL options in the *Connectivity Options* window were unselected; selecting all the KITL options in this window corrected the fault and a successful Ethernet connection with the OS on the EK was established. All the remote debugging tools such as Remote File Viewer, Process Viewer, Registry Editor, Performance monitor and Kernel tracker became available after the connection was established. Several utilities built into the OS such as *ipConfig* also became available after a successful connection.

The application *ipConfig* was launched in VS to obtain the IP address allocated to the EK by the Ethernet network during the boot process. It was decided to try and connect to the Telnet and FTP servers incorporated in the OS during its configuration. These tests would prove, if successful, that IP connections to the OS to obtain a command prompt and to execute file transfers were possible through the Ethernet. Telnet and FTP clients, PuTTY and FileZilla, were loaded onto the development PC and attempts to establish connections with the EK from each were made.

Connections to both servers were unsuccessful as the default usernames and passwords in WinCE were unknown and were required to complete successful logins.

Online research showed that the *anonymous* username and password login details of the Telnet (Microsoft Corporation, 2008) and FTP (FTDI, 2010*d*) servers may be enabled by making specific modifications to the keys FTPD and TELNETD in the WinCE registry by utilising the Registry Editor in PB. These registry configurations are discussed in further detail in Appendix G; it is also shown how the OS may be initialised with the modified settings after every system startup.

Successful connections could be established with the Telnet and FTP servers once the necessary adjustments were made to the registry. Simple file manipulation tasks and directory listings could be performed at the Telnet command prompt and files could be uploaded and downloaded and directories manipulated via the FTP server. These simple tests proved that that the deployed Windows CE image was functional, that the PB toolchain used to build the OS image was functioning correctly and that remote communications with the EK was possible via an Ethernet IP-based network.

3.2.1.4 .NET Compact Framework

Software applications to be executed on the WinCE OS may be written in either native or unmanaged code, or managed code (Pavlov and Belevsky, 2008*c*). Unmanaged or Win32 code is usually written in C or C++ and compiled to form executable code that is native, or specific, to the processor on which it is to be executed.

Unmanaged code is considered to operate at a lower level than managed code, and may be optimised during compilation to have a small footprint or high performance and provides the greatest access to the hardware platform on which it is being executed. The complexity of unmanaged application development is increased due to several factors such as the need for additional code to manage memory and security requirements of the application and the disposal of programming objects after they have served their purpose. These tasks are usually unrelated to the intended function of the application and are often neglected resulting in elevated testing costs and time periods.

Application development using managed code removes many of these additional complexities from the software developer's responsibility at the expense of a small amount of processing overhead. In addition, managed application development promotes the the development of software with fewer errors, is more compact and readable.

The .NET Compact Framework (.NETCF) is a scaled down version of the .NET Framework which presents an object-oriented API that has been designed to overcome the problems of Win32 software development. A typical example of this is that .NETCF groups common Application Programming Interface (API) functions into classes, and collections of classes into namespaces which assist in aiding the productivity of application development in an efficient development environment (Phung, 2009*b*). It was decided to include the .NETCF in the OS image to permit the evaluation of application development using a managed language with the inherited benefits mentioned previously.

.NETCF is listed as a software module in the WinCE OS catalogue. Choices of release versions and base configurations of.NETCF are available: .NETCF 2.0 or .NETCF 3.5, default and headless. The *headless* configuration of .NETCF is required if the target platform lacks a graphical user interface (GUI), keyboard or display and does not include any libraries that support forms, controls or drawing applications (Microsoft Corporation, 2002*b*).

The EK and final gateway design did not include a GUI and therefore the evaluation OS image built previously included the headless version of .NETCF 3.5. The .NETCF 3.5 - headless version was selected as it included bug fixes and additional functionality missing in .NETCF 2.0 that may have been required during the development of the gateway application.

Managed code applications may be written in languages such as Visual C#.NET or Visual Basic.NET and require the .NETCF to be included in the OS image at build time in order to be executed. An advantage of writing applications in managed languages is that the software is not compiled for a specific hardware

platform nor processor architecture, but rather the Common Intermediate Language (CIL).

CIL relies on the Common Language Runtime (CLR) and Base Class Libraries (BCL) provided by .NETCF in order to operate on the hardware platform. This architecture permits the porting of applications written in a managed language to different processor architectures and hardware platforms without requiring low-level technical details of the new target (Microsoft Corporation, 2002*c*). The test and final gateway applications were written in Visual C# which simplified the application development, reduced the development time and made it easier to modify existing or add new functionality to the applications.

3.2.1.5 Test application, deployment and verification

A small test application was written in C# first to investigate the process required to develop applications capable of being executed on .NETCF and deployed to the EK. It served as a base for the development and debugging of software snippets and modules that were used in the gateway application. The functionality of various hardware components on the EK such as initialising, writing and reading files to and from the SD card, the registration and the reception of data from the wireless sensor node on the USB hub and the transfer of data files via FTP over an IP network to a remote location were exercised and verified in an additional application.

A simple menu-driven interface, accessible through a terminal emulator on a PC via the UART, was also developed in this application and permitted user interaction with the application and control over the verification of each developed software module.

It was necessary to obtain and install Visual Studio 2008 Professional onto the development PC in order to develop the applications for .NETCF 3.5.

The test application development began by creating a new Visual C# project for a Smart Device and .NET Framework 3.5 selected in the project setup wizard. An application that created a new instance of a serial port, configured, opened and transmitted a text message to it was written and built successfully generating an executable file.

The Platform Builder debugging tools were activated in VS and a connection with the EK processor established before the deployment and execution of the test application occurred. The connection was established by connecting the EK to the development PC by means of an Ethernet cable and executing the *Attach Device* function, as done previously during the evaluation of the OS. An RS232 cable was connected between port COM1 on the EK and a UART on the development PC, *HyperTerminal* launched and configured to accept data at the same rate as that of COM1 on the EK.

The test application was deployed to the EK by copying it to the OS build `release` directory on the development PC. Files copied to the `release` directory are uploaded to the EK automatically by the debugger in VS. A list of applications on the EK may found in VS, under the *Target, Run Programs* menu. The test application was found in the list of applications shortly after it was copied to the `release` directory. The test application in the list was highlighted and launched. The debugging window in VS displayed the library files that were required to launch and execute .NETCF 3.5 and the test application.

The following events were displayed in the VS debugging output window during the execution of the test application: a serial port was opened, the baud rate configured and text transmitted - which was displayed by *HyperTerminal*. Characters from *HyperTerminal* were sent to the application and different messages echoed back depending on the text sent thus confirming functional communications with the test application and the ability of the application to parse and respond to the received data.

The following conclusions from this evaluation were made:

- the deployment of an application to the platform was successful
- the test application was functional
- the C# toolchain for application development was complete and functional
- the PB debugging system was functional and reported useful debugging data
- a framework with a simple user interface for the development and debugging of software modules was created

3.3 Software module development

Several software modules and functionality that were required in the final gateway application were developed and debugged by using the framework created in the test application from the previous section. The software modules developed were verified and copied into the final application without the need for much additional modification and are discussed in the following section.

3.3.1 Windows registry

The WinCE registry is a database that stores configuration information and settings for applications, OS kernel and hardware drivers (Microsoft Corporation, 2010*b*). Writing and reading data to and from the registry was important functionality that was used extensively by the test and gateway applications for the storing and retrieving of configuration data. The OS incorporates a *hive-based* registry which stores its data inside files or hives in the OS filesystem on the NAND flash of the EK. The data in the registry is therefore persistent and is not lost nor altered during a power interruption or system reset.

The registry is organised in a tree format where each subtree, identified by a constant value or HKEY, is formed from branches called keys. Keys may contain subkeys and data entries which may be modified programmatically by an application.

In addition, an application is able to create new and delete keys in the registry when required.

.NETCF provides a Registry class within the Microsoft.Win32 namespace that provides several functions which permit registry key manipulation. This class was added to the test application and verified by creating several new keys and populating them with data. The altering and deleting of these keys was tested and each step verified in the PB debugging window and *Remote device Registry Editor* tool.

3.3.2 SD/MMC memory card

The specification of the gateway application required the use of a persistent memory for the storage of data received from the sensor network. It was decided that an SD or MMC card provided the most compact means of storing vast amounts of data.

The *SDConnect* and *Folder* values of the *Flashloader* and *StorageManager* registry keys were modified to *1* and *SDcard* respectively to enable and mount the SD card for access at the launch of the OS (see Appendix G). The *Folder* value specifies the directory to which the SD card is bound once the card is mounted after the initialisation of the OS. The default folder value was modified to *SDcard* which provided a conveniently short name for the memory card during the application development and verification. The PB debugger displayed several messages confirming the initialisation and mounting of the memory card during subsequent launches of the OS. In addition, the functionality of memory card was further verified by the writing and reading of a file containing text to it by the test application.

3.3.3 USB host interface

The TelosB wireless node has a USB plug permitting its insertion into the USB A socket on the EK. It was found that the node could not be registered on the

OS as a USB driver and extra registry settings were required to be installed first.

The TelosB node incorporates a USB to serial UART converter manufactured by FTDI (University of California, 2010). An online search of FTDI's website found that a Virtual COM Port (VCP) driver was available for WinCE 6.0 being executed on ARM processor architectures (FTDI, 2010*f*). The driver was installed onto the EK by copying the driver files *FTDIPORT.INF* and *ftdi_ser.dll* to the *Windows* directory of the OS filesystem using the automatic file download function of PB. Several keys and values were added to registry in the *Registry Editor* in PB to configure and enable the drivers at the next OS launch according to the document Windows_CE_Installation_Guide available from the FTDI website (FTDI, 2010*e*).

The additional registry settings and names, with the directory paths, of the driver files were added to the *project.reg* and *project.bib* files[1] respectively to ensure that the registry settings and VCP drivers were automatically included in the OS images of subsequent builds, as shown in Appendix G.

The enumeration and initialisation of the TelosB node on the OS and its UART assignment were displayed in the PB debugging output window during the launch of the OS following a system reset. *COM0*, the UART assigned to the wireless node, was opened and configured by the application which permitted any data transmitted by the node to be received by the test application, stored in a memory buffer and transmitted to *HyperTerminal* via the UART.

3.3.4 Low-level API calls and PInvoke

It may be necessary at specific times during the development of a managed application to call API functions located in WinCE system Dynamic Link Library (DLL) files when the required functionality is not available in .NETCF. Examples of these occasions are the establishment of an Internet connection and setting the OS real time clock. Low-level API calls are performed by Platform Invocation

[1]These files are located in the OS `release` directory.

Services (PInvoke) and data marshalling (Microsoft Corporation, 2003*a*) which permits managed code to call unmanaged functions that are implemented in OS DLLs.

PInvoke involves three steps: declaration, invocation, and error handling. At design time, the declaration will include the entry point, or function required, and the module, or DLL, from where the function will be called. The declaration and wrapping of the entry point function within a method would typically be done in a utility class. Invoking the entry point is done by calling the newly declared wrapper method where required. Error-handling when using PInvoke must be taken into careful consideration as the generated error exceptions in the event of a fault may cause unknown operation or even a complete system failure.

Two error types exist:

- a *NotSupportedException* exception may be thrown if invalid arguments or data is passed to the method
- a *MissingMethodException* exception may be thrown if if the entry point in the DLL being called does not exist

The application is required to have a mechanism for dealing with these exceptions elegantly to avoid undefined system operation.

Data marshalling is required when passing arguments from .NETCF to unmanaged code because the datatypes used in each are different. Simple datatypes such as integers and booleans are converted automatically, however more complex types such as strings, arrays and structures require the use of pointer arguments (Microsoft Corporation, 2003*b*) which may increase the complexity of the application development significantly.

The PInvoke service and data marshalling was required during the development of the FTP and Time and Date control software modules.

3.3.5 File transfer

The specification of this study required that the transfer of stored files of data collected from the sensor node to remote file servers on the Internet should be possible by several connection technologies. Ethernet and GSM were selected as the technologies to accomplish this requirement and it was decided to employ the File Transfer Protocol (FTP) to ensure that the captured data files were transferred across the Internet without corruption to a remote server.

The software to accomplish an FTP transfer via GSM was written during the gateway application development and is discussed in section 3.4.5; the software required for an Ethernet transfer is discussed further.

It was found that .NETCF did not automatically include FTP client functionality that could be easily included into a user application. An online search revealed several commercial companies that offered the necessary software modules or source code to provide FTP client functionality in .NETCF. Two problems with these offerings were found:

high cost
: as at the time of writing the cost from Rebex.net for their FTP solution was 249USD (Rebex, 2011)

system requirements
: the commercial solutions usually require a platform with a GUI which was not available on the hardware in this study

The commercial offerings were not considered viable for inclusion into the application.

A search of the Microsoft MSDN website found that Internet client connectivity, including support for FTP functionality, is incorporated into the Windows Internet Services *WinInet* DLL (Microsoft Corporation, 2010*c*). This DLL was incorporated into the evaluation OS image by enabling the *Windows Internet*

Services catalogue item during the configuration. It was found during the OS configuration that *WinInet* was dependent on the inclusion of display support in order to permit the OS build to complete without a fatal error. The catalogue item *Multiple-Monitor Support* was enabled to satisfy the display support requirement of *WinInet*; it was found that this addition had no effect on the performance of the OS.

PInvoke was used to call Win32 Internet control API functions located in the *WinInet* DLL to transfer a file via FTP over an Ethernet connection to a remote server by the following steps:

- the function *InternetOpen* was called to initialise the *WinInet* DLL for use by the application and to define the Internet connection characteristics; a session handler number for subsequent function calls was also obtained
- the function *InternetConnect* logged onto the remote FTP server with the password and username details and successfully established a new FTP session
- the function *FtpPutFile* was called and handled the creation of a new file on the server as well as the uploading of the file
- the FTP session and Internet connections were terminated by the function *CloseInternet* after the file was successfully transferred

It was found that the transfer of files by FTP via an Ethernet connection using PInvoke and API calls was successful and very efficient.

3.3.6 User IO

The gateway developed in this study was specified with versatility in mind. A requirement of the hardware specification was the provision of general purpose IO (GPIO) that may be used to control any external equipment such as a GSM modem with an appropriately buffered electrical interface.

A GPIO driver is included in the BSP for the EK and was enabled during the configuration of the OS.

A general description of the Windows CE GPIO driver is given in a technical design document released by Adeneo, the authors of the EK BSP (Adeneo Embedded, 2007). According to the document, the GPIO driver is a *stream driver* as it is accessed through a Win32 file IO interface.

An extensive online search was conducted for further information regarding the usage of this driver from the .NETCF or its invocation from PInvoke, however none was found. It was fortuitous that a code snippet written in unmanaged C++ was included in Adeneo's technical design document which demonstrated the GPIO driver's usage. The code example showed the members of several structures required by the Win32 function *DeviceIOcontrol* and how it was called to alter the state of GPIO pins.

A month of experimentation followed to implement GPIO functionality by using PInvoke to call the GPIO driver through *DeviceIOcontrol* from C#, however all efforts were unsuccessful due to the complex data marshalling problems encountered.

A temporary workaround was implemented to permit GPIO control from .NETCF. The C++ code example demonstrating the use of the GPIO driver was placed into a new Win32 *Smart Device* project in VS and modified to alter the state of any GPIO pin on any port bank of the gateway processor depending on the values of four arguments passed to the code. The code, as shown in Appendix F.13, was compiled to generate an executable application, *IO_control.exe*, which was downloaded to the EK using PB. *IO_control* was verified and found to be functional by invoking it from *Run Programs* in PB and monitoring the voltage level change on a specified GPIO pin.

The test application was modified to invoke *IO_control.exe* and pass the port pin parameters and required state to it whenever a change in the state of a GPIO pin

was required. This method of GPIO control was found to be slow and unsuitable for any high speed switching applications but very well suited to the slow control requirements such as the control of the power of a GSM modem.

3.3.7 System clock adjustment

The setting of the OS clock and calendar was the final requirement to be fulfilled by the test application. A PInvoke to the *SetLocalTime* function within the *Coredll* DLL provided the required time and date adjustment functionality. The system clock and calendar were used in the gateway application for the timestamping of captured data received from the wireless node and for the scheduler that performed the regular file transfers to a remote server.

The software modules developed in the test application were used in the development of the gateway application which is discussed in the sections that follow.

3.4 The Gateway Application

The primary function of the gateway application is the capturing and storing of data received from the wireless node connected to the gateway and the scheduled transferring of the data to a remote server located on the Internet. The previous section discussed and provided details about the development and testing of functional software modules required to achieve these aims. In addition to these, the gateway application was equipped with several utility functions (section 3.4.4) which permit a user to test the gateway configuration or perform system troubleshooting.

The functionality of the gateway application has been divided into five broad classes of operation which are discussed in the following sections. The diagrams used throughout the discussions provide a broad indication of the flow of the software and do not reflect every conditional path available in the source code. A full listing of the gateway application's source code may be found in Appendix F.

3.4.1 Application initialisation

The method *Main* in the public class *Program* instantiates an instance of *GatewayEngine* which is executed first after the application is launched from the OS command prompt or automatic launch by the OS after a system reset. Its primary function is to initialise: the application from the configuration values in the registry, the user interface, FTP and logging threads, UARTs and the GSM modem. It also incorporates the main programme loop which ensures the continuous operation of the application until it is terminated. Once a termination request is received, the Gateway Engine class ensures that the operation of application threads, UARTs and modem are terminated correctly to prevent undefined system behaviour.

The application incorporates a text-based user interface that permits the configuration of the application and is accessible through either one of several connections: Ethernet, UART or GSM modem. The windows registry was configured to force the OS to launch the application after the initialisation of the OS (see Appendix G) was complete and to permit access to the application's configuration menus through a UART terminal emulator such as *HyperTerminal* or GPRS connection through the GSM modem. The application was designed in this manner to ensure that access to the application was possible after a system reset in the absence of an Ethernet network connection. In addition, it is not possible to terminate the application through the terminal nor GPRS connection as this would render the gateway inoperable; the application may only be launched and terminated manually in a command shell accessible through a Telnet client via an Ethernet connection.

The selection of a particular user interface may be configured by passing one of the following arguments to the application at its invocation. Additional options permit the application to terminate multiple application instances, prevent the GSM modem from initialising and permit a resetting of the gateway configuration. The permitted control arguments are:

- `eth` steers the user interface to the current command shell accessible through a Telnet client via an Ethernet connection
- `kill` terminates any executing instances of the gateway application
- `nogprs` prevents the initialising of the GSM modem after an application launch
- `reset` resets the application configuration to default values; the user interface is through a terminal emulator via the UART
- `eth reset` resets the application registry configuration; the user interface is through a Telnet client via an Ethernet connection

The application steers the user interface to a terminal via a UART if no argument is passed to the application at launch time.

Figure 3.2 shows the flow of the initialisation of the application and the contents of the main programme loop found in the class *GatewayEngine*.

The class *ConsoleUserInterface* is executed in a separate thread which permits the configuration of the application by a user via the selected interface without interfering with the operation of the logging and FTP threads. *ConsoleUserInterface* displays a list of commands in a menu to assist the user to review and configure the application settings stored in the registry. Further details may be found in section 3.4.4.

The values stored in the registry keys may be reset to default hard-coded values by the method *createRegistrykeys* if either of the arguments `reset` or `eth reset` are passed to the application at launch.

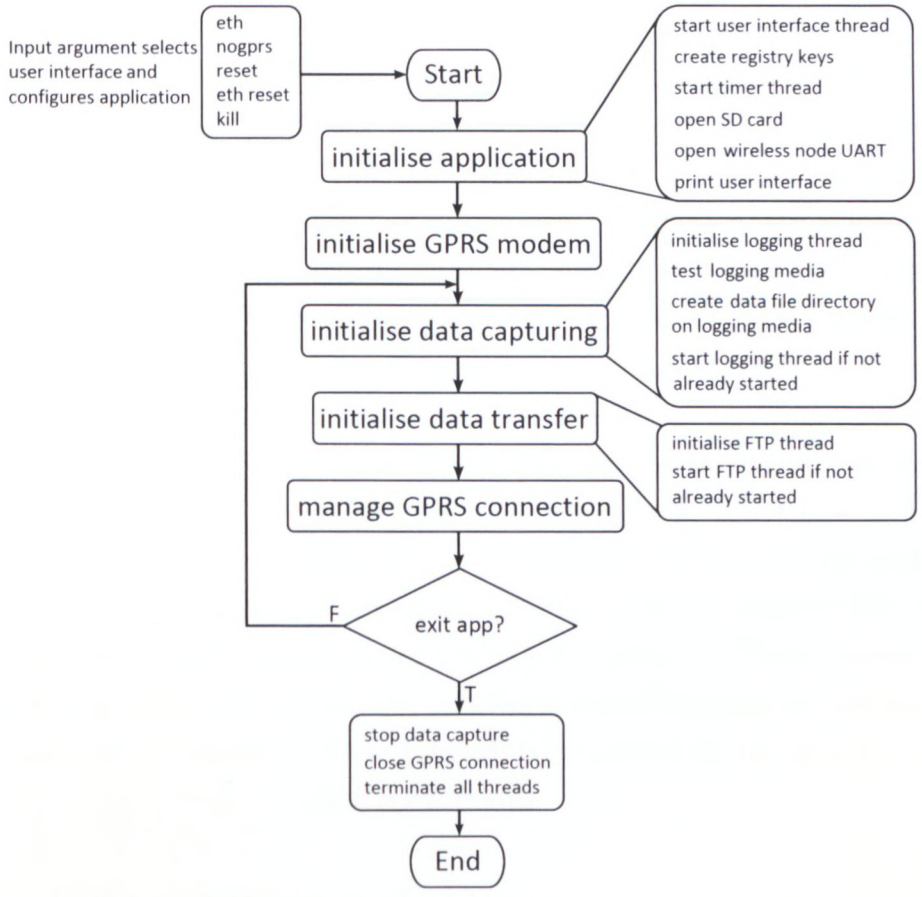


Figure 3.2: The class *GatewayEngine*

The method *openSDcard* ensures that the values in the registry are set to ensure that the memory card is initialised after the start of the OS. A test of the directory structure of the card is performed to determine if (a) the card exists (b) the card is mounted with the correct path name in the OS. Data packet capturing is not started if either of these tests fail.

The GSM modem is only powered up and initialised to accept a remote GPRS connection from the Internet if either *no argument* or an *invalid argument* is passed to the application. The power supply on the communication module is cycled to ensure that the GSM modem is in a reset state and disconnected from the GSM network before its initialisation begins.

The data logging and FTP threads are initialised only if they're not already in execution. The presence of the memory card storage media is confirmed and a new logging directory is created if it does not already exit. The logging and FTP initialisation methods are part of the method *runningLoop*, a continuous loop that ensures the continual capture of data and FTP functionality when enabled. The GPRS connection is managed by *modemConnectionManager* which permits the application to keep track of the connection state of the GSM modem at all times during GPRS and FTP connections.

The application may be terminated by entering *quit* at the prompt in the main menu or *kill apps* in the *utils* menu. *kill apps* sets a value in the *RemoteKill* registry key which forces the termination of all instances of the gateway application that may be executing on the OS which is useful if an application has stopped functioning and cannot be terminated via its user interface. These commands are functional only when the application is being accessed through a command shell via a Telnet client; resetting of the gateway is permitted by the *reboot* command and is functional via all interface connections.

3.4.2 Data reception and storage

3.4.2.1 COM0 Data Received event

Packets of data of a defined size are received from the wireless node through *COM0*, a virtual COM port provided by the installed FTDI VCP driver, which is opened and configured by the class *UART* in the source code. *COM0*'s *DataReceived* event in the class *UART*, as shown in Figure 3.3, is triggered after the specified number of bytes have been received from the node.

The number of bytes received *bytesReceived* is added to the pointer *MOTE_UART_DataIn_* to test that their sum is less than *MAX_IN_BUFFER_SIZE*. The new data is copied to the buffer *rawNodeDataIn* and the pointer *MOTE_UART_DataIn_* incremented by *bytesReceived* if this condition is true otherwise *MOTE_UART_DataIn_* wraps to zero to prevent a buffer overflow, lost data and undefined application operation.

The received data is copied to a TCP socket if the flag *streamData* is set by the user interface. This permits a remote user to monitor the raw data packets received from the wireless sink node.

DataReceived event is executed in a separate thread which is managed entirely by .NETCF. The code for the *DataReceived* event was kept to a minimum to ensure that it was executed in as short a time as possible to permit high data throughputs.

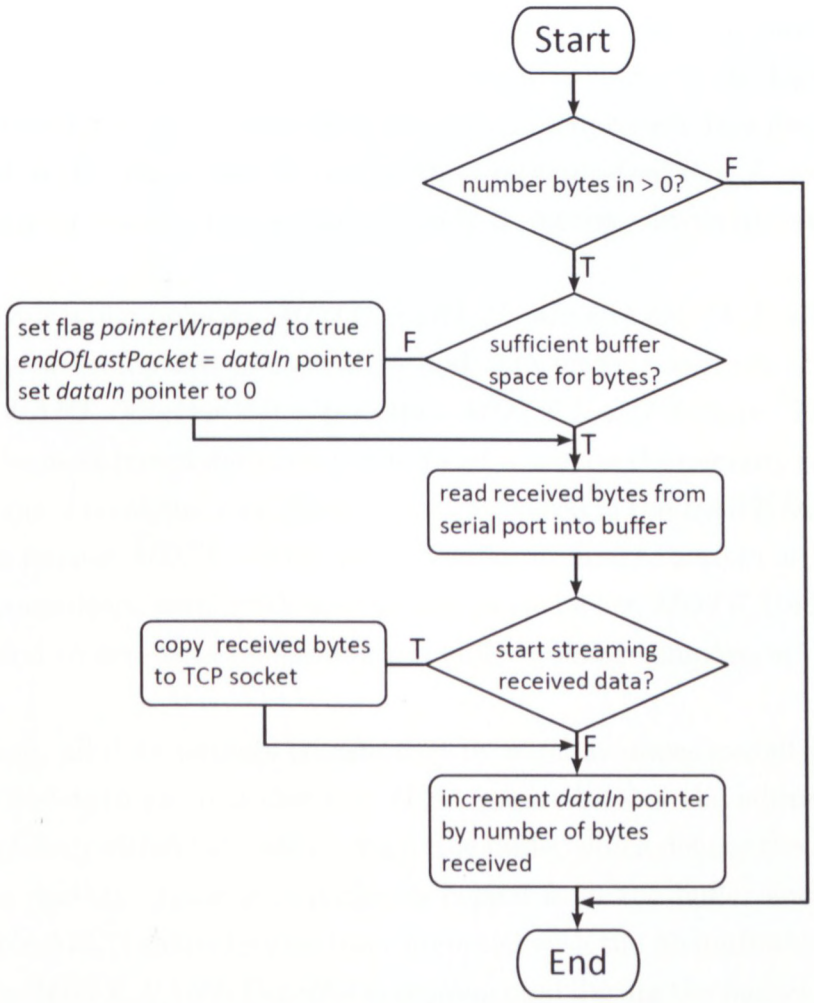


Figure 3.3: The *DataReceived* event is triggered when a defined number of bytes are received in *COM0*'s UART buffer.

3.4.2.2 storeData method

The purpose of the method *storeData* in the class *LoggingThread* is as follows:

- create a new data file when a change of date is detected
- compress the previous day's captured data file and modify its extension
- parse the data stored in the buffer *rawNodeDataIn* for packet headers
- convert packets from binary into equivalent ASCII character codes
- place a timestamp derived from the system clock into the converted data
- write the timestamped data to a data file on the SD memory card

storeData starts by comparing the system date with the current logging date in order to determine if a new day has started and if the previous day's data file needs to be compressed. If a new day is detected, a listing of the logging directory on the memory card is performed and any uncompressed data files found are compressed in the *gzip* format by the method *compressLogFiles*. A new data file with the current system date is created ready to receive converted data.

The in and out buffer pointers, *MOTE_UART_DataIn* and *MOTE_UART_DataOut* respectively, are compared to each other and data packets read out of the buffer at *MOTE_UART_DataOut* if it is **less** than *MOTE_UART_DataIn*. This ensures that only the most recent data is converted and stored on the memory card. A flag *pointerWrapped* is set and *endOfLastPacket* initialised in the *UART.DataReceived* event when pointer *MOTE_UART_DataIn* wraps around to zero in order to override this comparison. *pointerWrapped* is only cleared when *MOTE_UART_DataOut* wraps around to zero thus reinstating the buffer pointer comparison.

By definition, all data packets transmitted by wireless nodes executing *TinyOS* begin and end with an identifier hex *7E*. A parser locates the addresses of the packet identifiers within the data stored in the buffer which defines the boundaries of the data packets. Each data packet is copied from the buffer and converted into readable ASCII characters and concatenated with the *StringBuilder* array *sb*. The pointer *MOTE_UART_DataOut* is incremented during the packet conversion and wrapped around to zero if *endOfLastPacket* is exceeded.

Ten converted packets are accumulated in *sb* before being written to the data file to limit the number of function calls to the to the system clock, to reduce the number accesses to the memory card and data file. *packetCounter* reset to zero and *sb* is reinitialised with a timestamp ready for the next set of packets.

dataStore executes within a continuous loop that is exited when the gateway application is terminated, as shown in Figure 3.4.

3.4.3 File transfer

The class *FTPThread* contains the method *sendLogFiles* that is responsible for the scheduled or forced transferring of compressed data files stored on the memory card to a remote server located on the Internet by either the Ethernet or GSM connections. The connection options, server IP address, usernames and passwords may be configured by the user in the *ftp* menu item in the user interface, as shown in section 3.4.4. User-configurable iteration counters and timers are implemented to ensure that several attempts are made to establish a connection to the Internet and to complete the FTP file transfer if either fails during its operation.

This section discusses the control loop structure of *sendLogFiles* and file transfer via an Ethernet connection only; the transfer of files via a GSM connection is discussed in section 3.4.5.

sendLogFiles consists of three nested loops: the first loop is continuous and contains the code to perform the file transfer. It is exited when the gateway application is terminated; the second establishes a connection, at the scheduled time, to the Internet and remote FTP server and initiates an FTP session; the third performs the file transfer via the configured connection type and the termination of the FTP session and Internet connection.

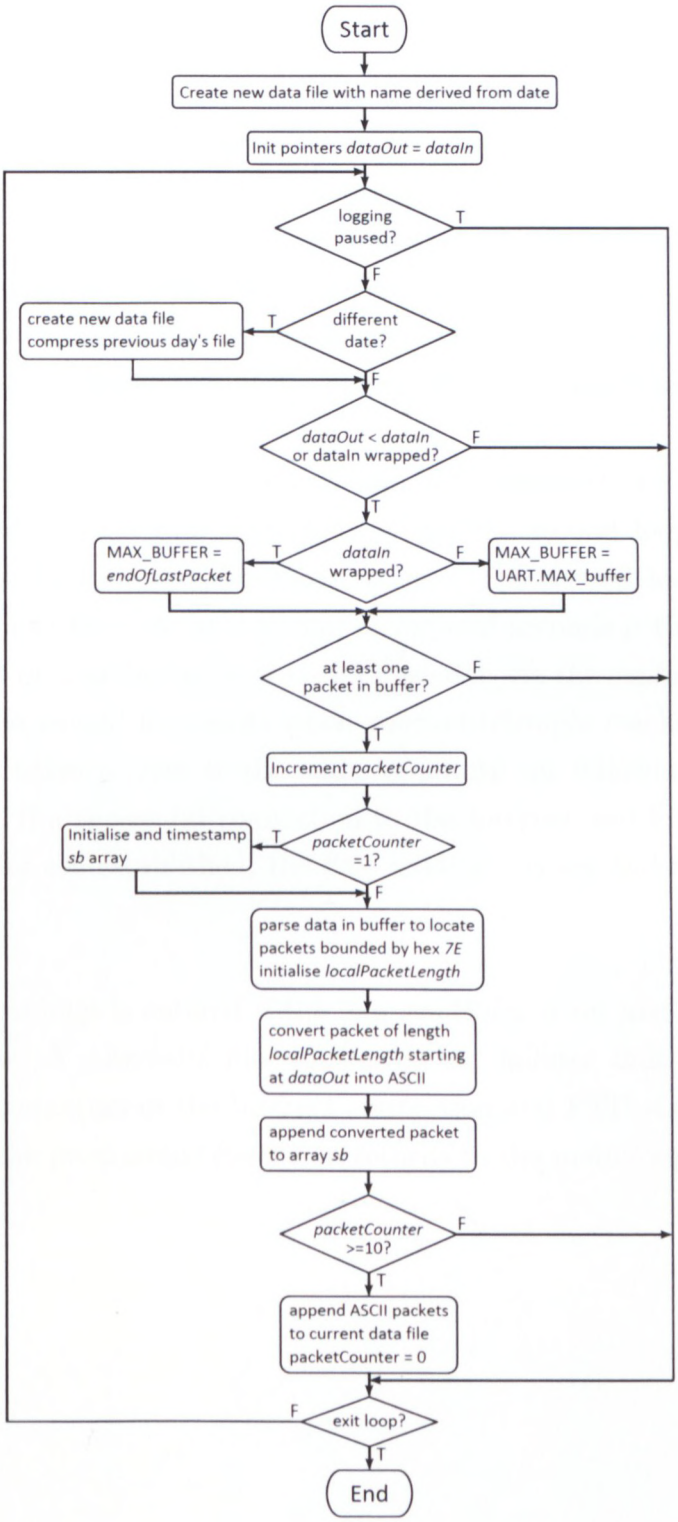


Figure 3.4: The *dataStore* method checks for new data stored in buffer *rawNodeDataIn*, which it parses to locate packets bounded by the identifier hex 7E. The packets of data are converted to ASCII characters and accumulated in *sb*. Ten accumulated packets are written to the data file at a time to reduce the number of accesses to the memory card.

3.4.3.1 *sendLogFiles* control loops

The first or main loop in *sendLogFiles* polls the registry to obtain the most recent configuration of the data file directory and the scheduled file transfer time and is executed every ten seconds to reduce the frequency of access to the registry and system clock, as shown in Figure 3.5. A comparison of the system clock and configured transfer time sets a flag *startFileTransfer* if a match is found. The time comparison may be overridden by the flag *forceSendLogs_* which is set by a user wishing to force the transfer of any stored files before the scheduled transfer time.

The variables *connectAttempts*, *interval* and *FTPtransport* are initialised with configuration values read from the registry and the second loop entered if either *startFileTransfer* or *forceSendLogs_* is set. The second loop iterates and decrements *connectAttempts* at a period of *interval* seconds if the Internet connection attempt or establishment of an FTP session via the configured transport option fails. The second loop exits when *connectAttempts* reaches zero and the programme execution returns to the main loop until the following scheduled file transfer occurs. If a successful connection to the Internet and FTP session with the remote server are established, the flag *sendFiles* is set and the second loop exited.

The third control loop is entered if the flag *sendFiles* is set and sends the compressed data file. A successful file transfer or any failures that occur during it result in the termination of the Internet connection and FTP session. The third loop exits and the programme execution returns to the main loop.

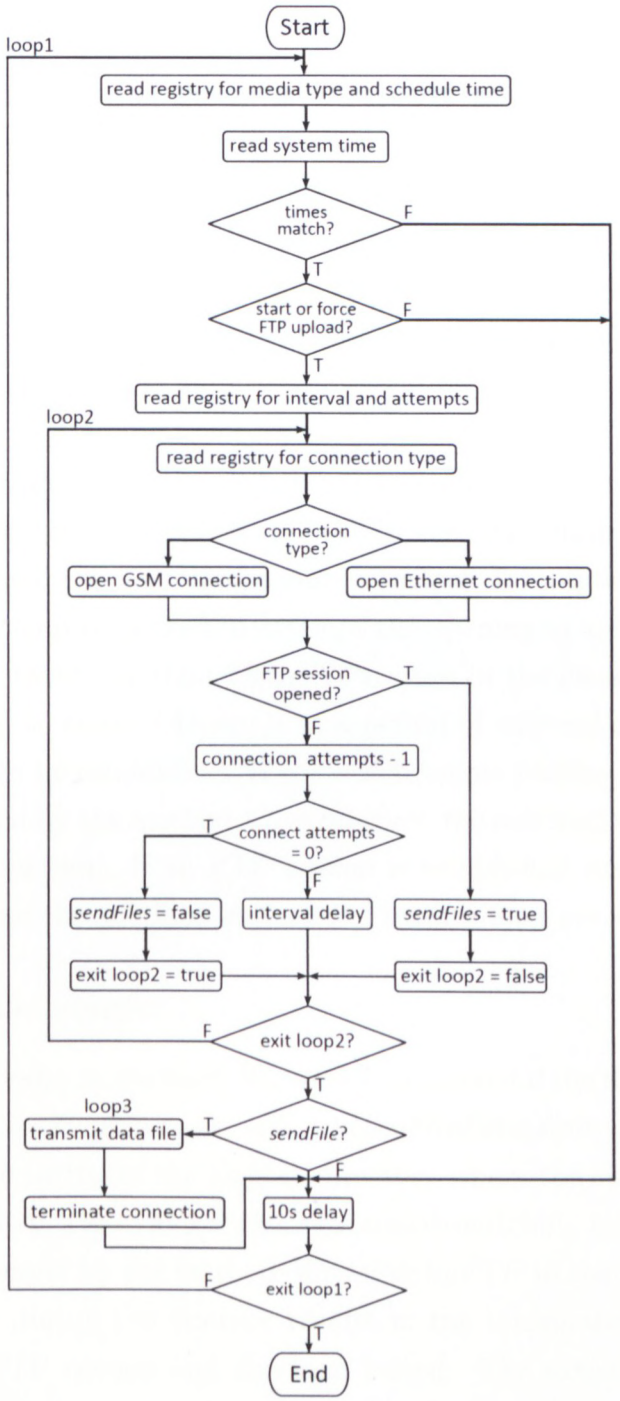


Figure 3.5: The FTP file transfer process is performed by *sendLogFiles* and consists of three loops: the first or main loop executes every 10s and is exited when the application is terminated; the second loop attempts to connect to the Internet and establish an FTP session; the third loop performs the file transfer and terminates the FTP and Internet connection.

3.4.3.2 FTP session initialisation

The second control loop, as shown in Figure 3.6 attempts to connect to the Internet and establish an FTP session with a remote server at the IP address retrieved from the registry. The server login details are retrieved from the registry and initialise the variables *serverIP*, *userName* and *passWord*. The user configuration determines the connection options: Ethernet or GSM modem. This section discusses FTP via an Ethernet connection only; FTP via a GSM connection is discussed in section 3.4.5.3.

The method *OpenInternet* in the class *Win32* is called to instruct the OS to open an Internet connection on the Ethernet port. If the connection attempt fails, the second loop exits and the main loop resumes control. If an Internet connection is established, a sub loop is entered to attempt the opening of an FTP session with the remote server by the method *OpenFTPsession* in the class *Win32*. The sub loop is iterated up to *connectAttempts* at a period of *interval* seconds if an FTP session is unable to be established. If *connectAttempts* reaches zero, the Internet connection is closed by the method *CloseInternet*, the sub loop exited and control returns to the main loop. If an FTP session is established successfully, the flag *sendFiles* is set and the sub loop and second control loop are exited.

3.4.3.3 FTP file transfer

The third control loop, as shown in Figure 3.7, is entered if the flag *sendFiles* is set and the variables *logFileExtension* and *sentLogFileExtension* are initialised from the registry. A file listing of the logging directory where the stored data files are located is performed. Those files with an extension matching *logFileExtension* are sent to the FTP server by the method *WriteFileByFTP* in the class *Win32*. Any error that occurs during the transfer results in the termination of the Internet connection and FTP session and the loop exited. The extensions of data files transferred successfully are renamed to *sentLogFileExtension* which ensures that the same files are not transmitted again in subsequent FTP transfers; the Internet connection and FTP session are terminated and the loop exited.

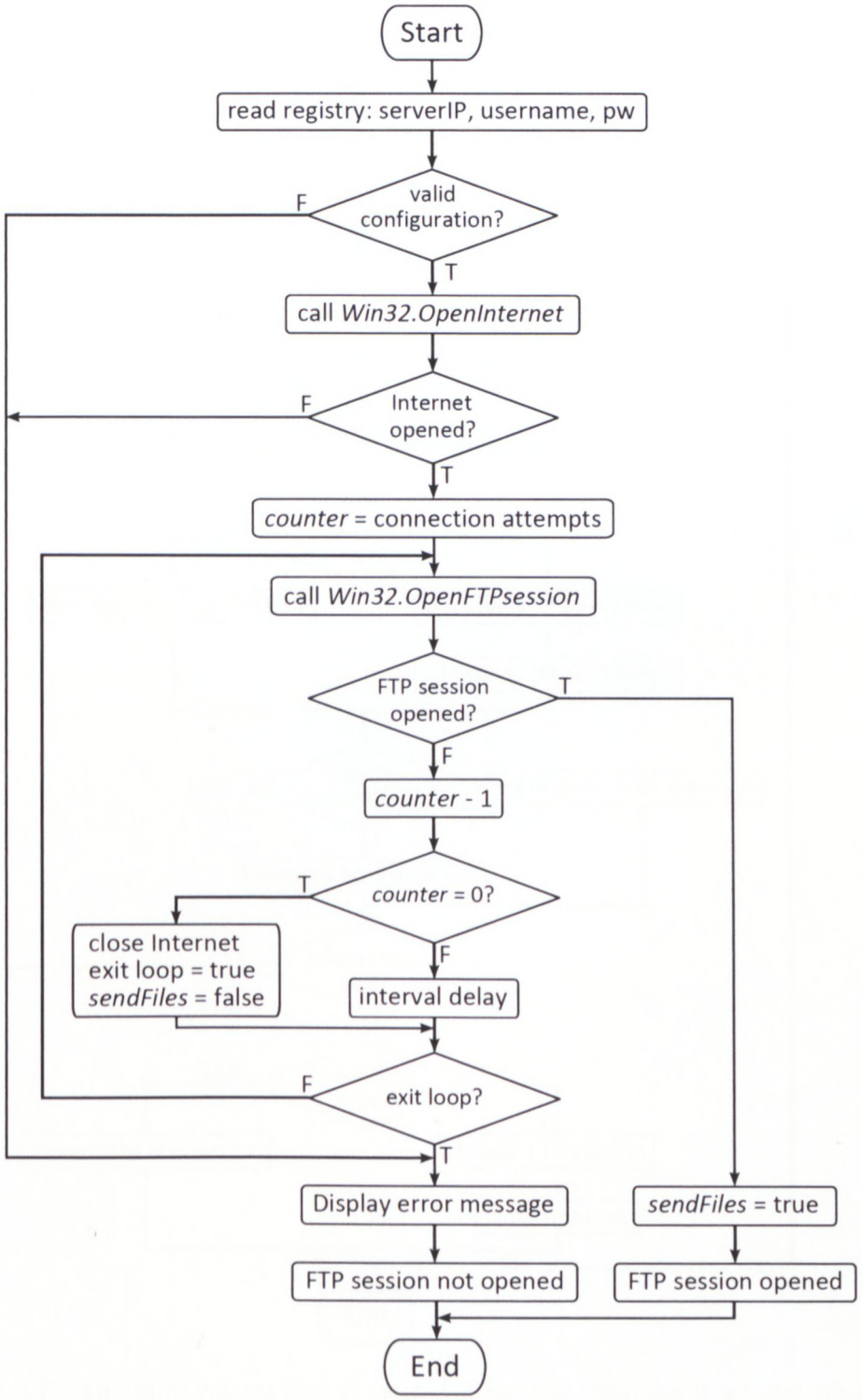


Figure 3.6: An FTP session is established in the second control loop before a data file is uploaded to a remote server. A sub-loop decrements *counter* to ensure that several attempts are made to establish an FTP session.

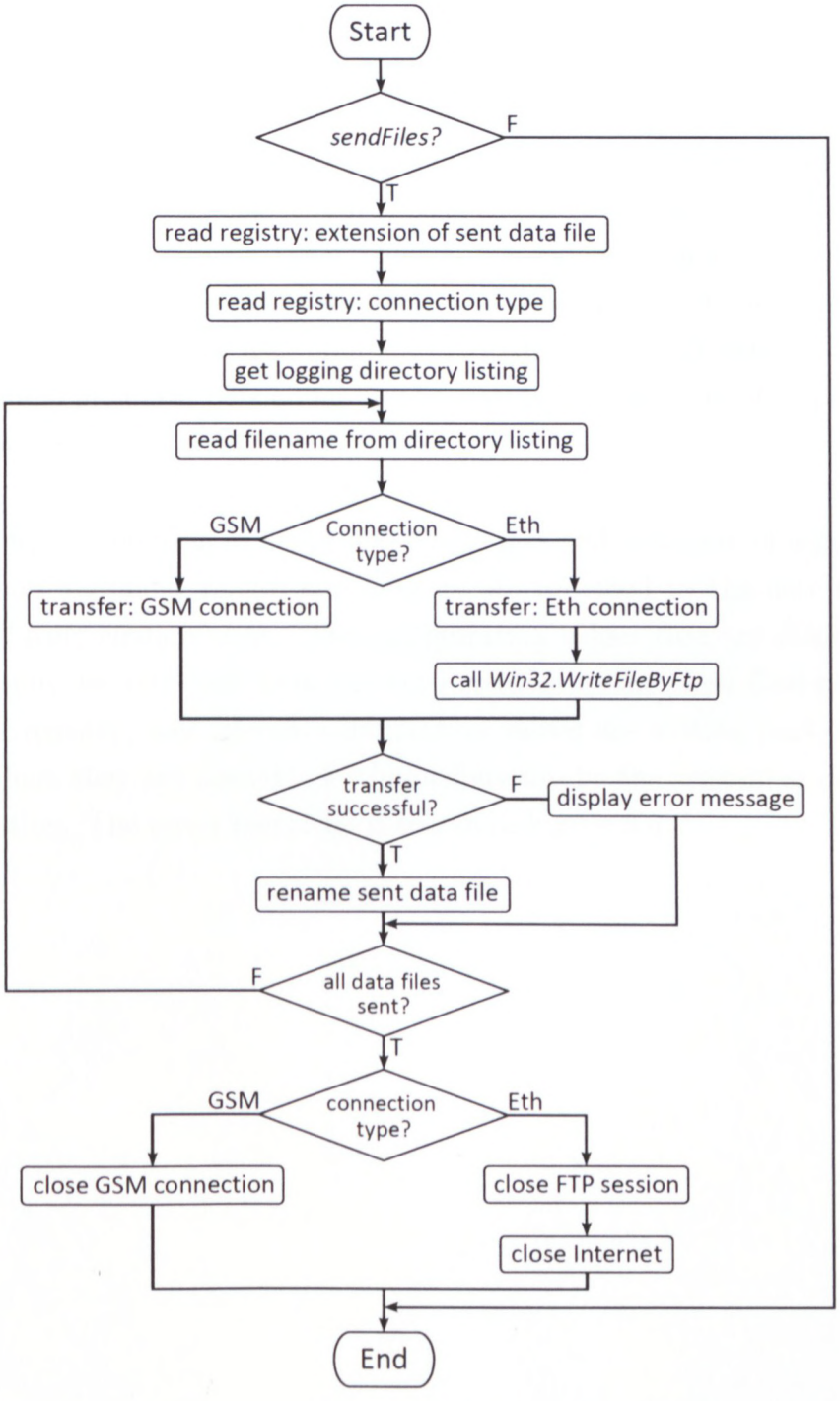


Figure 3.7: The third control loop is entered if the flag *sendFiles* is set and transfers all previously unsent compressed data files. The FTP session and Internet connection are closed after the file transfers.

3.4.4 The user-interface

The gateway application incorporates a user interface (UI) found in the class *ConsoleUserInterface* that permits the configuration of the application and troubleshooting of the gateway hardware. The UI is text-based and is presented as a hierarchical system of menus and sub menus each pertaining to a specific function offered by the application. The UI runs independently of the data logging and file transfer operations permitting real-time control of the application and gateway hardware. The user interface is accessible through either a Telnet client or terminal emulator depending on the arguments passed to the application, as discussed in section 3.4.1.

Assistance in the form of explanatory messages and examples of acceptable configuration argument ranges and formats are provided to the user in each sub menu during configuration. The configuration values that are displayed in the sub menus are retrieved from the keys created by the class *GatewayEngine* in the OS registry; any altered configuration values are written back to the same keys where they are available for immediate use by the remaining classes in the application. The menu hierarchy is shown in Figure 3.8.

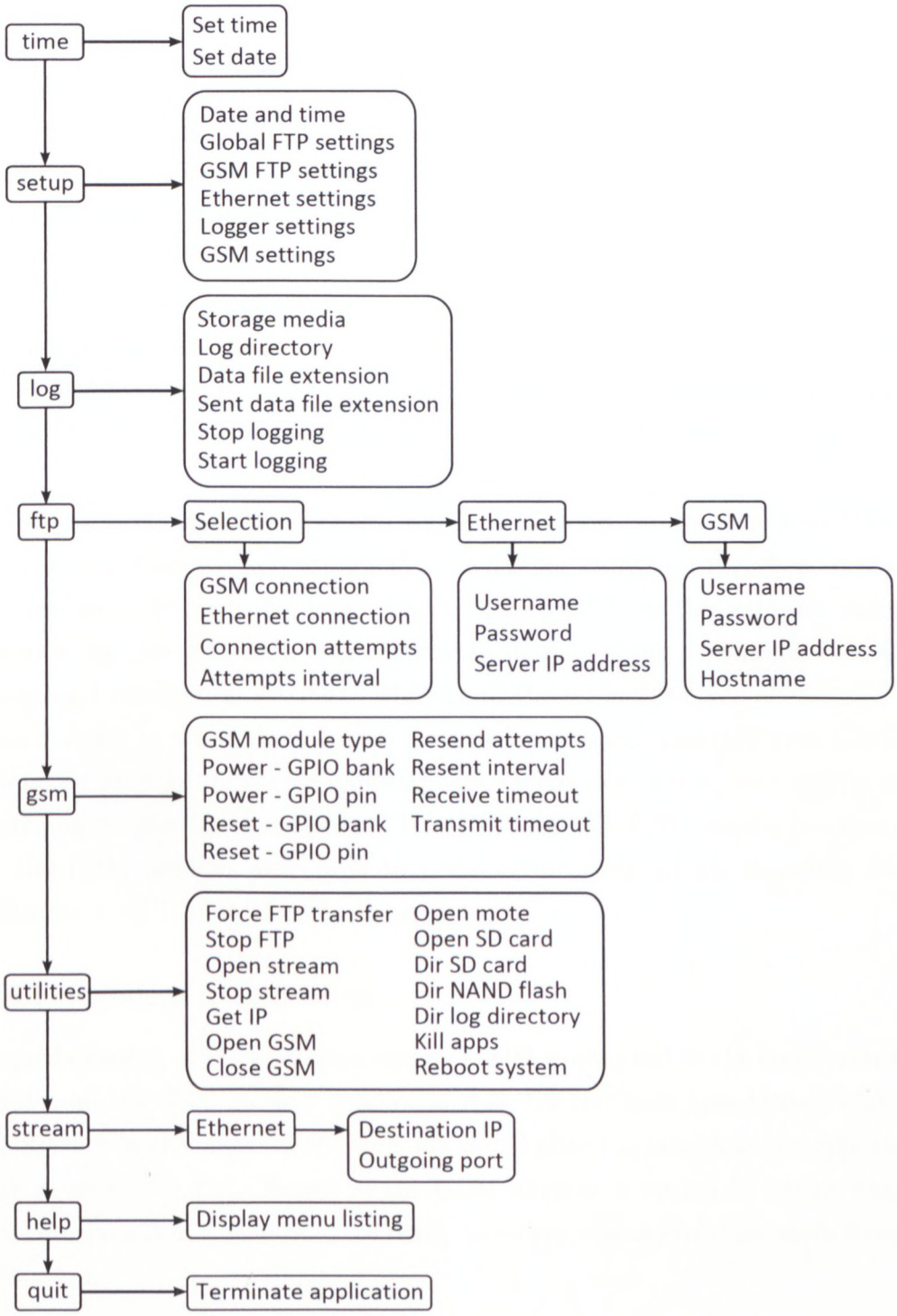


Figure 3.8: Gateway application user interface structure

3.4.5 GSM control

Little mention has been made thus far about the GSM functionality of the gateway communication module. The design process was structured in this sequence in order to have a stable and operational software framework on which to begin the development of the more intricate GSM control software.

An investigation to ascertain how a GPRS connection is established by WinCE was undertaken. It was found that *Dial-Up Networking* and *Connection Manager* modules are employed by WinCE to manage GSM modem control and to establish GPRS connections to the Internet (Microsoft Corporation, 2004).

In addition, it was found the use of *Dial-Up Networking* and *Connection Manager* have complex PInvoke and argument marshalling requirements when controlled programmatically from .NETCF. The lack of a GUI on the gateway platform prevented the use of in-built user controls in these modules to configure the GSM modem and connection settings. The use of these modules was abandoned and it was decided to write the software necessary to control and configure the GSM modem, to manage its power requirements, connection status, and timing when interfacing to the GSM network. The FTP and TCP/IP stacks incorporated into the GSM modem were used to perform the required file transfers and to implement a GPRS interface to the gateway.

3.4.5.1 Modem initialisation

The initialisation of the microprocessor's UART connected to the communication module and the GSM modem is performed by the methods *openModem* and *initGPRSserver* in the class *GSM* which are called after the launch of the application (refer to section 3.4.1). Power to the GSM modem is cycled to ensure that its serial interface is in a default state ready to receive commands and data from the application.

A period of approximately 20 seconds is required for the GSM modem to be registered on the GSM network. The AT command *ATE0* is sent to the modem to

test the functionality of the serial interface and to terminate the default echoing of transmitted characters back to the application. The modem responds with *OK* which confirms the the functionality of the serial interface. The modem power is cycled and the initialisation is attempted again if an *OK* response is not received.

The following configuration sequence is applied to initialise the modem to receive GPRS connections from the Internet:

PDP context definition

The method *definePDPcontext* configures the Packet Data Protocol (PDP) context which determines the packet structure of the data to be transmitted across the network and the Access Point Name (APN) connection to the GSM network. The modem is configured to send *IP*-type data and to connect to Vodacom's *unrestricted* APN which permits incoming connections to a device registered on the GSM network.

GPRS socket configuration

The method *configureGPRSsockets* configures a socket within the defined PDP context which permits a GPRS connection from a remote client on the Internet. The GPRS socket is configured so that the TCP/IP packet size is 1000 bytes and the transmit timer is set to 50ms. The transmit timer ensures that a packet which is only partially full is transmitted to the Internet.

Socket answering behaviour configuration

The method *configureSocketAutoanswer* configures the modem to answer an incoming GPRS connection on the defined socket automatically and to send the response *CONNECT* to the application.

PDP context activation

The method *activatePDPcontext* activates the PDP context defined at the start of the initialisation sequence. The network allocates a public IP address to the modem which is sent to the application.

Firewall configuration

The method *configureFirewall* configures a firewall incorporated into the modem as a means of filtering the connection requests to the modem. An IP filter may be configured through the user interface which permits connections from a range of IP addresses to be accepted.

Listening socket configuration

The method *configureListeningSocket* configures the modem to receive GPRS connections on a specific IP port that is bound to the socket defined earlier. The IP port number may be configured through the user interface.

3.4.5.2 GPRS communication

The modem is configured to receive connection requests from remote TCP/IP clients on the Internet after the completion of the initialising sequence. The method *modemDataReceiver* in the class *UART* receives responses sent by the modem over the serial interface. The responses are parsed for text such as *OK* or *CONNECT* and are tracked by a state machine in the method *modemConnectionManager* which determines the operation of the application.

If a valid connection request is received from a remote client, the modem sends the response *CONNECT* which changes the application's operational state from *LISTEN* to *CONNECT*. The *CONNECT* state is detected by the methods *printToConsole* and *readFromConsole* in the class *ConsoleUserInterface* which steers the application's user interface to the GSM modem's UART. The modem wraps the text to be sent with TCP/IP headers and transmits the resulting packets to the client via the Internet. Similarly, packets sent by the remote user are received by the modem and the TCP/IP header data stripped away permitting the text to be sent to the application via the serial interface.

All data received from the modem is parsed by *modemDataReceiver* for the responses *NO CONNECT* or *ABORT* which indicate that the remote client has been disconnected or that a network error has occurred. The application's operational state is changed to *NO CARRIER* and the GPRS listening socket is reset

and reconfigured to accept subsequent GPRS connections. The application's operational state is changed back to *LISTEN* which steers the user interface back to the interface-type configured at the application launch.

3.4.5.3 FTP initialisation

The PDP context established during the initialisation of the GSM modem permits the transfer of files by FTP to a remote file server by means of its inbuilt FTP stack. The GSM modem requires additional configuration in order to transfer files at the scheduled time or when forced to do so by a user (refer to section 3.4.3.1).

sendLogFiles calls the method *openFTPlink*, if the user has selected a file transfer via the GSM connection, which configures the modem to establish a session with a remote FTP server before the start of a file transfer. *openFTPlink* performs the following additional configuration:

GPRS socket shutdown

The modem is configured to receive incoming requests for communications on the defined GPRS socket. The GPRS socket must be disabled temporarily to permit the transfer of data via FTP and reinstated at its conclusion. The socket is shutdown by the method *shutdownSocket* and the application's operational state changed to *FTP_INIT*.

Modem handshaking

The method *setModemHandshaking* configures the modem's serial interface to use *Xon/Xoff* handshaking to control the stream of data sent to it during a file transfer to prevent data buffer overflows from occurring. The rate of data transmission is dependent on the throughput capability of the GSM network at the time of transfer which may be slower than the data transmission rate of the application.

Data is sent a byte at a time to the modem's serial interface by the method *sendBuffer* in the class *UART* until the modem responds with an *Xoff* character when its buffer cannot receive further data.

modemDataReceiver receives this character and pauses the transmission of data to the modem. The modem sends an *Xon* character once the data in its buffer has been transmitted which resumes the flow of data to the modem.

Establishing an FTP session

The method *openFTP* in the class *GSM* establishes a session with a remote file server at the IP address or hostname and logs on to it with the configured username and password. An IP address for the hostname is resolved by a Domain Name System (DNS) query performed automatically by the modem. It may take several seconds for a session to be established with the server and is dependent on the data throughput of the GSM network at the time of the transfer. The flag *sendFiles* is set after the session is established.

If any of the above configuration steps fail, the file transfer is abandoned, *openFTPlink* is exited and the programme execution returns to the calling loop (refer to section 3.4.3.2) where several subsequent attempts to send the file are made.

3.4.5.4 FTP file transfer

The transmission of a file begins after the successful establishment of an FTP session with the file server. The third operational loop, as discussed in section 3.4.3.1, is entered if the flag *sendFiles* is set and the variables *logFileExtension* and *sentLogFileExtension* are initialised from the registry. A file listing of the logging directory where the stored data files are located is performed. The name of a file with an extension matching *logFileExtension* is passed to the method *putFTPfile* in the class *GSM* which creates a new file of the same name on the server.

The method *sendFile* in the class *FTPThread* (refer to Figure 3.9) transfers the data file to the modem for transmission to the newly created file on the server.

Any error that occurs during the transfer results in the termination of the FTP session and the loop is exited. The extensions of data file transferred successfully is renamed to *sentLogFileExtension* which ensures that the same files are not transmitted again in subsequent FTP transfers. The transfer process is reiterated for each file found in the directory file listing as shown in Figure 3.7.

The method *closeFTP* terminates the FTP session by calling the method *disconnectingFTPsession*. The modem *Xon/Xoff* handshaking configuration is returned to *none* by the method *clearModemHandshaking* as handshaking control is not implemented elsewhere in the application. The application's operational state is changed to *FTP_CLOSED* which is detected by *modemConnectionManager*. *modemConnectionManager* calls *enableSocket* which reinstates the GPRS connection or configures the modem to receive subsequent GPRS connection requests. The application's operational state is changed to *CONNECT* or *LISTEN* depending on the state of the application before the initialisation of the FTP transfer.

The programme execution returns to the main control loop in *sendLogFiles*.

3.4.5.5 The operation of *sendFile*

The method *sendFile* sends a data file to the GSM modem which transmits it to a remote file server after the FTP session has been initialised and a new file created on the server. Figure 3.9 shows the programme flow of *sendFile* which is discussed in detail in the following section.

The data file to be transferred is opened for reading and the variable *bytesRemaining* initialised to the number of bytes contained in the file. The file is read in packets of up to 300 bytes which are sent to the modem a byte at a time by the method *sendBuffer* in the class *UART*. *bytesRemaining* is decremented by the number of bytes sent to the modem during the transmission and indicates that the entire file has been transferred when it reaches zero.

Xon/Xoff handshaking is implemented to ensure that the modem's input buffer is not over loaded with data from the application, resulting in possible undefined modem operation or data loss. *Xon/Xoff* handshaking utilises two control characters, $Xon = 17$ and $Xoff = 19$, to control data flow and are sent to the application via the serial interface. *Xon* is sent when more data may be sent and *Xoff* when the data transfer must stop. Handshaking is implemented in one direction only, from modem to application, as large amounts of data are not received by the application during a file transfer thus negating the need for handshaking in the other direction.

The method *modemDataReceiver*, which executes in the thread *modemHandshakeThread*, receives the *Xon* and *Xoff* control characters which set or clear the flag *stopTX*. *stopTX* is utilised by *sendBuffer* to stop or resume data transfer to the modem. A timer, which is clocked by an application tick, is implemented in *sendBuffer* to ensure that the method is exited if an *Xon* character is never received from the modem preventing the application from remaining in *sendBuffer* indefinitely. The programme execution leaves *sendBuffer* and returns to *sendFile* once the data has been sent or timeout has occurred.

The loop in *sendFile* iterates until *bytesRemaining* reaches zero or a timeout from *sendBuffer* is detected and the data file is closed for reading.

An error message is displayed on the user interface if a timeout in *sendBuffer* occurred to alert the user of a potential file transmission error. An escape sequence consisting of three consecutive + characters is sent to the modem to bring it out of data mode and place it into command mode by the method *sendEscapeSequence* in the class *UART*. *sendFile* is exited and programme execution returns to *sendLogFiles*.

If no timeout error occurred in *sendBuffer* and *bytesRemaining* reached zero, a timer is started to ensure that the final packet in the modem buffer is transmitted before sending the escape sequence to it.

sendFile is exited and the programme execution returns to *sendLogFiles*.

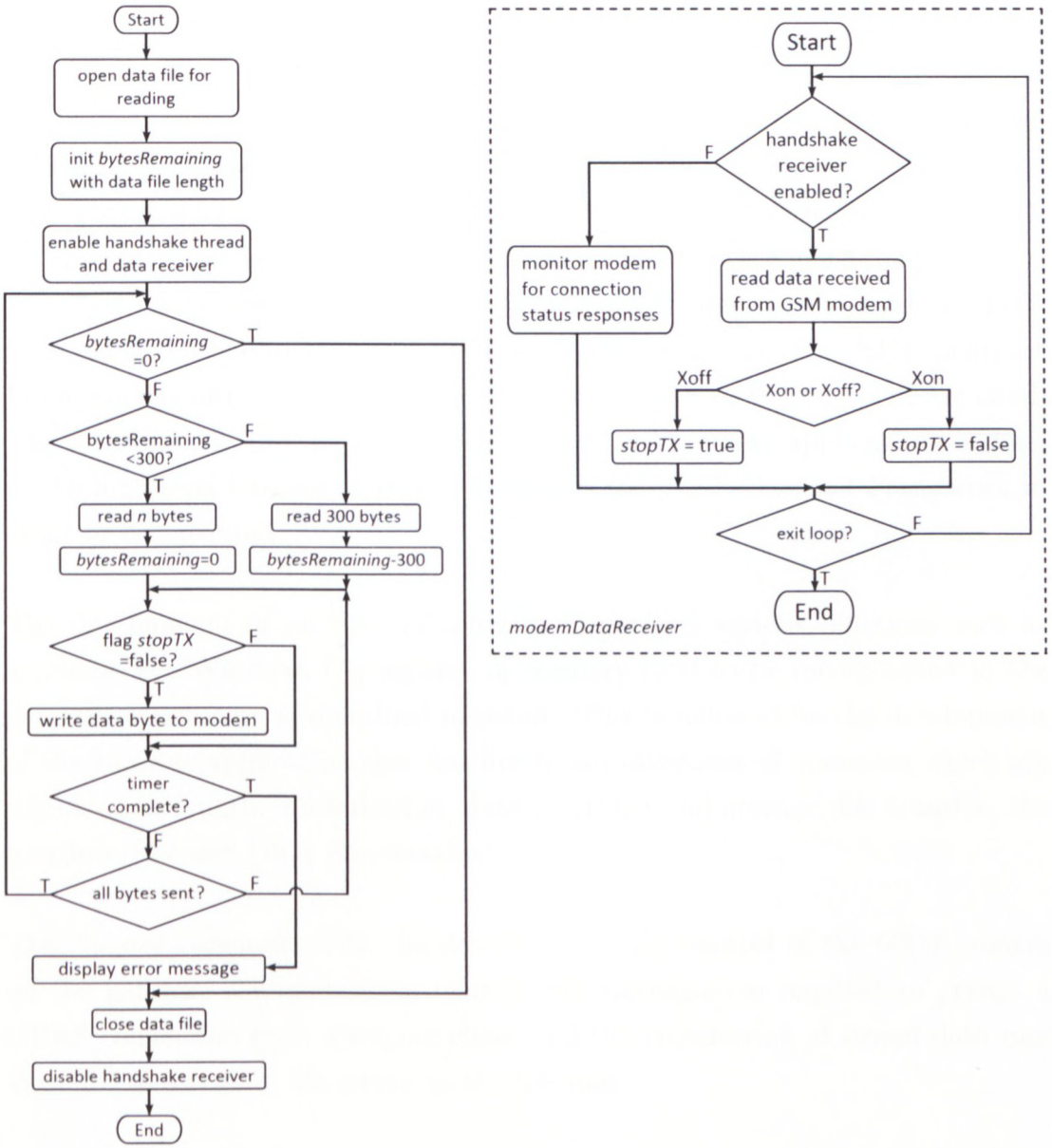


Figure 3.9: The methods *sendFile* and *modemDataReceiver*

3.5 Summary

This chapter discusses the choice of a suitable multitasking operating system for the gateway and the reasons for selecting Windows Embedded CE. A discussion of the tools Platform Builder and Visual Studio required to configure, build and deploy a Windows CE image to the hardware platform is given. The purpose of the bootloader programmes *FIRSTBOOT* and *EBOOT* generated during the building of Windows CE, their deployment to the hardware and configuration is described. A procedure to deploy the Windows CE image to the hardware platform using the bootloaders and Platform Builder is given. The .NET Compact Framework is introduced and reasons for its inclusion in Windows CE are cited. The development and deployment of the test and gateway applications is done in the high-level language C# which requires the .NET Compact Framework in order to be executed.

The development of modules of software to perform various functions such as accessing the Windows CE registry or memory card to be incorporated in the gateway application is described in detail. This is followed by the development of the gateway application that has five broad categories of operation which are discussed at length: initialisation, data reception and storage, file transfer, the user interface and GSM functionality.

The chapter concludes with the description of the control of the GSM modem on the gateway communication module, the initialisation required to permit a GPRS connection from a remote client and the transferring of stored data files via FTP to a remote file server on the Internet.

Chapter 4

Hardware Development

4.1 Overview

The following chapter discusses the development and implementation of the final hardware design of the gateway based on the requirements and specification listed in section 1.4.2. A high-level block diagram divides the design proposal into functional modules with their interfaces which are discussed in detail. Particular attention is paid to the power supply which is required to supply its output voltages in a specific sequence at startup. The chapter concludes with details relating to the layout and production of the printed circuits boards required for the study.

4.2 Gateway system modules

A gateway was required to receive and store packets of data sent by a wireless sensor node through its USB interface in a persistent memory. Periodic transfers of the stored data to a file server located on the Internet was also required. In addition, a configurable design that permitted several Internet connection types to be used for the file transfers was required. A fixed Ethernet interface was included in the design which provided a wired connection for access to the Internet. In order to satisfy the requirement of an additional, *configurable* Internet connection, the concept of a separate communication module incorporating any one of several Internet connection interfaces that connected directly to the main gateway platform was conceived. The communication module designed for this study incorporated a GSM modem permitting the deployment of the gateway into environments without access to a wired Ethernet connection.

Due to the remote nature of some of the possible deployment environments, the gateway was required to be able to be powered from a battery source and the power consumption required to be less than 3,5W during normal operation even at elevated ambient temperatures.

A block diagram in Figure 4.1 shows the topology of the overall system and the modules with their interconnections relating to the system functionality required to satisfy the specification. Details, such as part numbers and power consumption, regarding the specific components employed in the design are reserved for discussion in section 4.3.

The ARM9 microprocessor selected in section 2.2.2 was the central component in the design as it provided the necessary peripheral functionality required by the specification.

USB interfaces

The microprocessor incorporates a USB 2.0 compliant host port which was required as an interface to the wireless sensor sink node linking the gateway to a wider network of sensors.

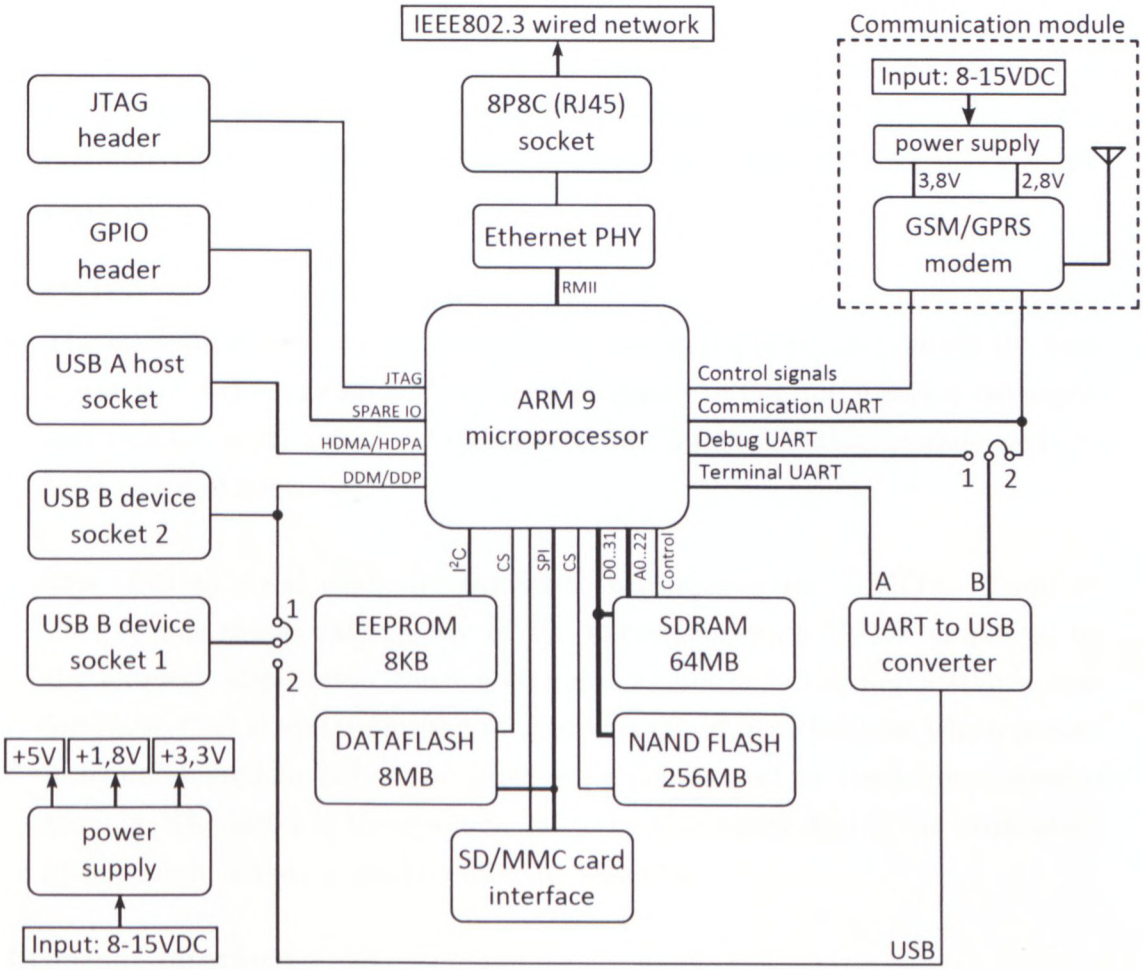


Figure 4.1: Gateway top-level block diagram

The USB host was terminated in a *USB A* socket and connected to the gateway's 5V supply rail to supply the node with power.

The design incorporated a *USB B* socket that serves a dual function depending on the setting of several configuration jumpers:

- the gateway may be configured to behave as a USB device when connected to a PC for example when the bootloaders were written to the gateway by SAM-BA
- the USB to serial converter required a suitable socket when connected to a PC executing a terminal emulator

Provision in the design was made for two *USB B* sockets in case both of the requirements listed above were needed simultaneously; only socket 1 was populated during the gateway PCB assembly as the dual functionality was not required in this study.

UARTs

The microprocessor has three UARTs: a debugging port used during the configuration of the bootloaders and for the display of OS initialisation messages, and two for serial communications with the communication module and the USB to serial converter.

The USB to serial converter provided two independent UARTs, A and B. Port A was connected directly to the microprocessor's UART employed by the gateway application when a terminal emulator is the application's user interface; Port B was connected to a set of configuration jumpers which permit it to be steered to either the processor's debug port or the communication module. The latter of these was found to be a necessity during the verification of the communication module after its assembly.

Ethernet interface

An 8 position 8 contact (8P8C), commonly referred to as an RJ45, socket was used in the design to provide the gateway with a connection to a wired Ethernet network. A Physical Interface (PHY) IC links the 8P8C connector to the microprocessor's Reduced Media Independent Interface (RMII) bus. The RMII interface is a reduced version of the full MII as it uses between 6 and 10 of the original 16 signals only and is capable of supporting data rates of 10Mbps and 100Mbps between the PHY and microprocessor.

User IO

The specification required that several unbuffered IO pins on the microprocessor were to made available on a PCB header to permit their use for the control of auxiliary equipment. The microprocessor control signals *wakeup* and *shutdown* were made available on the header though they remained unused.

JTAG interface

The microprocessor's JTAG interface was made available on a PCB header socket to permit debugging functions such as single-stepping through low-level code or boundary scans when necessary and was included for the sake of a holistic design. The header for this interface was not populated during the PCB assembly as the JTAG interface was never required during this study.

System memories

The design incorporates several memory devices that serve different functions. The 256MB NAND flash memory stores the Windows CE image, hive-based registry and the gateway application. It has an 8-bit parallel data bus which is connected to lines 0 to 7 of the microprocessor's 32-bit parallel data bus.

The microprocessor incorporates an SDRAM controller which provided the necessary signals to control two SDRAM memories connected to the microprocessor's data bus. Each memory IC has a 16-bit parallel interface that was connected to lines D0 to D15 and D16 to D31 of the data bus respectively. The combined memory capacity of the two ICs equates to 64MB which corresponds to a total of 16384 32-bit words.

An 8MB dataflash was used in the design to store the bootloader applications *FIRSTBOOT*, *EBOOT* and the configuration of NAND flash. The dataflash has an SPI interface which is shared with the SD/MMC memory card interface.

SD/MMC memory card interface

An SD/MMC memory card was chosen to provide the persistent storage medium for the data received from the wireless node as they provided excellent storage

densities in a very compact form factor. Many of these memory cards support multiple electrical interfaces such as the SPI bus, one-bit and four-bit SD buses. A four-bit SD interface was implemented in the design which permits data transfer rates up to 100Mbps.

Power supply

The power supply is responsible for providing constant voltages to the component supply rails in a hardware design. It is also responsible for ensuring that the hardware is protected against reverse polarity connections from the main power source.

A two-stage power supply consisting of a switched mode pre-regulator followed by linear regulators was incorporated into the design. A switched mode pre-regulator ensured that power losses and heat dissipation were kept to a minimum if an elevated voltage source was applied to the gateway. This resulted in improved efficiency of the supply as well as constant power consumption over a wide range of input voltages. The specification required the gateway to be polarity protected and functional over an input voltage range of 8V to 15V DC.

The output of the pre-regulator was required to supply the linear voltage regulators and the USB host socket to power the wireless node and USB to serial converter with 5V. Two low dropout (LDO) linear regulators were used to deliver 1,8V and 3,3V to the microprocessor core and the remainder of the circuit respectively from the 5V supply.

Communication module

A GSM modem was incorporated into the communication module designed for this study. Other technologies suitable for connecting to the Internet such as Bluetooth or WiFi radio modules may be incorporated into subsequent redesigns of the communication module provided that the communication and control interface remains compatible with the interface on the main gateway PCB.

The communication and control interface consists of a 9-wire serial port implementation including handshake and data signals, 3,3V from the gateway PCB and four control lines. Three of the control lines permit the gateway application to manage the power supply of the communication module and status of the modem. The fourth control line permits the monitoring of the modem's power status, however this was not implemented in the gateway application.

The power supply of the communication module used a similar design topology as the gateway's supply and was specified to operate over an input voltage range of 8V to 15V DC. The switch mode pre-regulator was configured to regulate its output to 3,8V which supplies power to the GSM modem and an additional linear regulator; the linear regulator's output voltage of 2,8V supplies the GSM modem's communication interface.

4.3 Design implementation

This section discusses the implementation of each of the functional modules introduced in the previous section in detail. Schematic diagrams of the final gateway and communication module designs may be found in Appendix B.

4.3.1 Microprocessor and memories

The AT91SAM9260 microprocessor (U4) by Atmel was selected for use in the gateway design as it incorporates the peripheral features required to fulfill the specification of this study (refer to section 2.2.2). The circuitry of the microprocessor and memory components form the central processing core of the design was based on the microprocessor's datasheet (Atmel Corporation, 2009*d*) and the circuit schematic supplied with the AT91SAM9260 evaluation kit (Atmel Corporation, 2007). The circuitry of the processing core was incorporated unmodified into the design of the gateway to ensure that the bootloaders and OS developed previously still functioned after their deployment to the new gateway hardware.

Clock sources

The microprocessor incorporates a clock generator module which consists of a main oscillator, two phase locked loops (PLL) and slow clock oscillator which derive their clocking sources from external 18.432MHz and 32.768kHz crystals (Y1, Y2) respectively. The Real-Time Timer (RTT) module is used by the OS to count elapsed seconds and receives its clock source from either the slow clock or internal RC oscillator depending on the state of the *OSCSEL* pin; *OSCSEL* was pulled high, enabling the slow clock oscillator source.

The values of the PLL second-order filter network (R13, C10, C11) on the *PLLRCA* pin were set to the same values as those on the evaluation kit schematic to ensure that its operation was functional in the final design. The PCB traces to the network were kept short to minimise degraded performance and high quality X7R capacitors and 1% resistors were used during its assembly.

Micrprocessor reset

The processor pins, shutdown *SHDN* and wakeup *WKUP* were pulled up to the 1,8V core supply with 1MΩ and 100kΩ resistors respectively and were made available on a PCB header (P4) in case of a future requirement.

The microprocessor reset pin *NRST* has active low, bi-directional functionality and was pulled high to the 3,3V supply rail with a 1kΩ resistor. *NRST* may behave as an output and be asserted by the the microprocessor to execute a reset of devices connected to it; or it may behave as an input to reset the processor and other devices when asserted low by a pushbutton switch (S1) connected to it. The Ethernet PHY and dataflash memory are connected to the *NRST* and are held in a reset state while *NRST* is asserted low.

Memory interface

The External Bus Interface (EBI) facilitates the control and access of external memory devices such as the NAND and SDRAM memories and ensures the successful transmission of data between these memories and the Memory Controller of the ARM processor core.

SDRAM memories

The SDRAM controller provides the microprocessor with a configurable interface to 16-bit or 32-bit SDRAM devices and supports 8, 16 or 32-bit data accesses. In addition, the SDRAM controller supports several CAS latencies 1, 2 and 3 and optimises the read accesses depending on its frequency. All the address clocking, the timing of the data transfers on the bus and control signals such as RAS, CAS and bank select required to interface to an SDRAM memory are generated by the SDRAM controller.

Two 32MB SDRAM memory ICs (U5, U6), the MT48LC16M16A2P-75 (Micron Technology Incorporated, 2011) by Micron, were utilised in the design. Each IC has a 16-bit parallel interface that was connected to lines D0 to D15 and D16 to D31 of the microprocessor's data bus respectively. Several of the microprocessor address lines A0 to A17 were connected to the SDRAM address inputs and control line *DQML*; the remaining control inputs CS, WE, CAS and RAS were connected to the corresponding outputs on the microprocessor, as shown in the datasheet (Atmel Corporation, 2009*a*).

NAND flash

The 256MB NAND flash IC (U7), a K9F2G08U0B (Samsung Electronics, 2008) by Samsung, has an 8-bit parallel data interface that is connected to D0 to D7 of the microprocessor data bus. The remaining control signals CE, ALE, RE, WE, CE and R/B are connected to the microprocessor pins according to the datasheet (Atmel Corporation, 2009*b*). The NAND flash is controlled by dedicated circuitry in the EBI which negates the need for external interface circuitry.

NAND memories contain invalid blocks with bits that are faulty. Samsung identifies bad blocks by programming their first pages with data other than hex *FF* which must be detected by the user application and avoided in normal use. Additional bad blocks in the NAND flash may occur with time and use. An Error Corrected Code controller (ECC) is implemented in the microprocessor to detect and correct errors in the data retrieved from the NAND flash. The

ECC is has the ability to correct single bit errors and to detect 2-bit random errors; data with two or more bit errors is not able to be corrected by the ECC.

Dataflash

The 8MB Dataflash IC (U8), an AT45DB642D (Atmel Corporation, 2011) by Atmel, has an 8-bit parallel and a serial SPI interface consisting of SCLK, MOSI, MISO and CS signals. The SPI interface was implemented to reduce the number of connections to the microprocessor and to simplify the PCB routing.

SD/MMC interface

The microprocessor incorporates a MultiMedia Card Interface (MCI) that supports the MultiMedia Card (MMC) specification V3.11, SDIO specification V1.0 and SD Memory Card specification V1.0. The MCI module contains the logic and error detection that performs command and data transmissions with a memory card and handles any responses thereby limiting the overhead to the microprocessor. SD card communications are based on a 9-pin interface consisting of a clock, command line, four data lines and three power lines. A four-bit SD and SPI interfaces were implemented in the design to satisfy the minimum MMC and SD specifications shown in the documents listed and terminate in a memory card connector CON4.

4.3.2 Ethernet PHY

The Ethernet PHY interface used in this design was based on the circuit schematic supplied with the AT91SAM9260 evaluation kit (Atmel Corporation, 2007) to ensure that the bootloaders and OS remained functional when deployed to the new gateway hardware.

The EK schematic incorporates a DM9161A Ethernet PHY interface device by Davicom which implements the physical layer functions defined by the IEEE 802.3 standard. The DM9161A is a low power, physical layer interface IC for 100Base-TX Fast Ethernet and 10Base-T Ethernet connections. It provides a direct interface to unshielded, twisted pair category 5 cable (UTP5) used for Fast

Ethernet or UTP3 regular Ethernet connections. Connection to the Medium Access Control (MAC) within the microprocessor is through an RMII interface, a reduced version of the full MII interface. The RMII interface uses between 6 and 10 of the MII's 16 signals and is capable of supporting data rates of up to 100Mbps between the PHY and microprocessor.

The EMAC module incorporated within the microprocessor implements an Ethernet MAC also compatible with the IEEE 802.3 standard and provides address checking, statics and control registers, transmit and receive blocks and a DMA interface as shown in the datasheet (Atmel Corporation, 2009*c*).

A substitute for the DM9161A device was required as it was not locally available. The DM9161B (U10) was readily available and found to be a suitable, equivalent device after a comparison of the datasheets (Davicom Semiconductor Incorporated, 2009*a*) was performed. The only difference found was an additional Power Saving Control register in the DM9161B which was not found in the DM9161A. The default configuration of this register was found to have no effect on the operation of the Ethernet interface in the evaluation of the final design.

The RMII bus and DM9161B are clocked at a rate of 50MHz which was supplied by a FXO-HC736R fixed oscillator X1. Three LEDs, D9, D10 and D11 display the state of the Ethernet connection: ACT, 100Mbps and full duplex mode.

The DM9161B required a transformer to couple it to an Ethernet network. The transformer provides electrical isolation between (a) the network and gateway, (b) the transfer of signals between them without distortion and (c) the rejection of common mode noise that may be present on the Ethernet cable. The transformers may be supplied as discrete components requiring a separate RJ45 connector and matching networks to be placed nearby on the PCB or integrated directly into the connector.

The latter option was desirable for this design as it offered a space-saving advantage on the PCB area, reduced the required component count and costs and pro-

vided a single-component magnetic coupling solution required by the DM9161B's datasheet (Davicom Semiconductor Incorporated, 2009*b*). A J00-0061NL (Pulse Electronics, 2010) integrated connector by Pulse Electronics was used on the evaluation kit and is an equivalent component for the H1102NL as listed in Table 2 of the DM1961B datasheet.

The J00-0061NL was found to be unavailable at the time of the design and the J00-0064NL (CON5) offered as an equivalent. The difference between these components is the omission of the "Bob Smith" (National Semiconductor Corporation, 2008) termination network in the J00-0064NL which reduces signal noise resulting from common mode current flows in the magnetics and the susceptibility to noise resulting from unused wire pairs on the Ethernet cable.

Operational problems with the Ethernet interface were discovered during the evaluation of the OS deployment and verification, as discussed in section 5.5.2.

4.3.3 USB to serial converter

The FT2232D (U11) (FTDI, 2010*b*) is a dual USB to serial port manufactured by FTDI that provides a highly configurable and versatile interface between a PC and an embedded device via a USB cable. The FT2232D was found to be suitable for inclusion in the design as the gateway required two independent UARTs (a) to configure the bootloader *EBOOT*, (b) for the verification of the communication module and (c) as a local interface to a terminal emulator such as *HyperTerminal*.

The FT2232D is connected to a PC via a USB connection which is handled by the IC entirely and requires no intervention from the target hardware. FTDI supply a Virtual Comm Port (VCP) software driver which assigns each of the UARTs a unique COM port number and access to either port from any application being executed on the PC when the USB cable is connected. The configuration of the baudrate, handshaking type, parity and the number of stop bits of each UART is permitted by the VCP driver and each may be individually configured. The configuration of the FT2232D was carried out by *FT_Prog* (FTDI, 2010*a*),

a free configuration utility supplied by FTDI. The configuration is stored in a 93C46 EEPROM (U12) and retrieved by the FT2232D when required.

The gateway's USB B socket (CON1) may be configured to connect to either the microprocessor's USB interface or the FT2232D by jumpers J12, J13 and J14 included in the design. The use of configuration jumpers to select the function of the socket permitted the purchase and use of fewer components.

An additional USB B socket (CON2) was also incorporated but left unpopulated during the PCB assembly to permit the simultaneous use of the microprocessor's USB port and the FT2232D if required.

The values of the remaining components required by the FT2232D circuit were shown in the datasheet and based on a the *USB self powered configuration* schematic (FTDI, 2010*c*) and implemented in the gateway design. Further details of the power supply requirements are discussed in section 4.3.5.

UART B's TX, RX, CTS and RTS signals on the FT2232D were connected directly to UART 1 on the microprocessor as this port was assigned by the application as the local user interface accessed through a terminal emulator. J9, J10 and J11 permit UART A to be steered to either the communication module or the microprocessor's debugging UART. An additional function that permitted the snooping of the serial transfers between the microprocessor and the communication module was included in the design to permit the verification of commands to and from the module when required.

4.3.4 USB host interface

The microprocessor incorporates a USB host port (UHP) and interfaces an external USB connection with the application. The host port complies with the USB 2.0 Full speed and Low speed specifications. The port provides several half-duplex communication ports operating at 12Mbps and is capable of communicating with up to 127 USB devices.

The TelosB wireless node required a USB host connected to a USB A socket for its serial communications with the gateway. A Y99000 USB A socket (CON3) was included in the design to provide a suitable node connection. The termination resistors R61 and R62 were set to 27Ω according to the test circuit shown in the datasheet and the evaluation kit schematic.

A 450mA resettable thermal circuit breaker, SMDC016F (F1) was placed in series with the +5V supply to the host socket to protect the gateway's power supply against short circuits or high current loading by equipment inserted into the socket.

4.3.5 Power supply

The gateway power supply unit (PSU) was required to provide stable DC voltages to three rails in the design: 3,3V for all IC Vcc or Vdd supplies and GPIO lines, 1,8V for the processor core and its peripheral modules and 5V for the USB host socket and USB to serial converter. The gateway specification in section 1.4.2 required the gateway to operate over an input DC voltage range of 8V to 15V, be polarity protected in case of a reversed supply connection and to be as efficient as possible.

The PSU consists of two stages to reduce power losses over the required input voltage range: an efficient switched mode, step down converter was selected as a pre-regulator to reduce the input voltage to 5V which in turn supplies two low-dropout (LDO) linear regulators. The LDOs regulate the 5V input to 3,3V and 1,8V as shown in figure 4.2.

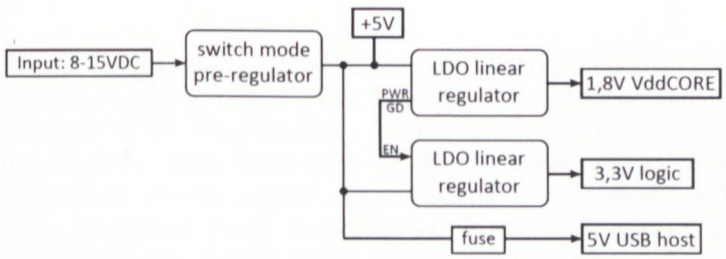


Figure 4.2: A switched mode regulator supplies two LDO regulators in a two-stage supply.

Table 4.1 shows the anticipated current draw in mA at 25°C by the 5V, 3,3V and 1,8V supply rails. The currents listed were obtained from the datasheets of the components selected for use in the design and are based on an operating duty cycle of 100%. A reduced operating duty cycle, and therefore reduced current draw, is expected in the built circuit.

IC	5V rail	3,3V rail	1,8V rail
AT91SAM9260 @ 180MHz			
• VddCORE	-	-	130
• PIO controller	-	-	1,8
• USART	-	-	5,4
• USB host port	-	-	2,52
• USB device port	-	-	3,6
• Two wire interface	-	-	3,78
• SPI	-	-	1,8
• Memory card interface	-	-	5,4
• Timer Counter blocks	-	-	1,08
• EMAC Ethernet block	-	-	15,84
K9F2G08U0B NAND	-	15	-
2 x MT48LC16M16A2P-75 SDRAM	-	250	-
AT45DB642D Dataflash	-	10	-
24LC64 EEPROM	-	1	-
SD card	-	50	-
DM9161B Ethernet interface	-	168	-
FT2232 USB converter	-	30	-
M93C46 EEPROM	-	1	-
TelosB WSN node	25	-	-
Total 3,3V rail current	525	-	-
Total 1,8V rail current	171,22	-	-
TOTALS:	**721,22**	**525**	**171,22**

Table 4.1: Anticipated currents (mA) shown are based on datasheet values required by the components within the gateway circuit at 25°C. It is noted that the calculated power consumption of the 5V rail is 3,6W if 721,22mA is drawn from it. This result is based on the components operating at a duty cycle of 100%, however in practice, a lower duty cycle is anticipated resulting in a lowering of the average current drawn and therefore a reduction in the gateway's power consumption. Additional power losses also occur within the power supply regulators (not shown) thus adding to the overall power budget of the gateway. These losses are load dependant and are thus influenced by the operational duty cycle of the gateway components.

4.3.5.1 Pre-regulator

A switching converter was used as a pre-regulator as they dissipate less power than an equivalent linear regulator during elevated input voltage or output loading conditions.

An LM2738YM (U2) (National Semiconductor Corporation, 2011*b*) by National Semiconductor is a switched mode, PWM step-down DC/DC buck converter and was selected to convert the gateway's input DC supply voltage to 5V which supplies the 1,8V and 3,3V LDO linear regulators and the USB host socket, CON3.

The LM2738YM has the following characteristics required by the design:

- wide input voltage range of 3V to 20V
- load current capability up to 1,5A
- switching frequency of 550kHz thereby permitting the use of components such as inductors and capacitors with reduced electrical values and therefore smaller PCB footprints
- internal soft-start and thermal shutdown protection
- conversion efficiencies of up to 90% ensuring reduced power loss
- compact 8-Pin eMSOP packaging

Table 4.1 shows an anticipated current of 721,22mA required to be delivered on the 5V supply rail during normal gateway operation at 25°C. The LM2738YM is capable of supplying double this current. The calculations used to determine the component values for the LM2738YM circuit assumed an additional current draw of 360,61mA or 50% of the tabulated result to accomodate any unforeseen rise in current demand. An increase in supply current may result from factors such as an increase in the ambient temperature of the gateway or different sensor node inserted into CON3 and drawing more than 25mA as drawn by the TelosB node used in this study.

A portion of the total power consumed by the gateway is lost as heat in the power supply regulators. These losses are a function of the *efficiency* of the reg-

ulator which may vary depending on its input supply and loading conditions. The LM2738YM is shown in its datasheet to have an efficiency exceeding 90% over the input voltage range and anticipated current consumption of the gateway. This results in an additional wasted maximum power dissipation of 247,51mW as shown by the calculations in Appendix A.1.4.

A boost voltage was required to drive the gate of the main internal switching NMOS of the LM2738 and may be derived by several means depending on the input and output voltages of the regulator. As 5V was required from the output of the regulator, the required boost voltage was derived from the converter output, as shown in the LM2738 datasheet (National Semiconductor Corporation, 2011*a*), which reduced the number of components required by the regulator circuit.

The value of the inductor required for the LM2738 converter was found by the calculations in Appendix A.1.1. The values of the inductors were calculated for the highest and lowest specified input voltage and the results averaged to find the value of a suitable *best fit* inductor.

A Panasonic *ELLCTV220M* 22μH inductor was selected for the design as it has a rating of 2,7A, very low DC resistance and is available in a magnetically shielded SMD package (Panasonic Industrial Company, 2010).

The choice and placement of input capacitors for the switch mode converter was important to ensure its optimal performance. Capacitors with a low Equivalent Series Inductance (ESL) were selected in order to reduce the impedance of the supply to the regulator to limit input voltage drops during switching transients. The ESL of leaded capacitors at the 550kHz switching frequency of the LM2738 exhibit high impedances resulting in possible unstable device operation; SMD-packaged multilayer ceramic capacitors (MLCC) which have very low ESL ratings were therefore used in the design and placed nearby the converter. In addition, these capacitors were required to be constructed with the X7R dielectric which offers stable operation over an extended temperature range.

A 10μF MLCC X7R capacitor, a C1210C106K3RAC by Kemet (KEMET, 2011), was used as the input capacitor for the LM2738YM as it satisfied the requirements discussed above.

Similarly, the output capacitor was required to have a low Equivalent Series Resistance (ESR) and low ESL to reduce output ripple and to bypass high frequency noise generated by the switching components and inductor. An MLCC capacitor was thus ideal for this application.

A minimum of 22μF was recommended to maintain the stability of the regulator feedback control loop, however the output capacitance was increased by an order of magnitude by means of a parallel combination of four 47μF MLCC capacitors and a 470μF electrolytic capacitor to ensure that the supply had a low output impedance and minimal ripple.

The pre-regulator circuit required a Schottky diode as they have fast switching times and forward voltage drops lower than regular silicon diodes. An RSX201L-30 (D7) was selected as it has a continuous forward current rating of 2A, a voltage drop of 0,44V, a low peak reverse current of 150μA and is enclosed in an SOD-106 SMD package (Rohm Semiconductor Ltd, 2011).

The output voltage of the LM2738 is adjustable and was set by resistors R7 and R9 whose values were calculated by the equation shown in Appendix A.1.2 and found to be 52,3kΩ and 10kΩ respectively.

Input polarity protection to the pre-regulator was provided by series diode D5. An RSX201L-30 was selected as its forward current capacity was sufficient for the application and it was deemed unnecessary to add an extra component type to the bill of materials.

4.3.5.2 Supply rail LDO regulators

Two low dropout (LDO) linear regulators were required by the gateway to supply 1,8V to the microprocessor core and peripherals, and a 3,3V supply to power the microprocessor IO ports, memories and remaining components within the circuit.

An MCP1825 (Microchip Technology Incorporated, 2008*a*) by Microchip is fixed voltage LDO linear regulator that is capable of supplying 500mA at 1,8V from a 5V input. It has a *power good* (PWRGD) output pin that switches to logic high when the output voltage has reached at least 92% of it's rated ouput value, a low quiescent current and is able to respond quickly to load transients. It is available in a 5-lead DDPAK case providing good heat dissipation characteristics when soldered to a PCB. These attributes made this device a suitable choice for this application.

The 3,3V rail is supplied by an ST Microelectronics LD29150PTR (ST Microelectronics, 2010*a*) adjustable LDO linear regulator. It requires two external resistors to set the output voltage to the required level and is capable of supplying 1,5A from a 5V input. The values of R10 and R65 were calculated by the equations shown in Appendix A.1.3. The LD29150 incorporates internal current and thermal limiting and is available in a compact PPAK package. In addition, it has an *inhibit* (INH) input control pin that permits the output voltage to be switched *on* when set to logic high.

100nF and 47μF MLCC capacitors were placed nearby to the inputs and outputs of both regulators to avoid unstable regulation.

4.3.5.3 Power sequencing

The gateway power supply was required to comply with a specific sequence at startup, as shown in figure 4.3, to guarantee reliable booting of the microprocessor (Atmel Corporation, 2009*d*). The following conditions were required:

- The 1,8V rail that supplies the microprocessor core, *VddCORE*, shall reach a level of 1,5V or greater before any voltage is applied to the 3,3V rail, which

supplies *VddIOM*, *VddP0* and *VddP1* and the microprocessor IO ports

- The 3,3V rail shall reach and exceed 2V within 166μs after *VddCORE* has reached 1,5V and 2,9V within 352μs after *VddIO* has reached 2V.

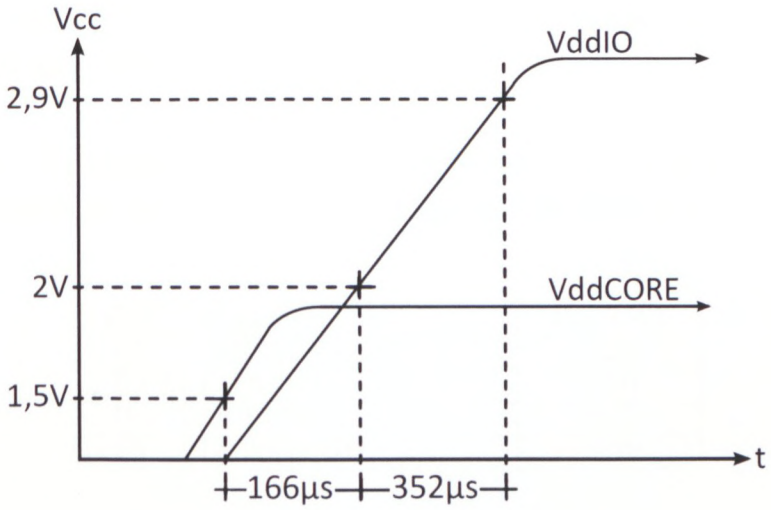


Figure 4.3: Specified power up sequence

The *PWRGD* output signal of the MCP1825 1,8V regulator was connected to the *EN* input of the LD29150PTR 3,3V regulator to ensure that this specification was met, as shown in section 5.3.2. The *PWRGD* output signal rises as the output voltage of the MCP1825 reaches 1,65V. A logic high signal on the *EN* input of the LD29150 switches its output voltage *on* which begins to rise and reaches 2V within the specified 352μs.

4.3.5.4 Thermal considerations

The regulators employed in the design of the power supply required heatsinks to remove excess heat generated during operation. Fixing leaded regulators to heatsinks is usually a simple operation as compared to that of equivalent SMD devices. Derating factors such as the possible ambient operating temperature of the PCB, maximum junction temperatures of the regulators and current drawn by the circuit at elevated temperatures and a load applied to the USB connector were considered during the design of a suitable heatsink.

In order to save space and cost, the copper planes on the PCB were used as the heatsink for the power supply regulators. This design relied on natural radiation and conduction to dissipate the heat from the planes into the surrounding air. Forced cooling was not considered a viable option due to increased power consumption of the gateway by the cooling mechanism, dust and insect ingress into the enclosure and added costs.

Appendix A.1.4 contains the equations used to calculate the anticipated internal power dissipation of each regulator as summarised in Table 4.2:

Ref	**Regulator**	**P_{int} (W)**
U2	LM2738, V_{in}=8V	0,247
U2	LM2738, V_{in}=15V	0,204
U1	MCP1825	0,822
U3	LD29150-ADJ	1,429

Table 4.2: Calculated regulator internal power dissipation

The total power to be dissipated by the copper planes on the PCB was the sum of the power dissipated by each regulator. The thermal resistance of the copper plane affects the effectiveness of the heat transfer away from the regulators and is dependent on its thickness and length. The thermal resistance is proportional to length and inversely proportional to cross-sectional area. Copper has a much higher thermal conductivity than the PCB substrate material and breaks in the planes caused by tracks surrounding the devices were minimised to ensure maximum heat transfer and efficiency of the heatsink. In addition, thermal vias were used to transfer the heat from one side of the PCB to the other as well as to the internal planes increasing the effective surface area of the heatsink.

The individual heatsink surface area required by each regulator was calculated by means of the equations in Appendix A.1.5 and the results summed to find the total approximate heatsink surface area needed for the three regulators as shown in Table 4.3.

Ref	Regulator	PCB Area cm^2
U1	MCP1825	17,306
U2	LM2738	5,814
U3	LD29150-ADJ	37,051
	TOTAL AREA	**60,171**

Table 4.3: Calculated heatsink surface area requirements for the power supply regulators

The area of the heatsink required for the power supply regulators to operate within a safe limit was found to be 60,171cm^2. This area was made available in the PCB layout and is discussed in section 4.4.

4.3.6 Configuration jumpers

Several jumper switches were included in the gateway design to permit the configuration of the USB connection and serial communications ports and the invocation of the bootloader *ROMboot*.

The microprocessor on the evaluation kit was packaged in a BGA-style case which incorporated several extra IO pins (Atmel Corporation, 2009*e*) that are not available on the equivalent leaded QFP device selected for the gateway design. The microprocessor in the BGA package used port *A31* in conjunction with a system reset to invoke *ROMboot* thus ignoring any previously installed bootloaders and permitting a USB connection to SAM-BA (refer to section 3.2.1.2).

Jumpers J1 and J2 were incorporated into the gateway design to permit the disconnection of the chip select lines of the AT45DB642 dataflash or K9F2G08U0B NAND flash memories from the microprocessor thereby disabling them. A USB connection with SAM-BA is attempted if the memories are disabled directly after a microprocessor reset. Replacing J1 and J2 enables the memories for normal access after a connection with SAM-BA is established.

J10 and J11 configure the connections of the UARTs from the microprocessor, communication module and FT2232 USB to serial converter. J10 and J11 steer port A on the FT2232 to either the microprocessor debug UART or to the com-

munication module, permitting a serial link with a PC.

J9 steers the *transmit* signal of the communication module to either the microprocessor or FT2232 via J10; the *receive* signal is permanently connected to the microprocessor and J11. The FT2232 may be configured to *snoop* on the communications between the microprocessor and the communication module, as shown in Figure 4.4, by a combination of jumpers J9, J10 and J11.

J12 and J13 configure the USB destination from CON1 to either the FT2232 or the microprocessor when using SAM-BA. Provision was made in the circuit design to permit the simultaneous use of the FT2232 and the microprocessor's USB connection if required. CON2 was not populated during the assembly of the final gateway PCB and may be added if required.

Figure 4.4 summarises the jumper configuration positions permitted on the gateway PCB:

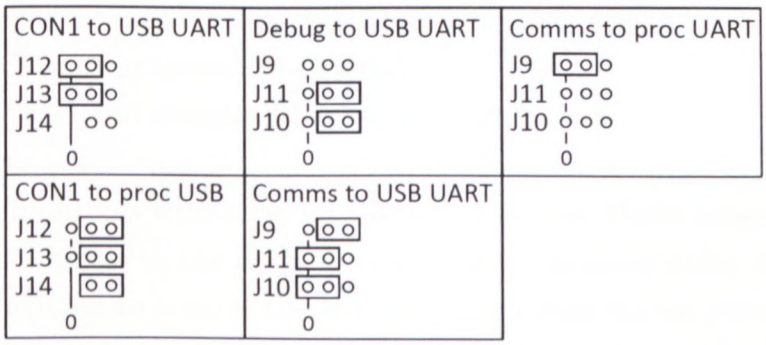


Figure 4.4: Gateway PCB configuration jumper positions

4.3.7 Communication module

The specification in section 1.4.2 of the study required a configurable means of connecting the gateway to the Internet. A separate communication module incorporating a GSM module was designed to permit an additional means of connecting the gateway to the Internet.

The communication module incorporates a UART with modem handshaking signals signals such as RTS and CTS and signals permitting the control and monitoring of the module by the microprocessor. The schematic diagram of the communication module may be found in Appendix B.7.

4.3.7.1 GSM/GPRS module

The GM862-GPS module, by Telit (Telit Communications, 2011*b*), was selected for this study as discussed in section 2.3. It is a quad band GSM/GPRS modem that incorporates a 20 channel high sensitivity GPS receiver.

In addition, the module has a CMOS level UART, embedded TCP/IP stack with support for GPRS, FTP and SMTP protocols, an extended operating temperature range, built-in SIM card holder and several modes of operation that draw approximate current values as shown:

- Power-off, $< 26\mu$A
- Idle, where the module is registered and in power saving mode, 2.6mA
- GPRS Class 10 transmission, 370mA
- GPS receiver and antenna LNA supply, 75mA

The module interfaces with a circuit through a 50-way Molex connector MOD1 which provides access to the required control and communications signals. Only the signals required to control the module's connection status, communications and handshaking were implemented as the remaining functionality offered by the module was not required by the specification.

A GSM antenna connects to the module via a 50Ω MMCX connector and was required to exhibit a gain of less than 3dBi, have an impedance of 50Ω and be able to accept a peak power level of 2W or greater and be suitably tuned to operate up to 1800MHz.

4.3.7.2 Power supply

The communication module was required to operate at the same 8V to 15V DC supply voltage as the gateway PCB. Supply polarity protection was included to ensure the safety of the module during an accidental supply reversal. A two-stage supply, as used for the gateway, was implemented in the communication module design. The GSM module requires a voltage of 3,8V for its main supply and 2,8V for its serial interface which were supplied by a switched mode converter and linear regulator respectively.

It was decided to use the same switched mode converter, LM2738YM (National Semiconductor Corporation, 2011*b*) (U3), as was used for the gateway power supply pre-regulator to deliver the required supply voltage as it:

- is capable of a 1,5A continuous load current
- permits an adjustable output voltage
- is efficient and has a low quiescent current
- is fast-switching and is suitable for supplying a burst-type load

The LM2738 converter circuit was configured in a similar manner to the gateway where the drive voltage for the NMOS switch was derived from its output (National Semiconductor Corporation, 2011*a*). The output voltage was determined by the ratio of resistors R12 and R16 as shown by the calculations in Appendix A.2.2.

A 2,8V supply was required to supply the UART of the GSM module. The module incorporates a regulator that is able to supply the required voltage, however it is limited to a maximum output of 1mA and is sensitive to electrical noise and fast-rising signals. It was decided to incorporate an additional linear regulator into the design to supply the 2,8V rail instead. An MCP1725 adjustable LDO linear regulator by Microchip (Microchip Technology Incorporated, 2007) was selected as it has a low quiescent current, 500mA load current capacity which was more than required by the rail and is packaged in a compact 8-pin SOIC package.

The values of the voltage setting resistors R13 and R17 were calculated by the equations shown in Appendix A.2.3.

A 470μF electrolytic capacitor was placed just after the reverse polarity diode D3 to ensure that the supply impedance to the communication module was kept low and that a reservoir of charge was available during peaks of high current demand during GSM transmission. 10μF and several 47μF MLCC capacitors were mounted nearby to the inputs and output pins of both regulators to ensure optimal regulator performance.

GSM packets are transmitted at a rate of approximately 216Hz resulting in current spikes up to 2A in magnitude being drawn from the module's *VBATT* supply pins. Two 100μF tantalum capacitors with very low ESR ratings were placed in parallel directly beside the connections to the GSM module to ensure that maximum transfer of current occurred without loss.

4.3.7.3 Voltage level translation

The UART on the GM862 module operates at level of 2,8V and the microprocessor UART at 3,3V. It was necessary to address the difference in operating voltages to ensure that no damage was caused to the module's UART by the higher microprocessor levels and to ensure that the integrity of the data being transferred in either direction was maintained. Two TXS0104E (Texas Instruments, 2008) bidirectional voltage level translators were incorporated in to the design to accommodate the difference in signal voltages.

The level translators are supplied by the 2,8V and 3,3V of the communication module and gateway power supplies respectively. The level translators permit the power of the communication module to be switched *off* by the microprocessor without consequence to the microprocessor's UART and ensures that the communication module does not try to draw current through any of the digital signals from the microprocessor.

The GSM module has two control inputs: one that permits the module to be powered on or off and the other resets the module. These pins are pulled up to an internal supply by internal pull-up resistors and only require an external pushbutton or open collector switch connected to system ground to function. *Digital transistors*, the DTA114EKA (Q2, Q4) and DTC114EKA (Q1, Q3) (Rohm Semiconductor Ltd, 2009*a*,*b*) by Rohm, were connected to form a buffer circuit permitting the 3,3V microprocessor signals to control the power and reset signals on the GSM module operating at 2,8V.

The LM2738 permits its output voltage to be switched *on* or *off* depending on the state of its enable pin *EN*. *Digital transistors*, Q6 and Q7 were configured as a buffer permitting a 3,3V microprocessor GPIO pin to control the *EN* pin which is pulled up to the input supply voltage by R11. The ability to control the power supply on the communication module was used by the gateway application to ensure that the GSM module was always reset to a default state before the application attempted communications with it.

Status LEDs D6 and D2 were incorporated into the design to indicate the presence of power to the communication module and the connection status of the GSM module with the GSM network.

4.4 The Gateway PCB design

4.4.1 Layout considerations

One of the requirements of the gateway developed in this study was that it should be possible for the gateway to be deployed in external environments being monitored by a WSN. The gateway PCB, communication module and WSN sink node would need to be enclosed in a moisture and dust resistant enclosure. A Gewiss *GW44207* enclosure was investigated and found to be a suitable enclosure; its internal dimensions used to set the overall size of the gateway PCB and the positions of the communication module and sink node. An area slightly larger than the size of a TelosB WSN node was removed from the side of the gateway PCB in

order to accommodate it when inserted into the USB hub connector, as shown in the photographs of the manufactured PCB in Appendix C. This layout permitted the most efficient use of the available area within the enclosure and prevented the node from fouling the sides of the enclosure [1].

It was decided to mount the communication module above the gateway PCB on support pillars to reduce the lateral footprint within the enclosure. A flexible 20 way ribbon cable connection between the gateway PCB and communication module permits access to the gateway PCB jumper legend (refer to Figure 4.4) and the ability to access the underside of the communication module during system configuration or repair without interrupting its operation.

Components were placed on both sides of the gateway and communication module PCBs in order to reduce the total surface area of the PCBs and to reduce the lengths of signal traces where possible. The heights of the components were taken into account in areas where the communication module overlaps the gateway PCB.

The TelosB node communicates with a WSN through a small Planar Inverted F (PIF) antenna etched on to its PCB. The node was positioned to the side of the gateway PCB in order to gain as much distance as possible between it, the switching power supply regulator and high speed memory buses of the microprocessor. These components and signals may emit broadband electromagnetic radiation or harmonics resulting in electromagnetic interference (EMI) which may degrade the RF signal reaching the sink node. Specialised equipment located in an RF controlled anechoic chamber is required to accurately measure and characterise any emitted or conducted interference.

The data bus between the microprocessor and SDRAM memories is 32-bits wide and is accompanied by the address bus and several control lines. It was decided to place these components nearby to each other and the microprocessor and to

[1]The gateway, communication module and WSN sink node PCBs have not yet been built into an enclosure at the time of this writing

complete the routing of the interlinking traces first during the PCB layout. This was done in order to keep the lengths of the traces as short and uninterrupted by vias and other components as possible to assist with minimising radiated emissions and to promote signal integrity. The NAND flash memory was placed on the bottom PCB layer directly beneath the microprocessor to permit a connection of its 8-bit data bus to the SDRAM data bus with minimal interruption.

The DM9161B Ethernet controller was positioned as close as possible to the microprocessor to minimise EMI as the RMII bus that interfaces the two operates at a clock frequency of 50MHz.

Peripheral devices such as the SD card slot and the communication module's data interface operate from signals with less critical timing requirements and were positioned towards the extremities of the PCB. Several spare IO lines and the JTAG interface of the microprocessor were routed to headers positioned on the edge of the PCB to permit easy user-access.

The power plane embedded within the PCB was connected to the output of the 3,3V regulator and forms a low-impedance path to all the IC supplies ensuring optimal supply performance. All Vcc and Vdd supply pins of the ICs in the circuit were decoupled with 100nF MLCCs and positioned as close as possible to the pin to assist with high speed switching current demands.

The ground plane performs an important function in assisting with the reduction of emitted EMI by providing a continuous and shortened return path for high speed signals and the IC supply return currents. The return currents find the most direct path back to their source through the ground plane and was therefore important to ensure that the plane was not split nor interrupted unnecessarily. Splitting a ground plane increases its inductance and adds length to the current return paths resulting in high-frequency voltage drops divided across the two planes. This forms an efficient radiating dipole structure which leads to increased EMI and magnetic coupling of noise into other parts of the circuit.

In addition to the electrical purposes of ground planes, they are used as a heatsink to dissipate heat generated by the power supply regulators. The power supply regulators were placed to the side of the PCB to ensure a compact layout and efficient supply interconnections. Thermal vias were placed near to the tabs of the regulator ICs to draw generated heat away and to conduct it into the ground plane embedded within the PCB and on the bottom side. The area of uninterrupted ground plane was found to be greater than the calculated amount required for the regulators as discussed in section 4.3.5.4.

4.4.2 PCB manufacture

There are several PCB substrate laminates available such as FR-2 or Phenolic, FR-4, Polyimide and Teflon. FR-4 is a flame retarding composite consisting of woven fibreglass matting impregnated with an epoxy binder and is a very common substrate material used in PCB laminates. It is cost effective, exhibits an acceptable Relative Dielectric Constant over a wide frequency spectrum and is mechanically stable in dry and humid conditions. FR-4 was used in the PCBs for this study due to these properties.

The number of PCB layers in the design was limited to a maximum of 4 in order to keep the manufacturing costs to a minimum.

Electroless Nickel Immersion Gold (ENIG) was applied to the copper traces and groundplanes to:

- improve the surface planarity to assist with the optimal placement of SMD components
- reduce trace oxidation occurring after a prolonged exposure to the atmosphere thereby improving the ability to solder the components to the PCB
- to bring the finished thickness of the traces and groundplanes to 35μm

The PCBs were finished with a green soldermask and white silkscreen legend on the exterior layers. A flying-probe test was performed on the PCBs to ensure that all the trace nets were electrically continuous.

Photographs of the gateway and communication module PCBs may be found in Appendix C.

4.5 Summary

This chapter discusses the design of the hardware required to meet the specification of the gateway defined in section 1.4.2. A top-level block diagram divides the gateway into functional blocks and the interconnections between them, the WSN node and communication module.

Details of the design of the circuit follow with reasons given for the selection of the each component. The power supply is an important feature in any electronic system. The power supply for the gateway is a two-stage supply consisting of a switched mode pre-regulator feeding two low dropout linear regulators that supply the board. The same design topology is used for the communication module. In order to ensure that the system boots correctly, the processor core supply voltage needs to be at a prescribed level before the peripheral supply is enabled.

An in-depth analysis of the thermal considerations for all the regulators is shown in order to ensure that sufficient cooling is provided for the power supply regulators by the PCB ground planes.

The gateway has several configuration jumpers that alter the operation of the USB connection and system UARTs. The functions of these are described and a table summarises the jumper settings.

The communication module is an auxiliary system that provides the gateway with an additional means of connecting to the Internet. Data communication with the module occurs through a microprocessor UART made available on a header. The Telit GM862-GPS GSM module was used in the design of the communication module as it incorporates several of the required Internet protocol stacks to satisfy the gateway specification.

A similar two-stage topology to the gateway power supply was used in the communication module and specific details about its design are given.

The communication module incorporates level translators to ensure that the integrity UART on the GSM module is not compromised by the UART on the microprocessor that operates at a higher voltage level. GSM module control signals are buffered by digital transistors to ensure that they are at the correct signal levels when driven by the microprocessor.

The final section of the chapter deals with several aspects of the PCB manufacture. The PCBs have four conductive layers and were manufactured from FR-4 material. The copper tracks and planes were gold-plated to ensure that oxidising was kept to a minimum before the assembly process. Green soldermasks and silkscreen legends were placed on both sides of the PCBs.

Chapter 5

Prototype evaluation

5.1 Overview

The following chapter discusses the assembly and evaluation of the gateway prototype design. Various aspects of the hardware and software operation are evaluated and the results shown.

5.2 Preliminary assembly considerations

The gateway and communication module PCBs were visually inspected for obvious defects once they'd been received from the manufacturer. A flying-probe test to check the continuity of the signal tracks was conducted by the manufacturer, however it was deemed safe practice to conduct an additional visual inspection before the population of any components took place.

A multimeter was used to check that no short circuits existed between any of the supply rails and the ground plane on the PCBs; no short circuits were found.

It was decided to begin the PCB assembly process by populating the power supply sections first and to ensure that these were completely functional before commencing with the soldering of the remaining components.

5.3 Power supply verification

5.3.1 Supply regulators

The assembly of the power supply commenced with the switched-mode 5V pre-regulator, as discussed in section 4.3.5.1, and associated components. Extreme care was exercised during the soldering of the LM2738YM to the PCB to avoid solder spikes due to the small package type, fine pitch of the pins and the necessity for the use of solder paste to bond the heatsink pad embedded underneath the component to the PCB.

A laboratory DC power supply was set to deliver 10V at a limited current of approximately 100mA. The power supply was connected to the gateway PCB first and the pre-regulator output on the 5V rail was measured with a digital multimeter and found to be at 5,01V. A similar test of the pre-regulator on the communication module showed that its output was at 3,795V.

The temperature of the pre-regulator components on both PCBs were checked manually to ensure that none of them were generating excessive heat which would indicate the presence of a possible fault with the regulator or PCB. None of the components were hot to the touch and power supply did not register any current being drawn on its display during the tests.

These initial tests were conducted without any loading connected to the pre-regulator outputs. Their output voltages were remeasured once the PCB assemblies were completed to confirm correct output voltages under load as shown in table 5.1.

The remaining linear regulators and associated components were soldered to the PCBs. It was found that the incorrect footprint for U3, the LD29150 3,3V fixed output regulator, on the gateway PCB had been placed onto the PCB during the design phase. The fixed regulator was replaced with an LD29150-ADJ, an adjustable regulator from the same family, packaged with the footprint currently placed on the PCB. A supply rail LED indicator D8 and its limiting resistor R10 were substituted by a resistor divider network to set the output voltage of the adjustable regulator to 3,3V. These resistors were selected by calculation, as shown in Appendix A.1.3.

	Target voltage	**Gateway PCB**	**GSM comms module**
U2, U3 Pre-regulators	5V, 3,8V	4,989V	3,783V
U1 MCP1825	1,8V	1,791V	-
U3 LD29150-ADJ	3,3V	3,282V	-
U4 MCP1725-ADJ	2,8V	-	2,84V

Table 5.1: Power supply regulator output voltages with complete PCB assemblies

5.3.2 Power supply start up sequence

The gateway microprocessor requires a specific start up sequence on its 1,8V and 3,3V supply rails as discussed in section 4.3.5.3.

The following specification must be met:

- The 1,8V processor core rail VddCORE must reach a level of 1,5V or greater before any voltage is applied to the 3,3V rail
- The 3,3V rail must reach and exceed 2V within 166μs after VddCORE has reached 1,5V and 2.9V within 352μs

In order to verify the power supply startup sequence, a two channel oscilloscope captured the voltages on the 1,8V and 3,3V supply rails as shown in figure 5.1:

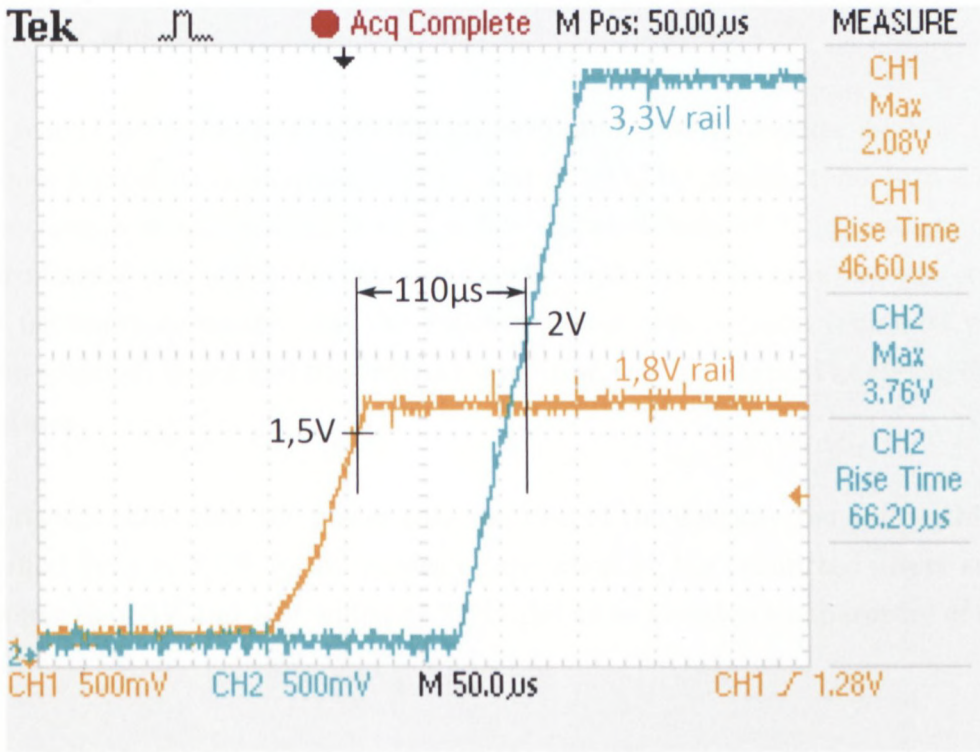


Figure 5.1: Start up sequence voltages generated by the gateway power supply.

Figure 5.1 shows that the voltage sequence generated by the power supply during startup adequately meets the timing requirements of the specification.

5.3.3 Gateway power consumption

The gateway specification in section 1.4 required an operational time of up to 18 hours when powered by a 12Ah battery and therefore components with a low power consumption were selected for the design where possible. The gateway's power consumption was required not to exceed an average of 3,5W during normal operation.

Figures 5.2 and 5.3 show the power consumed by the gateway at various stages of operation and represent a summary of the data captured in tables D.1 and D.2 found in appendix D.

The current measurements were performed by a digital multimeter connected in series with the DC supply to the gateway at the specified supply voltage extremities of 8V and 15V.

The results were obtained with the gateway and communication module PCBs at room temperature, nominally 21°C, and at 60°C by placing them into a small convection oven and heating it to simulate elevated ambient temperatures of the environments into which the gateway may be deployed. The heat test was considered necessary to ensure that the gateway power consumption remained within in the specified limits and that correct operation was maintained at elevated temperatures.

The results show that the power consumption of the gateway remains within the specified limit of 3,5W for all modes of operation at the permitted power supply extremities of 8V and 15V whilst at 21°C and at an elevated temperature of 60°C.

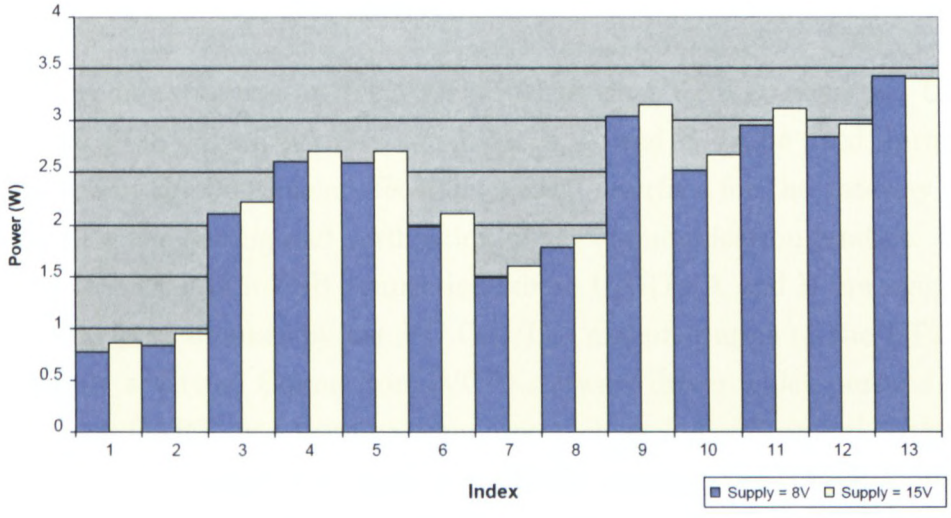


Figure 5.2: Power consumption at 21°C with the DC supply set at 8V and 15V.

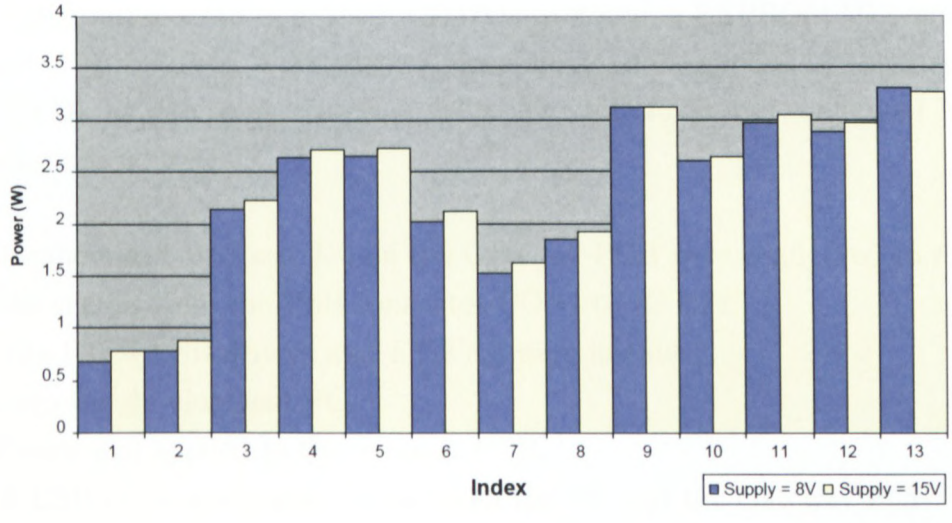


Figure 5.3: Power consumption at 60°C with the DC supply set at 8V and 15V.

Legend for figures 5.2 and 5.3			
	Index		**Index**
1	Gateway held in reset	8	Wireless node inserted
2	USB cable connected	9	Launch gateway app. with LAN
3	EBOOT execution	10	Gateway app., LAN disconnected
4	EBOOT, LAN connected	11	GSM modem initialising, GSM registration
5	Launch Windows CE	12	Modem in Listen mode
6	Windows CE	13	FTP session initialise, file send
7	Windows CE, LAN disconnected		

5.4 UART verification

The gateway incorporates an FT2232D USB to dual UART converter, U11 on the gateway PCB, which permits two UARTs, A and B, to be used during the configuration of the bootloader *Eboot*, as a local interface for the gateway application and for the testing and verification of the communication module. U11 is connected to a PC via a USB connection where UARTs A and B are assigned a unique COM port number by the PC OS. The manufacturers of the FT2232D, FTDI, supply a Virtual Comm Port (VCP) software driver which permits a user access to each UART by a terminal emulator such as *HyperTerminal*. *HyperTerminal* and an oscilloscope were used to verify the operation of UART A after the configuration of the FT2232D. UART B's operation will be checked during the verification of the gateway software in section 5.5.

The configuration settings of the FT2232D are stored in EEPROM U12 and were configured by *FT_Prog*, a proprietary setup application supplied by the manufacturer FTDI, as discussed in section 4.3.3. The FT2232D was configured by the following procedure:

- Jumpers J12, J13 and J14 on the Gateway PCB were configured to route the signals from the USB connector CON1 to IC U11.
- The FTDI USB drivers and *FT_Prog* were installed onto the development PC.
- Power was applied to the gateway PCB.
- A USB cable was connected between the PC and the Gateway PCB.
- The PC detected new hardware and completed the USB enumeration process with U11.
- *FT_Prog* was started following the successful installation of the new USB device.

The FT2232D was configured so that:

- it is self-powered i.e. it derives its power from the gateway and not the USB port.

- both the output serial ports would function as UARTs accessible by the VCP software drivers.
- the Product Descriptor string was set to "WSN Internet Gateway" which is displayed by a PC during the first USB enumeration session of the gateway.

HyperTerminal and the communication module were used to test the operation of the UARTs on the gateway. *HyperTerminal* configures a COM port on the PC which correlates to either UART A or B on the gateway and sends ASCII characters to it at every keyboard stroke. Similarly, any characters echoed back to PC via the UART and COM port are displayed by *HyperTerminal*. This may be used to perform a *loopback* test of the UART by connecting its TX and RX signals together and any character that is transmitted by *HyperTerminal*, at the the configured baud rate, should be echoed and displayed without corruption.

The GSM modem on the communication module incorporates a UART that operates at a different voltage level to that of the gateway, 2,8V and 3,3V respectively. Bi-directional level shifting ICs thus bridge the voltage difference as discussed in section 4.3.7.3. It was decided to perform a loopback test on UART A and to include the level shifters to ensure correct operation of the entire communications link at an elevated baud rate.

The following procedure was required in order to perform the loopback test to verify the operation of the UART and the level shifters:

- jumpers J9, J10 and J11 were configured to route UART A to the communication module.
- *HyperTerminal* on the development PC was started and COM port 6 configured to 115200 baud, no handshaking, 8 bits, 1 stop bit.
- the gateway and comms module were powered and the USB cable connected to the development PC.
- the centre pins of jumpers J2 and J4 on the communication module were linked.

- channel 1 of an oscilloscope was connected to the middle pin of J10 and channel 2 connected to the link on J2 and J4.
- a capital U was pressed on the keyboard.

The following result was obtained from the oscilloscope:

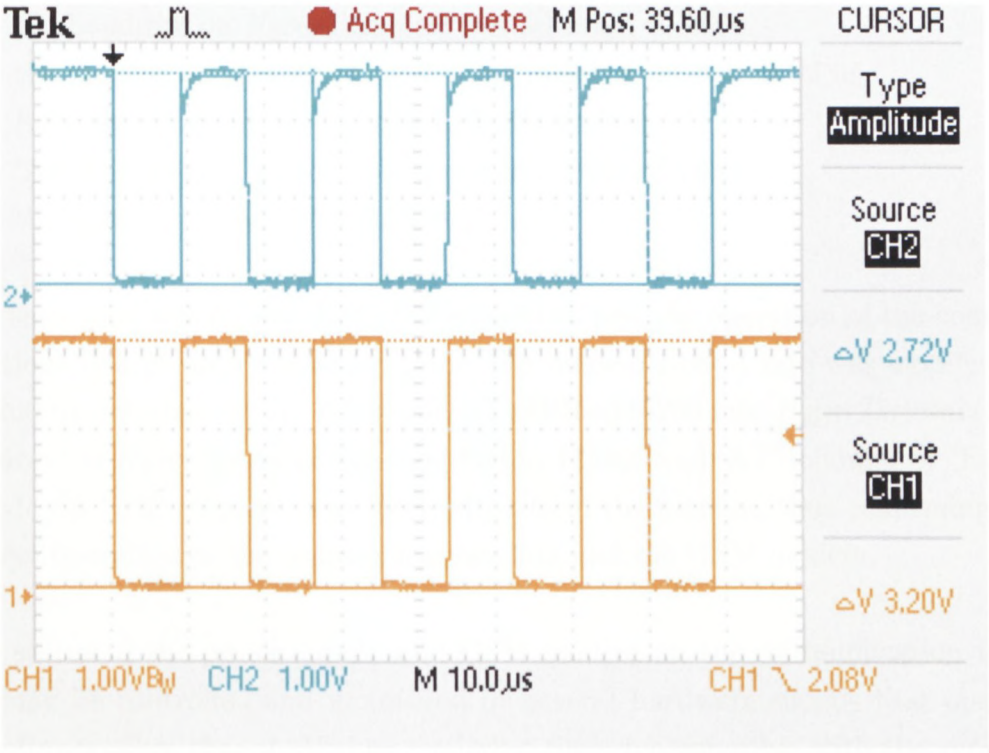


Figure 5.4: Channel 1 on the oscilloscope shows the output of UART A on the FT2232D when capital U is transmitted by *HyperTerminal* at 115200 baud and reaches a high level of 3,3V. Channel 2 shows the corresponding signal generated by the level shifter which only reaches a level of 2,8V. The voltage values displayed in the figure show the difference between the high and low voltage levels of the signals and are not referenced to signal ground.

It was found that a capital U was echoed back and displayed by *HyperTerminal* without corruption. Holding the U key down on the keyboard generated a stream of characters that were all echoed and displayed correctly. This proved that the loopback test was successful and that the FT2232D dual UART and communication module level shifters were deemed to be functioning correctly.

5.4.1 GSM modem and power control verification

The GSM modem on the communication module interfaces to the PCB via a 50pin Molex connector. Part of the loopback test was repeated in order to verify that the communication link with the GRPS modem was functional:

- the link between jumpers J2 and J4 was removed.
- the baudrate on *HyperTerminal* was adjusted to 9600.
- the modem was activated by holding pushbutton switch S2 in for a short period until status LED D2 flashed.
- 'AT' followed by 'Enter' was typed into *HyperTerminal.*
- the modem responded with 'OK'.

The same test was repeated at 115200 baud to test the operation of the communications link at an elevated bit rate. The modem's baud rate was modified to 115200 by entering the AT command AT+IPR=115200 into *HyperTerminal*; the baudrate in *HyperTerminal* was modified to 115200 and 'AT' followed by 'Enter' typed. An 'OK' response was received by from the modem, thus confirming the correct operation of the communications link and the GSM modem.

The status of the power supply and GSM modem on the communication module may be controlled and monitored by several hardware signals that operate at voltages other than 3,3V thus rendering them incompatible with the gateway microprocessor. Digital transistors Q1 to Q7 provide signal level-shifting and buffering of the signals and converts them to a suitable level for the microprocessor.

The communication module power supply may be disabled by placing a logic 0 onto pin 14 of connector P2. This permits the gateway processor to control the amount of power that is consumed by the system when required; it also permits the microprocessor to reset the modem if communications with it have failed.

An oscilloscope was used to capture a logic 0 to 1 transition on pin 14 during a power up event and the corresponding signal on the enable input of the switch mode pre-regulator U3:

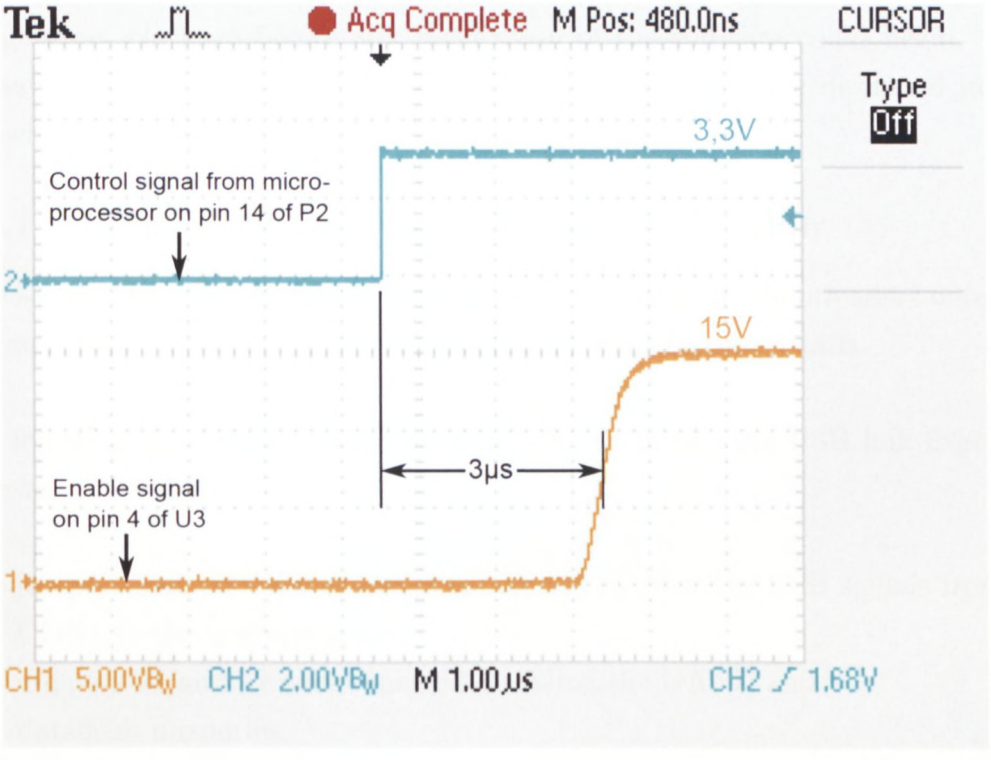


Figure 5.5: Channel 2 shows the transition from a logic 0 to 1 applied to the base of digital transistor Q6. This transition switches Q6 and Q7 *off* permitting the enable input, as shown on channel 1, of the pre-regulator IC U3 to rise to the module's supply voltage, in this case 15V, thus enabling the pre-regulator U3. It was noted that the switching of the digital transistors from a conductive to a non-conductive state took approximately $3\mu s$.

This test proved that the microprocessor was able to control the power supply on the communication module thus permitting overall system power control and the resetting of the GSM modem by the gateway application when required.

5.5 Software verification

A Windows CE operating system image was built using the toolchain assembled on the development PC according to the discussion in section 3.2.1.1. The bootloaders and OS image, as discussed in section 3.2.1.2, needed to be deployed to the gateway platform before any verification of the software could begin. The procedures used to deploy the image to the new platform are discussed in the following sections.

5.5.1 Bootloader deployment and verification

This section describes the procedures required to install the bootloaders onto the gateway using SAM-BA. Refer to section 3.2.1.2 for further details.

The following procedure was required in order to invoke the USB link from the gateway to SAM-BA:

- jumpers J12, J23 and J14 were configured to steer the USB signals from CON1 to the microprocessor.
- jumpers J1 and J2 were removed disabling the NAND and dataflash memories.
- power was applied to the gateway.
- SAM-BA was installed onto the development PC and started.
- a USB cable was connected between the gateway and the PC.
- after the USB enumeration of the gateway, the \usb\ARM0 connection and at91sam9260-ek board are selected in the SAM-BA drop down menu.

SAM-BA displayed a screen, as shown in figure 5.6, that permitted the writing, reading, erasing and verification of the memories on the gateway platform. Jumpers J1 and J2 were replaced enabling the dataflash and NAND memories for access by SAM-BA.

The bootloaders *FIRSTBOOT* and *EBOOT* were deployed to the dataflash memory by the following sequence:

- the script "Enable Dataflash (SPI0 CS1)" was selected and executed.
- the script "Send Boot File" was highlighted permitting the selection of *FIRSTBOOT.nb0* which was written to address 0x0.
- *EBOOT.nb0* was selected next by clicking the folder icon situated to the right of text box *Send File Name.*
- 0x4000 was written into the Address box and the file upload started by clicking the button labeled *Send File.*

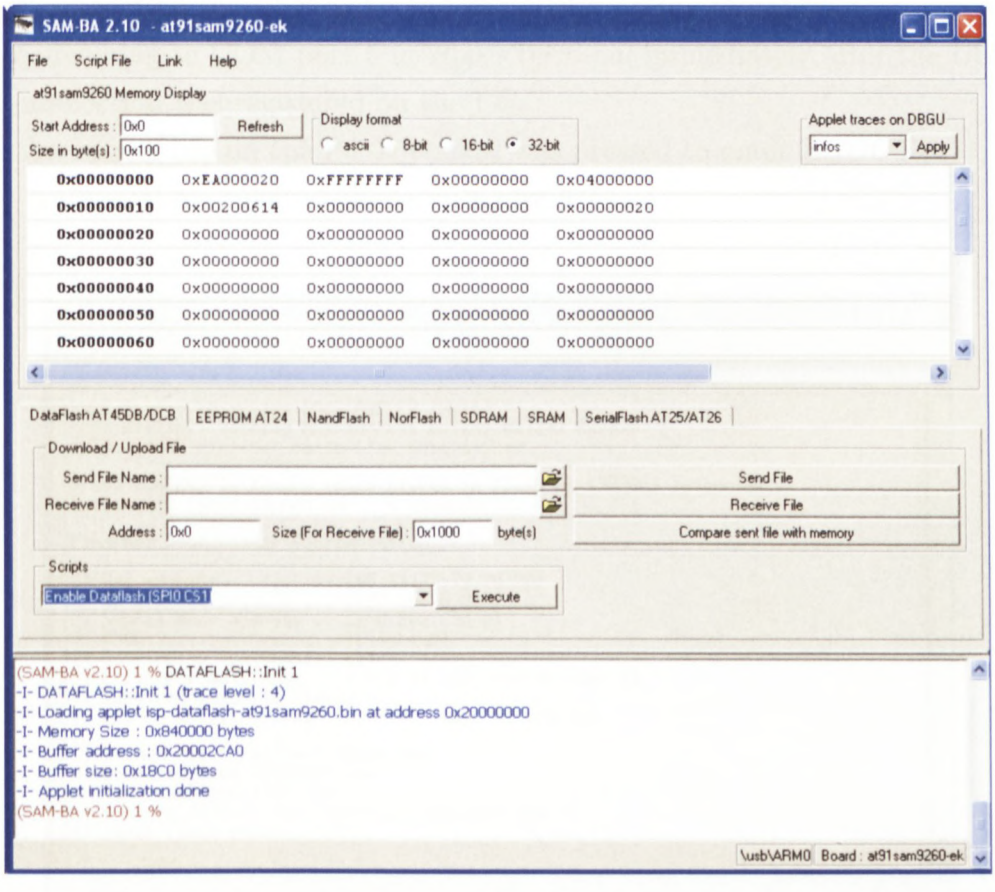


Figure 5.6: The SAM-BA application permits a user full control over the writing, reading, erasure and verification of the memories on the gateway platform.

SAM-BA was terminated and the USB cable and power supply removed from the gateway after the successful uploading of the bootloaders to the gateway.

The bootloader installation was verified by:

- configuring jumpers J12, J13 and J14 to steer the USB signals from CON1 to the FT2232D UART converter.
- configuring jumpers J9, J10 and J11 to steer the microprocessor debugging port to the FT2232D UART converter.
- starting *HyperTerminal* and configuring COM port 5 operate at 115200 baud, 1 stop bit, no handshaking.
- reconnecting the USB cable and the power supply to gateway.
- connecting to COM port 5 in *HyperTerminal* immediately after the USB connection was registered on the PC.
- the 'space' bar on the PC keyboard was pressed to enter *EBOOT*s configuration menu as shown in figure 5.7.

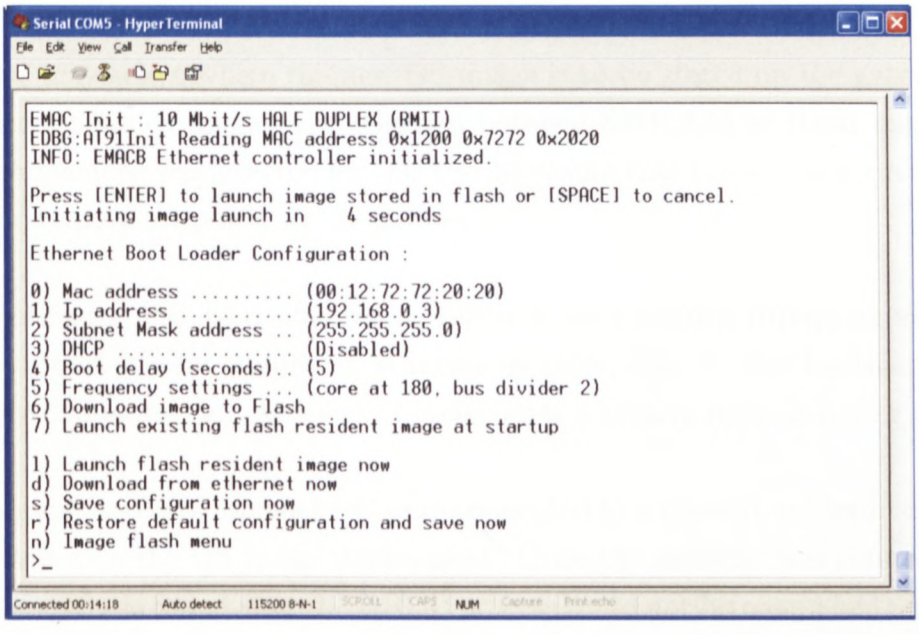


Figure 5.7: The menu display in *EBOOT* permits the configuration of the bootloader and the behaviour of the gateway following a reset.

The bootloader installation was proven to be fully functional by the results of the preceding procedures.

5.5.2 OS deployment and verification

The previous section dealt with the deployment of *EBOOT* onto the gateway platform. The following discussion shows how the OS image was installed onto the gateway by utilising *EBOOT*.

The configuring of *EBOOT* began by entering the **Image flash menu** and entering three OS parameters (**Physical Start Address**, **Starting ip** and **Total ROM size**) which were obtained from the file *makeimage.out* located in the OS build output directory. It was noted that these values may change every time a new OS image is generated.

The IP address of the gateway may be assigned automatically by enabling the DHCP function at menu item 3 or statically at item 1; a DHCP assigned IP was selected for this test.

The user may select where the new OS image is to be stored on the gateway by menu item 6 which toggles the selection between **SDRAM** or **flash** memories; the flash memory was selected for this test to ensure that the OS image remained after the removal of power to the gateway.

The bootloader may be configured to **Launch an existing image at startup** or **Download a new image at startup** by menu item 7. The bootloader was configured to launch an existing OS image after a system reset or power up.

The gateway and development PC were connected to a network router in order to commence with the OS image deployment. Once the gateway was connected to the router, it was found that the DHCP process could not be completed and after several attempts by the bootloader to restart the process, the gateway was reset. The router DHCP IP pool allocation was checked to determine if the gateway had been assigned an IP address, but none was found.

An investigation of the gateway PCB layout surrounding the Ethernet circuit revealed several potential issues that may prevent its operation at high bit rates. It was also suspected that the Ethernet socket CON5 containing the coupling magnetics may be at fault (section 4.3.2). The specified J00-0061NL connector was unavailable at the time of purchase and a similar device J00-0064NL was purchased instead. The J00-0061NL incorporates "Bob Smith" terminations that are missing in the J00-0064NL. In addition, it was found that the Ethernet connection was being attempted at 100Mbps only and that no attempts were being made at the slower rate of 10Mbps. It was not possible to force the router nor bootloader to attempt an Ethernet connection at 10Mbps; it was thus decided to modify *EBOOT*s source code to limit the Ethernet connection to 10Mbps. The modified source code was recompiled and redeployed to the gateway using the procedures discussed in the previous section. This problem is discussed further in section 6.3.

The new bootloader limited the connection of the Ethernet to 10Mbps, shown in figure 5.7, and the gateway was successfully assigned an IP address by the DHCP server within the router.

The OS image was downloaded to the flash memory from the development environment on the PC (refer to section 3.2.1.3) and the gateway reset in order to test the OS installation.

The retrieval of the OS image from the NAND flash and initialisation began successfully which was reported on *HyperTerminal*. However, the OS initialising process stopped and the gateway was reset when the Ethernet DHCP request attempt failed. An investigation of the hardware system-reset line *NRST* using an oscilloscope revealed that *NRST* was asserted during the initialisation, thus resetting the Ethernet interface IC and clearing the settings configured by *EBOOT*. The OS Ethernet driver files were modified to remove the assertion of *NRST*, to disable Ethernet auto-negotiation at 100Mbps and to force the Ethernet to connect at 10Mbps only.

The OS image was rebuilt in order to include the modifications made to the Ethernet drivers and redeployed to the gateway by *EBOOT*. The gateway was reset in order to test the new OS installation and to ensure that the initialisation completed successfully. No further errors were reported during the initialisation process. The time taken for the entire boot and OS launch process was approximately 28 seconds.

A Telnet client on the development PC was used to connect to the Windows CE Telnet Service on the gateway at the IP address allocated by the router and a directory listing of the file system requested, as shown in figure 5.8.

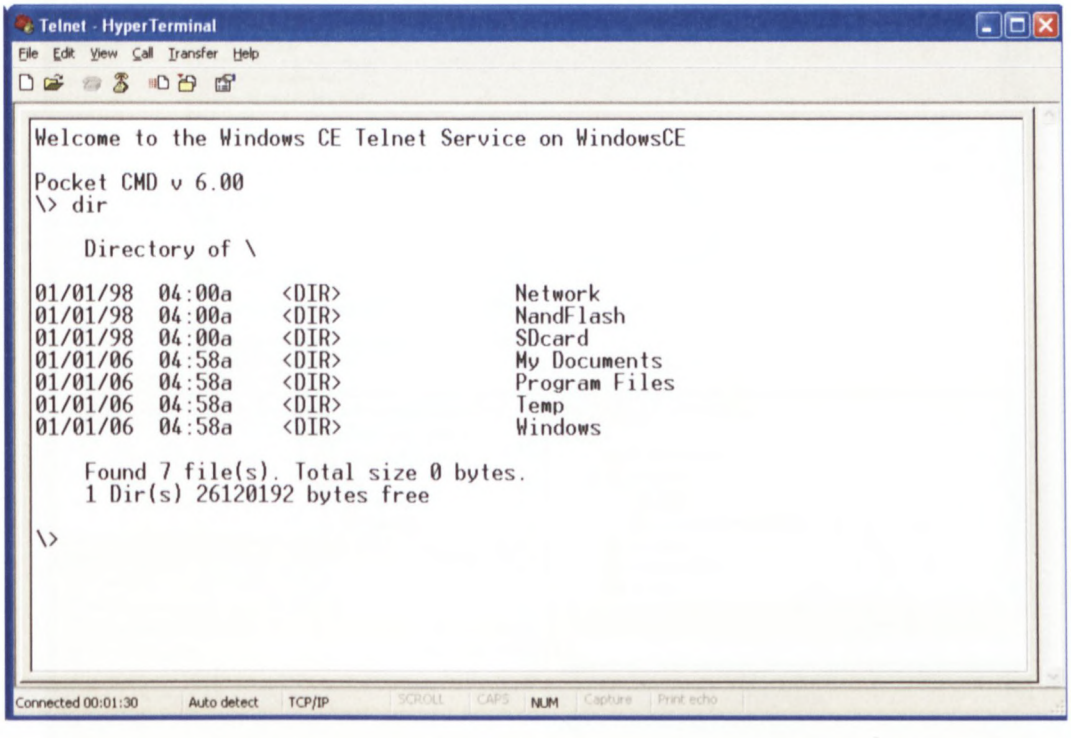


Figure 5.8: *HyperTerminal* was used as a Telnet client to log on to the gateway OS and request a directory listing of the file system.

This test proved that the Windows CE 6.0 OS had been successfully generated by the toolchain on the development PC, deployed and installed onto the gateway platform and was operational at the completion of the booting process.

5.5.3 Gateway application verification

The gateway application was deployed to the platform by an FTP client as shown in Figure 5.9 and launched in a Telnet terminal window. The success of these steps proved that the FTP server and the *.NET Compact Framework 3.5*, as discussed in section 3.2.1.4, embedded in the OS were both functional. The following sections discuss the verification of the evaluation application.

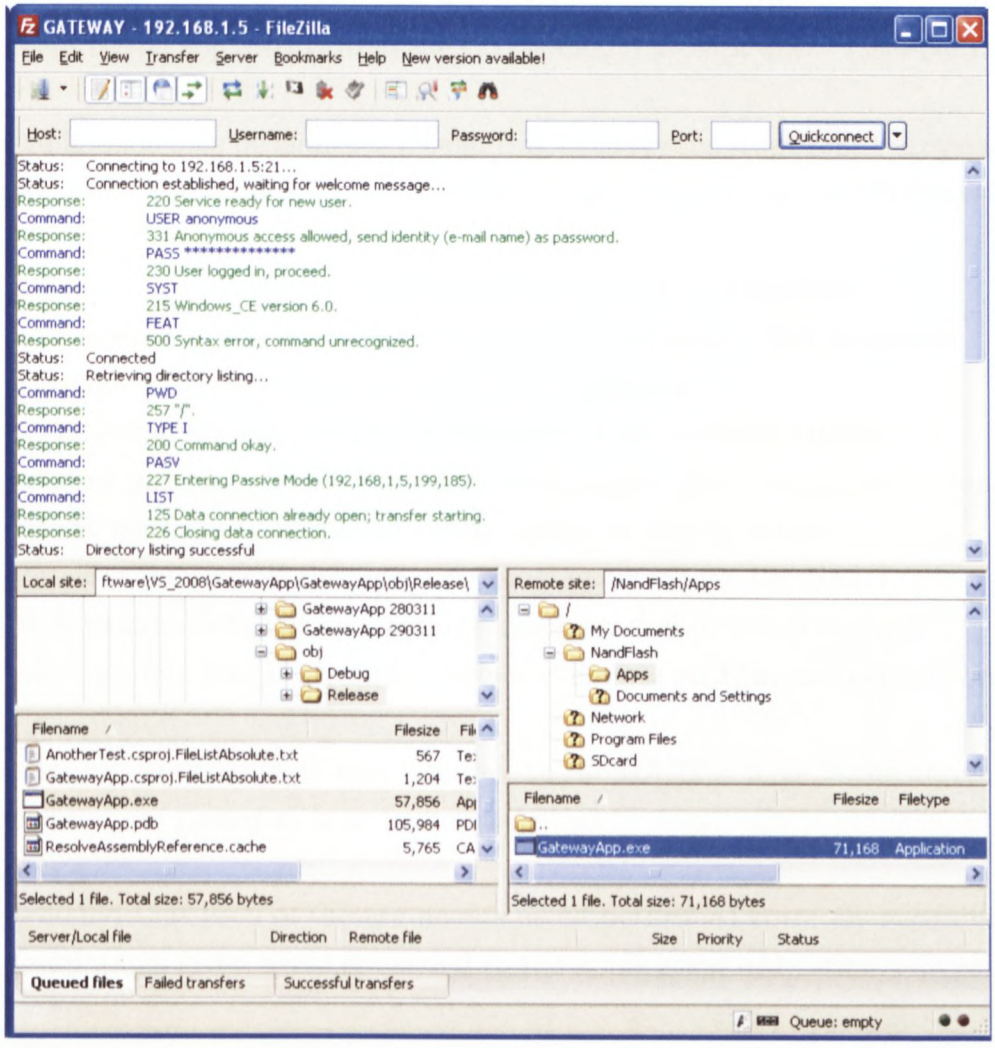


Figure 5.9: The exchange of commands between the FTP client *FileZilla* and Windows CE6.0 server to establish an FTP session, to login to the server and request a directory listing are shown in the Figure above. *FileZilla* was used to create the subdirectory *Apps* in *NandFlash* and to upload the gateway application *GatewayApp.exe* to it.

5.5.3.1 Application launch

The gateway hardware developed in this study did not include a display nor user interface and therefore relied on the use of a terminal emulator such as *HyperTerminal*, via the UART, or Telnet client via the Ethernet connection to permit the configuration of the application. It was necessary to include a means of selecting the application's user interface in order to reduce the complexities of programming a parallel user interface. This was achieved by permitting a user to pass an argument to the application from the OS command prompt only [1] (section 3.4.1).

The application may be launched by entering `gatewayapp <optional argument>` at the command prompt in the *\NandFlash\Apps* directory of the OS filesystem.

The following arguments are accepted by the gateway application:

- `eth` steers the user interface to the current command shell accessible through a Telnet client via an Ethernet connection
- `kill` terminates any executing instances of the gateway application
- `nogprs` prevents the GSM modem going online after an application launch
- `reset` resets the application configuration to default values; the user interface is through a terminal emulator via the UART
- `eth reset` resets the application's configuration to default values; the user interface is through a Telnet client via an Ethernet connection

The application defaults its user interface to the UART if none of the arguments listed above are passed to it at launch time.

It was found that each of the arguments listed performed correctly according to the descriptions given when appended to the application invocation.

[1] It was not possible to launch the gateway application from a UART terminal emulator as the OS command prompt was only accessible through a Telnet client running on a PC connected to the gateway via the Ethernet.

5.5.3.2 Application configuration

The gateway software specification required the application's configuration to be stored in a persistent memory to ensure that the gateway's functionality was maintained after power interruptions or system shutdown. Default configuration data was written into registry keys that were created by the application during its first execution on the gateway platform. Thereafter a user was permitted to adjust the configuration by means of the application's menu structure when necessary (refer to section 3.4.4). The registry is, by definition, stored as hives in the NAND flash memory, and is therefore persistent and unaffected by system resets or power interruptions.

5.5.3.3 Persistent data storage

The specification required the application to store and timestamp any data packets received from the WSN node in a persistent memory to preserve them in the event of a system reset or power interruption. The gateway incorporates a socket for the insertion of an SD or MMC card which is used by the OS or gateway application as the persistent storage medium for data. Figure 5.10 shows a snippet of timestamped data captured and stored on the memory card by the application.

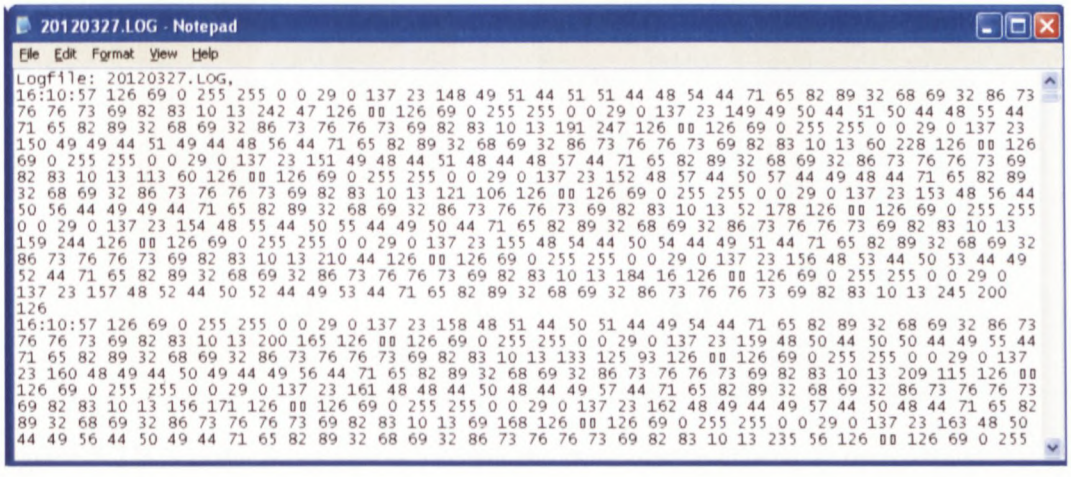


Figure 5.10: All data packets received from the WSN sink node were in 8-bit binary format which were converted by the application into equivalent ASCII characters and written to a data file on the memory card. Every eleventh packet was timestamped and the data delimited by white space characters to aid its future analysis. The data file name was composed from the gateway's system date and a new file created when a change in date was detected.

5.5.3.4 Packet repetition rate

As discussed previously, *all* data packets received from the wireless sink node were required to be stored in a persistent memory. The following evaluation was required to determine if any packets of data transmitted by the wireless node were lost by the gateway, and if so, to establish the transmission rate when this occurs.

A stable stream of packets was required to simulate a wireless network for the evaluation. A wireless node was programmed with the firmware in Appendix F.14 to transmit sequentially numbered packets each consisting of 42 bytes to the gateway application for up to an hour at a time to ensure a reasonable sample of packets. A timer *Tmr0* in the node's firmware was adjusted to alter the delay period t between each packet transmission, as shown in Figure 5.11 and Table 5.2:

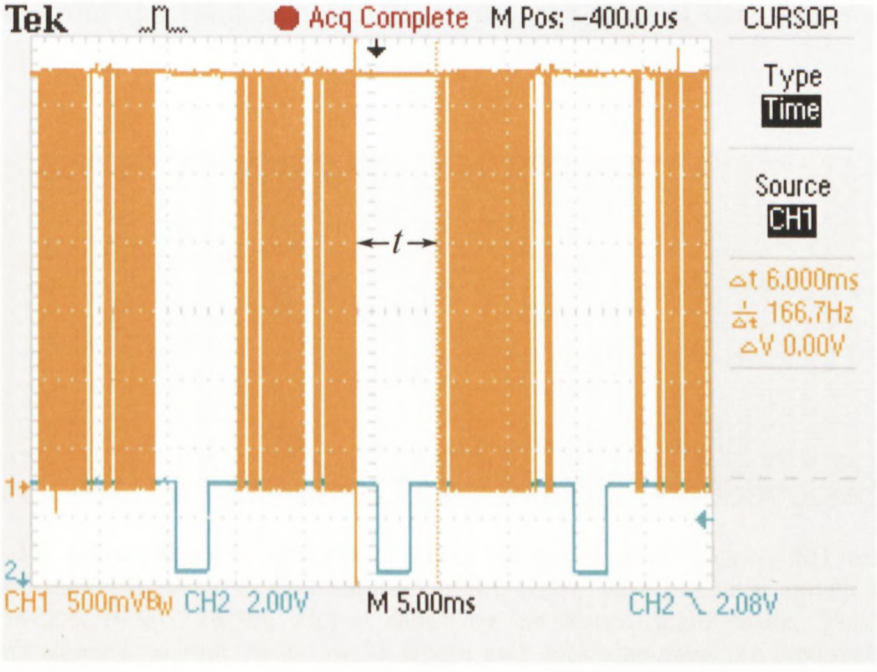


Figure 5.11: t is the delay period between the transmission of each 42 byte data packet and was adjusted by altering a timer value in the transmitting node's source code. The 5ms delay period between the packets was the shortest time achievable by the node application and was lengthened for each subsequent test.

Tmr0	***t* (ms)**
10	5,24
15	10,06
20	15,06
25	20,35
35	29,60
50	44,40
100	93,00
150	142,60
200	191,20

Table 5.2: The delay period t between each packet transmission is adjusted by *Tmr0* in the wireless node packet generator firmware.

Data packets were captured at each delay period t, converted and stored in files on the gateway SD memory card. Each file was imported into an Excel 2003[1] worksheet in order to determine the number of corrupt packets received or packets absent from the total sample. A snippet of a typical data file is shown in Figure 5.12:

15:37:43	126	69	0	255	255	0	0	29	0	137	16	206	48	51	44	56	51	44	52	51	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	20	168	126
	126	69	0	255	255	0	0	29	0	137	16	207	48	52	44	56	52	44	52	52	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	205	90	126
	126	69	0	255	255	0	0	29	0	137	16	208	48	53	44	56	53	44	52	53	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	123	4	126
	126	69	0	255	255	0	0	29	0	137	16	209	48	54	44	56	54	44	52	54	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	165	53	126
	126	69	0	255	255	0	0	29	0	137	16	210	48	55	44	56	55	44	52	55	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	192	164	126
	126	69	0	255	255	0	0	29	0	137	16	211	48	56	44	56	56	44	52	56	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	54	192	126
	126	69	90	65	10	13	151	235	126																																	
	126	69	0	255	255	0	0	29	0	137	16	215	49	50	44	57	50	44	53	50	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	73	218	126
	126	69	0	255	255	0	0	29	0	137	16	216	49	51	44	57	51	44	53	51	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	253	237	126
	126	69	0	255	255	0	0	29	0	137	16	217	49	52	44	57	52	44	53	52	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	36	31	126
15:37:43	126	69	0	255	255	0	0	29	0	137	16	218	49	53	44	57	53	44	53	53	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	65	142	126
	126	69	0	255	255	0	0	29	0	137	16	219	49	54	44	57	54	44	53	54	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	159	191	126
	126	69	0	255	255	0	0	29	0	137	16	220	49	55	44	57	55	44	53	55	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	170	188	126
	126	69	0	255	255	0	0	29	0	137	16	221	49	56	44	57	56	44	53	56	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	92	216	126
	126	69	0	255	255	0	0	29	0	137	16	222	49	57	44	57	57	44	53	57	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	57	73	126
	126	69	0	255	255	0	0	29	0	137	16	223	50	48	44	48	48	44	54	48	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	240	201	126
	126	69	0	255	255	0	0	29	0	137	16	224	50	49	44	48	49	44	54	49	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	66	69	126
	126	69	0	255	255	0	0	29	0	137	16	225	50	50	44	48	50	44	54	50	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	156	116	126
	126	69	0	255	255	0	0	29	0	137	16	226	50	51	44	48	51	44	54	51	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	249	229	126
	126	69	0	255	255	0	0	29	0	137	16	227	50	52	44	48	52	44	54	52	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	32	23	126
15:37:43	126	69	0	255	255	0	0	29	0	137	16	228	50	53	44	48	53	44	54	53	44	67	80	85	84	32	67	65	80	69	84	79	87	78	44	90	65	10	13	21	20	126

Figure 5.12: Shown above is a snippet of a data file stored on the gateway SD memory card where every eleventh packet is timestamped. Each consecutive packet is assigned a unique number (16 206, 16 207, 16 208, etc) as shown by the vertical highlighting. The horizontal highlighting shows a *corrupt* packet as its length and data alignment are incorrect. *Corrupt* packets were considered to be equal to *lost* packets and were included in the *packet loss* calculation. The number of *lost* packets may be calculated by finding the difference between the packet numbers adjacent to the discontinuity. As can be seen from the above, 3 packets have been lost.

[1] Excel 2003 permits a maximum of 65535 rows of data to be imported into a worksheet and is a known limitation of the package.

The wireless node assigns a sequential number to each packet transmitted. The packet numbers were copied into a graph to locate any lost or corrupted packets. A straight line graph without discontinuities was expected if no packet losses were present in the data file, as shown in Figure 5.13. Lost or corrupt packets were shown in the graph as discontinuities as in Figure 5.14.

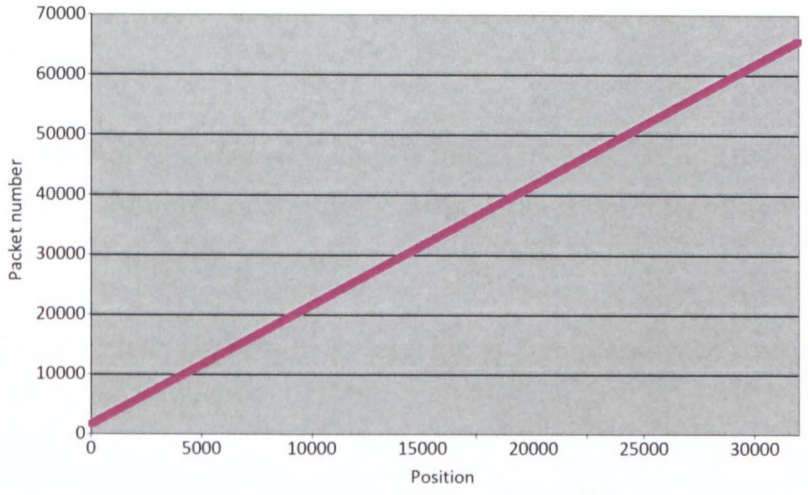


Figure 5.13: No packet loss was found when $t = 20$ms as the graph is continuous.

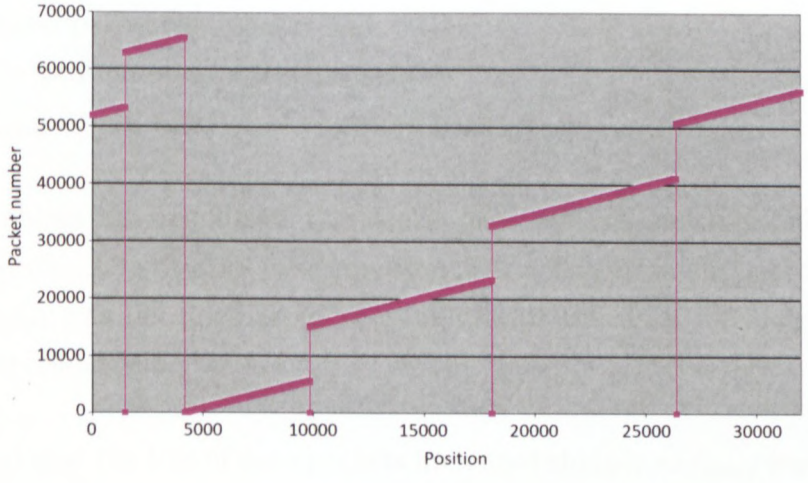


Figure 5.14: Packet losses are represented by discontinuities in the graph. This graph was generated by the data file captured at $t = 5{,}24$ms which resulted in a packet loss of 50,42%. The largest step in the graph at position 5000 was the result of the packet address wrapping to 0 from 65535.

The discontinuities of the graph in Figure 5.14 show the position in the data file where the gateway was not able to process the data packets being transmitted by the wireless node resulting in packet loss or corruption.

The number of *lost packets* at each discontinuity was calculated by finding the difference between the packet numbers of the packets adjacent to the discontinuity in the data file; the total number of *lost packets* was calculated by summing these differences.

The total number of *expected packets* was found by calculating the difference between the packet numbers of the last and first packets received and taking into account any packet number wrap around from 65535 to 0 as shown in Figure 5.14.

The percentage of lost packets in a data file is the quotient of *lost packets* and *expected packets* multiplied by 100:

$$Ploss_{tot} = \frac{packets_{lost}}{packets_{expected}} \times 100 \tag{5.1}$$

where:
$Ploss_{tot}$ - total percentage packet loss
$packets_{lost}$ - total number of lost packets
$packets_{expected}$ - total number of expected packets

Table 5.3 shows the results of the packet loss $Ploss_{tot}$ calculations after the listed data was substituted into equation 5.1. The data for $packets_{lost}$ and $packets_{expected}$ was obtained from the analysis of the data files imported into the Excel spreadsheets.

It was found that the loss of data packets increased sharply as t_{delay} was shortened below 15ms. It was suspected that the loss of data packets, as shown by the discontinuities in Figure 5.14, when t_{delay} = 5,24ms and t_{delay} = 10,06ms may be caused by the application's data receive buffer being overwhelmed by fresh data from the wireless node before the existing data in the buffer could be processed

and stored by the application resulting in a buffer overflow scenario. Further investigation of this problem would be required to confirm this suspicion and to find a suitable solution.

Tmr0	t_{delay} **(ms)**	tx_{rate} **(B/s)**	$packets_{lost}$	$packets_{expected}$	$Ploss_{tot}$(%)
10	5,24	2949	66621	132127	50,42
15	10,06	2204	9531	36266	26,28
20	15,06	1746	8	36786	0,02
25	20,35	1431	300	54627	0,55
35	29,60	1088	19	33598	0,06
50	44,40	772	5	13628	0,04
100	93,00	412	0	20429	0,00
150	142,60	277	0	4299	0,00
200	191,20	210	0	2659	0,00

Table 5.3: The percentage of packets lost increases as the delay period between the packets reduces. The values in *Tmr0* were entered into the packet delay timer on the wireless node. t_{delay} and $packets_{lost}$ are the resulting delay periods between the packet transmissions and percentage packet loss by the gateway application respectively. The rate transmission rate tx_{rate} data is from Table 5.4.

The following equation was used to determine the rate at which the data was transmitted to the gateway in *bytes per second*:

$$tx_{rate} = \frac{1}{T_{packet\,width} + t_{delay}} \times no.\,of\,bytes \tag{5.2}$$

where:

tx_{rate} - rate of data transmission (B/s)

$T_{packet\,width}$ - width of a data packet[2] (s)

t_{delay} - delay period between packet transmissions (s)

$no.\,of\,bytes$ - number of bytes in a data packet[2]

Table 5.4 shows the results of transmission rate calculations after the listed data was substituted into equation 5.2.

[2] $no.\,of\,bytes$ in a packet is fixed at 42 which equated to a $T_{packet\,width}$ of 9ms at 115200 baud.

Tmr0	t_{delay} **(ms)**	$T_{packet\,width} + t_{delay}$ **(ms)**	tx_{rate} **(B/s)**
10	5,24	14,24	2949
15	10,06	19,06	2204
20	15,06	24,06	1746
25	20,35	29,35	1431
35	29,60	38,60	1088
50	44,40	54,40	772
100	93,00	102,00	412
150	142,60	151,60	277
200	191,20	200,20	210

Table 5.4: *Tmr0* determines the data packet transmission rate.

Figure 5.15 shows the packet loss versus the rate of data transmission. A small increase in the packet loss was noted at a delay period of 20,35ms. It was suspected that a combination of coincidental events between the gateway's operating system, VCP drivers and the application may have caused this marginal increase in packet loss. Further investigation of this problem would be required to confirm this suspicion.

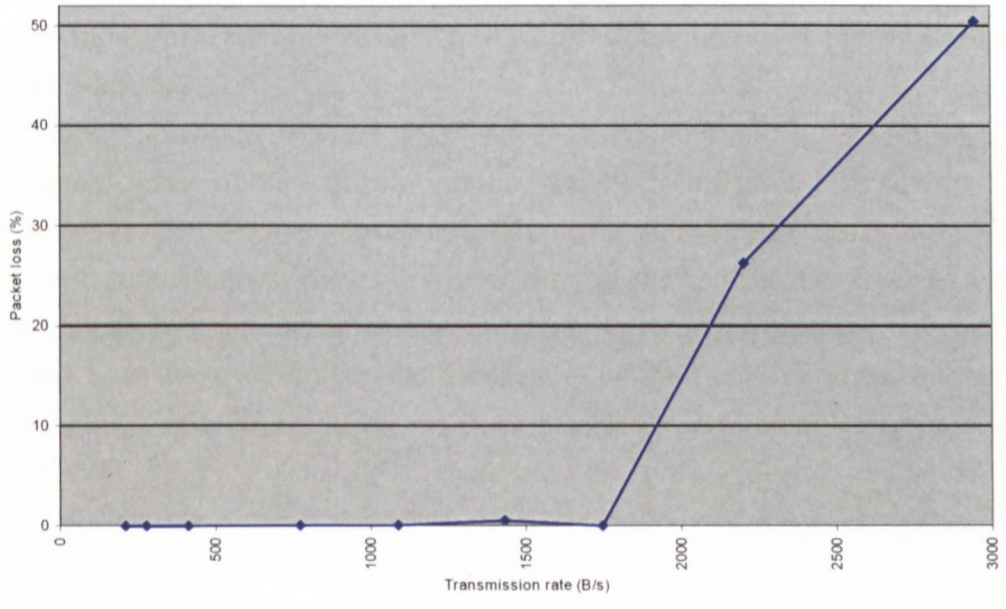


Figure 5.15: Packet loss increases sharply as the rate of data packet transmission increases. The small increase in packet loss at the transmission rate of 1431 B/s may be caused by coincidental events occurring in the operating system, VCP driver and application.

5.5.3.5 Stored data transferral

Scheduled transfers of the recorded data to remote file servers located on the Internet were required by the gateway specification. File transfers were configured to be executed by FTP via either the Ethernet connection or communication module at the scheduled time configured in the application's *log* and *ftp* menus.

The stored data files were compressed by the application prior to being transferred to limit the amount of data transmitted. Only files with the *.gz* extension that were successfully transmitted would have their extensions modified to *.gz1* to ensure that a repeat transmission of the original files did not occur.

The Ethernet and GSM Internet connections were both evaluated for FTP file transfers. The communication module was configured to FTP the compressed data file 20120301.LOG.gz to a remote file server located at the address *gatewayftp.dyndns.org* whose assigned IP is resolved following an automatic DNS query performed by the GSM service provider. After the successful completion of the file transfer, the application was reconfigured to permit the Ethernet to transfer the compressed file 20120302.LOG.gz to the file server on the local network at address 192.168.1.2.

Screenshots taken of the gateway application's selected user interface and the FTP server during the file transfer evaluations are shown in figures 5.16 and 5.17. A binary comparison was performed after the transfers on the files received by the server and the original files on the gateway memory card to check for corruptions that may have occurred during the transfers. The files were found to be identical proving that the FTP transfers via both interfaces were completely functional and reliable.

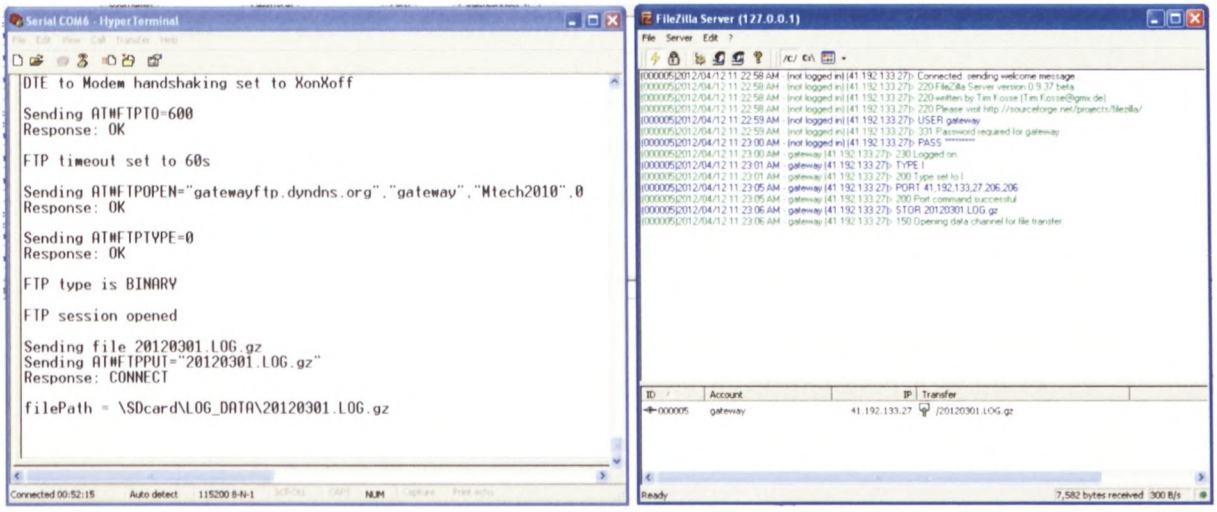


Figure 5.16: The GSM modem on the communication module was configured to execute the FTP transfer of the compressed data file 20120301.LOG.gz. The display on the left shows the AT commands required to initiate an FTP session with the file server located on the Internet at address *gatewayftp.dyndns.org*, shown in the display on the right. The IP address for *gatewayftp.dyndns.org* is resolved by an automatic DNS query performed by the GSM service provider when the *AT#FTOPEN* command is issued to the modem. The file transfer begins once the FTP session with the server is opened successfully. The length of time it takes to send a file depends on the GSM network's Quality of Service level and how busy the GSM network's service is at the time of transmission.

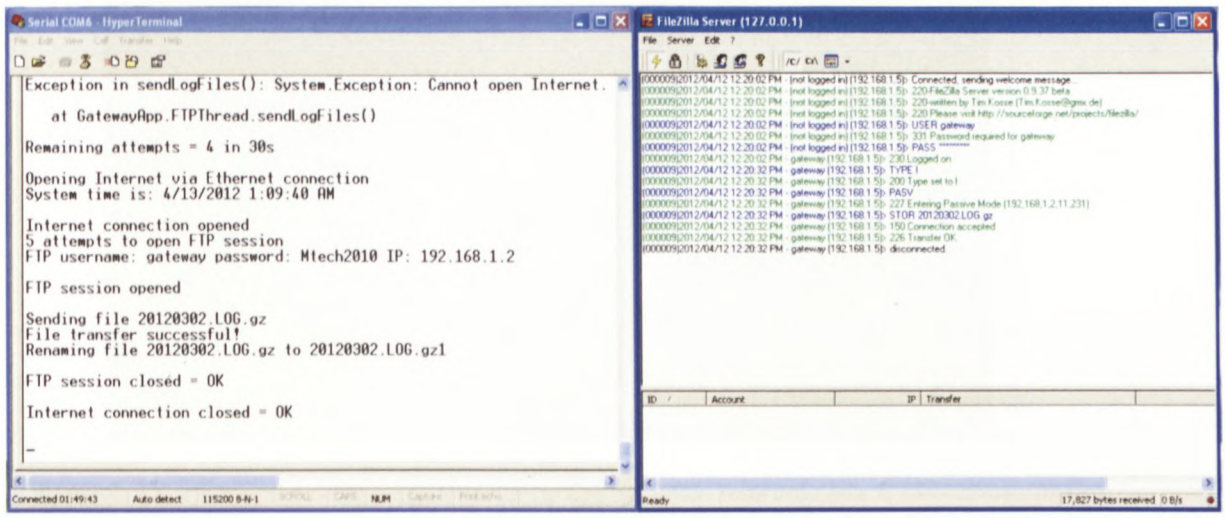


Figure 5.17: The gateway application may be configured to transfer a compressed data file via the Ethernet connection to a file server on the local network. The username and password were transmitted to the server at the designated IP address; an Internet connection and FTP session were established and the file transferred successfully to the server.

5.5.3.6 Remote gateway connection

The gateway is likely to be deployed into remote locations that will not permit the configuration of the gateway application via an Ethernet or USB cable connected to a nearby network or PC. The communication module is able to accept a connection from an IP-based client terminal connected to the Internet permitting a remote user access to the gateway. The GSM modem required a SIM card with the `unrestricted` APN enabled by the GSM service provider before a public IP address could be assigned to the modem and any incoming GPRS connections accepted.

Figure 5.18 shows a successful GPRS connection and text transfer between a client terminal and the GSM modem on the gateway's communication module following its configuration by the application. In this instance, the GSM modem was configured to accept incoming connections from a range of IP addresses 41.241.0.0 to 41.241.255.255 on TCP port 49500. Only a client with an IP address within the range of permissible addresses defined by the combination of the modem's firewall and IP filter mask was permitted to connect to the gateway.

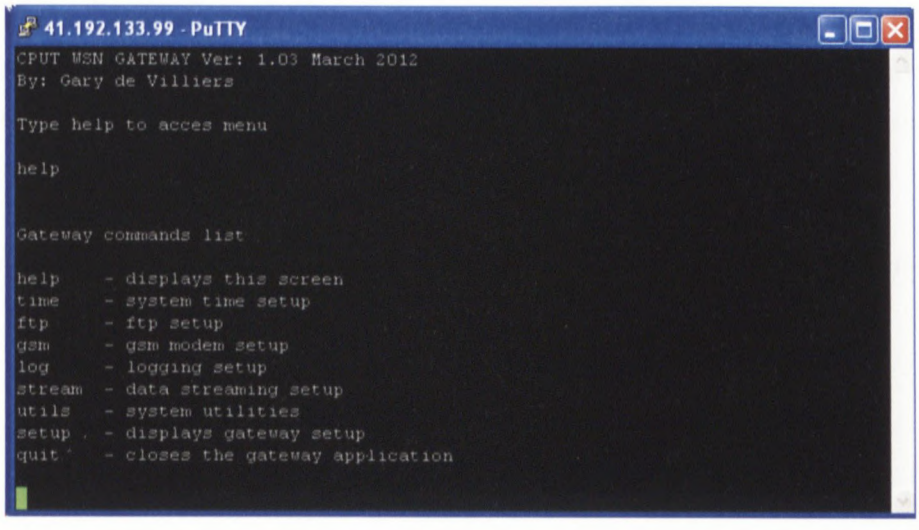


Figure 5.18: A user was able to connect to the gateway remotely via a GPRS connection using a TCP/IP client such as PuTTY which sends ASCII characters to an IP address over a TCP/IP connection on any TCP port which permitted the user to modify the configuration or operation of the gateway. At the time of this writing, only IP addresses in the range 41.241.0.0 to 41.241.255.255 on port 49500 were permitted to connect to the gateway.

5.6 Gateway hardware cost

The specification in section 1.4 required the gateway to be designed in such a manner as to minimise the costs of its manufacture and to incorporate components that were sourced from component manufacturing companies represented by local suppliers only. Only the costs for the production and assembly of the hardware has been taken into account; a costing for the software development tools and licenses has not been performed.

The final component costings were separated into two bills of materials as shown in tables E.1 and E.2 in Appendix E respectively.

The construction cost of the gateway was R1622,97 (approximately 194.43 USD) and R1145,73 (approximately 137.25 USD) excluding 14% VAT.
The total cost was R2768,70 (approximately 331.70 USD).

The cost of the hardware developed in the study was found to be considerably lower than the commercial offering by National Instruments and the Atmel development kit employed by Steenkamp, et al.(Steenkamp et al., 2009). The Gumstix Network starter pack was advertised for sale at 327 USD and did not include a GSM/GPRS modem module; thus the cost of gateway developed in this study was less than the Gumstix offering.

The development of the gateway in this study may be deemed *low cost* when compared to the gateway technologies listed in table 1.1.

5.7 Summary

This chapter discusses the evaluation of the gateway hardware platform, the deployment of the bootloaders, Windows Embedded CE OS and the gateway application software developed for it.
The following evaluations of the gateway hardware were conducted:

- visual inspection and electrical continuity checks of all PCBs before assembly
- power supply pre-regulator output verification
- power supply start up sequence
- gateway power consumption at supply voltage extremities and various temperatures
- USB to serial port converter verification
- communication module UART level shifter and control verification

The hardware platform was found to be functional and the power consumption remained within the specified limit of 3,5W during all modes of system operation.

The bootloaders, OS and gateway application were deployed to the platform and verified sequentially, as summarised below:

- bootloaders written to dataflash memory by SAM-BA
- *EBOOT* configured through a terminal emulator via the UART
- Ethernet interface found not to function at 100Mbps; *EBOOT* modified to force Ethernet connection to 10Mbps
- OS deployed to NAND flash memory via Ethernet from development PC
- OS Ethernet driver modified to force connection at 10Mbps
- OS verified; Telnet and FTP servers functional
- gateway application deployed to gateway via FTP
- application launch, configuration, data storage, data transfers via Ethernet and GSM media verified
- gateway packet data receiving rate ability characterised

The final cost of the gateway hardware construction in comparison to other gateway implementations concludes the chapter.

Chapter 6

Conclusions

6.1 Overview

The following chapter presents the general conclusions of the study and provides the answers to the research questions posed in section 1.2. Several problems encountered during the evaluation of the assembled hardware and the development of the gateway application are discussed. The chapter concludes with recommendations for future work to improve the performance of the hardware and gateway application developed during this study.

6.2 General conclusions

Wireless Sensor Networks and several typical deployments were studied in order to establish the types of equipment used to permit remote access to the network from the Internet and how generated data was handled. It was found that gateway devices, several with Internet connectivity, were implemented with equipment such as laptop computers, IC evaluation kits and complex single board computers, with varying degrees of success and cost. Based on the findings of the study, a specification for a generic, low cost gateway with several Internet connection options was developed to provide answers to the following research questions:

1. **Was it possible to design a gateway that is generic in nature to suit a variety of WSNs?**
   This study produced a functional, generic gateway that has the ability to interface to several WSNs by means of a wireless sink node which interfaces with the gateway through a dedicated USB host socket on the gateway PCB. The application written for the gateway permits the storage and forwarding of any packets received from a Wireless Sensor node executing the *TinyOS* operating system.

2. **Was it possible to design a low-power gateway with sufficient resources to execute a multi-tasking operating system, provide persistent data storage and scheduled transfers of the data to remote locations via the Internet?**
   Windows Embedded CE is a multi-tasking operating system and was found to be a suitable choice for use in this study as it permitted the inclusion of the *.NET Compact Framework* which supported the development of the gateway application in Visual C#, a high-level object oriented language promoting the ease of future modifications and additions to the gateway application.

   Persistent storage for the data packets received from the wireless sensor node was provided by an SD memory card. The received packets were converted from their raw 8-bit binary format into equivalent ASCII characters to form readable data which was timestamped and stored in files organised by the FAT

filesystem implemented automatically by the operating system.

Windows Embedded CE implements a real time clock which was used by a scheduler in the gateway application to perform regular uploads of stored data by using the FTP protocol to remote file servers on the Internet. The *Gzip* compression algorithm was applied by the application to data files stored on the SD card when a change in date was detected to limit the amount of data transmitted over the Internet.

The gateway was designed to have a power consumption of less than 3,5W during all stages of its operation permitting it to be powered from a moderately sized battery and photovoltaic charging system when required.

3. **Was it possible to design a gateway that provided several Internet communication interfaces?**
   The gateway designed in this study has a fixed wired Ethernet port and a removable communication module featuring a GSM modem which both permit access to the Internet. In order to ensure that the gateway functionality was kept generic and its means of connecting to the Internet configurable, the idea of an auxiliary module featuring a specific Internet connection technology was conceived. A full modem UART with handshaking signals was terminated on a generic header which connects to a corresponding header on the communication module via a flexible ribbon cable permitting the easy removal and replacement of the communication module when required. Only one communication module featuring a GM862-GPS modem and its supporting software was developed in this study.

4. **Was it possible to design a gateway that included an interface for debugging and configuring a WSN and the gateway via the Internet?**
   The gateway application has a user interface that is accessible via the Internet through either the Ethernet or GSM connections. The same user interface is also accessible through a serial port on the gateway permitting a local user the

opportunity to configure or troubleshoot the gateway. The user interface permits the adjustment of the gateway application's configuration settings which are stored in the Windows registry, which is persistent by definition.

In addition to the scheduled uploading of stored data, the application permits the forwarding of raw packets of data received from the WSN node to a remote TCP/IP server via an Ethernet connection only when debugging or monitoring of the WSN is required.

5. **Was it possible to design a gateway that was cost effective and constructed with components obtainable from South African electronics suppliers?**
   It was decided that all the components required for the gateway be procured from locally based agents and suppliers that represent the international manufacturers of the components. This was done to simplify and expedite the procurement process and to ensure that the costs of the components were kept to a minimum. The overall cost of the gateway and communication module hardware developed in this study was found to be less than the costs of gateway implementations examined previously.

6.3 Problems encountered

The Ethernet interface on the gateway PCB was unable to establish a connection to a network at 100Mbps and it was suspected that the PCB layout surrounding the Ethernet connector CON5 may have been the cause of this problem. In addition, it was suspected that the absence of the "Bob Smith" termination in the magnetics of the Ethernet connector used in the design may be a contributing factor in preventing Ethernet connections at 100Mbps. As a temporary workaround, the bootloader *EBOOT* and the Windows CE Ethernet driver supplied with the AT91SAM9260 board support package were modified to force the Ethernet to connect at 10Mbps to ensure Ethernet connectivity and to overcome this problem.

The control of general purpose IO port pins on the microprocessor by the gateway application was found to be complex due to the data marshaling required to call a low-level IO function within the operating system.

A workaround solution was implemented where an auxiliary application *IO_Control.exe* was invoked and passed arguments to control the state of individual IO pins (refer to section 3.3.6).

The lack of a graphical user interface on the gateway platform prevented the use of the *Dial Up Networking* functionality incorporated in the Windows CE OS for the control of the GSM modem on the communication module. This resulted in the development of a bespoke modem control class within the gateway application to permit incoming GPRS connections and FTP file transfers via the Internet, as discussed in section 3.4.5.

6.4 Future work and recommendations

The present gateway prototype employs a powerful multitasking operating system which required a complex hardware platform to support and execute it resulting in a moderate power consumption of up to 3,5W for normal operation. This power consumption prohibits the long term deployment of the gateway without a battery charging system or other permanent power source. This results in future research to develop a hardware platform with further reduced power consumption which may be achieved in several ways:

- redesign the power supply to completely exclude the use of linear regulators supplying the low voltage rails as these devices waste significant power in the form of heat

- respecify the gateway to exclude the use of a full multitasking operating system permitting the use of a smaller microcontroller with fewer peripherals and bespoke firmware tailored to promote minimised power consumption

- rewrite the application in *C++* to negate the need for the *.NET Compact Framework* which was shown to increase the gateway's power consumption when invoked

- investigate alternative operating systems such as OpenBSD that may offer reduced power consumption during their execution

- investigate the use and inclusion of a 3G GSM modem on an additional communication module to reduce the average power consumption by reducing the FTP file transfer time

The size of the gateway PCB may be reduced by utilising a microprocessor and memories encased in BGA-style packages even though these packages pose several disadvantages as discussed in section 2.2.2. Adding additional signal layers to the PCB will also result in a reduced system footprint and increased signal and power supply integrity.

The PCB layout surrounding the Ethernet connection should be investigated and modified if any potential problems are found to permit a connection at 100Mpbs. In addition, a connector incorporating the "Bob Smith" termination should be procured and incorporated into the revised design.

The present gateway application converts *all* packets received by the wireless node in 8-binary format to equivalent ASCII characters. This was found to create very large data files on the SD memory card resulting in unwieldy data manipulation requirements and lengthened file access times. The gateway application may be modified to address this issue by filtering the received data packets and only storing the relevant packets in the stored data files. In addition, the binary to ASCII conversion may be removed and the only the raw packet data stored resulting in smaller data files. A further recommendation is to monitor the size of the current data file and to create a new file should the initial one become too large resulting in several files for the day's data capturing.

The evaluation of the packet repetition rate in section 5.5.3.4 revealed that large packet losses occur at elevated data transmission rates.

It was suspected that an overflow of the data buffer may be contributing to this problem. In addition, a marginal increase in packet loss was found when the transmission rate was 1431B/s which was thought to be caused by coincident events occurring between the operating system, VCP driver and the application. Both of these issues require further investigation and solutions implemented to solve them.

References

Adeneo Embedded (2007), 'The GPIO Driver',
`http://read.pudn.com/downloads169/sourcecode/windows/777040/Documentation/AT91SAM9260EKCE6.0TDDv1.1.0A.pdf` p. 47. [25 April 2012]. 52

ARM holdings (2010*a*), 'ARM Company Milestones',
`http://www.arm.com/about/company-profile/milestones.php` .
[28 September 2011]. 22

ARM holdings (2010*b*), 'ARM Company Profile',
`http://www.arm.com/about/company-profile/index.php` .
[28 September 2011]. 23

ARM holdings (2012), 'ARM Processor Selector',
`http://www.arm.com/products/processors/selector.php` .
[22 February 2012]. 23

Atmel Corporation (2007), 'AT91SAM9260-EK evaluation kit schematic',
`http://www.atmel.com/Images/doc6234.pdf` p. 5. [25 April 2012]. 84, 87

Atmel Corporation (2009*a*), '32-bit SDRAM', *Atmel AT91SAM9260 Datasheet* p. 152. 86

Atmel Corporation (2009*b*), '8-bit NAND Flash', *AT91SAM9260 Datasheet* p. 153. 86

Atmel Corporation (2009*c*), 'Ethernet MAC 10/100 (EMAC)', *Atmel AT91SAM9260 Datasheet* p. 581. 88

Atmel Corporation (2009*d*), '*Atmel AT91SAM9260 Datasheet*', http://www.atmel.com/dyn/resources/prod_documents/doc6221.pdf . [1 November 2011]. 84, 96

Atmel Corporation (2009*e*), 'Unavailable Signals in 208-lead PQFP Package', *AT91SAM9260 Datasheet* p. 2. 99

Atmel Corporation (2010*a*), 'SAM-BA In-system Programmer', http://www.atmel.com/dyn/products/tools_devices.asp?category_id=163&family_id=605&subfamily_id=718&tool_id=3883 . [1 November 2011]. 39

Atmel Corporation (2010*b*), 'Windows Embedded CE BSP', http://www.at91.com/windows4sam/bin/view/Windows4SAM/WindowsEmbeddedCEBSP . [1 November 2011]. 36

Atmel Corporation (2010*c*), 'Windows4SAM', http://www.at91.com/windows4sam/bin/view/Windows4SAM/ . [27 March 2012]. 33

Atmel Corporation (2011), 'AT45DB642D Datasheet', http://atmel.com/dyn/resources/prod_documents/doc3542.pdf . [4 November 2011]. 87

Cirimele, R. (2004), 'BGA Rework A comparitive study of selective solder paste deposition for Area Array Packages', http://www.solder.net/stencilquik/papers/SMT_PANPAC_Paper_Rev032704.pdf . [28 September 2011]. 27

Crossbow Technologies (2009), 'Discontinued Products', http://bullseye.xbow.com:81/Products/productdetails.aspx?sid=229 . [14 February 2012]. 8

Crossbow Technologies (2010*a*), 'Crossbow - eKo Pro Series system', http://www.enviromon.co.za/eKo/eKo_Pro_datasheet.pdf . [14 February 2012]. 10

Crossbow Technologies (2010*b*), 'Crossbow - Stargate Gateway SPB400', `http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/Stargate_Datasheet.pdf` . [14 February 2012]. 7

Crossbow Technologies (2011), 'Gumstix - Overo Network pack', `https://www.gumstix.com/store/product_info.php?products_id=251` . [14 February 2012]. 9, 12

Crossbow Technologies (2012), 'Imote2 High performace wireless sensor node', `http://bullseye.xbow.com:81/Products/Product_pdf_files/Wireless_pdf/Imote2_Datasheet.pdf` . [5 March 2012]. 1

Davicom Semiconductor Incorporated (2009*a*), 'DM9161B Datasheet', `http://www.meworks.net/userfile/24247/DM9161B-DS-F01_101609.pdf` . [25 April 2012]. 88

Davicom Semiconductor Incorporated (2009*b*), 'Magnetic selection guide', *DM9161B-DS-F01_101609.pdf* p. 43. [25 April 2012]. 89

FTDI (2010*a*), '2232D Setup Utility', `http://www.ftdichip.com/Support/Utilities/FT_Prog_v2.4.2.zip` . [7 November 2011]. 89

FTDI (2010*b*), 'FT2232D Datasheet', `http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT2232D.pdf` . [7 November 2011]. 89

FTDI (2010*c*), 'FT2232D Datasheet', *DS_FT2232D.pdf* p. 24. [7 November 2011]. 90

FTDI (2010*d*), 'FTP Server Registry Settings', `http://msdn.microsoft.com/en-us/library/ee498908%28v=winembedded.60%29.aspx` . [1 November 2011]. 42

FTDI (2010*e*), 'Installation Guides', http://www.ftdichip.com/Support/Documents/InstallGuides/Windows_CE_Installation_Guide.pdf . [1 November 2011]. 48

FTDI (2010*f*), 'VCP Drivers', http://www.ftdichip.com/Drivers/VCP.htm . [1 November 2011]. 48

Hasler, A., Talzi, I., Tschudin, C. and Gruber, S. (2008), Wireless sensor networks in permafrost research - concept, requirements, implementation and challenges, *in* 'Proc. 9th Intl Conf. on Permafrost (NICOP 2008)', Vol. 1, pp. 669–674. 2

Keller, M. and Beutel, J. (2009), 'Technology for permasense', http://www.tik.ee.ethz.ch/~kellmatt/pub/publications/20090512_permasense_seminar_technology.pdf . [14 February 2012]. 8, 12

KEMET (2011), 'Commercial Ceramic Chips X7R Dielectric, KEM_C1002_X7R_SMD.pdf', http://www.kemet.com/kemet/web/homepage/kechome.nsf/weben/Ceramic%20Capacitors . [14 February 2012]. 95

Kollman, R. (2005), 'Constructing Your Power Supply - Layout Considerations', http://www.ti.com/lit/ml/slup230/slup230.pdf pp. 12–13. [03 May 2012]. 167, 168

Libelium Comunicaciones Distribuidas (2011*a*), 'Libelium: Meshlium Xtreme', http://www.libelium.com/products/meshlium . [28 September 2011]. 11

Libelium Comunicaciones Distribuidas (2011*b*), 'Meshlium Xtreme - 802.15.4/ZigBee Sensor Network Gateway', http://www.sensor-networks.org/index.php?page=1107407451 . [28 September 2011]. 11

Lorincz, K., Welsh, M., Marcillo, O., Johnson, J., Ruiz, M. and Lees, J. (2006), Deploying a wireless sensor network on an active volcano, *in* 'IEEE Internet Computing', pp. 18–25. 2, 6, 12

Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R. and Anderson, J. (2002), Wireless sensor networks for habitat monitoring, *in* 'WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications', ACM, New York, NY, USA, pp. 88–97. 2, 6, 12

Martinez, K., Ong, R. and Hart, J. (2004), Glacsweb: a sensor network for hostile environments, *in* 'The First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks', pp. 81–87. 2

McCarty, B. (1999), *Learning Debian GNU/Linux*, 1 edn, O'Reilly & Associates, Inc. Chapter 1, section 1.2.2. 22

Microchip Technology Incorporated (2007), 'MCP1725 datasheet', `http://ww1.microchip.com/downloads/en/DeviceDoc/22026b.pdf` . [14 November 2011]. 102, 170

Microchip Technology Incorporated (2008*a*), 'MCP1825 datasheet', `http://ww1.microchip.com/downloads/en/DeviceDoc/22056b.pdf` . [14 November 2011]. 96, 166

Microchip Technology Incorporated (2008*b*), 'Typical performace curves: Ground Current vs. Load Current.', *22056b.pdf* p. 12. [14 November 2011]. 166

Micron Technology Incorporated (2011), '48LC16M16A2 Datasheet', `http://www.micron.com/products/ProductDetails.html?product=products/dram/sdram/MT48LC16M16A2P-7E%20IT` . [4 November 2011]. 86

Microsoft Corporation (2002*a*), 'Bootloaders', `http://msdn.microsoft.com/en-us/library/ms836792.aspx#systemmemorymgmtwince_topic8` . [1 November 2011]. 34, 39

Microsoft Corporation (2002*b*), '.NET Compact Framework in Embedded Devices', `http://msdn.microsoft.com/en-us/library/ms836805.aspx#wincecompactfx_topic4` . [1 November 2011]. 43

Microsoft Corporation (2002*c*), '.NET Fundamentals', http://msdn.microsoft.com/en-us/library/ms836805.aspx#wincecompactfx_topic4 . [1 November 2011]. 44

Microsoft Corporation (2003*a*), 'An Introduction to P/Invoke and Marshaling on the Microsoft .NET Compact Framework', http://msdn.microsoft.com/en-us/library/aa446536.aspx . [1 November 2011]. 49

Microsoft Corporation (2003*b*), 'Marshalling Types During Platform Invoke (P/Invoke) on the Microsoft .NET Compact Framework', http://msdn.microsoft.com/en-us/library/aa446538.aspx . [1 November 2011]. 49

Microsoft Corporation (2004), 'Dial-up Networking', http://msdn.microsoft.com/en-us/library/ms897074.aspx . [1 November 2011]. 69

Microsoft Corporation (2008), 'Telnet Server Registry Settings', http://msdn.microsoft.com/en-us/library/aa927075.aspx . [1 November 2011]. 42

Microsoft Corporation (2009), 'Microsoft Delivers Rich User Experiences and Windows 7 Connectivity With New Windows Embedded CE Release', http://www.microsoft.com/windowsembedded/en-us/news/windows-embedded-ce-6-r3-release.aspx . [1 November 2011]. 34

Microsoft Corporation (2010*a*), 'Downloads You Need to Evaluate Windows Embedded CE 6.0', http://www.microsoft.com/windowsembedded/en-us/downloads/download-windows-embedded-ce6.aspx . [1 November 2011]. 35

Microsoft Corporation (2010*b*), 'Registry Fundamentals', http://msdn.microsoft.com/en-us/library/ee489817%28v=winembedded.60%29.aspx . [1 November 2011]. 46

Microsoft Corporation (2010*c*), 'Windows Internet Services (WinInet)', http://msdn.microsoft.com/en-us/library/ee492409%28v=WinEmbedded.60%29.aspx . [1 November 2011]. 50

Mouser Electronics (2012), http://za.mouser.com/ProductDetail/Atmel/AT91RM9200-EK/?qs=2mdvTlUeTfCOBeFg4SziwA%3d%3d . 15 February 2012]. 12

Musaloiu-Elefteri, R., Musaloiu-Elefteri, R. and Terzis, A. (2008), Gateway Design for Data Gathering Sensor Networks, *in* 'SECON', pp. 296–304. 7, 12

National Instruments Corporation (2010*a*), 'Ni wireless sensor network programmable gateway', http://www.ni.com/pdf/products/us/cat_NI9792.pdf . [26 April 2010]. 11

National Instruments Corporation (2010*b*), 'Wireless Sensor Network Devices', http://www.ni.com/wsn/whatis/ . [26 April 2010]. 11

National Instruments Corporation (2010*c*), 'Wireless Sensor Network Ethernet Gateway', http://www.ni.com/pdf/products/us/cat_wsn9791.pdf . [26 April 2010]. 11

National Semiconductor Corporation (2008), 'PHYTER Design & Layout Guide', http://www.national.com/an/AN/AN-1469.pdf p. 4. [7 November 2011]. 89

National Semiconductor Corporation (2011*a*), 'Boost function', http://www.national.com/profile/snip.cgi/openDS=LM2738 p. 8. [7 November 2011]. 94, 102

National Semiconductor Corporation (2011*b*), 'LM2738Y Datasheet', http://www.national.com/profile/snip.cgi/openDS=LM2738 . [7 November 2011]. 93, 102, 160

Panasonic Industrial Company (2010), 'Panasonic SMD Choke Coils', http://industrial.panasonic.com/www-data/pdf/AGM0000/AGM0000CE11.pdf . [14 November 2011]. 94, 161, 171

Pavlov, S. and Belevsky, P. (2008*a*), *Windows Embedded CE 6.0 Fundamentals*, Microsoft Press. p. 75. 22

Pavlov, S. and Belevsky, P. (2008*b*), *Windows Embedded CE 6.0 Fundamentals*, Microsoft Press. p. 5. 33

Pavlov, S. and Belevsky, P. (2008*c*), *Windows Embedded CE 6.0 Fundamentals*, Microsoft Press. p. 203. 42

Pfeil, C. (2007), 'BGA Breakout Challenges', `http://www.mentor.com/products/pcb-system-design/resources/bga-breakouts-routing/upload/bga-breakout-challenges.pdf` . [28 September 2011]. 27

Phung, S. (2009*a*), *Professional Microsoft Windows Embedded CE 6 .0*, Wiley Publishing, Inc. Chapter 2. 36

Phung, S. (2009*b*), *Professional Microsoft Windows Embedded CE 6 .0*, Wiley Publishing, Inc. Managed-Code applications, p. 202. 43

Polastre, J., Szewczyk, R. and Culler, D. (2005), Telos: enabling ultra-low power wireless research, *in* 'IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks', IEEE Press, Piscataway, NJ, USA, pp. 364–369. 4

Pulse Electronics (2010), 'J00-00xxNL Datasheet', `http://productfinder.pulseeng.com/products/datasheets/J414.pdf` . [7 November 2011]. 89

Raghavan, P., Lad, A. and Neelakandan, S. (2006), *Embedded Linux System Design and Development*, Auerbach Publications. p. 33. 22

Rebex (2011), 'FTP for .NET/.NET CF', `http://www.rebex.net/ftp.net/` . [1 November 2011]. 50

Rohm Semiconductor Ltd (2009*a*), 'DTC114E NPN digital transistor datasheet', `http://www.rohm.com/products/databook/tr/pdf/dtc114ee.pdf` . [14 November 2011]. 104

Rohm Semiconductor Ltd (2009*b*), 'DTC114E PNP digital transistor datasheet', `http://www.rohm.com/products/databook/tr/pdf/dta114ee.pdf` . [14 November 2011]. 104

Rohm Semiconductor Ltd (2011), 'Schottky barier diode datasheet', `http://www.rohm.com/products/databook/di/pdf/rsx201l-30.pdf` . [14 November 2011]. 95

Samsung Electronics (2008), 'K9F2G08U0B Datasheet', `http://www.szyuda88.com/uploadfile/1067/cfile/20105121786169.pdf` . [26 April 2012]. 86

Shanghai Jiao Tong University (2011), 'Course Project - Data collection with a wireless sensor network', `http://www1.cs.sjtu.edu.cn/~yzhu/courses/comnets_11fall/Course_project.htm` . [06 March 2012]. 5

ST Microelectronics (2010*a*), 'LD29150xxx datasheet', `http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00003403.pdf` . [14 November 2011]. 96, 162

ST Microelectronics (2010*b*), 'Typical characteristics: Quiescent current vs. output current (VI = 4.5 V)', *CD00003403.pdf* p. 12. [14 November 2011]. 166

Steenkamp, L., Kaplan, S. and Wilkinson, R. H. (2009), Wireless sensor network gateway, *in* 'IEEE Africon 2009', pp. 1–6. [26 April 2010]. 9, 12, 140

Stoianov, I., Nachman, L., Madden, S. and Tokmouline, T. (2007), PIPENET: A wireless sensor network for pipeline monitoring, *in* 'IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks', ACM, New York, NY, USA, pp. 264–273. 2, 7, 12

Telit Communications (2011*a*), 'GM862 Family Hardware User Guide', `http://www.telit.com/module/infopool/download.php?id=537` p. 41. [1 November 2011]. 170

Telit Communications (2011*b*), 'GM862-GPS Datasheet', `http://www.telit.com/module/infopool/download.php?id=165` . [1 November 2011]. 30, 101, 170

Test & Measurement World (2011), 'NI expands wireless-sensor network line', `http://www.tmworld.com/article/517519-Product_update.php` . [15 February 2012]. 12

Texas Instruments (2008), 'TXS01014E datasheet', `http://www.ti.com/lit/ds/sces651d/sces651d.pdf` . [14 November 2011]. 103

TinyOS (2010), 'Mission Statement', `http://www.tinyos.net/scoop/special/mission.html` . [26 April 2010]. 3

TS2 Technologie Satelitarne (2012), 'Eutelstat W3A Broadband Satellite in South Africa', `http://www.ts2.pl/en/Africa-W3A` . [14 February 2012]. 12

Tschudin, C., VonderMuehll, D. and Gruber, S. (2009), 'PermasenseII', `http://www.permasense.ch/projects/permasense-ii.html` . [06 March 2012]. 8

University of California (2010), 'Serial-to-Mote', `http://openwsn.berkeley.edu/wiki/OpenSerialToMote` . [1 November 2011]. 48

Willow Technologies Ltd (2012), 'TelosB mote platform', `http://www.willow.co.uk/TelosB_Datasheet.pdf` . [05 March 2012]. 4

Woo, A., Tong, T. and Culler, D. (2003), Taming the underlying challenges of reliable multihop routing in sensor networks, *in* 'SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems', ACM, New York, NY, USA, pp. 14–27. 3

Appendix A

Appendix A: Power supply calculations

Power supply component and thermal calculations

A.1 Gateway power supply

A.1.1 LM2738 inductor selection

The following equations found in the LM2738 datasheet (National Semiconductor Corporation, 2011*b*) were used to determine the value of the converter inductor L1:

$$V_{sw} = I_{out} \times R_{DSon} \tag{A.1}$$

where:
V_{sw} = internal switch voltage drop
I_{out} = output current
R_{DSon} = On-resistance of the internal switching MOS

$$D = \frac{V_o + V_d}{V_{in} + V_d - V_{sw}} \tag{A.2}$$

where:
D = duty cycle
V_o = output voltage
V_{in} = input voltage
V_d = catch diode voltage drop

$$I_{Lpk} = I_{out} + \Delta i_L \tag{A.3}$$

where:
$\Delta i_L = 0.1 \times I_{out}$

$$T_s = \frac{1}{f_s} \tag{A.4}$$

where:
f_s = switching frequency

The value of the inductor L may be found by

$$L = \left(\frac{DT_s}{2\Delta i_L}\right) \times (V_{in} - V_{out}) \tag{A.5}$$

The following design values were substituted in to the equations A.1, A.3 and A.4:

I_{out} = 1,081A - from table 4.1 with 50% safety margin added

R_{DSon} = 250mΩ - datasheet value f_s = 550kHz - switching frequency
V_o = 5V - output voltage

The following results were obtained from equations A.1, A.3 and A.4:
V_{sw} = 0,27V
I_{Lpk}= 1,189A
T_s = 1,818μs

It was necessary to investigate the influence of the input voltage range on the duty cycle of the converter. Two calculations were performed at the extremeties of the input voltage range by substituting the above results and the following design values into equation A.2:

V_{in1} = 8V
V_{in2} = 15V
V_d = 0,44V - catch diode forward drop

The duty cycle varies as follows:
$D_1 = 0,665$ if V_{in1} = 8V
$D_2 = 0,358$ if V_{in2} = 15V

Substituting these results into equation A.5 yields two inductor values suitable for each input voltage extremity:
L_{min} = 18,1μH if V_{in} = 8V
L_{max} = 32,5μH if V_{in} = 15V

A standard inductor value of 22μH was selected for the converter circuit. A Panasonic *ELLCTV220M* inductor was chosen as it has a rating of 2,7A which exceded the peak current of 1,189A, very low DC resistance and was available in a magnetically shielded SMD package (Panasonic Industrial Company, 2010).

A.1.2 LM2738 output voltage

The output voltage of the pre-regulator is set by the ratio of resistors R7 and R9, R_1 and R_2 respectively and is determined by:

$$R_1 = \left(\frac{V_o}{V_{ref}} - 1\right) \times R_2 \tag{A.6}$$

where:
V_o - output voltage
V_{ref} - reference voltage, set at 0,8V

R_2 = 10kΩ - datasheet recommendation

Substituting V_o = 5V into equation A.6 yields R_1 = 52,5kΩ.

52,5kΩ is a non-standard resistor value, however the *E96* resistor range includes a 52,3kΩ value which was selected for the design.

Recalculating the output voltage with the adjusted resistor value yields:
V_o = 4,98V.

A.1.3 LD29150 output voltage

The output voltage of the LD29150PTR (ST Microelectronics, 2010*a*) adjustable linear regulator is set by R10 and R65, R_1 and R_2 respectively and determined by:

$$R_1 = \frac{(V_o - V_{ref}) \times R_2}{V_{ref}} \tag{A.7}$$

where:
V_o - output voltage
V_{ref} - reference voltage, set at 1,23V
R_2 = 10kΩ - nominated value

Substituting V_o = 3,3V into equation A.7 yields R_1 = 16,829kΩ.

16,829kΩ is a non-standard resistor value, however the *E48* resistor range includes a 16,9kΩ value which was selected for the design.

Recalculating the output voltage with the adjusted resistor value yields:
V_o = 3,309V.

A.1.4 Regulator power dissipation

The following equations calculate the anticipated power to be dissipated by the three regulators in the gateway power supply.

A.1.4.1 LM2738 pre-regulator

The internal power $P_{internal}$ generated by the LM2738 switched mode converter may be calculated by:

$$P_{internal} = P_{cond} + P_q + P_{swf} + P_{swr} \tag{A.8}$$

where:

P_{cond} - conduction power loss in the NMOS switch

P_q - quiescent power loss

P_{swf} - switching loss falling

P_{swr} - switching loss rising

The terms of equation A.8 are calculated by the following equations:

$$P_{cond} = \left({I_{out}}^2 \times D\right)\left[1 + \frac{1}{3} \times \left(\frac{\Delta i_L}{I_{out}}\right)^2\right] R_{DSon} \tag{A.9}$$

where:

I_{out} - output current

D - duty cycle

R_{DSon} - NMOS on resistance

$\Delta i_L = 0.1 \times I_{out}$

$$P_q = I_q \times V_{in} \tag{A.10}$$

where:

I_q - quiescent operating current

V_{in} - input voltage

$$P_{swf} = \frac{1}{2}\left(V_{in} \times I_{out} \times f_s \times t_{fall}\right) \tag{A.11}$$

$$P_{swr} = \frac{1}{2}\left(V_{in} \times I_{out} \times f_s \times t_{rise}\right) \tag{A.12}$$

where:

f_s - switching frequency

t_{fall} - NMOS switch-off time

t_{rise} - NMOS switch-on time

The following design values were substituted into equation A.9 to calculate P_{cond} at the input supply voltage extremeties:

$D_1 = 0{,}665$ at $V_{in} = 8V$, from equation A.2

$D_2 = 0{,}358$ at $V_{in} = 15V$, from equation A.2

$I_{out} = 1{,}081$A - from table 4.1 with 50% safety margin added

$R_{DSon} = 250\text{m}\Omega$ - datasheet value
$\Delta i_L = 0{,}1081\text{mA}$

Therefore,
$P_{cond1} = 194{,}27\text{mW}$ at $V_{in} = 8V$
$P_{cond2} = 104{,}59\text{mW}$ at $V_{in} = 15V$

The following design values were substituted into equation A.10 to calculate P_q at the input supply voltage extremeties:
$I_q = 1{,}9\text{mA}$ - datasheet value
$V_{in1} = 8V$
$V_{in2} = 15V$

Therefore,
$P_{q1} = 15{,}2\text{mW}$ at $V_{in} = 8V$
$P_{q2} = 28{,}5\text{mW}$ at $V_{in} = 15V$

The following design values were substituted into equation A.11 to calculate P_{swf} at the input supply voltage extremeties:
$f_s = 550\text{kHz}$ - datasheet value
$t_{fall} = 8\text{ns}$ - approximated datasheet value
$V_{in1} = 8V$
$V_{in2} = 15V$

Therefore,
$P_{swf1} = 19{,}02\text{mW}$ at $V_{in} = 8V$
$P_{swf2} = 35{,}67\text{mW}$ at $V_{in} = 15V$

The switching *rise* and *fall* times were assumed to be equal, ($t_{fall} = t_{rise}$) according to the datasheet. Therefore:
$P_{swr1} = P_{swf1} = 19{,}02\text{mW}$ at $V_{in} = 8V$
$P_{swr2} = P_{swf2} = 35{,}67\text{mW}$ at $V_{in} = 15V$

All of the terms required to calculate $P_{internal}$ at each input voltage extremity have been found.
From equation A.8, $P_{internal} = P_{cond} + P_q + P_{swf} + P_{swr}$

Therefore:
$P_{internal1} = 247{,}51\text{mW}$ at $V_{in} = 8V$
$P_{internal2} = 204{,}43\text{mW}$ at $V_{in} = 15V$

It has been shown that the most internal power is dissipated by the pre-regulator when the input voltage is at its lowest value.

A.1.4.2 Linear regulators

The following general equations calculate the power dissipation of a linear regulator:

$$P_{tot} = P_{ldo} + P_{I_{gnd}} \tag{A.13}$$

where:
P_{tot} - total power dissipated
P_{ldo} - power dissipated under load
$P_{I_{gnd}}$ - quiescent power under load

$$P_{ldo} = (V_{in} - V_{out}) \times I_{out} \tag{A.14}$$

where:
V_{in} - input voltage
V_{out} - output voltage
I_{out} - output current

$$P_{I_{gnd}} = V_{in} \times I_{V_{in}} \tag{A.15}$$

where:
$I_{V_{in}}$ - current drawn by the loaded regulator
V_{in} - input voltage

3,3V LDO regulator LD29150 power dissipation

The following design values were substituted into equation A.14 to calculate P_{ldo}:

$V_{in} = 5$V - pre-regulator output voltage
$V_{out} = 3{,}3$V
$I_{out} = 787{,}5$mA - from table 4.1 with 50% safety margin added

Therefore,

$P_{ldo} = 1{,}339$W

The following design values were substituted into equation A.15 to calculate $P_{I_{gnd}}$:

V_{in} = 5V - pre-regulator output voltage
$I_{V_{in}}$ = 18mA - datasheet value (ST Microelectronics, 2010*b*)

Therefore,
$P_{I_{gnd}} = 90\text{mW}$

All of the terms required to calculate P_{tot} by equation A.13 have been found, therefore, P_{tot} = 1,429W

1,8V LDO regulator MCP1825 power dissipation

The equation A.13 may be applied to calculate the power dissipation of the MCP1825 regulator (Microchip Technology Incorporated, 2008*a*).

The following design values were substituted into equation A.14 to calculate P_{ldo}:
V_{in} = 5V - pre-regulator output voltage
V_{out} = 1,8V
I_{out} = 256,83mA - from table 4.1 with 50% safety margin added

Therefore,
$P_{ldo} = 821{,}86\text{mW}$

The following design values were substituted into equation A.15 to calculate $P_{I_{gnd}}$:
V_{in} = 5V - pre-regulator output voltage
$I_{V_{in}}$ = 150μA - datasheet value (Microchip Technology Incorporated, 2008*b*)

Therefore,
$P_{I_{gnd}} = 750\mu\text{W}$

All of the terms required to calculate P_{tot} have been found by equation A.13, therefore, P_{tot} = 822,61mW

A.1.5 Groundplane heatsink calculations

Thermal calculations for the cooling of electronic components may be simplified by means of an electrical circuit analogy: a heat source is the equivalent of a constant current source, heat flow is the equivalent of current, temperature is the equivalent of voltage and conduction, convection and radiation paths the equivalent of resistance (Kollman, 2005).

Therefore, the cooling circuit may be represented by:

$$T_j = P_{diss} \times (R_{\theta_{JC}} + R_{\theta_{CS}} + R_{\theta_{SA}}) + T_{amb} \tag{A.16}$$

where:
T_j = IC junction temperature
P_{diss} = power dissipated by an IC
$R_{\theta_{JC}}$ = junction-case thermal resistance
$R_{\theta_{CS}}$ = case-sink thermal resistance
$R_{\theta_{SA}}$ = sink-ambient thermal resistance
T_{amb} = ambient temperature

The heat generated by a chip within an IC flows through a junction to the case which has a thermal resistance $R_{\theta_{JC}}$ normally specified by the IC manufacturer. $R_{\theta_{CS}}$ is the thermal resistance of the interface between the case and the heatsink which is negligible if the device is soldered directly to the heatsink or copper plane and may therefore be ignored in the following calculations. The heatsink is required to dissipate the heat into the surrounding air by radiation and conduction and has a thermal resistance $R_{\theta_{SA}}$ and is a critical factor in determining the overall effectiveness of the cooling circuit.

The copper groundplanes on the gateway PCB were used as the heatsink for the power supply regulators. Heat flows from the regulators into the plane where it is conducted laterally; cooling is provided by the heat being conducted and radiated into the surrounding air in contact with the surface area of the plane. The analysis of this problem becomes complex as the only known quantities are the power generated within the device P_{diss} and the junction-case thermal resistance, $R_{\theta_{JC}}$.

The following equations determine an *approximate* surface area of the heatsink plane required to keep the regulator junction at a safe temperature that will not cause damage to the components nor PCB.

Temperature rise ΔT is proportional to power dissipation P_{diss} and inversely proportional to surface area S_a. The proportionality constant h is the *heat transfer coefficient* and is approximately $0{,}001W/(cm^2.°C)$ for still air (Kollman, 2005).

Therefore temperature rise may be calculated by:

$$\Delta T = \frac{P_{diss}}{S_a \times h} \tag{A.17}$$

Substituting h into the equation A.17,

$$\Delta T = \frac{1000.P_{diss}}{S_a} \tag{A.18}$$

From equation A.16 it can be shown that

$$\Delta T_{R_{sa}} = P_{diss} \times R_{\theta_{SA}} \tag{A.19}$$

therefore if $\Delta T = \Delta T_{R_{sa}}$,

$$\frac{1000.P_{diss}}{S_a} = P_{diss} \times R_{\theta_{SA}} \tag{A.20}$$

then

$$R_{\theta_{SA}} = \frac{1000}{S_a} \tag{A.21}$$

The surface area required is found by substituting equation A.21 into equation A.16, removing $R_{\theta_{CS}}$ and changing the subject to S_a:

$$S_a = \frac{1000.P_{diss}}{T_j - T_{amb} - (P_{diss}.R_{\theta_{JC}})} \tag{A.22}$$

The thermal resistance of the copper plane affects the effectiveness of the heat transfer away from the regulators and is proportional to its length and inversely proportional to its cross-sectional area. Breaks and discontinuities in the plane caused by tracks surrounding the devices should be minimised to ensure maximum heat transfer as copper has a much higher thermal conductivity than the FR4 PCB substrate material.

In addition, thermal vias may be used to transfer the heat from one side of the PCB to the other as well as to planes within the PCB thus increasing the effective surface area of the heatsink.

The individual heatsink surface area required by each regulator is calculated by means of equation A.22 and the results summed to find the total approximate surface area required.

A conservative maximum junction temperature $T_j = 110°C$ and ambient temperature $T_{amb} = 60°C$ are used throughout the calculations. The P_{diss} value for each regulator has been calculated in section A.1.4.

The following design values were substituted into equation A.22 to calculate surface area S_{a_1} for the LM2738:

$R_{\theta_{JC}} = 30°C/W$ - datasheet value

$P_{diss} = 247,51mW$ - from section A.1.4

Therefore,

$S_{a_1} = 5,814cm^2$

The following design values were substituted into equation A.22 to calculate surface area S_{a_1} for the LD29150:

$R_{\theta_{JC}} = 8°C/W$ - datasheet value

$P_{diss} = 1,429W$ - from section A.1.4

Therefore,

$S_{a_2} = 37,051cm^2$

The following design values were substituted into equation A.22 to calculate surface area S_{a_1} for the MCP1825:

$R_{\theta_{JC}} = 3°C/W$ - datasheet value

$P_{diss} = 822,61mW$ - from section A.1.4

Therefore,

$S_{a_3} = 17,306cm^2$

The total required surface area is found by: $S_{a_{tot}} = S_{a_1} + S_{a_2} + S_{a_3}$
Therefore $S_{a_{tot}} = 60,171cm^2$ which was made available on the groundplanes during the design of the gateway PCB.

A.2 Communication module power supply

The power supply for the communication module was based on the same topology as the gateway power supply and consists of an LM2738 switched mode converter supplying power to a linear regulator. The switched mode pre-regulator was required to reduce the input voltage range of 8V to 15V DC to 3,8V as specified by the GSM module's datasheet (Telit Communications, 2011*b*). An MCP1725 adjustable LDO linear regulator (Microchip Technology Incorporated, 2007) to further reduce this voltage to 2,8V was included into the design to provide power to the level translators.

A.2.1 LM2738 inductor selection

The equations A.1, A.2, A.3, A.4, A.5 in section A.1.1 may be used to calculate the value of the inductor required for the communication module's pre-regulator.

The following were substituted into equations A.1, A.3 and A.4:

I_{out} = 500mA - average current required (Telit Communications, 2011*a*)
Δi_L = 50mA - 0.1 × I_{out}, datasheet definition
R_{DSon} = 250mΩ - NMOS on-resistance
f_s = 550kHz - switching frequency

The following results were obtained from equations A.1, A.3 and A.4:

V_{sw} = 0,125V
I_{Lpk}= 0,55A
T_s = 1,818μs

It was necessary to investigate the influence of the input voltage range on the duty cycle of the converter. Two calculations were performed at the extremeties of the input voltage range by substituting the above result and the following design values into equation A.2:

V_{sw} = 0,125V
V_o = 3,8V - output voltage
V_d = 0,44V - catch diode forward drop
V_{in1} = 8V
V_{in2} = 15V

The duty cycle varies as follows:

$D_1 = 0,201$ if V_{in1} = 8V
$D_2 = 0,277$ if V_{in2} = 15V

Substituting these results into equation A.5 yields two inductor values suitable for each input voltage extremity:

$L_{min} = 15{,}34\mu$H if $V_{in} = 8$V

$L_{max} = 56{,}4\mu$H if $V_{in} = 15$V

A standard inductor value of 33μH was selected for the converter circuit. A Panasonic *ELLCTV330M* inductor was chosen as it has a rating of 2,7A which exceded the peak current of 0,55A, very low DC resistance and was available in a magnetically shielded SMD package (Panasonic Industrial Company, 2010).

A.2.2 LM2738 output voltage

The output voltage of the LM2738 is adjustable and was determined by resistors R13 and R17, R_1 and R_2 respectively.

Substituting $V_o = 3{,}8$V and $R_2 = 10$kΩ, a recommended value, into equation A.6 yields $R_1 = 37{,}5$kΩ.

37,5kΩ is a non-standard resistor value, however the *E96* resistor range includes a 37,4kΩ value which was selected for the design.

Recalculating the output voltage with the adjusted resistor value yields:
$V_o = 3{,}792$V.

A.2.3 MCP1725 output voltage

The MCP1725 output voltage is determined by equation A.23 and is set to 2,8V:

$$R_1 = \left[\frac{V_o - V_{ref}}{V_{ref}}\right] \times R_2 \tag{A.23}$$

where:
V_o - output voltage
V_{ref} - reference voltage, set at 0,41V
R_2 - 10kΩ, datasheet recommendation

Substituting $V_o = 2{,}8$V and $R_2 = 10$kΩ into equation A.23 yielded
$R_1 = 58{,}29$kΩ.

58,29kΩ is a non-standard resistor value, however the *E48* resistor range includes a 59kΩ value which was selected for the design.

Recalculating the output voltage with the adjusted resistor value yields:
$V_o = 2{,}83\text{V}$.

A.2.4 Thermal considerations

The maximum power dissipated by the gateway pre-regulator was found to be 247,51mW when the input voltage was = 8V at a maximum load current of 1,081A. The area of heatsink calculated to maintain safe operating conditions for the gateway pre-regulator was found to be $5,814cm^2$.

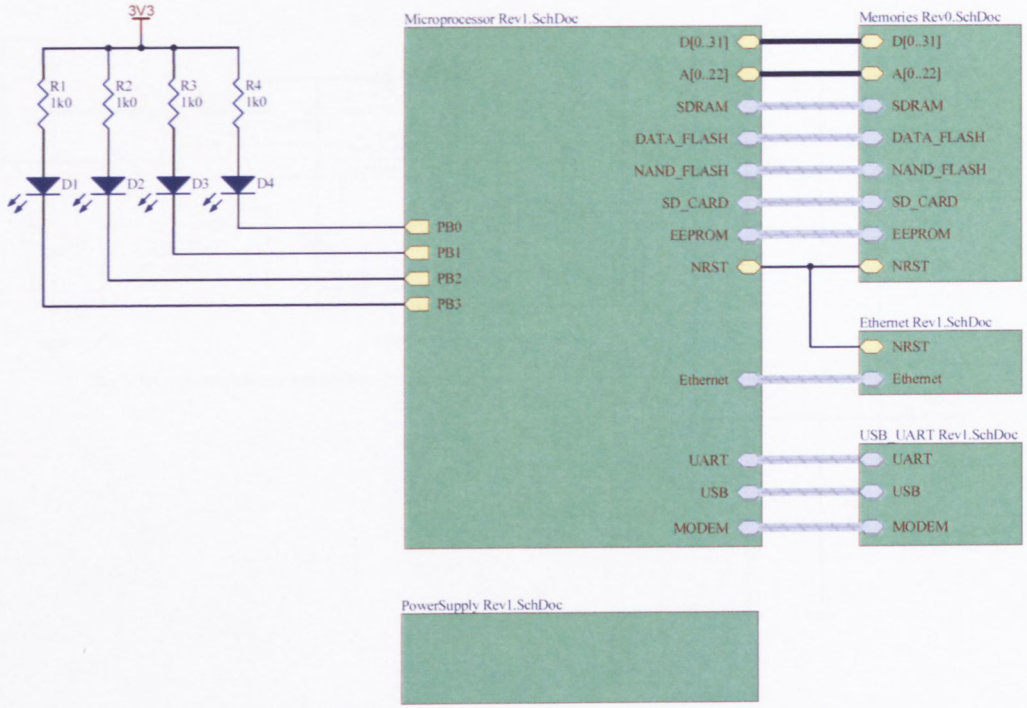
It was therefore reasoned that the area of the communication module's ground-plane, which totals approximately $39,7cm^2$, would act as an adequate heatskink for its pre-regulator.

The loading of the MCP1725 is minimal as the only components being driven are the level translators U1 and U2 and an indicator LED D6. It was reasoned that the power generated within the regulator would be easily dissipated by the component's package and through its legs into the surrounding groundplane.

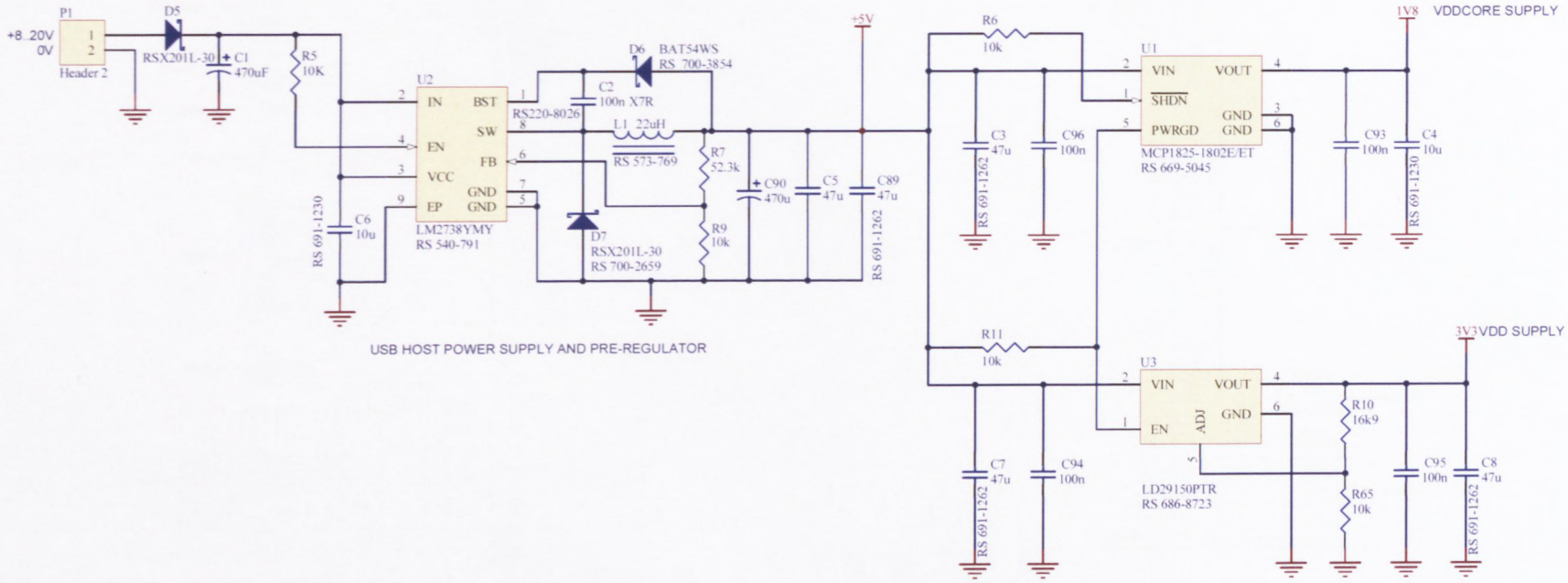
Appendix B

Appendix B: Schematic diagrams

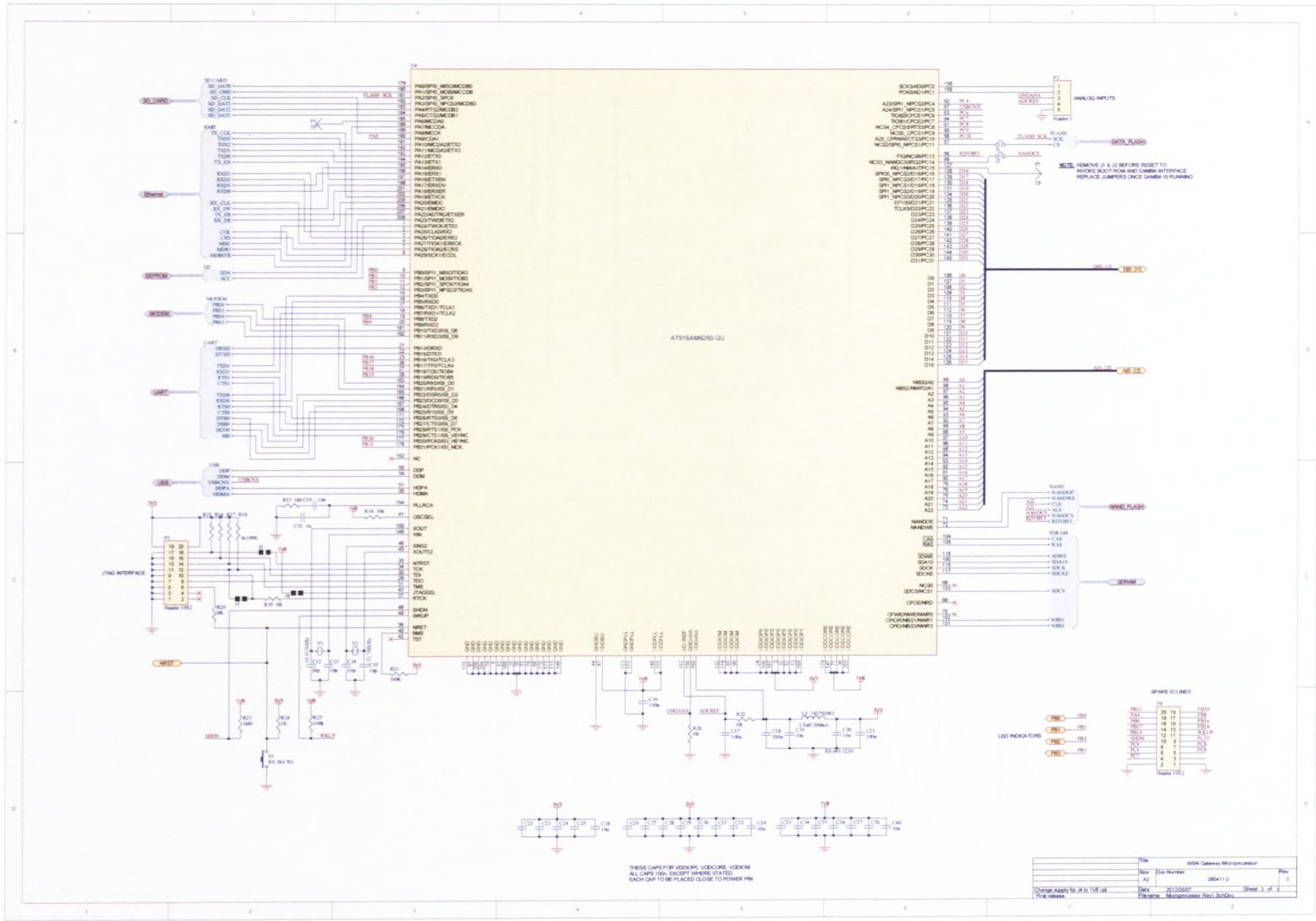
Schematic Circuit diagrams

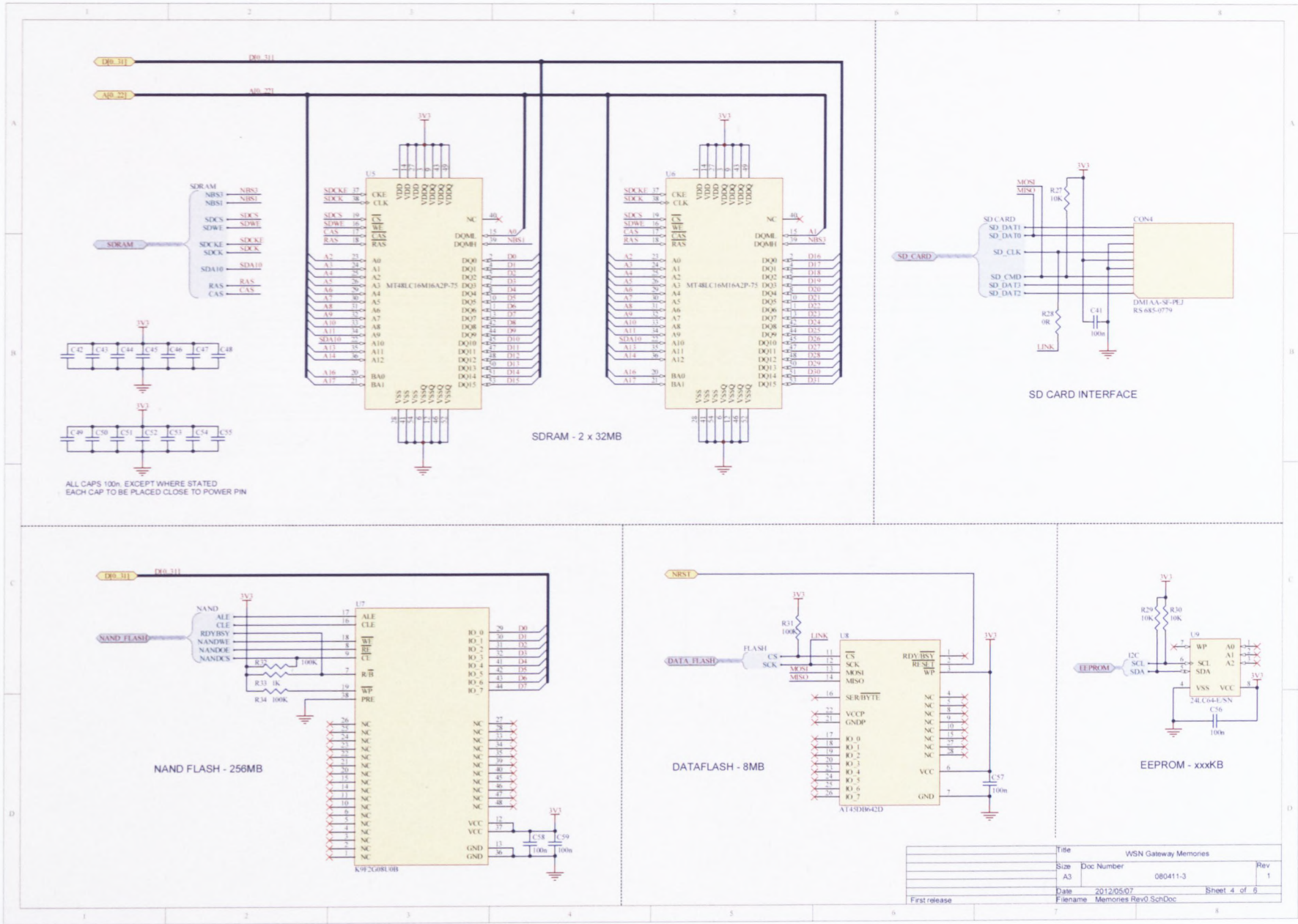


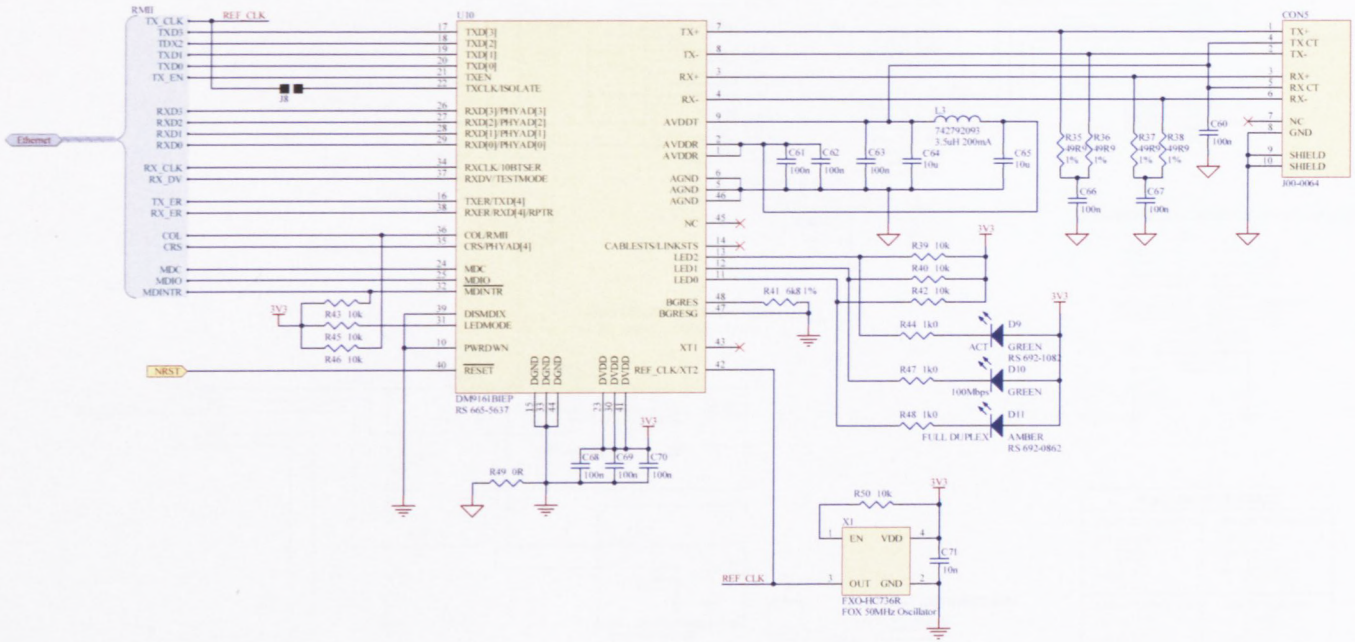
	Title	WSN Gateway Top Level Schematic	
	Size A4	Doc Number 080411-0	Rev 1
	Date 2012/05/07		Sheet 1 of 6
First release	Filename TopSheet Rev0.SchDoc		



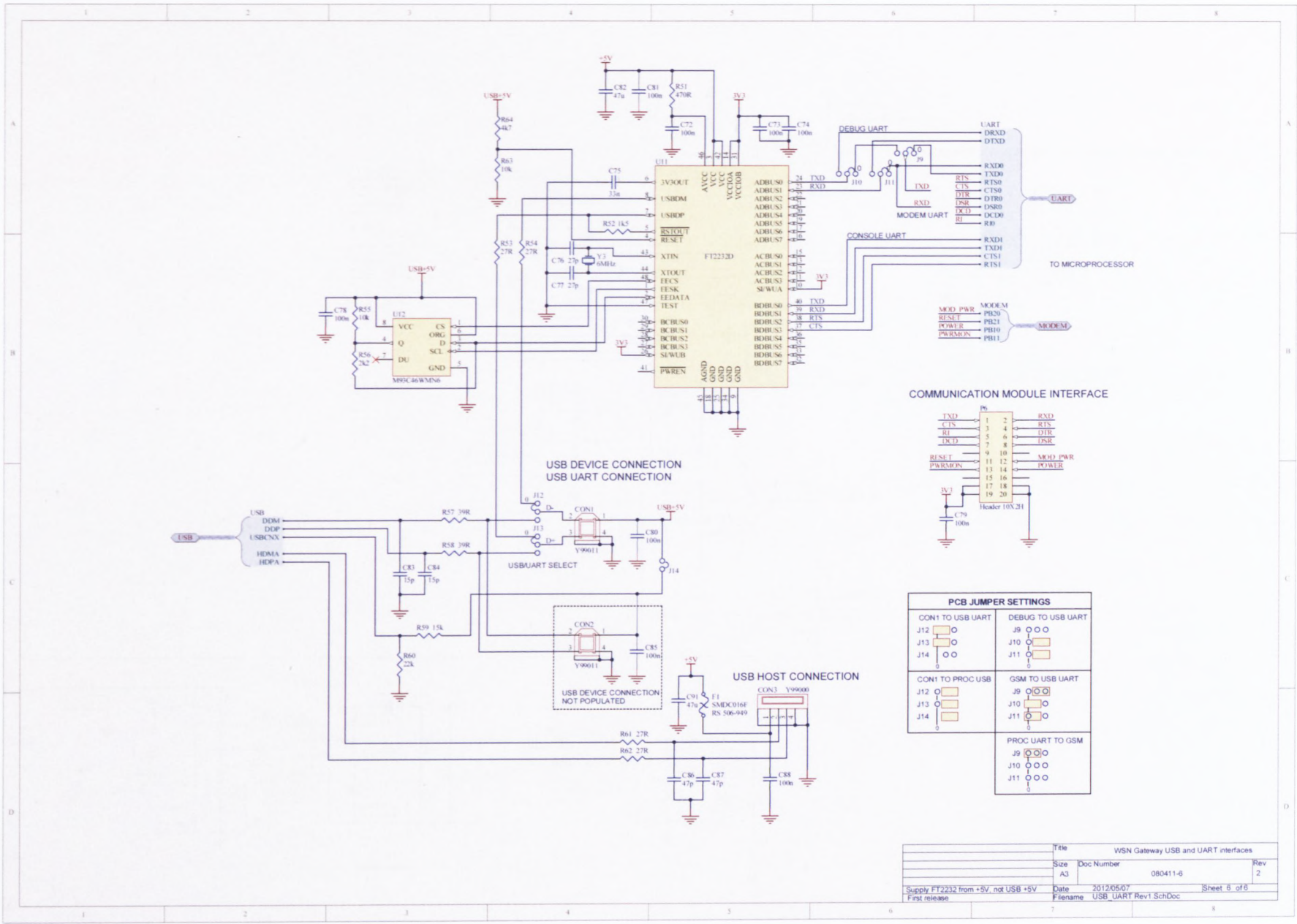
	Title	WSN Gateway Regulated Power Supply	
	Size	Doc Number	Rev
	A4	080411-1	2
Link PWRGD on U1 to EN on U3			
Change U3, add divider network	Date	2012/05/07	Sheet 2 of 6
First release	Filename	PowerSupply Rev1.SchDoc	

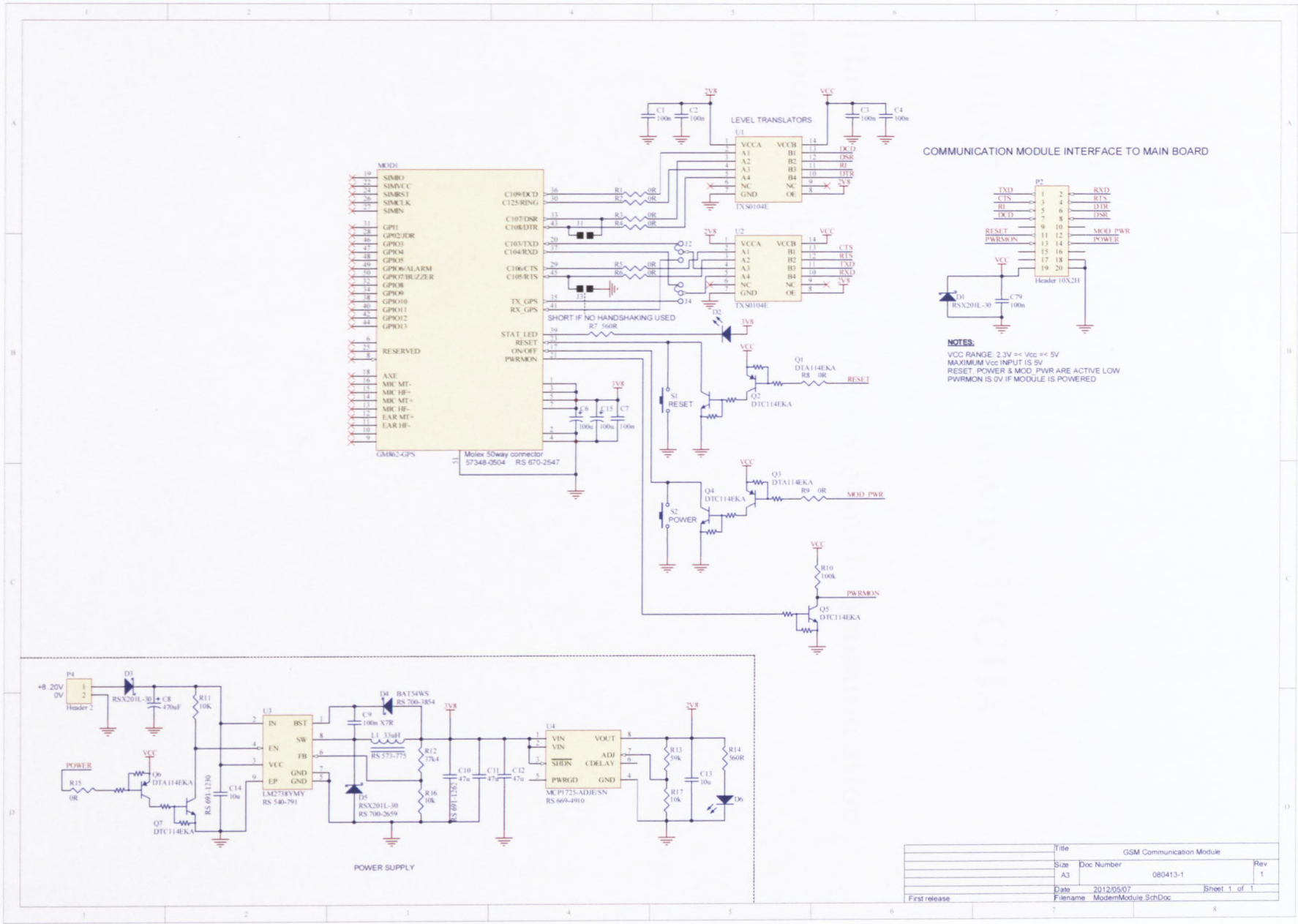

AT91SAM9260-QU
SD_CARD
Ethernet
EEPROM
MODEM
UART
USB
JTAG INTERFACE
NRST
FLASH SCK
ANALOG INPUTS
DATA_FLASH
NAND_FLASH
SDRAM
LED INDICATORS
SPARE IO LINES
NOTE: REMOVE J1 & J2 BEFORE RESET TO INVOKE BOOT ROM AND SAMBA INTERFACE REPLACE JUMPERS ONCE SAMBA IS RUNNING
THESE CAPS FOR VDDIOP0, VDDCORE, VDDIOM ALL CAPS 100n EXCEPT WHERE STATED EACH CAP TO BE PLACED CLOSE TO POWER PIN
WSN Gateway Microprocessor
080411-2
2012/06/07
Microprocessor Rev1.SchDoc
Change supply for J4 to 1V8 rail
First release


SDRAM - 2 x 32MB
ALL CAPS 100n, EXCEPT WHERE STATED
EACH CAP TO BE PLACED CLOSE TO POWER PIN
SD CARD INTERFACE
NAND FLASH - 256MB
DATAFLASH - 8MB
EEPROM - xxxKB
Title WSN Gateway Memories
Size A3
Doc Number 080411-3
Rev 1
Date 2012/05/07
Sheet 4 of 6
First release
Filename Memories Rev0.SchDoc



	Title	WSN Gateway UART and USB interface	
	Size A3	Doc Number 080411-5	Rev 2
Ground pin 47	Date	2012/05/07	Sheet 5 of 6
First release	Filename	Ethernet Rev1.SchDoc	


USB DEVICE CONNECTION
USB UART CONNECTION
USB HOST CONNECTION
USB DEVICE CONNECTION NOT POPULATED
USB/UART SELECT
COMMUNICATION MODULE INTERFACE
DEBUG UART
MODEM UART
CONSOLE UART
TO MICROPROCESSOR
PCB JUMPER SETTINGS
CON1 TO USB UART
DEBUG TO USB UART
CON1 TO PROC USB
GSM TO USB UART
PROC UART TO GSM
FT2232D
M93C46WMN6
Header 10X2H
Title WSN Gateway USB and UART interfaces
Size A3
Doc Number 080411-6
Rev 2
Date 2012/05/07
Sheet 6 of 6
Filename USB_UART Rev1.SchDoc
Supply FT2232 from +5V, not USB +5V
First release


LEVEL TRANSLATORS
U1
TXS0104E
U2
TXS0104E
MOD1
GM862-GPS
Molex 50way connector
57348-0504 RS 670-2547
SHORT IF NO HANDSHAKING USED
R7 560R
RESET
POWER
Q1 DTA114EKA
Q2 DTC114EKA
Q3 DTA114EKA
Q4 DTC114EKA
Q5 DTC114EKA
R8 0R
R9 0R
R10 100k
MOD_PWR
PWRMON
COMMUNICATION MODULE INTERFACE TO MAIN BOARD
P2
Header 10X2H
NOTES:
VCC RANGE: 2.3V =< Vcc =< 5V
MAXIMUM Vcc INPUT IS 5V
RESET, POWER & MOD_PWR ARE ACTIVE LOW
PWRMON IS 0V IF MODULE IS POWERED
POWER SUPPLY
U3
LM2734YMY
RS 540-791
U4
MCP1725-ADJE/SN
RS 669-4910
D4 BAT54WS
RS 700-3854
D5 RSX201L-30
RS 700-2659
L1 33uH
RS 573-775
Title GSM Communication Module
Size A3
Doc Number 080413-1
Rev 1
Date 2012/05/07
Sheet 1 of 1
First release
Filename ModemModule.SchDoc

Appendix C

Appendix C: Gateway PCBs

Photographs of the gateway and communication module PCBs

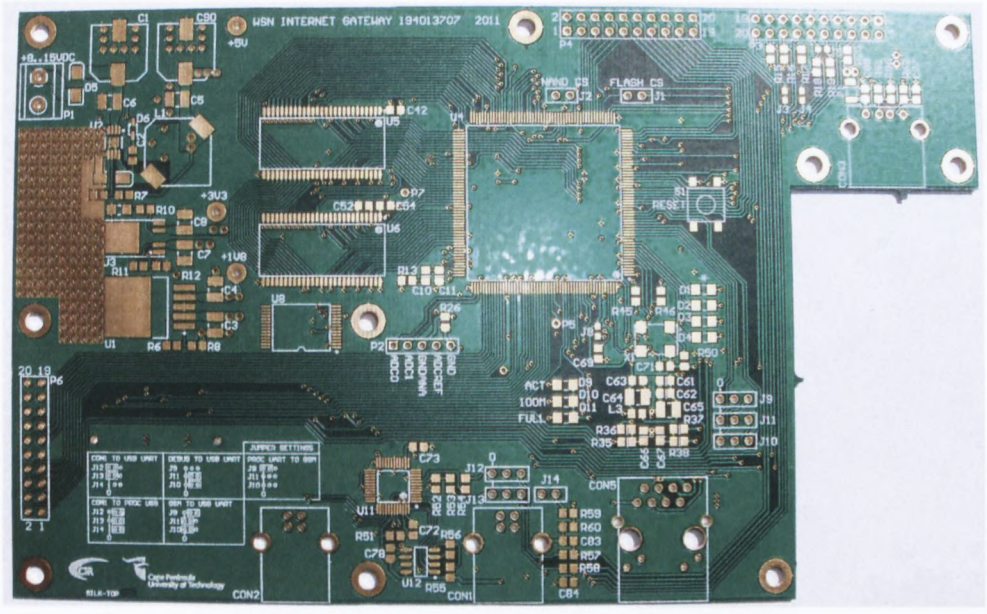


Figure C.1: Gateway PCB, top layer

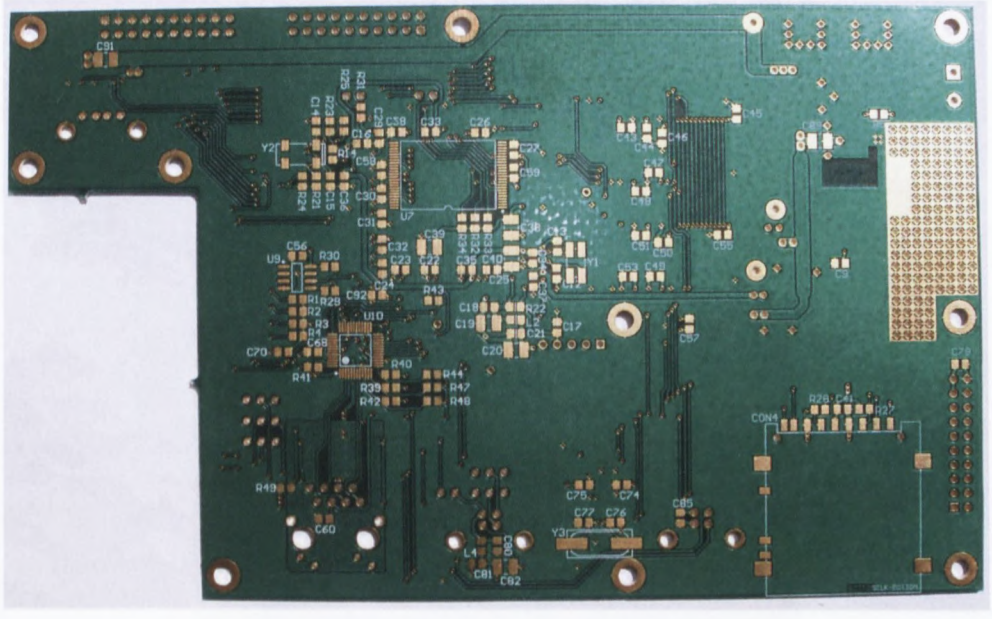


Figure C.2: Gateway PCB, bottom layer

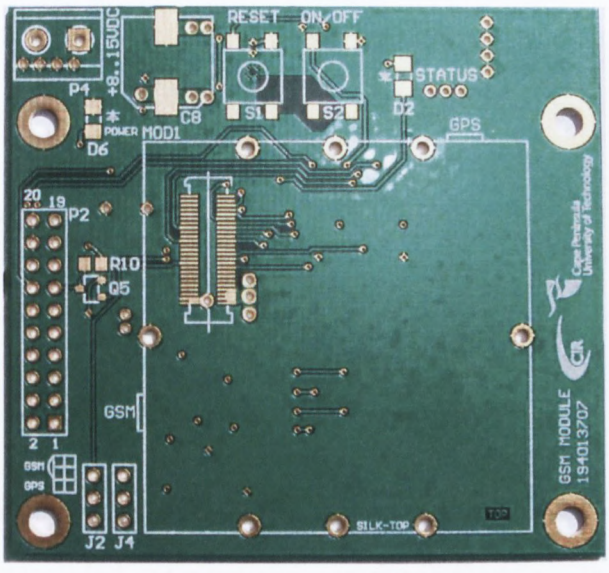


Figure C.3: Communication module, top layer

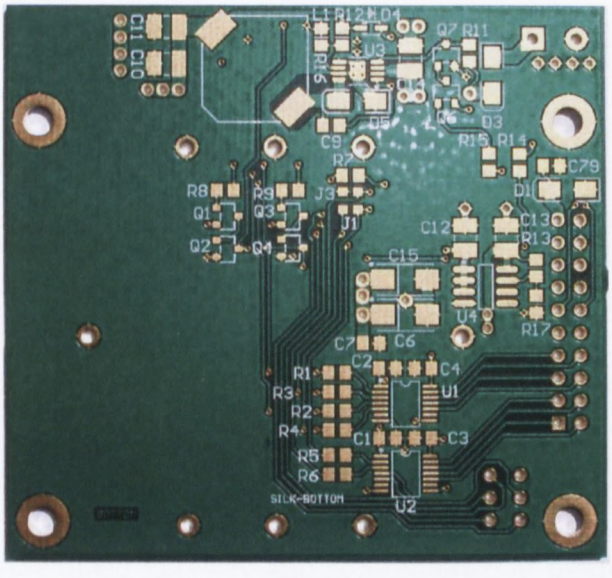


Figure C.4: Communication module, bottom layer

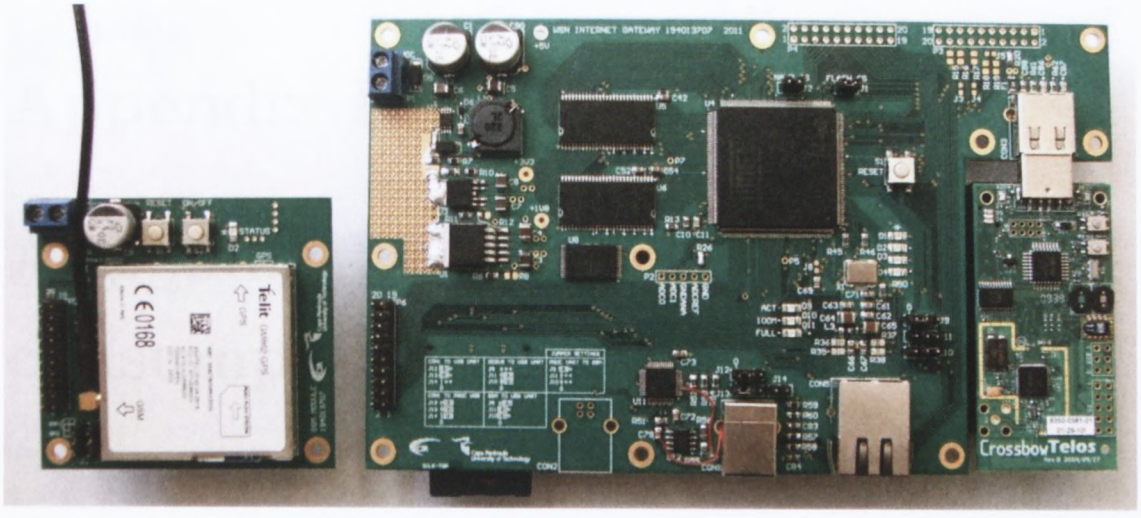


Figure C.5: The assembled gateway and communication module PCBs, top layer

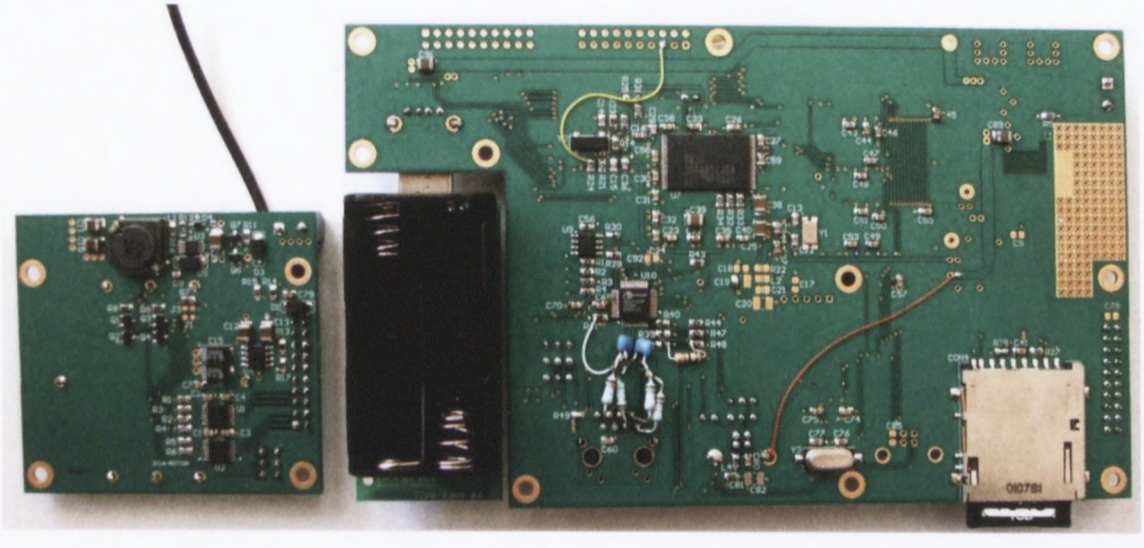


Figure C.6: The assembled gateway and communication module PCBs, bottom layer

Appendix D

Appendix D: Gateway power consumption results

Tabulated results of the gateway power consumption

Figures 5.2 and 5.3 in section 5.3.3 are derived from the data presented in tables D.1 and D.2.

The column, *System state*, indicates the operational mode of the gateway with corresponding test conditions listed in column *Condition*. The supply current measurements are listed in column I_{tot} (mA); the calculated current drawn by the Ethernet and USB serial interfaces and the wireless node is shown in column I_{calc} (mA); the total power consumed during each state is the product of the applied voltage and current drawn and is shown in column P_{tot} (W). Each state of operation is assigned an index number in column *Index* which is used as a cross reference in the results shown in section 5.3.3.

Figure D.1 shows the gateway and communication module being heated in a small oven to obtain current measurements whilst at an elevated temperature.

			Supply voltage: 8V			Supply voltage: 15V		
Index	**System state**	**Condition**	I_{tot}**(mA)**	I_{calc} **(mA)**	P_{tot} **(W)**	I_{tot}**(mA)**	I_{calc} **(mA)**	P_{tot} **(W)**
1	**Gateway held in reset**	USB & LAN cables, communication module disconnected Wireless node removed	97		0,776	57		0,855
2	**Insert USB cable**	USB converter current calculated	105	12	0,84	63	6	0,945
3	**EBOOT executing**		263		2,104	148		2,22
4	**Insert LAN cable**	LAN interface current calculated	326	63	2,608	180	32	2,7
5[a]	**Launch Windows CE**	Disconnect USB cable immediately	324		2,592	180		2,7
6	**Windows CE only**		248		1,984	140		2,1
7	**Windows CE without LAN**	LAN cable disconnected	186		1,488	106		1,59
8	**Insert wireless node**	Wireless node current calculated	223	37	1,784	125	19	1,875
9	**Launch gateway application**	Insert LAN cable to launch app	380		3,04	210		3,15
10	**Gateway application only**	LAN cable disconnected	315		2,52	178		2,67
11[a]	**GPRS modem initialising and network registration**		369		2,952	208		3,12
12	**GPRS modem in 'Listen' mode**		355		2,84	198		2,97
13[a]	**FTP session initialisation and file transfer**		428		3,424	227		3,405

Table D.1: Gateway power consumption at 21°C

[a]These operating states exhibit many current peaks which were averaged to generate a meaningful result.

			Supply voltage: 8V			Supply voltage: 15V		
Index	**System state**	**Condition**	I_{tot}**(mA)**	I_{calc} **(mA)**	P_{tot} **(W)**	I_{tot}**(mA)**	I_{calc} **(mA)**	P_{tot} **(W)**
1	**Gateway held in reset**	USB & LAN cables, communication module disconnected Wireless node removed	85		0,68	52		0,78
2	**Insert USB cable**	USB converter current calculated	98	8	0,784	59	7	0,885
3	**EBOOT executing**		268		2,144	149		2,35
4	**Insert LAN cable**	LAN interface current calculated	330	62	2,64	181	32	2,715
5[a]	**Launch Windows CE**	Disconnect USB cable immediately	331		2,648	182		2,73
6	**Windows CE only**		252		2,016	141		2,115
7	**Windows CE without LAN**	LAN cable disconnected	190		1,52	108		1,62
8	**Insert wireless node**	Wireless node current calculated	230	40	1,84	128	20	1,92
9	**Launch gateway application**	Insert LAN cable to launch app	390		3,12	208		3,12
10	**Gateway application only**	LAN cable disconnected	325		2,6	176		2,64
11[a]	**GPRS modem initialising and network registration**		371		2,968	203		3,045
12	**GPRS modem in 'Listen' mode**		361		2,888	198		2,97
13[a]	**FTP session initialisation and file transfer**		414		3,312	218		3,27

Table D.2: Gateway power consumption at 60°C

[a]These operating states exhibit many current peaks which were averaged to generate a meaningful result.

Figure D.1: Gateway and communication module were heated in a small oven to 60°C to verify that the power consumption remained within the 3,5W limit required by the specification.

Appendix E

Appendix E: Bill of Materials

Gateway and communication module bills of materials and component costings

Quantity	Value	PartType	Designator	Description	Supplier	Cost R (ex VAT)
Resistors						
1	15k	Res3	R 59	1% Resistor		0.19
1	1k5	Res3	R 52	1% Resistor		0.19
1	1M0	Res3	R 23	1% Resistor		0.19
1	22k	Res3	R 60	1% Resistor		0.19
1	2k2	Res3	R 56	1% Resistor		0.19
1	470R	Res3	R 51	1% Resistor		0.19
1	16k9	Res3	R 10	0.1% Resistor	RS 708-6499	3.31
1	6k8	Res3	R 41	1% Resistor		0.19
13	10k	Res3	R5, R6, R9, R11, R14, R27, R29, R30, R39, R40, R42, R43, R45, R46, R50, R55	1% Resistor		2.47
2	39R	Res3	R57, R58	1% Resistor		0.38
4	27R	Res3	R53, R54, R61, R62	1% Resistor		0.76
4	49R9	Res3	R35, R36, R37, R38	1% Resistor		0.76
6	0R	Res3	R19, R20, R22, R26, R28, R49	1% Resistor		1.14
8	1k0	Res3	R2, R3, R4, R13, R24, R33, R44, R47, R48	1% Resistor		1.52
9	100k	Res3	R15, R16, R17, R18, R21, R25, R31, R32, R34	1% Resistor		1.71
1	52k3	Res3	R7	0.1% Resistor	RS 708-6644	3.31
Capacitors						
2	470uF	SMD Electrolytic	C1, C90	Polarized Electrolytic Capacitor (Surface Mount)	RS 715-1708	7.42
1	1n	Cap Semi	C11	Capacitor (Semiconductor SIM Model)		0.18
1	33n	Cap Semi	C75	Capacitor (Semiconductor SIM Model)		0.18
2	10n	Cap Semi	C10, C71	Capacitor (Semiconductor SIM Model)		0.36
2	15p	Cap Semi	C83, C84	Capacitor (Semiconductor SIM Model)		0.36
2	27p	Cap Semi	C76, C77	Capacitor (Semiconductor SIM Model)		0.36
2	47p	Cap Semi	C86, C87	Capacitor (Semiconductor SIM Model)		0.36
4	10p	Cap Semi	C12, C13, C14, C15	Capacitor (Semiconductor SIM Model)		0.72
59	100n	Cap Semi	C2, C9, C16, C17, C18, C21, C22, C23, C24, C25, C26, C27, C28, C29, C30, C31, C32, C33, C34, C35, C36, C37, C41, C42, C43, C44, C45, C46, C47, C48, C49, C50, C51, C52, C53, C54, C55, C56, C57, C58, C59, C60, C61, C62, C63, C66, C67, C68, C69, C70, C72, C73, C74, C78, C79, C80, C81, C85, C88	Capacitor (Semiconductor SIM Model)	RS220-8026	31.27
6	47u	Cap Semi	C3, C5, C7, C8, C82, C89	X5R Ceramic multilayer cap	RS 691-1262	67.02
9	10u	Cap Semi	C4, C6, C19, C20, C38, C39, C40, C64, C65	X7R Ceramic multilayer cap	RS 691-1230	51.57
ICs						
2		MT48LC16M16A2P-75	U5, U6	256Mb CMOS DRAM, 16Mb x 16Bit, 54-Pin TSOP	AVNET	50.60
1		MCP1825-1802E/ET	U1	500 mA, LDO Regulator, 5-Pin DDPAK	RS 669-5045	8.69
1		M93C46WMN6	U12	1Kb (x8/x16) Serial Microwire Bus EEPROM	RS 686-2413	1.11
1		LM2738YMY	U2	550kHz/1.6MHz 1.5A DC-DC Switching Regulator	RS 540-791	29.85
1		LD29150P2T33R	U3	3V3 1.5A Fixed positive regulator	RS 686-9189	10.71
1		K9F2G08U0B	U7	Samsung 256MB NAND Flash	EBV	233.26
1		FT2232D	U11	Dual USB UART/FIFO Chip, LQFP-48	RS 406-578	52.99
1		DM9161BIEP	U10	Davicom 10/100Mbps Fast Ethernet transceiver	RS 665-5637	51.28
1		AT91SAM9260-QU	U4	Atmel ARM9 microprocessor	Arrow Altech	152.82
1		AT45DB642D	U8	Atmel 8MB DataFlash	RS 696-3064	72.33
1		24LC64-E/SN	U9	64Kbit, 400kHz, 2.5V, I2C Serial EEPROM, 8-Pin SOIC	RS 454-084	5.06

Quantity	Value	PartType	Designator	Description	Supplier	Cost (ex VAT)
Diodes						
1		BAT54WS	D6	Schottky Diode	RS 700-3854	0.56
8		LTST-C21TGKT	D1, D2, D3, D4, D8, D9, D10, D11	Amber LED, 1206 SMD, Green LED, 1206 SMD	RS 692-0862	7.28
2		RSX201L-30	D5, D7	Schottky Diode	RS 700-2659	1.74
Inductors						
3	3.5uH	742792093	L2, L3, L4	Inductor 200mA	RS 358-6557	1.92
1	22uH	Inductor Iron	L1	Magnetic-Core Inductor	RS 573-769	13.49
Crystals						
1	32.768kHz	FSRLF327	Y2	32.768kHz Crystal Oscillator	RS 547-6862	4.21
1	18.432MHz	E5SB18.4320F16E33	Y1	18.432MHz Crystal Oscillator	RS 675-4839	7.28
1	6MHz	LFXTAL026900	Y3	6MHz Crystal Oscillator	RS 478-9820	2.29
1		FXO-HC736R	X1	50MHz Oscillator	RS 672-0882	18.16
Jumpers						
5		JUMPER_3	J9, J10, J11, J12, J13	3 way jumper selector		1.45
4		JUMPER_2_SMD	J3, J4, J5, J8	2 way SMD solder jumper		0.50
3		JUMPER_2	J1, J2, J14	2 way jumper selector		0.50
Headers						
1		Header 10X2H	P6	Header, 10-Pin, Dual row, Right Angle		9.99
2		Header 4	P2, P5	Header, 4-Pin		3.38
1		Header 2	P1	Header, 2-Pin		1.69
Connectors						
2		Y99011	CON1, CON2	USB B Right angle socket	Mantech Y99011	4.50
1		Y99000	CON3	USB A Right angle socket	Mantech 72M2542	4.50
1		J00-0064	CON5	RJ45 Ethernet Connector with magnetics		
1		DM1AA-SF-PEJ	CON4	SD card socket SMD standard orientation	RS 685-0779	21.21
Miscellaneous						
1				4 layer PCB: setup, flying probe test	Trax PCB	667.33
1		B3S-1000	S1	SMD Pushbutton switch	RS 183-701	5.53
1		SMDC016F	F1	Thermal Fuse	RS 506-949	3.58
					TOTAL	1622.97

Figure E.1: Gateway bill of materials and component costings

Quantity	Value	PartType	Designator	Description	Supplier	Cost (ex VAT)
Resistors						
1	100k	Res3	R10	1% Resistor		0.19
2	560R	Res3	R14	1% Resistor		0.38
5	10k	Res3	R11, R12, R13, R16, R17	1% Resistor		0.95
9	0R	Res3	R1, R2, R3, R4, R5, R6, R8, R9, R15	1% Resistor		1.71
1	37k4	Res3	R7	0.1% Resistor	RS 708-6597	3.31
1	59k	Res3	R13	0.1% Resistor	RS 708-6663	3.31
Capacitors						
2	10u	Cap Semi	C13, C14	X7R Ceramic multilayer cap	RS 691-1230	5.10
3	47u	Cap Semi	C10, C11, C12	X5R Ceramic multilayer cap	RS 691-1262	10.02
7	100n	Cap Semi	C1, C2, C3, C4, C5, C7, C9	Capacitor (Semiconductor SIM Model)	RS 220-8026	0.86
1	470uF	SMD Electrolytic	C8	Polarized Electrolytic Capacitor (Surface Mount)	RS 715-1708	3.71
2	100u	SMD Tantalum	C6, C15	Polarized Capacitor (Surface Mount), TR3C107K6R3C0150 Tantalum SMD capacitor	RS 684-5149	2.90
ICs						
1		LM2738YMY	U3	550kHz/1.6MHz 1.5A Step-Down DC-DC Switching Regulator	RS 540-791	29.85
1		MCP1725-ADJE/SN	U4	500 mA, Low Voltage, Low Quiescent Current LDO Regulator, 8-Pin SOIC 150mil	RS 669-4910	5.42
2		TXS0104E	U1, U2	Bidirectional level shifter	RS 709-8745	28.96
Transistors						
3		DTA114EKA	Q1, Q3, Q6	PNP Digital transistor	RS 246-1687	1.59
4		DTC114EKA	Q2, Q4, Q5, Q7	NPN Digital transistor	RS 246-1693	2.12
Diodes						
1		BAT54WS	D4	Schottky Diode	RS 700-3854	0.56
2		LTST-C21TGKT	D2, D6	Green LED, 1206 SMD	RS 692-1082	1.82
3		RSX201L-30	D1, D3, D5	Schottky Diode	RS 700-2659	1.74
Inductors						
1	22uH	Inductor Iron	L1	Magnetic-Core Inductor	RS 573-769	13.49
Jumpers						
2		JUMPER_2	J1, J3	2 way SMD solder jumper		
2		JUMPER_3	J2, J4	3 way jumper selector		1.45
Headers						
1		Header 2	P4	Header, 2-Pin		1.69
1		Header 10X2H	P2	Header, 10-Pin, Dual row		9.99
Miscellaneous						
2		B3S-1000	S1, S2	SMD Pushbutton switch	RS 183-701	5.53
1		GM862-GPS	MOD1	Telit GM862 GSM module	Brabek, ref: Brian	577.00
1		52991-0508		Molex 50way male connector	RS 670-2538	18.37
2				Antenna SMA to SMB	Brabek, ref: Brian	80.00
1				4 layer PCB: setup, flying probe test	Trax PCB	333.71
					TOTAL	1145.73

Figure E.2: Communication module bill of materials and component costings

Appendix F

Appendix F: Application source code

Prototype gateway application source code

F.1 GatewayEngine.cs
F.2 Win32.cs
F.3 UART.cs
F.4 ConsoleUserInterface.cs
F.5 GSM.cs
F.6 Logging.cs
F.7 LoggingThread.cs
F.8 FTPThread.cs
F.9 GPIO.cs
F.10 SocketStream.cs
F.12 Main.cs
F.12 Modem.cs
F.13 IO_Control.c++
F.14 PACKETTXApp.nesC
F.15 PACKETTXC.nesC

```
//FILENAME: GatewayEngine.cs
//V1.04     22-04-2012  Reword main menu text
//          10-03-2012  Added nogprs, reset, eth reset, kill command line switches
//                      eth - ethernet IO console
//                      nogprs - stops GSM server starting automatically
//                      reset - defaults all registry values
//                      eth reset - defaults all registry values, ethernet IO console
//                      kill - stops all iterations of GatewayApp running
//V1.03     01-03-2012  Added remote TCP/IP interface functionality
//                      Added get local IP address function to Win32 class
//                      Added data streaming over TCP port - outgoing only
//                      Added stop ftp functionality to user menu - breaks FTP foreach loop andcloses internet
    connection after file send
//          27-02-2012  Added logger registry key Packet size - adjustable packet size from sink node
//V1.02     23-02-2012  Added GZ compression to data logger.  FTP zip files instead
//V1.01     05-2011     Initial release
using System;
using System.IO;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using System.Runtime.InteropServices;
using System.Diagnostics;
using Microsoft.Win32;          //For accessing the registry

namespace GatewayApp
{
    public class GatewayEngine
    {
        DataRecorder dataLogger = new DataRecorder();

        ConsoleUserInterface.consoleInput userConsole;

        //IPAddress localIPaddress = IPAddress.Parse("127.0.0.1");

        private const string version = "1.04";

        public void mainEngine(string[] controlArgs)
        {
            bool exitLoop;
            bool startGPRSserver = false;

            if (controlArgs.Length != 0)   //An argument is passed to GatewayApp
            {
                if (controlArgs[0] == "eth") userConsole = ConsoleUserInterface.consoleInput.Ethernet;
                else if (controlArgs[0] == "nogprs") userConsole = ConsoleUserInterface.consoleInput.UART;
                else if (controlArgs[0] == "reset") //Reset all registry keys to default values.  Create new keys if
     nec
                {
                    userConsole = ConsoleUserInterface.consoleInput.UART;
                    Registry.SetValue(@"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp", "Keys exist", 0);
                }
                else if (controlArgs[0] == "eth reset")
                {
                    userConsole = ConsoleUserInterface.consoleInput.Ethernet;
                    Registry.SetValue(@"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp", "Keys exist", 0);
                }
                else if (controlArgs[0] == "kill")
                {
                    Win32.killApps();
                    return;
                }
                else //No parameter, default to UART console and start GPRS server
                {
                    startGPRSserver = true;
                    userConsole = ConsoleUserInterface.consoleInput.UART;
                    //return;
                }
            }
            else //No parameter, default to UART console and start GPRS server
            {
                startGPRSserver = true;
                userConsole = ConsoleUserInterface.consoleInput.UART;
                //return;
            }

            try
            {
                ConsoleUserInterface.softwareVersion = version;
                ConsoleUserInterface.selectConsoleInput(userConsole);   //Opens UART if nec
                ConsoleUserInterface.printToConsole("\n\rGatewayApp.exe started\n\r");

                GPIO.setGPIO("B", 2, true);     //Turn off LED2 on gateway PCB - processor defaults it ON

                int FTDIlatency=Convert.ToInt32(Registry.GetValue(@"HKEY_LOCAL_MACHINE\Drivers\USB\ClientDrivers\
    FTDI_DEVICE", "LatencyTimer", 2));
                ConsoleUserInterface.printToConsole(String.Format("FTDI latency = {0}\n\r",FTDIlatency));
                if (FTDIlatency != 16)
                {
                    Registry.SetValue(@"HKEY_LOCAL_MACHINE\Drivers\USB\ClientDrivers\FTDI_DEVICE", "LatencyTimer",
    16);
                }

                ConsoleUserInterface.startCommsThread();
                Win32.createRegistryKeys();
                Win32.startTimerThread();
                Win32.openSDcard();

                //Start GSM only for UART console interface
                if (userConsole != ConsoleUserInterface.consoleInput.Ethernet)
                {
                    //Start GPRS server by default if user does not pass gprs to command line
                    if (startGPRSserver == true)
                    {
                        if (GSM.openModem() == true)
                        {
                            //If request to exit application does not exist, open GPRS server
```

```
                    if (Win32.exitGatewayApp == false)
                    {
                        //Only reset gateway if exitApplication request does not exit
                        if (GSM.initGPRSserver() == false && Win32.exitGatewayApp == false)
                        {
                            ConsoleUserInterface.printToConsole("System rebooting in 2s\n\r");
                            Thread.Sleep(2000);
                            Win32.systemReset();  //Hardware reset of system
                            Thread.Sleep(10000);
                        }
                    }
                }
            }
        }
        dataLogger.openMoteUART();
        ConsoleUserInterface.printHelpScreen();

        exitLoop = false;
        int counter = 20;
        while (exitLoop == false)
        {
            if (runningLoop() == false) exitLoop = true;    //Waits for exitApplication exit request
            else
            {
                counter--;
                if (counter == 0) exitLoop = true;
                else Thread.Sleep(5000); //Wait 5s before trying to start application again
            }
        }

        try
        {
            dataLogger.closeMoteUART();
            UART.CloseSerialPort(UART.MODEM_UART);
            ConsoleUserInterface.closeConsoleUART();

            if (userConsole == ConsoleUserInterface.consoleInput.Ethernet)
            {
                ConsoleUserInterface.printToConsole(String.Format("\rGatewayApp closed.  Press Enter key...\↙
n\r"));
            }
            try { UART.modemHandshakeThread.Abort(); }
            catch { }
            try { FTPThread.threadRunning = false; }
            catch { }
            try
            {
                LoggingThread.loggingThreadRunning = false;
                //LoggingThread.dataLoggingThread.Abort();
            }
            catch { }
            try { Win32.realTimeInterruptThread.Abort(); }
            catch { }
            try { ConsoleUserInterface.userInterface.Abort(); }
            catch { }
            return;
        }
        catch { }
        return;
    }
    catch(Exception msg)
    {
        ConsoleUserInterface.printToConsole(String.Format( "\rGatewayEngine failed. Exception: {0}",msg));
    }
}

/// <summary>
/// Main running loop
/// </summary>
/// <returns>true to attempt to start the loop again, false quits the application</returns>
private bool runningLoop()
{
    bool exitLoop=false;
    DataRecorder.LogError loggerError;

    try
    {
        //Wait here until application exit requested
        do
        {
            //Setup and start data logging
            loggerError = dataLogger.setupLogging();
            if (loggerError != DataRecorder.LogError.NoError && loggerError != DataRecorder.LogError.↙
LoggingAlreadyStarted)
                ConsoleUserInterface.printToConsole(String.Format("Cannot start Data Recorder: {0}",↙
loggerError.ToString()));

            //FTP files to remote server
            FTPThread.startFTPthread();

            if (ConsoleUserInterface.exitApplication == true || Win32.testForKillApps()==true)
            {
                exitLoop = true;
                continue;
            }

            GSM.modemConnectionManager();
            Thread.Sleep(0);
        }
        while (exitLoop == false);
        return false;
    }
    catch (Exception msg)
    {
        ConsoleUserInterface.printToConsole(msg.ToString());
        return true;
    }
```

```
//FILENAME: Win32.cs
using System;
using System.Text;
using System.Runtime.InteropServices;
using System.IO;
using System.IO.Ports;
using System.Threading;
using Microsoft.Win32;
using System.Net;
using System.Net.Sockets;

namespace GatewayApp
{
    /// <summary>
    /// Low level access procedures
    /// </summary>
    //public class Win32
    public static class Win32
    {
        #region Definitions
        /// <summary>
        /// Public System time structure
        /// </summary>
        public struct SYSTEMTIME
        {
            public ushort Year;
            public ushort Month;
            public ushort DOW;
            public ushort Day;
            public ushort Hour;
            public ushort Minute;
            public ushort Second;
            public ushort Millisecond;
        }

        public static Thread realTimeInterruptThread = new Thread(timerThread);
        private static bool _50ms_toggle_;
        private static bool timerThreadStarted;

        /// <summary>
        /// Name of application using internet connection
        /// </summary>
        public const string HTTP_USER = "GatewayApp";
        public const UInt32 HKEY_LOCAL_MACHINE = 0x80000002;
        private const UInt32 FLAG_ICC_FORCE_CONNECTION = 1;

        #region Registry Key definitions

        /// <summary>
        /// Reg key: HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\LOGGER
        /// </summary>
        public const string LoggerKey = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\LOGGER";

        /// <summary>
        /// Reg key: @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\GSM"
        /// </summary>
        public const string GSMkey = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\GSM";

        /// <summary>
        /// Reg key: @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\GSM\GPRS"
        /// </summary>
        public const string GPRSkey = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\GSM\GPRS";

        /// <summary>
        /// Reg key: @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\STREAM"
        /// </summary>
        public const string StreamKey = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\STREAM";

        /// <summary>
        /// @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\FTP"
        /// </summary>
        public const string FTPkey = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\FTP";

        /// <summary>
        /// Reg key: HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\FTP\GSM
        /// </summary>
        public const string FTPGSMkey = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\FTP\GSM";

        /// <summary>
        /// Reg key: HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\FTP\ETH
        /// </summary>
        public const string FTPETHkey = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\FTP\ETH";

        #endregion

        /// <summary>
        /// Internet and FTP constants found in WinInet.h
        /// </summary>
        private const UInt32 INTERNET_OPEN_TYPE_PRECONFIG                   = 0;   // use registry configuration
        private const UInt32 INTERNET_OPEN_TYPE_DIRECT                      = 1;   // direct to net
        private const UInt32 GATEWAY_INTERNET_ACCESS                        = 2;
        private const UInt32 INTERNET_OPEN_TYPE_PROXY                       = 3;   // via named proxy
        private const UInt32 INTERNET_OPEN_TYPE_PRECONFIG_WITH_NO_AUTOPROXY = 4;   // prevent using java/script/INS
        private const UInt32 INTERNET_FLAG_ASYNC                            = 0x10000000; // this request is
asynchronous (where supported)

        private const UInt32 INTERNET_INVALID_PORT_NUMBER   = 0;          // use the protocol-specific default

        private const UInt32 INTERNET_DEFAULT_FTP_PORT      = 21;         // default for FTP servers
        private const UInt32 INTERNET_DEFAULT_HTTP_PORT     = 80;         //    "     "  HTTP   "
        private const UInt32 INTERNET_DEFAULT_HTTPS_PORT    = 443;        //    "     "  HTTPS  "
        private const UInt32 INTERNET_DEFAULT_SOCKS_PORT    = 1080;       // default for SOCKS firewall servers.

        private const UInt32 INTERNET_SERVICE_FTP    = 1;
        private const UInt32 INTERNET_SERVICE_HTTP   = 3;

        private const UInt32 INTERNET_FLAG_PASSIVE = 0x08000000;          // used for FTP connections
```

```
    private const UInt32 INTERNET_FLAG_NO_CACHE_WRITE  =  0x04000000;  // don't write this item to the cache
    private const UInt32 INTERNET_FLAG_DONT_CACHE      =  INTERNET_FLAG_NO_CACHE_WRITE;
    private const UInt32 INTERNET_FLAG_RESYNCHRONIZE   =  0x00000800;  // asking wininet to update an item if it
 is newer
    private const UInt32 INTERNET_FLAG_HYPERLINK       =  0x00000400;  // asking wininet to do hyperlinking
semantic which works right for scripts
    private const UInt32 INTERNET_FLAG_NEED_FILE       =  0x00000010;  // need a file for this request
    private const UInt32 INTERNET_FLAG_MUST_CACHE_REQUEST = INTERNET_FLAG_NEED_FILE;

    private const UInt32 FTP_TRANSFER_TYPE_ASCII    = 0x00000001;
    private const UInt32 FTP_TRANSFER_TYPE_BINARY   = 0x00000002;

    private const UInt32 INTERNET_FLAG_TRANSFER_ASCII   = FTP_TRANSFER_TYPE_ASCII;     // 0x00000001
    private const UInt32 INTERNET_FLAG_TRANSFER_BINARY = FTP_TRANSFER_TYPE_BINARY;    // 0x00000002

    private const int POWER_STATE_RESET = 0x800000;
    private const int POWER_FORCE = 4096;

    private const int IOCTL_HAL_REBOOT = 0x101003C;  //The value of IOCTL_HAL_REBOOT was found by my app Reboot.
exe on WinCE

    /// <summary>
    /// Win32 Handles
    /// </summary>
    private static IntPtr InetHandle;
    private static IntPtr FTPsession;

    private static string serverIP_ = "";
    private static bool InternetOpen_;

    private static bool SDcardReady_ = false;
    private static bool exitGatewayApp_ = false;

    public static string serverIP
    {
        get { return serverIP_; }
        set { serverIP_ = value; }
    }

    public static bool isSysClockSet
    {
        get { return sysClockSet_; }
    }

    /// <summary>
    /// Timer bit that toggles every 50ms
    /// </summary>
    public static bool _50msToggle
    {
        get { return _50ms_toggle_; }
    }

    /// <summary>
    /// Shows if timer thread has started
    /// </summary>
    public static bool RTIthreadStarted
    {
        get { return timerThreadStarted; }
    }

    /// <summary>
    /// True if SD card available for access
    /// </summary>
    public static bool SDcardSetup
    {
        get { return SDcardReady_; }
    }

    /// <summary>
    /// Exit GatewayApp when exitApplication is true
    /// </summary>
    public static bool exitGatewayApp
    {
        get { return exitGatewayApp_; }
        set { exitGatewayApp_ = value; }
    }

    /// <summary>
    /// System clock set? (true/false)
    /// </summary>
    private static bool sysClockSet_;

    #endregion

    #region DLL_imports

    [DllImport("Coredll.dll", EntryPoint = "SetSystemPowerState", SetLastError = true)]
    private static extern UInt32 SetSystemPowerState(string psState, int stateFlags, int Options);

    [DllImport("Coredll.dll", EntryPoint = "SetLocalTime", SetLastError = true)]
    private static extern bool SetLocalTime(ref SYSTEMTIME Time);

    [DllImport("Coredll.dll", EntryPoint = "GetDiskFreeSpaceEx", SetLastError = true)]
    public static extern bool GetDiskFreeSpaceEx(string dirName, ref long FreeBytesAvailable, ref long
TotalBytesAvailable, ref long TotalFreebytesAvailable);

    [DllImport("Coredll.dll", SetLastError = true, CharSet = CharSet.Auto)]
    public static extern UInt32 GetLastError();

    [DllImport("Coredll.dll")]
    public static extern int RegFlushKey(UInt32 hKey);

    [DllImport("Coredll.dll")]
    public static extern UInt32 RegOpenKeyEx(UInt32 hKey, string lpSubKey, UInt32 ulOptions, UInt32 samDesired,
out UInt32 phkResult);
```

```
    [DllImport("Coredll.dll")]
    public static extern UInt32 RegCloseKey(UInt32 hKey);

    [DllImport("Coredll.dll")]
    public static extern bool KernelIoControl(int dwIoControlCode,
                                              byte[] inBuf,
                                              int inBufSize,
                                              byte[] outBuf,
                                              int outBufSize,
                                              ref int bytesReturned);

    [DllImport("wininet.dll", SetLastError = true, CharSet = CharSet.Auto)]
    private static extern UInt32 InternetAttemptConnect(UInt32 dwReserved);

    [DllImport("wininet.dll", SetLastError = true, CharSet = CharSet.Auto)]
    private static extern bool InternetCheckConnection(string lpszURL,
                                                       UInt32 dwFlags,
                                                       UInt32 dwReserved);

    [DllImport("wininet.dll", SetLastError = true, CharSet = CharSet.Auto)]
    private static extern IntPtr InternetOpen(string lpszAgent,
                                              UInt32 dwAccessType,
                                              string lpszProxyName,
                                              string lpszProxyBypass,
                                              UInt32 dwFlags);

    [DllImport("wininet.dll", SetLastError = true, CharSet = CharSet.Auto)]
    private static extern bool InternetCloseHandle(IntPtr hInternet);

    [DllImport("wininet.dll", SetLastError = true, CharSet = CharSet.Auto)]
    private static extern IntPtr InternetConnect(IntPtr hInternet,
                                              string lpszServerName,
                                              UInt32 nServerPort,
                                              string lpszUsername,
                                              string lpszPassword,
                                              UInt32 dwService,
                                              UInt32 dwFlags,
                                              UInt32 dwContext);

    [DllImport("wininet.dll", SetLastError = true, CharSet = CharSet.Auto)]
    private static extern bool FtpPutFile(IntPtr hConnect,
                                          string lpszLocalFile,
                                          string lpszNewRemoteFile,
                                          UInt32 dwFlags,
                                          UInt32 dwContext);

    #endregion

    /// <summary>
    /// Performs a system reboot
    /// </summary>
    public static void systemReset()
    {
        GPIO.setGPIO("C", 6, false);    //Direct IO connection to processor reset line - ugly method of resetting system!
    }

    /// <summary>
    /// Writes a registry value that causes instances of GatewayApp to terminate
    /// </summary>
    public static void killApps()
    {
        Registry.SetValue(@"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp", "RemoteKill", 1);
        exitGatewayApp_ = true;
    }

    /// <summary>
    /// Polls registry for command to close application
    /// </summary>
    /// <returns></returns>
    public static bool testForKillApps()
    {
        exitGatewayApp_ = Convert.ToBoolean(Registry.GetValue(@"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp", "RemoteKill", 1));
        return exitGatewayApp_;
    }

    #region FTP methods

    public static UInt32 AttemptInternetConnection()
    {
        return InternetAttemptConnect(0);
    }

    /// <summary>
    /// Checks if connection to the Internet can be established
    /// </summary>
    /// <param name="URL"></param>
    /// <returns>True if connection is made, else false</returns>
    public static bool CheckInternetConnection(string URL)
    {
        return InternetCheckConnection(URL, FLAG_ICC_FORCE_CONNECTION, 0);
    }

    /// <summary>
    /// Establishes the characteristics of the internet connection
    /// </summary>
    /// <param name="HTTPuser">Name of application calling this function</param>
    /// <returns>True if connection is opened, else false if already opened</returns>
    public static bool OpenInternet(string HTTPuser, ref UInt32 ErrorNumber)
    {
        string nullStr = null;

        //Internet already opened: do not open a new session
        if (InternetOpen_ == true) return false;

        InetHandle = InternetOpen(HTTPuser, INTERNET_OPEN_TYPE_DIRECT, nullStr, nullStr, INTERNET_FLAG_ASYNC);
```

```
        ErrorNumber = GetLastError();

        if (ErrorNumber == 0)InternetOpen_ = true;
        else InternetOpen_=false;
        return InternetOpen_;
    }

    /// <summary>
    /// Closes present internet connection
    /// </summary>
    /// <param name="err">Error number</param>
    /// <returns>True indicates successful closure, else false</returns>
    public static bool CloseInternet(ref UInt32 ErrorNumber)
    {
        bool result;

        result = InternetCloseHandle(InetHandle);
        ErrorNumber = GetLastError();
        if (result == true) InternetOpen_ = false;
        return result;
    }

    /// <summary>
    /// Opens an FTP session
    /// </summary>
    /// <param name="IP"></param>
    /// <param name="Username"></param>
    /// <param name="Password"></param>
    /// <returns>Error code. 0 if session opened successfully</returns>
    public static bool OpenFTPsession(string IP,string Username,string Password, ref UInt32 ErrorNumber)
    {
        ConsoleUserInterface.printToConsole(String.Format("FTP username: {0} password: {1} IP: {2}\n\r",Username ↙
,Password,IP));
        FTPsession = InternetConnect(InetHandle, IP.Trim(), INTERNET_INVALID_PORT_NUMBER, Username, Password,    ↙
INTERNET_SERVICE_FTP, INTERNET_FLAG_PASSIVE, 0);

        ErrorNumber = GetLastError();
        if (ErrorNumber == 0) return true;
        else return false;
    }

    /// <summary>
    /// Uploads file via FTP to remote file server
    /// </summary>
    /// <param name="fileToStore">Local file to send</param>
    /// <param name="newFileName">Name of new file on remote server</param>
    /// <returns>True if file upload was successful, else false</returns>
    public static bool WriteFileByFTP(string fileToStore, string newFileName)
    {
        return FtpPutFile(FTPsession, fileToStore, newFileName, INTERNET_FLAG_TRANSFER_BINARY, 0);
    }

    /// <summary>
    /// Closes present internet connection
    /// </summary>
    /// <returns>True indicates successful closure</returns>
    public static bool CloseFTPsession(ref UInt32 ErrorNumber)
    {
        bool result;

        result = InternetCloseHandle(FTPsession);
        ErrorNumber = GetLastError();
        return result;
    }

    #endregion

    #region Time and clock methods

    public static void SetSystemClock(SYSTEMTIME NewTime)
    {
        SetLocalTime(ref NewTime);
        sysClockSet_ = true;
    }

    /// <summary>
    /// Converts DateTime to struct SYSTEMTIME
    /// </summary>
    /// <param name="Type: DateTime"></param>
    /// <returns>"struct SYSTEMTIME"</returns>
    public static SYSTEMTIME ConvertDateTime(DateTime TimeNow)
    {
        SYSTEMTIME x = new SYSTEMTIME();

        x.Year = (ushort)TimeNow.Year;
        x.Month = (ushort)TimeNow.Month;
        x.Day = (ushort)TimeNow.Day;
        x.DOW = (ushort)TimeNow.DayOfWeek;
        x.Hour = (ushort)TimeNow.Hour;
        x.Minute = (ushort)TimeNow.Minute;
        x.Second = (ushort)TimeNow.Second;
        x.Millisecond = (ushort)TimeNow.Millisecond;
        return x;
    }

    /// <summary>
    /// Compares a date to current system date
    /// </summary>
    /// <param name="DateToCheck">Date to compare</param>
    /// <returns>True if dates differ, else false</returns>
    public static bool IsDateDifferent(DateTime DateToCheck)
    {
        DateTime SystemDateNow = DateTime.Today;

        if(DateTime.Compare(DateToCheck,SystemDateNow)!=0)return true;
        else return false;
    }
```

```
    /// <summary>
    /// Compares a time to current system time
    /// </summary>
    /// <param name="TimeToCheck">Time to compare</param>
    /// <returns>True if hour and minute match, false</returns>
    public static bool TimeMatch(SYSTEMTIME TimeToCheck_)
    {
        SYSTEMTIME SystemDateNow_;

        SystemDateNow_=ConvertDateTime(DateTime.Now);

        if (SystemDateNow_.Hour == TimeToCheck_.Hour && SystemDateNow_.Minute == TimeToCheck_.Minute) return
true;
        else return false;
    }

    /// <summary>
    /// Starts a timer thread that toggles a bit every 50ms
    /// </summary>
    public static void startTimerThread()
    {
        if (RTIthreadStarted == false)
        {
            realTimeInterruptThread.IsBackground = true;
            realTimeInterruptThread.Start();
            ConsoleUserInterface.printToConsole("Timer thread started\n\r");
        }
    }

    /// <summary>
    /// Toggles a bit every 50ms
    /// </summary>
    public static void timerThread()
    {
        timerThreadStarted = true;
        while (true)
        {
            _50ms_toggle_ = !_50ms_toggle_;
            Thread.Sleep(50);
        }
    }

    #endregion

    #region Data storage methods

    /// <summary>
    /// Enable SD card for data storage
    /// </summary>
    /// <returns>True if successful, else false</returns>
    public static bool openSDcard()
    {
        SDcardReady_ = false;
        Registry.LocalMachine.CreateSubKey(@"SOFTWARE\FLASHLOADER");
        Registry.SetValue(@"HKEY_LOCAL_MACHINE\SOFTWARE\FLASHLOADER\", "SDConnect", 1);
        Registry.SetValue(@"HKEY_LOCAL_MACHINE\System\StorageManager\Profiles\SDMemory\", "Folder", "SDcard");


        try
        {
            DirectoryInfo dirInfo = new DirectoryInfo("\\SDcard");
            DirectoryInfo[] dirList = dirInfo.GetDirectories();  //Get directory listing of dirInfo

            ConsoleUserInterface.printToConsole("SDcard opened successfully\n\r");
            return true;
        }
        catch (DirectoryNotFoundException)
        {
            ConsoleUserInterface.printToConsole("SD card not found.\n\rNo data logging allowed, only system
setup\n\rInsert SDcard into slot and reboot system\n\r");
            return false;
        }
        catch (Exception msg)
        {
            ConsoleUserInterface.printToConsole(String.Format("Exception in Win32 class: {0}\n", msg.Message));
            return false;
        }
    }

    /// <summary>
    /// Close SD card in registry for valid test at next start of Gateway app
    /// </summary>
    public static void closeSDcard()
    {
        Registry.SetValue(@"HKEY_LOCAL_MACHINE\SOFTWARE\FLASHLOADER\", "SDConnect", 0);
        if(SDcardReady_==true)ConsoleUserInterface.printToConsole("Closing SDcard\r");
        else ConsoleUserInterface.printToConsole("Resetting SDcard setting in registry\r");
    }

    /// <summary>
    /// Checks if media volume exists and if capacity is sufficent
    /// </summary>
    /// <param name="currentPath">Media path to test</param>
    /// <param name="exists">Result of media exist test</param>
    /// <param name="threshold">Minimum amount of space allowed on media</param>
    /// <param name="enoughCapacity">Result of media capacity test</param>
    public static void doesVolumeExist(string currentPath, ref bool exists, float threshold, ref bool
enoughCapacity)
    {
        doesVolumeExist(currentPath, ref exists, threshold, ref enoughCapacity, false);
    }

    /// <summary>
    /// Checks if media volume exists and if capacity is sufficent
    /// </summary>
    /// <param name="currentPath">Media path to test</param>
```

```
    /// <param name="exists">Result of media exist test</param>
    /// <param name="threshold">Minimum amount of space allowed on media</param>
    /// <param name="enoughCapacity">Result of media capacity test</param>
    /// <param name="ignoreThreshold">Ignore media capacity test</param>
    public static void doesVolumeExist(string currentPath,ref bool exists, float threshold, ref bool
enoughCapacity, bool ignoreThreshold)
    {
        long FreeBytesAvailable = 0;
        long TotalBytesAvailable = 0;
        long TotalFreebytesAvailable = 0;

        exists = false;
        GetDiskFreeSpaceEx(currentPath, ref FreeBytesAvailable, ref TotalBytesAvailable, ref
TotalFreebytesAvailable);
        if (TotalBytesAvailable != 0) exists = true; //0 bytes available if volume does not exist

        if (ignoreThreshold == true) return;

        enoughCapacity = true;
        if (AvailableStorage(FreeBytesAvailable, TotalBytesAvailable) < threshold) enoughCapacity = false;
    }

    /// <summary>
    /// Calculate remaining storage capacity on volume
    /// </summary>
    /// <param name="FreeBytesAvailable"></param>
    /// <param name="TotalBytesAvailable"></param>
    /// <returns>Space available as a percentage</returns>
    private static float AvailableStorage(long FreeBytesAvailable, long TotalBytesAvailable)
    {
        float BytesAvailable, TotalVolumeBytes;

        BytesAvailable = FreeBytesAvailable;
        TotalVolumeBytes = TotalBytesAvailable;
        return (BytesAvailable / TotalVolumeBytes) * 100;
    }

    /// <summary>
    /// Perform directory listing on selected media
    /// </summary>
    /// <param name="media">Options: sd, usb, hdd</param>
    public static void dirListing(string media)
    {
        long FreeBytesAvailable = 0;
        long TotalBytesAvailable = 0;
        long TotalFreebytesAvailable = 0;

        string path = "\\";

        switch (media)
        {
            case "sd":
                path="\\SDcard";
                break;

            case "usb":
                path = "\\Hard Disk";
                break;

            case "hdd":
                path = "\\NandFlash";
                break;

            case "log":
                path = String.Format("\\{0}\\{1}\\", Registry.GetValue(Win32.LoggerKey, "Media", "\\"), Registry
.GetValue(Win32.LoggerKey, "Directory", ""));
                break;
        }

        try
        {
            Win32.GetDiskFreeSpaceEx(path, ref FreeBytesAvailable, ref TotalBytesAvailable, ref
TotalFreebytesAvailable);

            if (TotalFreebytesAvailable > 0)
            {
                ConsoleUserInterface.printToConsole(String.Format("\n\rDirectory of {0}:\\\n\r",path));

                DirectoryInfo dirInfo = new DirectoryInfo(path);
                DirectoryInfo[] dirList = dirInfo.GetDirectories();  //Get directory listing of dirInfo

                foreach (DirectoryInfo dirNext in dirList)
                {
                    ConsoleUserInterface.printToConsole(String.Format("\r{0}", dirNext.ToString().Trim()));

                }

                FileInfo[] fileList = dirInfo.GetFiles("*.*");
                foreach (FileInfo fileNext in fileList)
                {
                    ConsoleUserInterface.printToConsole( String.Format("\r{0}", fileNext.ToString().Trim()));
                }
                ConsoleUserInterface.printToConsole(String.Format("\r\n{0} File(s)", fileList.Length));
                ConsoleUserInterface.printToConsole(String.Format("\r{0} Dir(s)\r\n", dirList.Length));
            }

            ConsoleUserInterface.printToConsole(String.Format("{0} Free Bytes: {1}\r",path, FreeBytesAvailable))
;
            ConsoleUserInterface.printToConsole(String.Format("{0} Volume size: {1}\r",path,
TotalBytesAvailable));
            ConsoleUserInterface.printToConsole(String.Format("{0} Free Bytes: {1}\n\r",path,
TotalFreebytesAvailable));
        }
        catch { }
    }
    #endregion
```

```
    #region Registry methods

    /// <summary>
    /// Checks if registry keys exist.  Create if they do not exist and set all to default values
    /// </summary>
    public static void createRegistryKeys()
    {
        int keyExist;
        string key = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp";

        ConsoleUserInterface.printToConsole("Reading registry\n\r");

        keyExist = Convert.ToInt32(Registry.GetValue(key, "Keys exist", 0));
        if (keyExist != 1)    //Assume no registry keys for the app exist, so create them with default values
        {
            ConsoleUserInterface.printToConsole("Creating registry keys and resetting to default values\n\r");

            Registry.LocalMachine.CreateSubKey(@"Software\WSN_Gateway\GatewayApp");
            Registry.LocalMachine.CreateSubKey(@"Software\WSN_Gateway\GatewayApp\FTP");
            Registry.LocalMachine.CreateSubKey(@"Software\WSN_Gateway\GatewayApp\FTP\ETH");
            Registry.LocalMachine.CreateSubKey(@"Software\WSN_Gateway\GatewayApp\FTP\GSM");
            Registry.LocalMachine.CreateSubKey(@"Software\WSN_Gateway\GatewayApp\LOGGER");
            Registry.LocalMachine.CreateSubKey(@"Software\WSN_Gateway\GatewayApp\GSM");
            Registry.LocalMachine.CreateSubKey(@"Software\WSN_Gateway\GatewayApp\GSM\GPRS");
            Registry.LocalMachine.CreateSubKey(@"Software\WSN_Gateway\GatewayApp\STREAM");

            ConsoleUserInterface.printToConsole("Writing default registry values\n\r");

            key = LoggerKey;

            Registry.SetValue(key, "Media", "SDcard");
            Registry.SetValue(key, "Directory", "LOG_DATA");
            Registry.SetValue(key, "Extension", "LOG");
            Registry.SetValue(key, "Old extension", "gz1");
            Registry.SetValue(key, "Packet size", "41");
            Registry.SetValue(key, "Send time", "0010");
            Registry.SetValue(key, "Logger enabled", 0);   //Ensure logger is disabled if values in keys are not ↙
set

            key = GSMkey;

            Registry.SetValue(key, "Module type", "Telit GM862");
            Registry.SetValue(key, "Power IO bank", "B");
            Registry.SetValue(key, "Power IO number", "20");
            Registry.SetValue(key, "Reset IO bank", "B");
            Registry.SetValue(key, "Reset IO number", "21");

            Registry.SetValue(key, "Resend attempts", 5);
            Registry.SetValue(key, "Interval ms", 5000);
            Registry.SetValue(key, "Read timeout ms", 60000);
            Registry.SetValue(key, "Transmit timeout s", 30);

            key = GPRSkey;

            Registry.SetValue(key, "Out port", "");
            Registry.SetValue(key, "In port", "49500");
            Registry.SetValue(key, "Firewall mask", "255.255.0.0");
            Registry.SetValue(key, "Firewall IP", "0.0.0.0");
            Registry.SetValue(key, "Internet APN", "unrestricted");
            Registry.SetValue(key, "Update DNS", "www.dyndns.org");     //DNS server address - update with       ↙
public IP
            Registry.SetValue(key, "Check IP address", "checkip.dyndns.org");   //Address to ping to get public ↙
IP address

            key = StreamKey;

            Registry.SetValue(key, "Out port", "49500");
            Registry.SetValue(key, "In port", "");
            Registry.SetValue(key, "Destination IP", "0.0.0.0");

            key = FTPkey;

            Registry.SetValue(key, "Transport", 1);            //FTP transport default is GSM
            Registry.SetValue(key, "Attempts", 5);             //FTP connection attempts
            Registry.SetValue(key, "Attempts interval s", 30); //Interval s between Internet connect attempts

            key = FTPGSMkey;

            Registry.SetValue(key, "Username", "");
            Registry.SetValue(key, "Password", "");
            Registry.SetValue(key, "Hostname", "");
            Registry.SetValue(key, "Server IP", "");

            key = FTPETHkey;

            Registry.SetValue(key, "Username", "");
            Registry.SetValue(key, "Password", "");
            Registry.SetValue(key, "Hostname", "");
            Registry.SetValue(key, "Server IP", "");

            Registry.SetValue(@"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp", "Keys exist", 1);
            RegFlushKey(HKEY_LOCAL_MACHINE);    //Saves all registry settings in NandFlash hive
        }
        Registry.SetValue(@"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp", "RemoteKill", 0);
    }
    #endregion

    #region IP methods

    public static string getIPaddressLAN()
    {
        string hostName = Dns.GetHostName();
        IPHostEntry localIPinfo = Dns.GetHostEntry(hostName); //Dns.GetHostByName(hostName);
        return localIPinfo.AddressList[0].ToString();
    }

    #endregion
```

```
    }
}
```

```
//FILENAME: UART.cs
using System;
using System.IO.Ports;
using System.Collections;
using System.Threading;
using System.Collections.Generic;
using System.Text;
using Microsoft.Win32;

namespace GatewayApp
{
    public static class UART
    {
        #region Definitions

        public const int MOTE_UART = 0;
        public const int CONSOLE_UART = 4;
        public const int MODEM_UART = 2;

        private const int MAX_IN_BUFFER_SIZE_ = 400000;
        private const int MAX_MODEM_IN_BUFFER_SIZE_ = 100;

        private static string[] availablePorts = SerialPort.GetPortNames();
        private static bool[] comPortOpened_ = new bool[10] { false, false, false, false, false, false, false, false
    , false, false };

        private static SerialPort COM0 = new SerialPort();
        private static SerialPort COM2 = new SerialPort();
        private static SerialPort COM4 = new SerialPort();

        private static byte[] rawNodeDataIn = new byte[MAX_IN_BUFFER_SIZE_];  //Buffer to hold data received from
    sink node
        private static int endOfLastPacket_ = 0;     //Address of the end of the last packet received before the
    inpointer wraps to 0
        private static int MOTE_UART_DataIn_ = 0;

        private static bool pointerWrapped_ = false;
        private static int bytesReceived;
        private static int bytesRead;

        private static bool streamData_ = false;

        /// <summary>
        /// Length of packet from Mote
        /// </summary>
        private static int packetLength_ = 100;     //Start with an arbitary value - will be initialised from
    registry

        private static byte[] MODEM_UART_InBuffer_ = new byte[MAX_MODEM_IN_BUFFER_SIZE_];
        private static int MODEM_UART_DataIn_ = 0;
        private static string modemOnlineData;      //Data received from modem in online mode
        private static string modemOnlineDataCopy;  //A copy of data received from modem in online mode

        public static SerialPort consoleUART
        {
            get { return COM4; }
        }

        public static SerialPort modemUART
        {
            get { return COM2; }
        }

        private enum HandshakeControl
        {
            Xon = 17,
            Xoff = 19
        };

        public static Thread modemHandshakeThread = new Thread(modemDataReceiver);

        public static bool stopTX;                              //Used to stop/start data transmission to the modem
        private static bool startHandshakeRX_;                  //Starts/stops handshake data reception in handshake
     thread
        private static bool modemHandshakeThreadRunning_;       //Shows thread is running to prevent restart of
    thread
        private static int transmitTimer_;                      //Set to a time value in seconds on every Xon byte
    received
                                                                //Timesout after last Xon recieved to send escape
    sequence to modem
        private static int transmitTimerConst_;                 //Constant to set transmitTimer

        #endregion

        #region Class Properties

        /// <summary>
        /// Raw byte data received from sink node
        /// </summary>
        public static byte[] MOTE_UART_Raw_InBuffer
        {
            get { return rawNodeDataIn; }
        }

        /// <summary>
        /// Pointer to start address of recevied packet in buffer rawNodeDataIn
        /// </summary>
        public static int MOTE_UART_DataIn
        {
            get { return MOTE_UART_DataIn_; }
        }

        /// <summary>
        /// Set to true when buffer pointer wraps to zero
        /// </summary>
        public static bool pointerWrapped
        {
```

```
    set { pointerWrapped_ = value; }
    get { return pointerWrapped_; }
}

/// <summary>
/// Address of the end of the last packet written in MOTE_UART_Raw_InBuffer
/// </summary>
public static int endOfLastPacket
{
    get { return endOfLastPacket_; }
}

public static int MAX_IN_BUFFER_SIZE
{
    get { return MAX_IN_BUFFER_SIZE_; }
}

public static bool[] comPortOpened
{
    get { return comPortOpened_; }
}

/// <summary>
/// Length of packet from sink node
/// </summary>
public static int packetLength
{
    get { return packetLength_; }
}

/// <summary>
/// Start or stop sink node data streaming
/// </summary>
public static bool streamData
{
    set { streamData_ = value; }
    get { return streamData_; }
}

/// <summary>
/// A gate to start/stop receiving handshake signals in handshake thread
/// </summary>
public static bool startHandshakeRX
{
    get{return startHandshakeRX_; }
    set { startHandshakeRX_ = value; }
}

public static bool modemHandshakeThreadRunning
{
    get { return modemHandshakeThreadRunning_; }
}

/// <summary>
/// A timer that is set to trasmitTimerConstant
/// </summary>
public static int transmitTimer
{
    get { return transmitTimer_; }
    set { transmitTimer_ = value; }
}

/// <summary>
/// A constant that sets a timer after receiving a Xon control signal from the modem
/// </summary>
public static int transmitTimerConst
{
    set { transmitTimerConst_ = value; }
}

#endregion

/// <summary>
/// OpenSerialPort
/// </summary>
/// <param name="COM">COM port number to address</param>
/// <param name="baud">Baud rate of specified port</param>
/// <returns></returns>
public static bool OpenSerialPort(int COM,int baud)
{
    string COMname;
    bool portOpen = false;

    try
    {
        COMname="COM"+COM.ToString();
        int x = 0;
        ConsoleUserInterface.printToConsole("Available COM ports:\n\r");
        //Console.Write("\rAvailable COM ports:\n\r");

        foreach(string msg in availablePorts)
        {
            ConsoleUserInterface.printToConsole(String.Format( "{0} {1}\r",x, msg));
            //Console.Write(String.Format("{0} {1}\n\r", x, msg));
            x++;
        }
        switch (COM)
        {
            case MOTE_UART:
                packetLength_ = Convert.ToInt32(Registry.GetValue(Win32.LoggerKey, "Packet size", ""));
                COM0.PortName = COMname;
                COM0.BaudRate = baud;
                COM0.DataBits = 8;
                COM0.Handshake = Handshake.None;
                //COM0.Handshake = Handshake.RequestToSend;
                COM0.ReadTimeout = Timeout.Infinite;
                COM0.ReceivedBytesThreshold = packetLength_;
```

```
                //COM0.ReceivedBytesThreshold = 1;
                COM0.DataReceived += new SerialDataReceivedEventHandler(COM0_DataReceived);


                //endOfLastPacket_ = ((int)MAX_IN_BUFFER_SIZE_ / packetLength_) * packetLength_;
                endOfLastPacket_ = (int)MAX_IN_BUFFER_SIZE_ - 1;

                if (COM0.IsOpen == true) COM0.Close();
                if (COM0.IsOpen == false)
                {
                    COM0.Open();
                    portOpen = true;
                    comPortOpened_[MOTE_UART] = true;
                }
                break;

            case MODEM_UART:
                COM2.PortName = COMname;
                COM2.BaudRate = baud;
                COM2.DataBits = 8;

                COM2.Handshake = Handshake.None;

                COM2.ReadTimeout = 50;

                if (COM2.IsOpen == true) COM2.Close();
                if (COM2.IsOpen == false)
                {
                    COM2.Open();
                    portOpen = true;
                    comPortOpened_[MODEM_UART] = true;
                }
                break;

            case CONSOLE_UART:
                COM4.PortName = COMname;
                COM4.BaudRate = baud;
                COM4.DataBits = 8;
                COM4.Handshake = Handshake.None;
                COM4.ReadTimeout = 500;
                //COM4.ReadTimeout = Timeout.Infinite;        //Wait for ever for a console key entry
                if (COM4.IsOpen == true) COM4.Close();
                if (COM4.IsOpen == false)
                {
                    COM4.Open();
                    portOpen = true;
                    comPortOpened_[CONSOLE_UART] = true;
                }
                break;

            default: return false;
        }
    }
    catch (Exception msg)
    {
        ConsoleUserInterface.printToConsole(String.Format( "{0}\n", msg.Message));
    }
    return portOpen;
}

/// <summary>
/// Close UART
/// </summary>
/// <param name="COM"></param>
/// <returns>true when closed, false if still open</returns>
public static bool CloseSerialPort(int COM)
{
    string COMname;

    COMname = "COM" + COM.ToString();
    try
    {
        switch (COM)
        {
            case MOTE_UART:
                if (COM0.IsOpen == true)
                {
                    COM0.Close();
                    comPortOpened_[MOTE_UART] = false;
                }
                break;

            case MODEM_UART:
                if (COM2.IsOpen == true)
                {
                    ConsoleUserInterface.printToConsole("Closing MODEM UART\r");
                    COM2.Close();
                    comPortOpened_[MODEM_UART] = false;
                }
                break;

            case CONSOLE_UART:
                if (COM4.IsOpen == true)
                {
                    COM4.Close();
                    comPortOpened_[CONSOLE_UART] = false;
                }
                break;

            default: return false;
        }
        return true;
    }
    catch
    {
        ConsoleUserInterface.printToConsole(String.Format( "Error closing {0}\n", COMname));
    }
```

```
    return true;
}

public static void WriteToSerialPort(int COM,string Message)
{
    string COMname;

    COMname = "COM" + COM.ToString();
    try
    {
        switch (COM)
        {
            case MOTE_UART:
                COM0.WriteLine(Message);
                break;

            case MODEM_UART:
                //Console.WriteLine(String.Format("C1 {0}\n\r", Message));
                COM2.WriteLine(Message);
                break;

            case CONSOLE_UART:
                COM4.WriteLine(Message);
                break;
        }
    }
    catch (Exception msg)
    {
        ConsoleUserInterface.printToConsole(String.Format( "[13] Exception message: {0}\n", msg.Message));
    }
}

public static string ReadFromSerialPort(int COM)
{
    string message;
    try
    {
        switch (COM)
        {
            case MOTE_UART:
                message = COM0.ReadLine();
                break;

            case MODEM_UART:
                message = modemOnlineDataCopy;
                modemOnlineDataCopy = "";
                break;

            case CONSOLE_UART:
                message = COM4.ReadLine();
                break;

            default: return "";
        }
        return message;
    }
    catch (TimeoutException) { }
    return "";
}

public static string[] GetPortNames()
{
    return SerialPort.GetPortNames();
}

/// <summary>
/// Change a COM port baud rate
/// </summary>
/// <param name="COM">COM number</param>
/// <param name="baud">New baud rate</param>
/// <returns>True if successful, else false</returns>
public static bool changeBaud(int COM, int baud)
{
    try
    {
        switch (COM)
        {
            case MOTE_UART:
                COM0.BaudRate = baud;
                break;

            case MODEM_UART:
                COM2.BaudRate = baud;
                break;

            case CONSOLE_UART:
                COM4.BaudRate = baud;
                break;

            default: return false;
        }
        return true;
    }
    catch { return false; }
}

/// <summary>
/// Send +++ to modem to return to command mode
/// </summary>
/// <param name="guardTime">Time in ms between escape char +</param>
/// <param name="noResponse">True = no response, only send +++</param>
/// <returns></returns>
public static string sendEscapeSequence(int guardTime, bool noResponse)
{
    if (noResponse == false)
    {
```

```
            return sendEscapeSequence(guardTime);
        }
        else
        {
            COM2.Write("+");
            Thread.Sleep(guardTime);
            COM2.Write("+");
            Thread.Sleep(guardTime);
            COM2.Write("+");
            Thread.Sleep(guardTime);
            return "";
        }
    }

    /// <summary>
    /// Send +++ to modem to return to command mode
    /// </summary>
    /// <param name="guardTime">Time in ms between escape char +</param>
    /// <returns></returns>
    public static string sendEscapeSequence(int guardTime)
    {
        byte byteReceived;
        byte[] holder = new byte[3];
        string modemResponse;

        try
        {
            for (int i = 0; i < MAX_MODEM_IN_BUFFER_SIZE_; i++) MODEM_UART_InBuffer_[i] = 0;
            COM2.DiscardInBuffer();
            COM2.ReadTimeout = 10000;   //10s timeout to wait for modem response

            COM2.Write("+");
            Thread.Sleep(guardTime);
            COM2.Write("+");
            Thread.Sleep(guardTime);
            COM2.Write("+");
            Thread.Sleep(guardTime);

            MODEM_UART_DataIn_ = 0;
            while (true)
            {
                COM2.Read(MODEM_UART_InBuffer_, MODEM_UART_DataIn_, 1);
                byteReceived = MODEM_UART_InBuffer_[MODEM_UART_DataIn_];

                holder[0] = holder[1];
                holder[1] = holder[2];
                holder[2] = byteReceived;

                //Test for oK, erroR, connecT, no carrieR, CR LF
                if ((holder[0] == 'K' || holder[0] == 'R' || holder[0] == 'T') && holder[1] == '\r' && holder[2] ↙
 == '\n')
                {
                    modemResponse = Encoding.Default.GetString(MODEM_UART_InBuffer_, 2, MODEM_UART_DataIn_); // ↙
Check that this line is correct
                    return modemResponse;
                }
                MODEM_UART_DataIn_++;
                Thread.Sleep(0);
            }
        }
        catch
        {
            if (MODEM_UART_DataIn_ > 0)    //Some data has been received from the modem, so process the global ↙
buffer in case data is useable
            {
                modemResponse = Encoding.Default.GetString(MODEM_UART_InBuffer_, 2, MODEM_UART_DataIn_);
                return modemResponse;
            }
            else return "";  //ReadByte timeout occured, so return to calling function send command to modem ↙
again
        }
    }

    /// <summary>
    /// Sends a command to modem and returns modem response
    /// </summary>
    /// <param name="command">Modem AT command</param>
    /// <param name="timeout">Length of time in ms to wait before returning to caller</param>
    /// <returns>Returns modem response, or empty string if timeout occurs</returns>
    public static string sendCommand(string command, int timeout)
    {
        byte byteReceived;
        byte[] holder = new byte[3];
        string modemResponse;

        try
        {
            COM2.RtsEnable = true;
            for (int i = 0; i < MAX_MODEM_IN_BUFFER_SIZE_; i++) MODEM_UART_InBuffer_[i] = 0;
            for (int i = 0; i < 3; i++) holder[i] = 0;

            COM2.DiscardInBuffer();
            COM2.ReadTimeout = timeout;
            WriteToSerialPort(MODEM_UART, String.Format("{0}\r", command));

            MODEM_UART_DataIn_=0;
            while (true)
            {
                Thread.Sleep(0);

                COM2.Read(MODEM_UART_InBuffer_, MODEM_UART_DataIn_, 1);
                byteReceived = MODEM_UART_InBuffer_[MODEM_UART_DataIn_];

                holder[0]=holder[1];
                holder[1]=holder[2];
                holder[2] = byteReceived;
```

```
                //Test for (OK OR erroR OR connecT OR no carrieR) AND CR LF
                if((holder[0]=='K' || holder[0]=='R' || holder[0]=='T') && holder[1]=='\r' && holder[2]=='\n')
                {
                    modemResponse = Encoding.Default.GetString(MODEM_UART_InBuffer_, 2, MODEM_UART_DataIn_); // ↙
Check that this line is correct
                    return modemResponse;
                }
                MODEM_UART_DataIn_++;
            }
        }
        catch
        {
            if (MODEM_UART_DataIn_ > 0)    //Some data has been received from the modem, so process the global ↙
buffer in case data is useable
            {
                modemResponse = Encoding.Default.GetString(MODEM_UART_InBuffer_, 2, MODEM_UART_DataIn_);
                return modemResponse;
            }
            else return "";  //ReadByte timeout occured, so return to calling function send command to modem ↙
again
        }
    }

    /// <summary>
    /// Sends upto 300 bytes of data to the modem UART.
    /// </summary>
    /// <param name="buffer">Reference to buffer array holding data to send</param>
    /// <param name="bytesToSend">Number of bytes in buffer[] to send</param>
    /// <returns></returns>
    public static bool sendBuffer(ref byte[] buffer, int bytesToSend)
    {
        int data_out;
        byte[] X = new byte[5];
        bool old=false;
        int counter = 1200;     //1200 * 50ms = 1 minute

        data_out = 0;
        while (data_out < bytesToSend)
        {
            if(stopTX == false)  //Stop transmitting if Xoff received
            {
                COM2.Write(buffer, data_out, 1);
                counter=1200;
                data_out++;
            }
            if (old != Win32._50msToggle)   //Timer thread allows this timer to exit loop after 60s of ↙
inactivity
            {
                old = Win32._50msToggle;
                counter--;
                if (counter == 0) return false;
            }
            Thread.Sleep(0);
        }
        return true;
    }

    /// <summary>
    /// Thread to receive Xon/Xoff data from modem only during FTP
    /// </summary>
    static void modemDataReceiver()
    {
        byte[] X = new byte[20];

        stopTX = false;
        modemHandshakeThreadRunning_ = true;
        while (true)
        {
            Thread.Sleep(0);
            try
            {
                if (startHandshakeRX_ == true)   //Enable handshaking during file transfer only
                {
                    X[0] = 0;
                    COM2.DiscardInBuffer();
                    COM2.Read(X, 0, 1);

                    if (X[0] == (int)HandshakeControl.Xoff)
                    {
                        stopTX = true;          //Xoff received
                    }
                    else if (X[0] == (int)HandshakeControl.Xon)
                    {
                        stopTX = false;                         //Xon received - fill modem buffer
                        transmitTimer_ = transmitTimerConst_;    //Allow 15s after last Xon received before ↙
sending escape sequence
                        //ConsoleUserInterface.printToConsole(".");
                    }
                }
                else    //Once modem is in LISTEN mode by GSM.initGPRSserver(), the modem output needs to be ↙
polled to receive CONNECT message
                {       //Modem accepts client connection automatically if permitted by the firewall.
                    if (GSM.GPRSmodemStatus == GSM.GPRSstatus.LISTEN || GSM.GPRSmodemStatus == GSM.GPRSstatus. ↙
CONNECT)
                    {
                        COM2.ReadTimeout= 20;
                        modemOnlineData = "";
                        modemOnlineData = COM2.ReadLine();
                        if (modemOnlineData != "")
                        {
                            modemOnlineData.Trim();
                            //Print any received data from modem to console
                            if (GSM.GPRSmodemStatus != GSM.GPRSstatus.CONNECT)ConsoleUserInterface. ↙
printToConsole(modemOnlineData);

                            //If CONNECT received, modem in 'online data mode', steer Console IO to modem
                            if (modemOnlineData.Contains("CONN") == true)
```

```
                            {
                                GSM.GPRSmodemStatus = GSM.GPRSstatus.CONNECT;  //Modem out of online mode if NO CARRIER or ABORT found in data
                                ConsoleUserInterface.printSplashScreen();
                                COM2.WriteLine("Type help to acces menu\n\r");

                            }
                            else if (modemOnlineData.Contains("NO C") == true || modemOnlineData.Contains("ABOR") == true)
                            {
                                //Modem is offline, therefore steer Console IO to eth or UART
                                GSM.GPRSmodemStatus = GSM.GPRSstatus.NO_CARRIER;
                                ConsoleUserInterface.printToConsole(modemOnlineData);
                            }
                            modemOnlineDataCopy = modemOnlineData;
                        }
                    }
                }
                catch { }
            }
        }

        /// <summary>
        /// COM0 DataReceived event handler. Attaches system time to date and writes this into a circular buffer MOTE_UART_InBuffer
        /// </summary>
        /// <param name="sender"></param>
        /// <param name="e"></param>
        static void COM0_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            try
            {
                if (LoggingThread.pauseLogging == true) MOTE_UART_DataIn_ = 0;

                bytesReceived = COM0.BytesToRead;
                if (bytesReceived > 0)
                {
                    //Check there is enough space in rawNodeDataIn to hold a new packet.  Wrap pointer if not enough space
                    //if ((MAX_IN_BUFFER_SIZE_ - MOTE_UART_DataIn_) < bytesReceived || MOTE_UART_DataIn_ >= MAX_IN_BUFFER_SIZE_)
                    if((MOTE_UART_DataIn_ + bytesReceived) >= MAX_IN_BUFFER_SIZE_)
                    {
                        pointerWrapped_ = true;
                        endOfLastPacket_ = MOTE_UART_DataIn_;
                        MOTE_UART_DataIn_ = 0;
                    }

                    bytesRead = COM0.Read(rawNodeDataIn, MOTE_UART_DataIn_, bytesReceived);

                    //Streams sink node data to remote server
                    if (streamData_ == true)
                    {
                        if (ConsoleUserInterface.ActiveConsoleIO == ConsoleUserInterface.consoleInput.Ethernet)
                        {
                            if (SocketStream.socketActive == true)
                            {
                                SocketStream.TCPsocket.BeginSend(rawNodeDataIn, MOTE_UART_DataIn_, bytesRead, System.Net.Sockets.SocketFlags.None, null, System.Net.Sockets.SocketFlags.None);
                            }
                        }
                    }

                    MOTE_UART_DataIn_ = MOTE_UART_DataIn_ + bytesRead;

                }
            }

            catch (TimeoutException)    //Receive packet of shorter length
            {

            }
            catch (Exception msg)
            {
                ConsoleUserInterface.printToConsole(String.Format("{0}\n\r", msg.ToString()));
            }
        }
    }
}
```

```
//FILENAME: ConsoleUserInterface.cs
using System;
using System.Collections.Generic;
using System.Threading;
using System.Text;
using Microsoft.Win32;

namespace GatewayApp
{
    public static class ConsoleUserInterface
    {
        /// <summary>
        /// Console input selection
        /// </summary>
        public enum consoleInput : byte
        {
            UART = 1,
            Ethernet,
            MODEM
        };

        /// <summary>
        /// FTP transport selection
        /// </summary>
        public enum FTPtransmitter : int
        {
            GSM = 1,
            Ethernet
        };

        /// <summary>
        /// Stream transport selection
        /// </summary>
        public enum StreamTransmitter : int
        {
            GSM = 1,
            Ethernet,
            Serial
        };
        private static int CONSOLE_UART = UART.CONSOLE_UART;

        public static DataRecorder dataLogger = new DataRecorder();
        public static Thread userInterface = new Thread(userInterfaceThread);

        private static consoleInput consoleInput_ = consoleInput.UART;      //Default to all consoles
        private static int FTPtransport = (int)FTPtransmitter.GSM;          //Default FTP transport selection
        private static bool userInterfaceThreadRunning_;
        private static bool exitApp_;
        private static string softwareVersion_;

        private const bool textData = false;
        private const bool integerData = true;

        /// <summary>
        /// True if user interface thread is running
        /// </summary>
        public static bool userInterfaceThreadRunning
        {
            get { return userInterfaceThreadRunning_; }
        }

        /// <summary>
        /// Software version
        /// </summary>
        public static string softwareVersion
        {
            set { softwareVersion_ = value; }
        }

        /// <summary>
        /// Active console user interface
        /// </summary>
        public static consoleInput ActiveConsoleIO
        {
            get { return consoleInput_; }
        }

        /// <summary>
        /// True if request to exit gateway application is received from user
        /// </summary>
        public static bool exitApplication
        {
            get { return exitApp_; }
        }

        /// <summary>
        /// Opens a console UART for debugging
        /// </summary>
        public static void openConsoleUART()
        {
            if (UART.OpenSerialPort(CONSOLE_UART, 115200) == true) UART.comPortOpened[CONSOLE_UART] = true;
            else UART.comPortOpened[CONSOLE_UART] = false;
        }

        /// <summary>
        /// Close console UART
        /// </summary>
        public static void closeConsoleUART()
        {
            if (consoleInput_ == consoleInput.UART)
            {
                printToConsole("Closing CONSOLE UART\r");
                UART.CloseSerialPort(UART.CONSOLE_UART);
            }
        }

        /// <summary>
```

```
        /// Allows a user to select the console (UART/Ethernet) to be used for this session, opens UART
        /// <param name="choice">Console type</param>
        /// <param name="ver">Software version</param>
        /// </summary>
        public static void selectConsoleInput(consoleInput selection)
        {
            switch (selection)
            {
                case consoleInput.Ethernet:
                    consoleInput_ = consoleInput.Ethernet;
                    break;

                default:
                    consoleInput_ = consoleInput.UART;
                    openConsoleUART();
                    break;
            }
            printSplashScreen();
        }

        public static void printSplashScreen()
        {
            printToConsole(String.Format("CPUT WSN GATEWAY Ver: {0} March 2012\r", softwareVersion_));
            printToConsole("By: Gary de Villiers\n\r");
        }

//**CONSOLE METHODS*****************************************************************************************************************
    *************

        /// <summary>
        /// Prints messages to default console and UART COM4
        /// </summary>
        /// <param name="message">String to send to consoles</param>
        public static void printToConsole(string message)
        {
            try
            {
                if (GSM.GPRSmodemStatus == GSM.GPRSstatus.CONNECT)
                {
                    UART.WriteToSerialPort(UART.MODEM_UART, message);
                    //Console.WriteLine(String.Format("D1 {0}", message));
                }
                else
                {
                    if (consoleInput_ == consoleInput.UART) UART.WriteToSerialPort(UART.CONSOLE_UART, message);

                    else if (consoleInput_ == consoleInput.Ethernet) Console.WriteLine(message);
                }
            }
            catch { }
        }

        /// <summary>
        /// Reads characters from selected console
        /// </summary>
        /// <returns>String received from console</returns>
        public static string readFromConsole()
        {
            try
            {
                Thread.Sleep(0);
                if (GSM.GPRSmodemStatus == GSM.GPRSstatus.CONNECT)
                {
                    string temp;
                    temp = UART.ReadFromSerialPort(UART.MODEM_UART);
                    if(temp!="")
                    {
                        Console.WriteLine(String.Format("Text received B1 {0}", temp));
                        return temp.Trim();
                    }
                }
                else
                {
                    if (consoleInput_ == consoleInput.UART) return UART.ReadFromSerialPort(UART.CONSOLE_UART).Trim()
;
                    else if (consoleInput_ == consoleInput.Ethernet) return Console.ReadLine().Trim();

                }
                return "";
            }
            catch { return ""; }
        }

        /// <summary>
        /// Displays current gateway settings
        /// </summary>
        private static void printSetupScreen()
        {
            printToConsole("\n\rCurrent gateway settings:\n\r");
            printToConsole(String.Format( "Date and time: {0}\r",DateTime.Now));

            printToConsole("\n\rGlobal FTP settings:\r");
            displayFTPsettings();

            printToConsole("\n\rFTP GSM settings:\r");
            displayFTPsettingsGSM();

            printToConsole("\rFTP Ethernet settings:\r");
            displayFTPsettingsETH();

            printToConsole("\rGSM settings:\r");
            displayGSMsettings();

            printToConsole("\rLogger settings:\r");
            displayLoggerSettings();

            printToConsole("\rData streaming settings:\r");
```

```
    }

//**LOGGER METHODS*****************************************************************************************************************
    ************

    #region Logger setting options

    /// <summary>
    /// System Logger settings
    /// </summary>
    private static void printLoggerSetupScreen()
    {
        bool exitLoop = false;
        string message;

        displayLoggerOptions();
        while (exitLoop == false)
        {
            message = readFromConsole();
            if (message.Length > 1)
            {
                switch (message)
                {
                    case "media":
                        enterLoggerParameter("Media",false);
                        displayLoggerOptions();
                        break;

                    case "dir":
                        enterLoggerParameter("Directory",false);
                        displayLoggerOptions();
                        break;

                    case "ext":
                        enterLoggerParameter("Extension",false);
                        displayLoggerOptions();
                        break;

                    case "old":
                        enterLoggerParameter("Old Extension", false);
                        displayLoggerOptions();
                        break;

                    case "packet":
                        enterLoggerParameter("Packet size", true);
                        displayLoggerOptions();
                        break;

                    case "send time":
                        enterLoggerParameter("Send time",false);
                        displayLoggerOptions();
                        break;

                    case "start":
                        enableLogger(true);
                        displayLoggerOptions();
                        break;

                    case "stop":
                        enableLogger(false);
                        displayLoggerOptions();
                        break;

                    case "exit":
                        saveRegistry();
                        exitLoop = true;
                        break;

                    case "help":
                        displayLoggerOptions();
                        break;
                }
            }
        }
    }

    private static void enableLogger(bool start)
    {
        int enabled;
        //string key = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\LOGGER";
        string key = Win32.LoggerKey;

        if (start == true) Registry.SetValue(key, "Logger enabled", 1);
        else Registry.SetValue(key, "Logger enabled", 0);

        enabled = Convert.ToInt32(Registry.GetValue(key, "Logger enabled", 0));

        printToConsole(String.Format("Current time is: {0}\r", DateTime.Now));
        if (enabled == 0) printToConsole("Logger is stopped\r");
        else printToConsole("Logger is started\r");
    }

    /// <summary>
    /// Display logger setting options
    /// </summary>
    private static void displayLoggerOptions()
    {
        printToConsole("\n\rLogger settings:\n\r");
        displayLoggerSettings();

        printToConsole("\n\rOptions:\r");
        printToConsole("media    \t - SDcard, NandFlash, USBstick\r");
        printToConsole("dir      \t - Logging directory\r");
        printToConsole("ext      \t - <xxx>  Log file extension\r");
        printToConsole("old      \t - <xxx>  Compressed file extension after file FTP'd\r");
        printToConsole("packet   \t - size of packet from sink node\r");
```

```
            printToConsole("send time\t - <hhmm> Time at which logs are sent to remote site\r");
            printToConsole("start    \t - Start logging\r");
            printToConsole("stop     \t - Stop logging\r");

            printToConsole("exit\n\r");
        }

        /// <summary>
        /// Displays current Logger settings in registry
        /// </summary>
        private static void displayLoggerSettings()
        {
            int enabled;
            //string key = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\LOGGER";
            string key = Win32.LoggerKey;

            printToConsole(String.Format("media:    \t{0}\r", Registry.GetValue(key, "Media", "")));
            printToConsole(String.Format("dir:      \t{0}\r", Registry.GetValue(key, "Directory", "")));
            printToConsole(String.Format("ext:      \t{0}\r", Registry.GetValue(key, "Extension", "")));
            printToConsole(String.Format("old:      \t{0}\r", Registry.GetValue(key, "Old extension", "")));
            printToConsole(String.Format("packet size:\t{0}\r", Registry.GetValue(key, "Packet size", "")));
            printToConsole(String.Format("send time:\t{0}\r", Registry.GetValue(key, "Send time", "")));

            enabled = Convert.ToInt32(Registry.GetValue(key,"Logger enabled",0));
            if(enabled==1)printToConsole("logging:  \tYES\r");
            else printToConsole("logging:  \tNO\r");

            string logDirectory_ = String.Format("\\{0}\\{1}\\", Registry.GetValue(Win32.LoggerKey, "Media", "\\"), ↙
    Registry.GetValue(Win32.LoggerKey, "Directory", ""));
            printToConsole(String.Format("Logging path:   {0}\r", logDirectory_));

            printToConsole(String.Format("Current logfile: {0}\n\r",LoggingThread.currentLogFileName));
        }

        /// <summary>
        /// Allows user to enter new Logger parameters to registry
        /// </summary>
        /// <param name="parameter">Parameter in registry</param>
        /// <param name="parameterIsInteger">True if parameter to be written to registry is an integer</param>
        private static void enterLoggerParameter(string parameter, bool parameterIsInteger)
        {
            string key = Win32.LoggerKey;

            enterParameter(key, parameter, parameterIsInteger);
        }

        #endregion

//**GSM METHODS*****************************************************************************************************************↙
    ************

        #region GSM setting options

        /// <summary>
        /// System GSM settings
        /// </summary>
        private static void printGSMsetupScreen()
        {
            bool exitLoop = false;
            string message;

            displayGSMoptions();
            while (exitLoop == false)
            {
                message = readFromConsole();
                if (message.Length > 1)
                {
                    switch (message)
                    {
                        case "module":
                            enterGSMparameter("Module type",textData);
                            displayGSMoptions();
                            break;

                        case "powerbank":
                            enterGSMparameter("Power IO bank",textData);
                            displayGSMoptions();
                            break;

                        case "powerio":
                            enterGSMparameter("Power IO number",integerData);
                            displayGSMoptions();
                            break;

                        case "resetbank":
                            enterGSMparameter("Reset IO bank",textData);
                            displayGSMoptions();
                            break;

                        case "resetio":
                            enterGSMparameter("Reset IO number",integerData);
                            displayGSMoptions();
                            break;

                        case "attempts":
                            enterGSMparameter("Resend attempts",integerData);
                            displayGSMoptions();
                            break;

                        case "interval":
                            enterGSMparameter("Interval ms",integerData);
                            displayGSMoptions();
                            break;

                        case "rxtime":
                            enterGSMparameter("Read timeout ms",integerData);
                            displayGSMoptions();
```

```
                    break;

                case "txtime":
                    enterGSMparameter("Transmit timeout s",integerData);
                    displayGSMoptions();
                    break;

                case "exit":
                    saveRegistry();
                    exitLoop = true;
                    break;

                case "help":
                    displayGSMoptions();
                    break;
            }
        }
    }
}

/// <summary>
/// Display GSM setting options
/// </summary>
private static void displayGSMoptions()
{
    printToConsole("\n\rGSM settings:\n\r");
    displayGSMsettings();

    printToConsole("\n\rOptions:\r");
    printToConsole("module    \t - GSM module type\r");
    printToConsole("powerbank \t - <A..C>  Processor IO bank controlling module power\r");
    printToConsole("powerio   \t - <0..31> IO pin number controlling module power\r");
    printToConsole("resetbank \t - <A..C>  Processor IO bank controlling module reset\r");
    printToConsole("resetio   \t - <0..31> IO pin number controlling module reset\n\r");

    printToConsole("attempts \t - <0..100>   Command resend attempts on GSM RX timeout\r");
    printToConsole("interval \t - <0..90000> ms Time between command attempts\r");
    printToConsole("rxtime   \t - <0..90000> ms UART comms timeout\r");
    printToConsole("txtime   \t - <0..1000>  s  Time to send buffer at end of FTP upload\r");
    printToConsole("exit\n\r");
}

/// <summary>
/// Displays current GSM settings in registry
/// </summary>
private static void displayGSMsettings()
{
    string key = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\GSM";

    printToConsole(String.Format("module:    \t{0}\r", Registry.GetValue(key, "Module type", "")));
    printToConsole(String.Format("powerbank: \t{0}\r", Registry.GetValue(key, "Power IO bank", "")));
    printToConsole(String.Format("powerio:   \t{0}\r", Registry.GetValue(key, "Power IO number", "")));
    printToConsole(String.Format("resetbank: \t{0}\r", Registry.GetValue(key, "Reset IO bank", "")));
    printToConsole(String.Format("resetio:   \t{0}\n\r", Registry.GetValue(key, "Reset IO number", "")));

    printToConsole(String.Format("attempts:  \t{0}\r", Registry.GetValue(key, "Resend attempts", 0)));
    printToConsole(String.Format("interval:  \t{0}\r", Registry.GetValue(key, "Interval ms", 0)));
    printToConsole(String.Format("rxtime:    \t{0}\r", Registry.GetValue(key, "Read timeout ms", 0)));
    printToConsole(String.Format("txtime:    \t{0}\n\r", Registry.GetValue(key, "Transmit timeout s", 0)));
}

/// <summary>
/// Allows user to enter new GSM parameters to registry
/// </summary>
/// <param name="parameter">Parameter in registry</param>
/// <param name="parameterIsInteger">True if parameter to be written to registry is an integer</param>
private static void enterGSMparameter(string parameter, bool parameterIsInteger)
{
    string key = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\GSM";

    enterParameter(key, parameter, parameterIsInteger);
}

#endregion

//**FTP METHODS*****************************************************************************************************************
    ************

#region FTP setting options

/// <summary>
/// System FTP settings
/// </summary>
private static void printFTPsetupScreen()
{
    bool exitLoop = false;
    string message;

    displayFTPoptions();
    while (exitLoop == false)
    {
        message = readFromConsole();
        if (message.Length > 1)
        {
            switch (message)
            {
                case "gsm":
                    setupFTP_GSM();
                    displayFTPoptions();
                    break;

                case "eth":
                    setupFTP_ETH();
                    displayFTPoptions();
                    break;

                case "select":
```

```
                    selectTransmission();
                    displayFTPoptions();
                    break;

                case "exit":
                    saveRegistry();
                    exitLoop = true;
                    break;

                case "help":
                    displayFTPoptions();
                    break;

            }
        }
    }
}

/// <summary>
/// Display FTP setting options
/// </summary>
private static void displayFTPoptions()
{
    printToConsole("\n\rGlobal FTP settings:\n\r");
    displayFTPsettings();

    printToConsole("\rFTP GSM settings:\n\r");
    displayFTPsettingsGSM();

    printToConsole("\rFTP Ethernet settings:\n\r");
    displayFTPsettingsETH();

    printToConsole("select \t - Select transmission type: GSM or Ethernet\r");
    printToConsole("gsm \t - FTP settings for GSM transmission\r");
    printToConsole("eth \t - FTP settings for Ethernet transmission\r");

    printToConsole("exit\n\r");
}

/// <summary>
/// Display global FTP settings
/// </summary>
private static void displayFTPsettings()
{
    printToConsole(String.Format("Connection attempts: {0}\r", Registry.GetValue(Win32.FTPkey, "Attempts",
0)));
    printToConsole(String.Format("Connection interval: {0}s\r", Registry.GetValue(Win32.FTPkey, "Attempts
interval s", 30)));

    FTPtransport = Convert.ToInt32(Registry.GetValue(Win32.FTPkey, "Transport", 2));  //Default of 2 =
ethernet
    if (FTPtransport == (int)FTPtransmitter.GSM) printToConsole("FTP Transport:       GSM\r");
    else printToConsole("FTP Transport:       Ethernet\r");
}

#endregion

#region Setup FTP parameters for GSM

/// <summary>
/// Setup FTP parameters for GSM transmission
/// </summary>
private static void setupFTP_GSM()
{
    bool exitLoop = false;
    string message;

    displayFTPoptionsGSM();
    while (exitLoop == false)
    {
        message = readFromConsole();
        if (message.Length > 1)
        {
            switch (message)
            {
                case "username":
                    enterFTPparameter("Username", FTPtransmitter.GSM);
                    displayFTPoptionsGSM();
                    break;

                case "password":
                    enterFTPparameter("Password", FTPtransmitter.GSM);
                    displayFTPoptionsGSM();
                    break;

                case "serverip":
                    enterFTPparameter("Server IP", FTPtransmitter.GSM);
                    displayFTPoptionsGSM();
                    break;

                case "hostname":
                    enterFTPparameter("Hostname", FTPtransmitter.GSM);
                    displayFTPoptionsGSM();
                    break;

                case "exit":
                    exitLoop = true;
                    break;
            }
        }
    }
}

/// <summary>
/// Display FTP setting options for GSM
/// </summary>
private static void displayFTPoptionsGSM()
```

```
    {
        printToConsole("\n\n\rFTP setup for GSM transport:\n\r");
        displayFTPsettingsGSM();
        printToConsole("Options:\r");
        printToConsole("username\r");
        printToConsole("password\r");
        printToConsole("serverip\r");
        printToConsole("hostname\r");
        printToConsole("exit\n\r");
    }

    /// <summary>
    /// Displays current FTP settings for GSM transport
    /// </summary>
    private static void displayFTPsettingsGSM()
    {
        //string key = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\FTP\GSM";
        string key = Win32.FTPGSMkey;

        printToConsole(String.Format("username: \t{0}\r", Registry.GetValue(key, "Username", "")));
        printToConsole(String.Format("password: \t{0}\r", Registry.GetValue(key, "Password", "")));
        printToConsole(String.Format("hostname: \t{0}\r", Registry.GetValue(key, "Hostname", "")));
        printToConsole(String.Format("serverip: \t{0}\n\r", Registry.GetValue(key, "Server IP", "")));
    }

    private static void enterFTPparameter(string fieldName, FTPtransmitter select)
    {
        //string key = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\FTP";

        if (select == FTPtransmitter.Ethernet) enterParameter(Win32.FTPETHkey, fieldName, textData);  //{ key = ↙
key + @"\ETH"; }
        else if (select == FTPtransmitter.GSM) enterParameter(Win32.FTPGSMkey, fieldName, textData);  //{ key = ↙
key + @"\GSM"; }
    }

    #endregion

    #region Setup FTP parameters for Ethernet

    /// <summary>
    /// Setup FTP parameters for Ethernet transmission
    /// </summary>
    private static void setupFTP_ETH()
    {
        bool exitLoop = false;
        string message;

        displayFTPoptionsETH();
        while (exitLoop == false)
        {
            message = readFromConsole();
            if (message.Length > 1)
            {
                switch (message)
                {
                    case "username":
                        enterFTPparameter("Username", FTPtransmitter.Ethernet);
                        displayFTPoptionsETH();
                        break;

                    case "password":
                        enterFTPparameter("Password", FTPtransmitter.Ethernet);
                        displayFTPoptionsETH();
                        break;

                    case "serverip":
                        enterFTPparameter("Server IP", FTPtransmitter.Ethernet);
                        displayFTPoptionsETH();
                        break;

                    //case "hostname":
                    //    enterFTPparameter("Hostname", FTPtransmitter.Ethernet);
                    //    displayFTPoptionsETH();
                    //    break;

                    case "exit":
                        exitLoop = true;
                        break;
                }
            }
        }
    }

    /// <summary>
    /// Display FTP setting options for Ethernet
    /// </summary>
    private static void displayFTPoptionsETH()
    {
        printToConsole("\n\n\rFTP setup for Ethernet transport:\n\r");
        displayFTPsettingsETH();

        printToConsole("Options:\r");
        printToConsole("username\r");
        printToConsole("password\r");
        printToConsole("serverip\r");
        //printToConsole("hostname\r");
        printToConsole("exit\n\r");
    }

    /// <summary>
    /// Displays current FTP settings for Ethernet transport
    /// </summary>
    private static void displayFTPsettingsETH()
    {
        //string key = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\FTP\ETH";
        string key = Win32.FTPETHkey;
```

```
            printToConsole(String.Format("username: \t{0}\r", Registry.GetValue(key, "Username", "")));
            printToConsole(String.Format("password: \t{0}\r", Registry.GetValue(key, "Password", "")));
            //printToConsole(String.Format("hostname: \t{0}\r", Registry.GetValue(key, "Hostname", "")));
            printToConsole(String.Format("serverip: \t{0}\n\r", Registry.GetValue(key, "Server IP", "")));
        }

        #endregion

        #region FTP transport options

        /// <summary>
        /// Display FTP transmission options
        /// </summary>
        private static void displayTransmissionOptions()
        {
            printToConsole("\n\n\rTransport selection options:\r");

            FTPtransport = Convert.ToInt32(Registry.GetValue(Win32.FTPkey, "Transport", 2));  //Default of 2 =
    ethernet
            if (FTPtransport == (int)FTPtransmitter.GSM) printToConsole("FTP Transport: GSM\r");
            else printToConsole("FTP Transport: Ethernet\r");

            printToConsole(String.Format("Connection attempts: {0}\n\r", Registry.GetValue(Win32.FTPkey, "Attempts",
     0)));
            printToConsole(String.Format("Connection interval: {0}\n\r", Registry.GetValue(Win32.FTPkey, "Attempts
    interval s", 30)));

            printToConsole("gsm      \t - selects GSM transmission for FTP\r");
            printToConsole("eth      \t - selects Ethernet transmission for FTP\r");
            printToConsole("attempts \t - number of transmission connection attempts\r");
            printToConsole("interval \t - number of seconds between Internet connection attempts\r");
            printToConsole("exit\n\r");
        }

        /// <summary>
        /// Selects either GSM or Ethernet transmission
        /// </summary>
        private static void selectTransmission()
        {
            bool exitLoop = false;
            string message;
            //string key = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\FTP";

            displayTransmissionOptions();
            while (exitLoop == false)
            {
                message = readFromConsole();
                if (message.Length > 1)
                {
                    switch (message)
                    {
                        case "gsm":
                            Registry.SetValue(Win32.FTPkey, "Transport", (int)FTPtransmitter.GSM);  //1 = GSM
                            displayTransmissionOptions();
                            break;

                        case "eth":
                            Registry.SetValue(Win32.FTPkey, "Transport", (int)FTPtransmitter.Ethernet);  //Default
    of 2 = ethernet
                            displayTransmissionOptions();
                            break;

                        case "attempts":
                            enterParameter(Win32.FTPkey, "Attempts", true);
                            displayTransmissionOptions();
                            break;

                        case "interval":
                            enterParameter(Win32.FTPkey, "Attempts interval s", true);
                            displayTransmissionOptions();
                            break;

                        case "exit":
                            exitLoop = true;
                            break;
                    }
                }
            }
        }

        #endregion

//**USER PARAMTER ENTRY*****************************************************************************************************
    **********

        #region User parameter entry

        /// <summary>
        /// Enter a parameter to the registry
        /// </summary>
        /// <param name="key">Key path to write to</param>
        /// <param name="fieldName">Name of field to write to</param>
        /// <param name="writeInteger">If true, convert string data entered to integer</param>
        private static void enterParameter(string key, string fieldName, bool writeInteger)
        {
            string userInput;

            printToConsole(String.Format("Current {0}: {1}\r",fieldName, Registry.GetValue(key, fieldName, "")));

            printToConsole(String.Format("Enter a new {0}, 00 to leave unchanged or empty to clear field\r",
    fieldName));
            userInput = readUserInput();
            if (userInput == "") return;
            if (userInput == "empty") userInput = "";
            if (writeInteger == false) Registry.SetValue(key, fieldName, userInput);
            else
            {
```

```
            int result;
            if (userInput == "") result = 0;
            else result = Convert.ToInt32(userInput);
            Registry.SetValue(key, fieldName, result);
        }
    }

    /// <summary>
    /// Save the registry settings to persistent memory in hive
    /// </summary>
    private static void saveRegistry()
    {
        try
        {
            printToConsole("Saving settings...This may take a few seconds\n\r");
            Win32.RegFlushKey(Win32.HKEY_LOCAL_MACHINE);
        }
        catch (Exception msg)
        {
            printToConsole(String.Format("{0}\n\r",msg.ToString()));
        }
    }

    /// <summary>
    /// Reads user input from console
    /// </summary>
    /// <returns>Text user enters, or empty if 00 typed</returns>
    private static string readUserInput()
    {
        string userData;

        while (true)
        {
            userData = readFromConsole();   //At ReadTimeout, "" is returned from UART.ReadSerialPort

            if (userData != "")
            {
                if (userData == "00") return "";
                else return userData;
            }
        }
    }
    #endregion

//**TIME METHODS*****************************************************************************************************************
    ************

    #region Date and Time methods

    /// <summary>
    /// System time and date setup
    /// </summary>
    private static void printTimeSetupScreen()
    {
        bool exitLoop = false;
        string message;

        displayTimeOptions();
        while(exitLoop == false)
        {
            message = readFromConsole();
            if (message.Length > 1)
            {
                switch(message)
                {
                    case "set time":
                        setTime();
                        displayTimeOptions();
                        break;

                    case "set date":
                        setDate();
                        displayTimeOptions();
                        break;

                    case "exit":
                        exitLoop = true;
                        break;

                    case "help":
                        displayTimeOptions();
                        break;
                }
            }
        }
    }

    /// <summary>
    /// Displays available time options
    /// </summary>
    static void displayTimeOptions()
    {
        printToConsole("\n\rTime and Date settings:\n\r");
        printToConsole(String.Format("Current time is: {0}\r", DateTime.Now));
        printToConsole("set time\r");
        printToConsole("set date\r");
        printToConsole("exit\n\r");
    }

    /// <summary>
    /// Set system time
    /// </summary>
    private static void setTime()
    {
        Win32.SYSTEMTIME NewTime = new Win32.SYSTEMTIME();
        bool ExitLoop = false;
        string UserTime;
```

```
            NewTime = Win32.ConvertDateTime(DateTime.Now);

            printToConsole(String.Format("Current time is: {0}\r",DateTime.Now));
            printToConsole("Enter a new time (hhmmss) or 00 to leave time unchanged.\r");

            ExitLoop = false;
            while (ExitLoop == false)
            {
                UserTime = readFromConsole();
                if (UserTime.Length > 1)
                {
                    if (UserTime=="00")
                    {
                        printToConsole("OK\n\r");
                        ExitLoop = true;
                    }
                    else if (UserTime.Length >= 6)
                    {
                        printToConsole(UserTime.Trim().Substring(0, 6) + "\r");
                        NewTime.Hour = Convert.ToUInt16(UserTime.Substring(0, 2).Trim());
                        NewTime.Minute = Convert.ToUInt16(UserTime.Substring(2, 2).Trim());
                        NewTime.Second = Convert.ToUInt16(UserTime.Substring(4, 2).Trim());
                        Win32.SetSystemClock(NewTime);
                        printToConsole(String.Format("New set time is: {0}\r", DateTime.Now));
                        ExitLoop = true;
                    }
                }
            }
        }

        /// <summary>
        /// Set system date
        /// </summary>
        private static void setDate()
        {
            Win32.SYSTEMTIME NewTime = new Win32.SYSTEMTIME();
            bool ExitLoop = false;
            string UserDate;

            NewTime = Win32.ConvertDateTime(DateTime.Now);

            printToConsole(String.Format("Current date is: {0}\r", DateTime.Now));
            printToConsole("Enter a new date (yyyymmdd) or 00 to leave time unchanged.\r");

            ExitLoop = false;
            while (ExitLoop == false)
            {
                UserDate = readFromConsole();
                if (UserDate.Length > 1)
                {
                    if (UserDate=="00")
                    {
                        printToConsole("OK\n\r");
                        ExitLoop = true;
                    }
                    else if (UserDate.Length >= 8)
                    {
                        printToConsole(UserDate.Trim().Substring(0, 8) + "\r");
                        NewTime = Win32.ConvertDateTime(DateTime.Now);   // Read system time again now to avoid time
    being over written incorrectly
                        NewTime.Year = Convert.ToUInt16(UserDate.Substring(0, 4).Trim());
                        NewTime.Month = Convert.ToUInt16(UserDate.Substring(4, 2).Trim());
                        NewTime.Day = Convert.ToUInt16(UserDate.Substring(6, 2).Trim());
                        Win32.SetSystemClock(NewTime);
                        printToConsole(String.Format("New set date is: {0}\r", DateTime.Now));
                        ExitLoop = true;
                    }
                }
            }
        }

        #endregion

//**STREMAING METHODS*****************************************************************************************************************
    ******************

        #region Streaming methods

        /// <summary>
        /// System Streaming settings
        /// </summary>
        private static void printStreamSetupScreen()
        {
            bool exitLoop = false;
            string message;

            displayStreamOptions();
            while (exitLoop == false)
            {
                message = readFromConsole();
                if (message.Length > 1)
                {
                    switch (message)
                    {
                        case "eth":
                            setupStream_ETH();
                            displayStreamOptions();
                            break;

                        case "exit":
                            saveRegistry();
                            exitLoop = true;
                            break;
                    }
                }
            }
```

```
        }
        /// <summary>
        /// Display FTP setting options
        /// </summary>
        private static void displayStreamOptions()
        {
            printToConsole("\n\rEthernet stream settings:\n\r");
            displayStreamSettings();

            printToConsole("eth \t - Stream settings for Ethernet transmission\r");

            printToConsole("exit\n\r");
        }

        /// <summary>
        /// Display global Stream settings
        /// </summary>
        private static void displayStreamSettings()
        {
            printToConsole(String.Format("Destination IP address: {0}\r", Registry.GetValue(Win32.StreamKey,
    "Destination IP", "0.0.0.0")));
            printToConsole(String.Format("Outgoing port: {0}\r", Registry.GetValue(Win32.StreamKey, "Out port",
    "49500")));
            printToConsole(String.Format("Incoming port: {0}\n\r", Registry.GetValue(Win32.StreamKey, "In port",
    "")));
        }

        /// <summary>
        /// Setup streaming over Ethernet connection
        /// </summary>
        private static void setupStream_ETH()
        {
            bool exitLoop = false;
            string message;

            displayStreamOptionsETH();
            while (exitLoop == false)
            {
                message = readFromConsole();
                if (message.Length > 1)
                {
                    switch (message)
                    {
                        case "destip":
                            enterParameter(Win32.StreamKey, "Destination IP", false);
                            displayStreamOptionsETH();
                            break;

                        case "outport":
                            enterParameter(Win32.StreamKey, "Out port", false);
                            displayStreamOptionsETH();
                            break;

                        case "inport":
                            enterParameter(Win32.StreamKey, "In port", false);
                            displayStreamOptionsETH();
                            break;

                        case "exit":
                            exitLoop = true;
                            break;
                    }
                }
            }
        }

        /// <summary>
        /// Displays Ethernet streaming setup options
        /// </summary>
        private static void displayStreamOptionsETH()
        {
            printToConsole("\n\n\rStreaming setup for Ethernet transport:\n\r");
            displayStreamSettingsETH();

            printToConsole("Options:\r");
            printToConsole("destip  - Destination IP address\r");
            printToConsole("outport - Outgoing port 0 to 65535, default is 49500\r");
            printToConsole("inport  - Incoming port 0 to 65535, leave empty and outport is used\r");
            printToConsole("exit\n\r");
        }

        /// <summary>
        /// Display global Stream settings
        /// </summary>
        private static void displayStreamSettingsETH()
        {
            printToConsole(String.Format("Destination IP address: {0}\r", Registry.GetValue(Win32.StreamKey,
    "Destination IP", "0.0.0.0")));
            printToConsole(String.Format("Outgoing port: {0}\r", Registry.GetValue(Win32.StreamKey, "Out port",
    "49500")));
            printToConsole(String.Format("Incoming port: {0}\n\r", Registry.GetValue(Win32.StreamKey, "In port",
    "")));
        }

        #endregion

//**UTILS METHODS*****************************************************************************************************
    **************

        #region Utilities

        /// <summary>
        /// Utilities screen
        /// </summary>
        private static void printUtilsSetupScreen()
        {
            bool exitLoop = false;
```

```
        string message;

        displayUtilsOptions();
        while(exitLoop == false)
        {
            message = readFromConsole();
            if (message.Length > 1)
            {
                switch(message)
                {
                    case "force ftp":
                        int FTPtransport = Convert.ToInt32(Registry.GetValue(Win32.FTPkey, "Transport", 2));  // ↙
Default of 2 = ethernet
                        if(FTPtransport == (int)FTPtransmitter.Ethernet)
                        {
                            FTPThread.interruptFileSend = false;
                            FTPThread.forceFileSend = true;
                        }
                        else if (GSM.GPRSmodemStatus != GSM.GPRSstatus.CONNECT && GSM.GPRSmodemStatus != GSM.    ↙
GPRSstatus.LISTEN)
                        {
                            printToConsole("Open GSM modem before forcing FTP\n\r");
                        }
                        else
                        {
                            FTPThread.interruptFileSend = false;
                            FTPThread.forceFileSend = true;
                        }
                        displayUtilsOptions();
                        break;

                    case "stop ftp":
                        FTPThread.interruptFileSend = true;
                        printToConsole("Completing current file send operation\n\r");
                        break;

                    case "open stream":
                        if (consoleInput_ == consoleInput.Ethernet)
                        {
                            SocketStream.createNewSocket();
                        }
                        else printToConsole("Type start stream\r");
                        break;

                    case "start stream":
                        UART.streamData = true;
                        printToConsole("Start data streaming\r");
                        displayUtilsOptions();
                        break;

                    case "stop stream":
                        UART.streamData = false;
                        printToConsole("Stop data streaming\r");
                        displayUtilsOptions();
                        break;

                    case "get ip":
                        printToConsole(String.Format("\nLAN IP address: {0}\r",Win32.getIPaddressLAN()));
                        printToConsole(String.Format("GSM IP address: {0}\n\r",GSM.publicIPaddress));          ↙

                        break;

                    case "open gsm":
                        GSM.openModem();
                        GSM.initGPRSserver();
                        displayUtilsOptions();
                        break;

                    case "close gsm":
                        GSM.closeModem();
                        displayUtilsOptions();
                        break;

                    case "open mote":
                        dataLogger.openMoteUART();
                        displayUtilsOptions();
                        break;

                    case "open sd":
                        Win32.openSDcard();
                        displayUtilsOptions();
                        break;

                    case "dir sd":
                        Win32.dirListing("sd");
                        displayUtilsOptions();
                        break;

                    case "dir usb":
                        Win32.dirListing("usb");
                        displayUtilsOptions();
                        break;

                    case "dir nand":
                        Win32.dirListing("nand");
                        displayUtilsOptions();
                        break;

                    case "dir log":
                        Win32.dirListing("log");
                        displayUtilsOptions();
                        break;

                    case "kill apps":
                        if (GSM.GPRSmodemStatus == GSM.GPRSstatus.CONNECT || consoleInput_ == consoleInput.UART)
                        {
                            printToConsole("Cannot stop GatewayApp. Use reboot\n\r");
```

```
                        }
                        else
                        {
                            printToConsole("Stops all GatewayApp applications");
                            Win32.killApps();
                        }
                        break;

                    case "help":
                        displayUtilsOptions();
                        break;

                    case "reboot":
                        printToConsole("System rebooting in 2s\n\r");
                        Thread.Sleep(2000);
                        Win32.systemReset();  //Hardware reset of system
                        Thread.Sleep(10000);
                        break;

                    case "exit":
                        exitLoop = true;
                        break;
                }
            }
        }
    }

    /// <summary>
    /// Displays available time options
    /// </summary>
    static void displayUtilsOptions()
    {
        printToConsole("\n\rUtilities:\n\r");
        printToConsole("force ftp \t - starts FTP upload of any compressed log files\r");
        printToConsole("stop ftp  \t - terminates current FTP session\r");

        printToConsole("\nopen stream\t - open stream socket\r");
        printToConsole("start stream\t - starts data streaming from sink node\r");
        printToConsole("stop stream\t - stops data streaming from sink node\r");
        printToConsole("get ip    \t - Displays Ethernet allocated IP address\r");

        printToConsole("\nopen gsm\t - power gsm module, start TCP server\r");
        printToConsole("close gsm \t - power down gsm module\r");
        printToConsole("open mote \t - open MOTE COM port if failed previously\r");
        printToConsole("open sd   \t - open SD card if failed previously\r");

        printToConsole("\ndir sd    \t - directory listing of sd card\r");
        printToConsole("dir nand  \t - directory listing of Nand Flash\r");
        printToConsole("dir usb   \t - directory listing of USB flash drive\r");
        printToConsole("dir log   \t - listing of data logging directory\r");

        printToConsole("\nkill apps\t - causes all instances of GatewayApp to quit\r");
        printToConsole("reboot  \t - system hardware reboot\r");

        printToConsole("exit     \t - return to main menu\n\r");
    }

    #endregion

//**USER INTERFACE THREAD*****************************************************************************************************
    *************

    #region Thread main loop

    /// <summary>
    /// Display a list of commands and settings for the gateway
    /// </summary>
    public static void printHelpScreen()
    {
        printToConsole("\n\rGateway commands list\n\r");
        printToConsole("help  \t- displays this menu\r");
        printToConsole("time  \t- set system clock\r");
        printToConsole("ftp   \t- configure FTP\r");
        printToConsole("gsm   \t- configure GSM modem\r");
        printToConsole("log   \t- configure data logging\r");
        printToConsole("stream\t- configure data streaming\r");
        printToConsole("utils \t- system utilities\r");
        printToConsole("setup \t- displays gateway configuration\r");
        printToConsole("quit  \t- terminate gateway application\n\r");
    }

    /// <summary>
    /// Start console communications thread
    /// </summary>
    public static void startCommsThread()
    {
        if (userInterfaceThreadRunning == false)
        {
            userInterface.IsBackground = true;
            userInterface.Start();
            printToConsole("User interface thread started\n\r");
        }
    }

    /// <summary>
    /// Thread to provide a user interface via Ehternet and UART consoles for the gateway
    /// </summary>
    private static void userInterfaceThread()
    {
        string message;

        userInterfaceThreadRunning_ = true;
        while (true)
        {
            message = readFromConsole();
            if (message.Length > 1) //Prevent extra CRLF chars being sent to console and serial port UART
            {
```

```
                switch (message)
                {
                    case "help":
                        printHelpScreen();
                        break;

                    case "time":
                        printTimeSetupScreen();
                        printHelpScreen();
                        break;

                    case "setup":
                        printSetupScreen();
                        printHelpScreen();
                        break;

                    case "ftp":
                        printFTPsetupScreen();
                        printHelpScreen();
                        break;

                    case "gsm":
                        printGSMsetupScreen();
                        printHelpScreen();
                        break;

                    case "log":
                        printLoggerSetupScreen();
                        printHelpScreen();
                        break;

                    case "stream":
                        printStreamSetupScreen();
                        printHelpScreen();
                        break;

                    case "utils":
                        printUtilsSetupScreen();
                        printHelpScreen();
                        break;

                    case "quit":
                        if(GSM.GPRSmodemStatus==GSM.GPRSstatus.CONNECT || consoleInput_==consoleInput.UART)
                        {
                            printToConsole("Cannot close GatewayApp. Use utils, reboot\n\r");
                        }
                        else exitApp_ = true;
                        break;
                }
            }
        }
    }
    #endregion
}
}
```

```
//FILENAME: GSM.cs
using System;
using System.IO;
using System.Collections.Generic;
using System.Text;
using System.Threading;
using Microsoft.Win32;  //For accessing the registry
using System.Net;

namespace GatewayApp
{
    public static class GSM
    {
        private static int MODEM_UART = UART.MODEM_UART;

        static MODEM GSM_modem = new MODEM();                          //Create a new instance of modem

        #region GSM interface

        /// <summary>
        /// Parameters for GSM FTP
        /// </summary>
        public static FTPparams FTPparamsGSM
        {
            get { return GSM_FTP; }
            set { GSM_FTP = value; }
        }

        /// <summary>
        /// GPRS allocated public IP address
        /// </summary>
        public static string publicIPaddress
        {
            get { return publicIPaddress_; }
        }

        #endregion

        #region FTP declarations

        public struct FTPparams
        {
            /// <summary>
            /// Remote server hostname address eg gatewayftp.dyndns.org
            /// </summary>
            public string serverHostname;                                          //Remote server address
            /// <summary>
            /// Remote server IP address after DNS resolution
            /// </summary>
            public string serverIP;                                                //Remote server IP address after
    DNS resolution
            /// <summary>
            /// FTP login username
            /// </summary>
            public string username;
            /// <summary>
            /// FTP login password
            /// </summary>
            public string password;
            /// <summary>
            /// FTP timeout
            /// </summary>
            public int timeout;
        }

        private enum FTPTYPE
        {
            BINARY = 0,
            ASCII
        };

        private static FTPparams GSM_FTP = new FTPparams();

        #endregion

        #region GPRS declarations

        public struct GPRSparams
        {
            /// <summary>
            /// APN required for incoming connections on GSM network
            /// </summary>
            public string internetAPN;

            /// <summary>
            /// Firewall IP mask to allow a range of permitted incoming IP connections
            /// </summary>
            public string firewallMask;

            /// <summary>
            /// Permit a connection from the following IP(s) address only
            /// </summary>
            public string firewallIP;

            /// <summary>
            /// Incoming IP port
            /// </summary>
            public string listeningIPport;

            /// <summary>
            /// Outgoing IP port
            /// </summary>
            public string outgoingIPport;

            /// <summary>
            /// DNS service address
```

```
        /// </summary>
        public string DNSserviceAddress;

        /// <summary>
        /// Website address of allocated public IP address check
        /// </summary>
        public string checkIPaddress;
    }

    private static GPRSparams GPRS = new GPRSparams();

    public enum GPRSstatus : int
    {
        POWER_OFF = 0,
        POWER_UP,
        CONTEXT_INIT,
        LISTEN,
        CONNECT,
        COMMAND_MODE,
        NO_CARRIER,
        FTP_INIT,
        FTP_FILE_SENDING,
        FTP_CLOSED,
        MODEM_ERROR
    };

    private static string publicIPaddress_;
    private static GPRSstatus GPRSmodemStatus_ = new GPRSstatus();
    private static GPRSstatus initialConnectionState = new GPRSstatus();

    /// <summary>
    /// Permits reading and writing status of GPRS modem
    /// </summary>
    public static GPRSstatus GPRSmodemStatus
    {
        get { return GPRSmodemStatus_; }
        set { GPRSmodemStatus_ = value; }
    }

    #endregion

    /// <summary>
    /// Initialise modem UART to 9600 baud
    /// </summary>
    /// <returns>True if opened successfully</returns>
    private static bool openModemUART()
    {
        try
        {
            if (UART.comPortOpened[MODEM_UART] == false)
            {
                GSM_modem.baud = "";
                if (UART.OpenSerialPort(MODEM_UART, 9600) == false)return false;

                GSM_modem.baud = "9600";               //Set initial baud rate
                ConsoleUserInterface.printToConsole( "Modem UART opened at 9600\n\r");
            }
            else        //comPort is already opened, so reset the baud rate to 9600 for modem startup
            {
                UART.changeBaud(MODEM_UART, 9600);
            }
        }
        catch (Exception msg)
        {
            ConsoleUserInterface.printToConsole( String.Format("Error in GSM.openModemUART: {0}\r\n", msg.
Message.ToString()));
        }
        return true;
    }

    /// <summary>
    /// Init FTP parameters from registry
    /// </summary>
    private static void initFTPparameters()
    {
        //string key = @"HKEY_LOCAL_MACHINE\SOFTWARE\WSN_Gateway\GatewayApp\FTP\GSM";
        string key = Win32.FTPGSMkey;

        try
        {
            GSM_FTP.username = Convert.ToString(Registry.GetValue(key, "Username", ""));
            GSM_FTP.password = Convert.ToString(Registry.GetValue(key, "Password", ""));
            GSM_FTP.serverHostname = Convert.ToString(Registry.GetValue(key, "Hostname", ""));
            GSM_FTP.serverIP = Convert.ToString(Registry.GetValue(key, "Server IP", ""));
        }
        catch (Exception msg)
        {
            ConsoleUserInterface.printToConsole(String.Format("Exception in initFTPparameters: {0}\n\r", msg.
ToString()));
        }
    }

    /// <summary>
    /// Init GSM module parameters from registry
    /// </summary>
    private static void initGSMmoduleParameters()
    {
        string key = Win32.GSMkey;

        GSM_modem.type = Convert.ToString(Registry.GetValue(key, "Module type", "Telit"));
        GSM_modem.powerPortBank = Convert.ToString(Registry.GetValue(key, "Power IO bank", "B"));
        GSM_modem.powerIO = Convert.ToInt32( Registry.GetValue(key, "Power IO number", "20"));
        GSM_modem.resetPortBank = Convert.ToString(Registry.GetValue(key, "Reset IO bank", "B"));
        GSM_modem.resetIO = Convert.ToInt32( Registry.GetValue(key, "Reset IO number", "21"));

        //Number of times a command is sent to the modem in the event of comms issues
        GSM_modem.commsAttempts = Convert.ToInt32(Registry.GetValue(key, "Resend attempts", 5));
```

```
        //Interval between comms attempts
        GSM_modem.attemptInterval = Convert.ToInt32(Registry.GetValue(key, "Interval ms", 5000));
        //Time allowed for response on UART from modem before exception is thrown
        GSM_modem.readbyteWait = Convert.ToInt32(Registry.GetValue(key, "Read timeout ms", 60000));
        //Time allowed in s for data in module buffer to be sent after last Xon is received by Windows
        GSM_modem.transmitTimer = Convert.ToInt32(Registry.GetValue(key, "Transmit timeout s", 20));
    }

    /// <summary>
    /// Init GPRS paramters from registry
    /// </summary>
    private static void initGPRSparams()
    {
        string key = Win32.GPRSkey;

        GPRS.outgoingIPport = Convert.ToString(Registry.GetValue(key, "Out port", ""));
        GPRS.listeningIPport = Convert.ToString(Registry.GetValue(key, "In port", "49500"));
        GPRS.firewallMask = Convert.ToString(Registry.GetValue(key, "Firewall mask", "255.255.0.0"));
        GPRS.firewallIP = Convert.ToString(Registry.GetValue(key, "Firewall IP", "0.0.0.0"));
        GPRS.internetAPN = Convert.ToString(Registry.GetValue(key, "Internet APN", "unrestricted"));
        GPRS.DNSserviceAddress = Convert.ToString(Registry.GetValue(key, "Update DNS", "www.dyndns.org"));     /↙
/DNS server address - update with public IP
        GPRS.checkIPaddress = Convert.ToString(Registry.GetValue(key, "Check IP address", "checkip.dyndns.org"))↙
;   //Address to ping to get public IP address

        //These need to be added to the menu structure
        //GPRS.DNSserviceAddress = "";
        //GPRS.firewallMask = "255.255.0.0";
        GPRS.firewallIP = "41.246.25.38";  //Home IP address
        //GPRS.firewallIP = "155.238.33.36";   //Tech PC
        //GPRS.internetAPN = "unrestricted";
        //GPRS.listeningIPport = "49500";
        //GPRS.outgoingIPport = "";
    }

    /// <summary>
    /// Manages connection status of modem.  Called by GatewayEgine, runningLoop();
    /// </summary>
    /// <returns></returns>
    public static bool modemConnectionManager()
    {
        //Once FTP of files is complete, return the modem to connection state prior to starting FTP
        if (GPRSmodemStatus_ == GPRSstatus.FTP_CLOSED)
        {
            if (enableSocket() == false) return false;
            if (GPRSmodemStatus_ == GPRSstatus.CONNECT)
            {
                ConsoleUserInterface.printToConsole(String.Format("\n\rSystem time is: {0}\n\r", DateTime.Now));
                ConsoleUserInterface.printToConsole("FTP of logfiles via GPRS complete\r");
                ConsoleUserInterface.printToConsole("GPRS link opened\n\r");
            }
        }

        //Remote client has closed the connection, or network aborted the connection. Re-establish LISTEN mode   ↙
for new connection
        if (GPRSmodemStatus_ == GPRSstatus.NO_CARRIER)
        {
            if (configureListeningSocket() == false)
            {
                Thread.Sleep(1000);     //Wait for a while to prevent runaway comms with modem - this function   ↙
called by runningloop()
                ConsoleUserInterface.printToConsole("Modem error, cannot re-establish LISTEN mode\n\r");
                return false;
            }
            GPRSmodemStatus_ = GPRSstatus.LISTEN;
            ConsoleUserInterface.printToConsole("Modem in LISTEN mode\n\r");
        }
        return true;
    }

    /// <summary>
    /// Open modem UART, Power up and initialise modem
    /// </summary>
    /// <returns>True if modem initialise was OK</returns>
    public static bool openModem()
    {
        initGSMmoduleParameters();

        ConsoleUserInterface.printToConsole("Cycling comms module power\n\r");
        GPIO.setGPIO("B", 10, false);
        Thread.Sleep(1500);
        GPIO.setGPIO("B", 10, true);    //Cycle GSM communications board power
        Thread.Sleep(1500);

        if (openModemUART() == false) return false;

        ConsoleUserInterface.printToConsole("Powering modem\n\r");
        GPRSmodemStatus_ = GPRSstatus.POWER_UP;
        GPIO.setGPIO(GSM_modem.powerPortBank, GSM_modem.powerIO, false);
        Thread.Sleep(1000);
        GPIO.setGPIO(GSM_modem.powerPortBank, GSM_modem.powerIO, true);
        Thread.Sleep(15000);                    //Wait 15s for modem to powerup and registeron the network      ↙

        if (testModemComms() == false)          //Check modem comms is functional at 9600bps                     ↙
        {
            //Modem must be reset. Power down, exit this loop and allow modem comms to start again
            if (closeModem() == false)              //Attempt software power down of modem
            {
                //Force hardware power down
                GPIO.setGPIO(GSM_modem.powerPortBank, 10, false);
                Thread.Sleep(1500);
                GPIO.setGPIO(GSM_modem.powerPortBank, 10, true);
            }
            return false;
        }
```

```
        if (changeModemBaud(57600) == false)    //Up the comms baud rate to the modem
        {
            //Modem must be reset. Power down, exit this loop and allow modem comms to start again
            closeModem();                       //Attempt software power down of modem
            return false;
        }
        return true;
    }

    /// <summary>
    /// Configures modem to act as TCP/IP server for incoming IP connection
    /// </summary>
    public static bool initGPRSserver()
    {
        startModemDataReceiverThread();     //Starts thread to monitor modem for SRING response to outside
connections

        GPRSmodemStatus_ = GPRSstatus.CONTEXT_INIT;
        if (definePDPcontext() == false)    //Create PDP context for FTP and GPRS transfers
        {
            closeModem();
            return false;
        }

        if (configureGPRSsockets() == false) //Configures a second socket for GPRS transfers
        {
            closeModem();
            return false;
        }

        if (configureSocketAutoanswer() == false)    //Configures modem to autoanswer remote connection requests
        {
            closeModem();
            return false;
        }

        if (activatePDPcontext() == false)  //Activate PDP context
        {
            closeModem();
            return false;
        }

        if (configureFirewall() == false)   //Configure firewall
        {
            closeModem();
            return false;
        }

        if(configureListeningSocket()==false)   //Open a listening socket on port xxx
        {
            closeModem();
            return false;
        }
        GPRSmodemStatus_ = GPRSstatus.LISTEN;
        return true;
    }

    /// <summary>
    /// Separate thread in UART class to handle handshaking in FTP session, and receive connect requests from
remote users
    /// </summary>
    private static void startModemDataReceiverThread()
    {
        if (UART.modemHandshakeThreadRunning == false)          //Only start thread if not running, else
exception
        {
            UART.modemHandshakeThread.IsBackground = true;
            UART.modemHandshakeThread.Start();                  //Start modem handshake receiver thread

        }
    }

    /// <summary>
    /// Establish a GPRS context and FTP connection with remote server - called from FTPThread
    /// </summary>
    /// <returns>True if successful, else false</returns>
    public static bool openFTPlink()
    {
        initFTPparameters();    //Init parameters from registry

        //Make a copy of the original connection status before starting FTP to reinstate connection state after
FTP
        initialConnectionState = GPRSmodemStatus_;

        //Check on current modem connection satus, prepare for FTP
        if (shutdownSocket() == false)
        {
            return false;
        }

        GPRSmodemStatus_ = GPRSstatus.FTP_INIT;
        if (setModemHandshaking() == false) return false;     //Set DTE to modem handshake: XonXoff, modem to
DTE handshake: none

        if (setFTPtimeout(600) == false) return false;                      //Set FTP timeout to 60s

        if (openFTP() == false) return false;                               //Open FTP connection with remote
server

        if (setFTPtype(FTPTYPE.BINARY) == false) return false;              //Set the FTP trasmission type

        return true;
    }

    private static bool openFTP()
```

```
    {
        string response="";
        try
        {
            string temp;
            if (GSM_FTP.serverHostname == "") temp = GSM_FTP.serverIP;
            else temp = GSM_FTP.serverHostname;

            if (sendModemCommand(String.Format("AT#FTPOPEN=\"{0}\",\"{1}\",\"{2}\",0",temp, GSM_FTP.username,
GSM_FTP.password), ref response, "") == false)
            {
                ConsoleUserInterface.printToConsole( "Error resolving server IP address or comms timeout\n\r");
                return false;
            }
            return true;
        }
        catch
        {
            return false;
        }
    }

    /// <summary>
    /// Creates a new file on remote server and starts file upload
    /// </summary>
    /// <param name="newFileName">Name of file to upload</param>
    /// <returns>True if file creation is successful, else false</returns>
    public static bool putFTPfile(string newFileName)
    {
        GPRSmodemStatus_ = GPRSstatus.FTP_INIT;
        if (sendModemCommand(String.Format("AT#FTPPUT=\"{0}\"", newFileName), "CONNECT") == false)
        {
            //An FTP connection exists, but a new file can't be created on server, so send close FTP command.

            closeFTP();
            return false;
        }
        Thread.Sleep(2000);
        return true;
    }

    /// <summary>
    /// Closes an open FTP connection
    /// </summary>
    /// <returns>True if successful, else false</returns>
    public static bool closeFTP()
    {
        if (disconnectingFTPsession() == false) return false;

        if (clearModemHandshaking() == false) return false;

        GPRSmodemStatus_ = GPRSstatus.FTP_CLOSED;

        return true;
    }

    /// <summary>
    /// Disconnects modem from FTP server
    /// </summary>
    /// <returns></returns>
    private static bool disconnectingFTPsession()
    {
        int retries;

        retries = GSM_modem.commsAttempts;
        do
        {
            if (sendModemCommand("AT#FTPCLOSE", "") == false)
            {
                ConsoleUserInterface.printToConsole("Cannot close FTP connection, try sending escape sequence\n\
r");
            }
            else
            {
                ConsoleUserInterface.printToConsole("FTP connection closed\n\r");
                Thread.Sleep(2000);
                return true;
            }
            Thread.Sleep(2000);
            retries--;
            if (retries == 0)
            {
                ConsoleUserInterface.printToConsole("Cannot close FTP connection or return to command mode\n\r")
;
                GPRSmodemStatus_ = GPRSstatus.MODEM_ERROR;
                return false;
            }
        } while (true);
    }

    /// <summary>
    /// Sends +++ escape characters to return modem to command mode.  Default time between sends is 1s
    /// Sets GPRSstatus.MODEM_ERROR flag
    /// </summary>
    /// <returns></returns>
    public static bool sendEscapeSequence()
    {
        string modemResponse;
        int retries;

        if(GSM_modem.commsAttempts==0)GSM_modem.commsAttempts=5;
        retries = 0;
        ConsoleUserInterface.printToConsole( "\rSending escape sequence +++\n\r");

        Thread.Sleep(2000);

        do
```

```
        {
            modemResponse = UART.sendEscapeSequence(100);
            ConsoleUserInterface.printToConsole( String.Format("Response: {0}\r", modemResponse));
            if(Contains(modemResponse,"CARRIER")==true)
            {
                ConsoleUserInterface.printToConsole( "Modem put into command mode\n\r");
                return true;
            }
            Thread.Sleep(10000);    //Wait 10s between each resend
            retries++;
            ConsoleUserInterface.printToConsole( String.Format( "Attempt {0}: Sending +++ again\n\r",retries));
            if (retries == GSM_modem.commsAttempts)
            {
                ConsoleUserInterface.printToConsole( "Cannot put modem into command mode\n\r");
                GPRSmodemStatus_ = GPRSstatus.MODEM_ERROR;
                return false;
            }
        } while (true);
    }

    /// <summary>
    /// Set FTP timeout value
    /// </summary>
    /// <param name="ms">Time in 100ms intervals. 0 leaves modem timeout default.</param>
    /// <returns></returns>
    private static bool setFTPtimeout(int time_ms)
    {
        if (time_ms == 0) return true;

        if (sendModemCommand(String.Format("AT#FTPTO={0}",time_ms), "") == false)
        {
            ConsoleUserInterface.printToConsole( "Cannot write FTP timeout\n\r");
            return false;
        }
        ConsoleUserInterface.printToConsole( String.Format( "FTP timeout set to {0}s\n\r",time_ms/10));
        Thread.Sleep(2000);
        return true;
    }

    /// <summary>
    /// Sets type of FTP transmission
    /// </summary>
    /// <param name="type">Binary=0, ASCII=1</param>
    /// <returns>True if successful, else false</returns>
    private static bool setFTPtype(FTPTYPE type)
    {
        if (sendModemCommand(String.Format("AT#FTPTYPE={0}", Convert.ToByte(type)), "") == false)
        {
            ConsoleUserInterface.printToConsole("Cannot write FTP type\n\r");
            return false;
        }
        ConsoleUserInterface.printToConsole(String.Format("FTP type is {0}\n\r", type.ToString()));
        Thread.Sleep(2000);
        return true;
    }

    /// <summary>
    /// Creates PDP context for FTP and GPRS transfers on particular APN
    /// </summary>
    /// <returns></returns>
    public static bool definePDPcontext()
    {
        initGPRSparams();

        if (sendModemCommand(String.Format("AT+CGDCONT=1,\"IP\",\"{0}\",\"0.0.0.0\",{1},{2}",GPRS.internetAPN,0, ↙
0),"") == false)        //Activate  PDP context 1
        {
           ConsoleUserInterface.printToConsole("Cannot activate PDP context\n\r");
           return false;
        }
        Thread.Sleep(2000);

        //ConsoleUserInterface.printToConsole(String.Format("AT+CGDCONT=1,\"IP\",\"{0}\",\"0.0.0.0\",{1},{2}",   ↙
GPRS.internetAPN, 0, 0));
        return true;
    }

    /// <summary>
    /// Configures a second socket in the PDP context for GPRS transfers
    /// </summary>
    /// <returns></returns>
    public static bool configureGPRSsockets()
    {
        if (sendModemCommand("AT#SCFG=2,1,1000,0,600,50", "") == false)        //Activate  PDP context 1
        {
            ConsoleUserInterface.printToConsole("Cannot configure GPRS sockets\n\r");
            return false;
        }
        Thread.Sleep(100);
        return true;
    }

    /// <summary>
    /// Activates PDP context required to start a GPRS data connection
    /// </summary>
    /// <returns></returns>
    private static bool activatePDPcontext()
    {
        string response = "";

        if (sendModemCommand("AT#SGACT=1,1", ref response, "") == false)        //Activate  PDP context 1
        {
            ConsoleUserInterface.printToConsole("Cannot activate PDP context\n\r");
            return false;
        }

        if (sendModemCommand("AT#SGACT?", "1,1") == false)       //Check that context is activated
```

```
        {
            ConsoleUserInterface.printToConsole( "Cannot activate PDP context\n\r");
            return false;
        }

        int startIndex, lastIndex, length;
        startIndex = response.IndexOf((char)0x20) + 1;
        lastIndex = response.IndexOf((char)0x0A);
        length = lastIndex - startIndex;
        publicIPaddress_ = response.Substring(startIndex, length);
        ConsoleUserInterface.printToConsole(String.Format( "PDP context 1 activated\rIP adress allocated is {0}" ↙
, publicIPaddress_));

        Thread.Sleep(1000);
        return true;
    }

    /// <summary>
    /// Configures modem firewall
    /// </summary>
    /// <returns></returns>
    private static bool configureFirewall()
    {
        if (sendModemCommand(String.Format("AT#FRWL=1,\"{0}\",\"{1}\"",GPRS.firewallIP,GPRS.firewallMask),"") == ↙
 false)        //Setup firewall to accept incoming connections
        {
            ConsoleUserInterface.printToConsole("Cannot confgiure firewall\n\r");
            return false;
        }
        return true;
    }

    /// <summary>
    /// Configures listening socket 2 on select port
    /// </summary>
    /// <returns></returns>
    private static bool configureListeningSocket()
    {
        if (sendModemCommand(String.Format("AT#SL=2,1,{0},0",GPRS.listeningIPport),"") == false)        //Setup ↙
firewall to accept incoming connections
        {
            ConsoleUserInterface.printToConsole(String.Format("Cannot confgiure listening socket on port {0}\n\r ↙
",GPRS.listeningIPport));
            return false;
        }
        return true;
    }

    /// <summary>
    /// Configures the modem to automatically connect to remote user
    /// </summary>
    /// <returns></returns>
    private static bool configureSocketAutoanswer()
    {
        if (sendModemCommand("AT#SCFGEXT=2,0,0,0,1,0","") == false)
        {
            ConsoleUserInterface.printToConsole("Cannot confgiure socket to auto answer\n\r");
            return false;
        }
        return true;
    }

    /// <summary>
    /// Shutdown socket to place modem into command mode before FTP starts
    /// </summary>
    /// <returns></returns>
    private static bool shutdownSocket()
    {
        if (GPRSmodemStatus_ == GPRSstatus.LISTEN)  //Modem in LISTEN mode
        {
            GPRSmodemStatus_ = GPRSstatus.COMMAND_MODE;
            Thread.Sleep(50);    //Wait for Modem UART timeout to finish in UART.modemDataReceiver()

            //Close socket 2 for listening
            if (sendModemCommand("AT#SL=2,0", "") == false)
            {
                ConsoleUserInterface.printToConsole("Cannot shutdown socket\n\r");
                GPRSmodemStatus_ = GPRSstatus.MODEM_ERROR;
                return false;
            }
        }  //Send escape sequence to put modem into command mode from CONNECT state
        else if (GPRSmodemStatus_ == GPRSstatus.CONNECT)
        {
            GPRSmodemStatus_ = GPRSstatus.COMMAND_MODE;
            Thread.Sleep(50);   //Wait for Modem UART timeout to finish in UART.modemDataReceiver()

            //Send escape sequence
            UART.sendEscapeSequence(100, true);
        }
        return true;
    }

    /// <summary>
    /// Enables modem socket for CONNECT or LISTEN state depending on state before FTP commenced
    /// </summary>
    private static bool enableSocket()
    {
        if (initialConnectionState == GPRSstatus.LISTEN)
        {
            if (configureListeningSocket() == false)
            {
                GPRSmodemStatus_ = GPRSstatus.MODEM_ERROR;
                return false;
            }
            GPRSmodemStatus_ = GPRSstatus.LISTEN;
        }
        else if (initialConnectionState == GPRSstatus.CONNECT)
```

```
        {
            if (sendModemCommand("AT#SO=2", "CONNECT") == false)
            {
                ConsoleUserInterface.printToConsole("Cannot resume socket connection\n\r");
                GPRSmodemStatus_ = GPRSstatus.MODEM_ERROR;
                return false;
            }
            GPRSmodemStatus_ = GPRSstatus.CONNECT;
        }
        return true;
    }

    /// <summary>
    /// Checks for presence of SIM card in module
    /// </summary>
    /// <returns>True if present, else false</returns>
    private static bool testSIM()
    {
        if (sendModemCommand("AT+CPIN?", "READY") == true)
        {
            ConsoleUserInterface.printToConsole( "SIM card READY\n\r");
        }
        else
        {
            ConsoleUserInterface.printToConsole( "SIM card not found\n\r");
            return false;
        }
        Thread.Sleep(3000);
        return true;
    }

    /// <summary>
    /// First version of sendModemCommand without returning with modem response
    /// </summary>
    /// <param name="command"></param>
    /// <param name="search"></param>
    /// <param name="sendOnceOnly"></param>
    /// <returns></returns>
    public static bool sendModemCommand(string command, string search)
    {
        string dummy="";

        return sendModemCommand(command,ref dummy,search);
    }

    /// <summary>
    /// Sends a command to the modem and tries several times before giving up if comms error
    /// </summary>
    /// <param name="command">AT command to send</param>
    /// <param name="response">Response string sent by modem</param>
    /// <param name="search">String to search for in modem response. An empty string defaults search to OK. ERROR always searched for</param>
    /// <param name="sendOnceOnly">True to send command only once to allow for extended times for network responses</param>
    /// <returns>True if coms successful, else false</returns>
    private static bool sendModemCommand(string command, ref string response, string search)
    {
        int commsAttempts;
        string modemResponse;

        try
        {
            if (GSM_modem.commsAttempts == 0) GSM_modem.commsAttempts = 5;
            modemResponse = "";
            commsAttempts = 0;

            while (true)
            {
                ConsoleUserInterface.printToConsole( String.Format("Sending {0}\r", command));
                modemResponse = UART.sendCommand(command, GSM_modem.readbyteWait);  //Waits in this method until response received, or ReadOut timer occured
                response = modemResponse;   //Return the response from the modem to the calling method

                ConsoleUserInterface.printToConsole(String.Format( "Response: {0}\r",modemResponse));

                if (search != "")   //Only search for special responses if search string supplied, else default to ERROR or OK
                {
                    if(Contains(modemResponse,search.Trim())==true)
                    {
                        Thread.Sleep(50);
                        return true;
                    }
                }
                else if(Contains(modemResponse,"OK")==true)
                {
                    Thread.Sleep(50);
                    return true;
                }
                //if(Contains(modemResponse,"ERROR")==true)
                //{
                //    Thread.Sleep(50);
                //    return false;
                //}

                //If kill apps flag is set in the registry by user, return to caller
                if (Win32.testForKillApps() == true) return false;
                Thread.Sleep(GSM_modem.attemptInterval);
                commsAttempts++;
                if (commsAttempts >= GSM_modem.commsAttempts) throw new Exception("No comms with modem, or search string not found");
                ConsoleUserInterface.printToConsole( String.Format("Waiting for modem response.  Attempting resend {0}\n\r",commsAttempts));
                Thread.Sleep(0);
            }
        }
        catch (Exception msg)
```

```
        {
            ConsoleUserInterface.printToConsole( String.Format("Error in GSM.sendModemCommand(): {0}\r\n", msg.Message.ToString()));
            return false;
        }
    }

    /// <summary>
    /// Searches a string for a substring
    /// </summary>
    /// <param name="stringToSearch"></param>
    /// <param name="searchString"></param>
    /// <returns>True if found, else false</returns>
    private static bool Contains(string stringToSearch, string searchString)
    {
        if (stringToSearch.IndexOf(searchString) == -1) return false;
        else return true;
    }

    /// <summary>
    /// Check communications with modem at 9600bps
    /// </summary>
    /// <returns>True if modem comms OK, else false</returns>
    private static bool testModemComms()
    {
        if (sendModemCommand("ATE0","") == true)
        {
            ConsoleUserInterface.printToConsole( "Comms with modem OK\n\r");
            Thread.Sleep(2000);
            return true;
        }
        else return false;
    }

    /// <summary>
    /// Set DTE to modem data transfer handshaking to XonXoff, and modem to DTE handshaking to none
    /// </summary>
    /// <returns>True if successful, else false</returns>
    private static bool setModemHandshaking()
    {
        if (sendModemCommand("AT+IFC=0,1", "") == false)
        {
            ConsoleUserInterface.printToConsole( "Cannot set Modem handshaking\n\r");
            return false;
        }
        ConsoleUserInterface.printToConsole( "DTE to Modem handshaking set to XonXoff\n\r");
        Thread.Sleep(100);
        return true;
    }

    /// <summary>
    /// Clears all modem handshaking
    /// </summary>
    /// <returns>True if successful, else false</returns>
    private static bool clearModemHandshaking()
    {
        if (sendModemCommand("AT+IFC=0,0", "") == false)
        {
            ConsoleUserInterface.printToConsole("Cannot reset modem handshaking\n\r");
            return false;
        }
        ConsoleUserInterface.printToConsole("Modem handshaking reset\n\r");
        Thread.Sleep(100);
        return true;
    }

    /// <summary>
    /// Changes modem comms baud rate and MODEM_UART baud rate
    /// </summary>
    /// <param name="baud">Baud rate</param>
    /// <returns>True if modem and UART baud rates changed successfully, else false</returns>
    private static bool changeModemBaud(int baud)
    {
        if (sendModemCommand(String.Format("AT+IPR={0}",baud),"") == true)ConsoleUserInterface.printToConsole(String.Format("Modem baud rate set to {0}\n\r",baud));
        else
        {
            ConsoleUserInterface.printToConsole( "Cannot set Modem baud rate\n\r");
            return false;
        }

        if (UART.changeBaud(MODEM_UART, baud) == true)ConsoleUserInterface.printToConsole( String.Format( "UART baud rate set to {0}\n\r",baud));
        else
        {
            ConsoleUserInterface.printToConsole( "Cannot set UART baud rate\n\r");
            return false;
        }
        Thread.Sleep(2000);
        return true;
    }

    /// <summary>
    /// Power down the modem and reset MODEM UART to 9600bps for next modem startup
    /// </summary>
    /// <returns></returns>
    public static bool closeModem()
    {
        if (UART.comPortOpened[MODEM_UART] == false) return true;   //Return if COM port not opened

        if (GPRSmodemStatus_ == GPRSstatus.POWER_OFF) return true;
        GPRSmodemStatus_ = GPRSstatus.POWER_OFF;
        if (sendModemCommand("AT#SHDN","") == false)
        {
            ConsoleUserInterface.printToConsole( "Cannot shut modem down, cycling communications module power\n\r");
            GPIO.setGPIO("B",10 , false);
```

```
                Thread.Sleep(1500);
                GPIO.setGPIO("B", 10, true);
            }
            if (UART.changeBaud(MODEM_UART, 9600) == true) ConsoleUserInterface.printToConsole( "UART baud rate set ↙
    to 9600\n\r");
            else
            {
                ConsoleUserInterface.printToConsole( "Cannot set UART baud rate\n\r");
                return false;
            }
            return true;
        }
    }
}
```

```
//FILENAME: Logging.cs
using System;
using System.Collections.Generic;
using System.IO;
using System.Text;
using System.Threading;
using Microsoft.Win32;

namespace GatewayApp
{
    public class DataRecorder
    {
        public enum LogMedia : byte
        {
            SDCard = 0,
            NandFlash,
            USBstick
        };

        public enum LogError : byte
        {
            NoError = 0,
            LoggingAlreadyStarted,
            VolumeDoesNotExist,
            InsufficientSpaceOnVolume,
            InvalidMediaType,
            MoteCOMportNotOpen,
            CannotStartLogging
        };

        private string currentPath;                    //Path to the selected storage volume
        private int MOTE_UART = UART.MOTE_UART;

        /// <summary>
        /// Opens Vurtual UART to interface to MOTE
        /// </summary>
        /// <returns></returns>
        public bool openMoteUART()
        {
            if (UART.comPortOpened[MOTE_UART] == true)
            {
                ConsoleUserInterface.printToConsole("MOTE UART COM0 already opened\r");
                return true;
            }

            if (UART.OpenSerialPort(MOTE_UART, 115200) == true)
            {
                UART.comPortOpened[MOTE_UART] = true;
                ConsoleUserInterface.printToConsole("MOTE UART COM0 opened successfully\r");
                return true;
            }
            else
            {
                UART.comPortOpened[MOTE_UART] = false;
                ConsoleUserInterface.printToConsole("Cannot open MOTE UART.\n\rCheck MOTE is connected\n\rNo data logging allowed, only system setup\n\r");
                return false;
            }
        }

        /// <summary>
        /// Close Mote UART
        /// </summary>
        public void closeMoteUART()
        {
            if( UART.comPortOpened[MOTE_UART] == true)ConsoleUserInterface.printToConsole("Closing MOTE UART\r");
            UART.CloseSerialPort(0);
        }

        /// <summary>
        /// setupLogging() checks if logging has already started, media type and capacity available,
        /// creates logging directory, and if current logging date differs from system date,
        /// creates a new logging file if one does not exist or the system date has changed,
        /// starts logging thread.
        /// </summary>
        /// <returns>LogError message</returns>
        public LogError setupLogging()
        {
            bool volumeExists=false;
            bool enoughVolumeCapacity=false;
            string loggingMedia;                    //Device that stores logged data
            string loggingDir;                      //Logging directory
            int enableLogging;                       //Registry value to start/pause logging

            try
            {
                enableLogging = Convert.ToInt32(Registry.GetValue(Win32.LoggerKey, "Logger enabled", 0));
                if (enableLogging != 1 || UART.comPortOpened[MOTE_UART]==false)
                {
                    LoggingThread.pauseLogging = true;  //Ensure that logging pauses
                    return LogError.NoError;            //Logging not enabled in registry, return
                }

                //Check that logging is not already underway.  Exit function if this is true.
                if (LoggingThread.pauseLogging == false)
                {
                    return LogError.LoggingAlreadyStarted;
                }

                //Read setup in registry and check if storage medium is accessible/exists and there is space available to perform log
                loggingMedia = Convert.ToString(Registry.GetValue(Win32.LoggerKey, "Media", ""));

                switch (loggingMedia)
                {
                    case "SDcard":
                        currentPath = @"\SDcard\";
```

```
                    break;

                case "NandFlash":
                    currentPath = @"\NandFlash\";
                    break;

                case "USBstick":
                    currentPath = @"\Hard Disk\";
                    break;

                default:
                    //setupLogging = LogError.InvalidMediaType;
                    //throw new Exception("Invalid storage media type.");
                    return LogError.InvalidMediaType;
            }

            Win32.doesVolumeExist(currentPath, ref volumeExists, 1, ref enoughVolumeCapacity);
            if (volumeExists == false)
            {
                LoggingThread.pauseLogging = true;  //Ensure that logging stops
                Registry.SetValue(Win32.LoggerKey, "Logger enabled", 0);
                return LogError.VolumeDoesNotExist; //0 bytes available if volume does not exist
            }

            if (enoughVolumeCapacity == false)
            {
                LoggingThread.pauseLogging = true;  //Ensure that logging stops
                Registry.SetValue(Win32.LoggerKey, "Logger enabled", 0);
                return LogError.InsufficientSpaceOnVolume;
            }
            //Read name of logging directory from registry
            //If not empty string, check if dir exists, or create no logging dir
            //Update currentPath to include logging directory
            loggingDir = Convert.ToString(Registry.GetValue(Win32.LoggerKey, "Directory", ""));
            if (loggingDir != "")
            {
                currentPath = String.Format("{0}{1}\\", currentPath, loggingDir);
                if (System.IO.Directory.Exists(currentPath) == false)
                {
                    System.IO.Directory.CreateDirectory(currentPath);
                }
            }

            //Start storing data in logfile on storage volume
            if (startLogging() == false) return LogError.CannotStartLogging;
            return LogError.NoError;
        }
        catch (Exception)
        {
            return LogError.CannotStartLogging;
        }
    }

    private bool startLogging()
    {
        //Start the logging thread
        try
        {
            LoggingThread.currentPath = currentPath;
            LoggingThread.pauseLogging = false;
            LoggingThread.dataLoggingThread.IsBackground = true;
            if (LoggingThread.loggingThreadRunning == false) LoggingThread.dataLoggingThread.Start();
            return true;
        }
        catch (Exception)
        {
            return false;
        }
    }
  }
}
```

```
//FILENAME: LoggingThread.cs
using System;
using System.Threading;
using System.Collections.Generic;
using System.IO;
using System.IO.Compression;    //Implement file compression before FTP - see GZipStream class
using System.Text;
using Microsoft.win32;

namespace GatewayApp
{
    public static class LoggingThread
    {
        private static bool pauseLogging_ = true;               //Data logging control
        private static int MOTE_UART_DataOut;                   //Pointer to data being read out of UART ring buffer


        private static bool threadRunning = false;              //Is true when trhead is started

        private static string currentLogfileName_;              //Logfile presently being written to
        private static string currentPath_;                     //Path to the selected storage volume

        public static Thread dataLoggingThread = new Thread(storeData);   //Stores data in volume as it's received
    from UART

        #region Properties

        public static string currentLogFileName
        {
            get { return currentLogfileName_; }
        }
        public static string currentPath
        {
            set { currentPath_ = value; }
            get { return currentPath_; }
        }

        /// <summary>
        /// Read: true, logging thread is running. Do not start thread again
        /// Write: false, exit logging thread
        /// </summary>
        public static bool loggingThreadRunning
        {
            get { return threadRunning; }
            set { threadRunning = value; }
        }

        /// <summary>
        /// True continues logging, else pauses logging
        /// </summary>
        public static bool pauseLogging
        {
            get { return pauseLogging_; }
            set { pauseLogging_ = value; }
        }

        #endregion

        /// <summary>
        /// Thread created to store received data in selected storage volume
        /// </summary>
        public static void storeData()
        {
            StreamWriter logFile;
            DateTime currentLogDate = DateTime.Now;          //Logging date used to create current logfile


            Win32.SYSTEMTIME timeNow;
            bool volumeExists = false;
            bool enoughVolumeCapacity = false;
            int x;
            int packetCounter = 0;
            bool exitLoop = false;
            bool waitForNextPacket = false;
            int syncBytePointer;
            int remainingBytes;
            int MAX_IN_BUFFER_SIZE = UART.MAX_IN_BUFFER_SIZE;
            int localPacketLength;
            StringBuilder sb = new StringBuilder("", 10000);
            int sbLength = sb.Length;

            //Logging is starting, so create a logfile on selected media if necessary
            CreateNewLogFile(currentPath_, ref currentLogfileName_, ref currentLogDate);

            MOTE_UART_DataOut = UART.MOTE_UART_DataIn;       //Ensure that data being read from the InBuffer is the
    newest available
            pauseLogging_ = false;
            threadRunning = true;
            while (threadRunning == true)
            {
                if (pauseLogging_ == false)
                {
                    //Check if current logging date is the same as system date
                    if (Win32.IsDateDifferent(currentLogDate) == true)
                    {
                        //If logging date is different, create new file if logfile does not exist
                        CreateNewLogFile(currentPath_, ref currentLogfileName_, ref currentLogDate);
                    }

                    try
                    {
                        //Check that there is data in the buffer to be written to persistent storage
                        if (MOTE_UART_DataOut < UART.MOTE_UART_DataIn || UART.pointerWrapped == true)
                        {
                            //ConsoleUserInterface.printToConsole(String.Format("maxInBuffer {0} DataIn {1} DataOut
    {2}\n\r", MAX_IN_BUFFER_SIZE, UART.MOTE_UART_DataIn, MOTE_UART_DataOut));
                            if (UART.pointerWrapped == false)
                            {
```

```
                    MAX_IN_BUFFER_SIZE = UART.MAX_IN_BUFFER_SIZE;
                    //Check that there is at least one packet + 10 bytes available in the buffer to be processed
                    if ((UART.MOTE_UART_DataIn - MOTE_UART_DataOut) < (UART.packetLength + 10))
                    {
                        //remainingBytes = UART.MOTE_UART_DataIn - MOTE_UART_DataOut;

                        continue;
                    }
                }
                else  //Check if enough data available after pointer wrap around.  Leave function until enough data available
                {
                    MAX_IN_BUFFER_SIZE = UART.endOfLastPacket;
                    remainingBytes = MAX_IN_BUFFER_SIZE - MOTE_UART_DataOut;
                    if ((remainingBytes + UART.MOTE_UART_DataIn) < UART.packetLength + 10)
                    {
                        continue;
                    }
                }
                packetCounter++;

                //Initialise accumulator string with a timestamp at start of packet conversion
                if (packetCounter == 1)
                {
                    //Reset string builder
                    sb.Remove(0, sbLength);


                    timeNow = Win32.ConvertDateTime(DateTime.Now);

                    sb.Append(timeNow.Hour.ToString("00") + ":" + timeNow.Minute.ToString("00") + ":" + timeNow.Second.ToString("00") + " ");
                }
                else
                {
                    sb.Append(" ");
                }

                //Sync byte bounding each packet is 0x7E (126).  This value will never be found within a data packet.
                //The values 0x7D and 0x7E will be 'escaped' by 0x7D followed by the value (0x7D or 0x7E) XOR'd with 0x20 = 0x5D or 0x5E
                //Search for the first and second 0x7E, the addresses form the bounds of a packet

                syncBytePointer = MOTE_UART_DataOut;

                localPacketLength = 0;
                exitLoop = false;
                do
                {
                    //If DataOut pointer is not in sync with start of a packet and parsing starts in the middle of
                    //a packet, the end and start sync bytes will be found in consecutive addresses in the InBuffer putting the
                    //data stored in the logfile out of phase with the packets, eg 154 41 168 126 126 '\n\r' 69 0 255 255 0 0 28 etc
                    //Check for this condition and reduce the localPacketLength by 1 to get data back in sync with packet starts
                    if (UART.MOTE_UART_Raw_InBuffer[syncBytePointer] == (byte)0x7E)
                    {
                        //If second byte in buffer checked is a sync byte, reset packet counter, start of packet is found
                        if (localPacketLength == 1)
                        {
                            localPacketLength = 0;
                            MOTE_UART_DataOut = syncBytePointer;
                        }

                        if (localPacketLength > 1)
                        {
                            localPacketLength++;
                            exitLoop = true;
                            continue;
                        }

                        //Many 0s have been noticed in logfiles occassionally
                        //May be caused by syncBytePointer exceeding UART.DataIn pointer.
                        //Exit entire loop and wait for more data to arrive before converting to ASCII
                        if (localPacketLength > (UART.packetLength + 20))
                        {
                            waitForNextPacket = true;
                            exitLoop = true;
                            continue;
                        }

                    }

                    localPacketLength++;
                    syncBytePointer++;
                    if (syncBytePointer >= MAX_IN_BUFFER_SIZE) syncBytePointer = 0;

                } while (exitLoop == false);

                //Do not process data in buffer, wait until more has arrived
                if (waitForNextPacket == true)
                {
                    waitForNextPacket = false;
                    continue;
                }

                //Stream data to IP port
                //if (streamData_ == true)
                //{
                //    if (ConsoleUserInterface.ActiveConsoleIO == ConsoleUserInterface.consoleInput.Ethernet)
                //    {
```

```
                        //        if (SocketStream.socketActive == true)
                        //        {
                        //            SocketStream.TCPsocket.BeginSend(UART.MOTE_UART_Raw_InBuffer,
MOTE_UART_DataOut, localPacketLength, System.Net.Sockets.SocketFlags.None, null, System.Net.Sockets.SocketFlags.
None);
                        //        }
                        //    }
                        //}

                        //Convert a received binary packet to string equivalents ('0' -> 48 ',' -> 44)
                        for (x = 0; x < localPacketLength; x++)
                        {
                            sb.AppendFormat("{0} ", UART.MOTE_UART_Raw_InBuffer[MOTE_UART_DataOut]);

                            UART.MOTE_UART_Raw_InBuffer[MOTE_UART_DataOut] = (byte)0;
                            MOTE_UART_DataOut++;
                            if (MOTE_UART_DataOut >= MAX_IN_BUFFER_SIZE)
                            {
                                MOTE_UART_DataOut = 0;
                                UART.pointerWrapped = false;
                            }
                        }
                        sb.Append("\n\r");
                        sbLength = sb.Length;

                        if (packetCounter >= 10)
                        {
                            //Check if volume exists with sufficent space to perform data writing
                            //Win32.doesVolumeExist(currentPath_, ref volumeExists, 1, ref enoughVolumeCapacity)
;
                            //if (volumeExists == false) throw new Exception("Volume does not exist"); //0 bytes
 available if volume does not exist
                            //if (enoughVolumeCapacity == false) throw new Exception("Insufficient capacity on
volume for logging");

                            logFile = File.AppendText(currentPath_ + currentLogfileName_);
                            logFile.WriteLine(sb.ToString(0, sbLength).Trim());
                            logFile.Close();
                            packetCounter = 0;
                        }
                    }
                }
                catch (Exception msg)
                {
                    pauseLogging_ = true;   //Logging doesn't start again

                    ConsoleUserInterface.printToConsole(String.Format(" Error in Logging thread: {0}\r\n", msg.
Message.ToString()));
                }
            }
            else   //During logging pause ensure that the data Outpointer tracks the data Inpointer
            {
                MOTE_UART_DataOut = 0;
                packetCounter = 0;
            }
            Thread.Sleep(0);    //Release thread for a moment
        }
    }

    /// <summary>
    /// Checks if logfile exists for current system date.  Creates a new logfile if no logfile exists for the
day.
    /// </summary>
    /// <param name="VolumePath">Input: storage volume path</param>
    /// <param name="LogFileName">Output: name of current logfile</param>
    /// <param name="SysTime">Output: Timestamp when logfile created</param>
    private static void CreateNewLogFile(string VolumePath, ref string LogFileName, ref DateTime SysTime)
    {
        Win32.SYSTEMTIME DateNow;
        string tempLogFileName;
        DateTime TempSysTime = DateTime.Now;
        string fileExtension;

        try
        {
            fileExtension = Convert.ToString(Registry.GetValue(Win32.LoggerKey, "Extension", "LOG"));
            DateNow = Win32.ConvertDateTime(TempSysTime);
            tempLogFileName = DateNow.Year.ToString("0000") + DateNow.Month.ToString("00") + DateNow.Day.
ToString("00");

            //Create a log file if one does not exist for today
            if (File.Exists(String.Format("{0}{1}.{2}", VolumePath, tempLogFileName, fileExtension)) == false)
            {
                //Compress all files in log directory with LOG extension before a new logfile is created for the
 day
                compressLogFiles(VolumePath, fileExtension);

                ConsoleUserInterface.printToConsole(String.Format("Create log file: {0}{1}.{2}\n\r", VolumePath,
 tempLogFileName, fileExtension));
                StreamWriter logFile = new StreamWriter(String.Format("{0}{1}.{2}", VolumePath, tempLogFileName,
 fileExtension));

                //File header
                logFile.WriteLine(String.Format("Logfile: {0}.{1},", tempLogFileName, fileExtension));
                //Check if systemclock has been set: place error message in log file and on console

                //if (Win32.isSysClockSet == false)
                if (DateNow.Year == 2006)
                {
                    ConsoleUserInterface.printToConsole("System clock has not been set.\r\n");
                    logFile.WriteLine("System clock is not set,");
                }
                //Column headers
                //logFile.WriteLine(String.Format("Packet length = {0}",UART.packetLength));
                logFile.Close();

                //A new logfile is created: update log filename path and time variables with latest values
```

```
                //LogFileName = tempLogFileName + ".LOG";
                LogFileName = String.Format("{0}.{1}", tempLogFileName, fileExtension);
                SysTime = TempSysTime;
            }
            string temp = String.Format("{0}.{1}", tempLogFileName, fileExtension);
            if (LogFileName != temp) LogFileName = temp;  //Init with logfile name if file exists on disk, and on first execution of this method
        }
        catch (ThreadAbortException) { }
        catch (Exception msg)
        {
            ConsoleUserInterface.printToConsole(String.Format(" Error in DataLogging.CreateNewLogFile(): {0}\r\n", msg.Message.ToString()));
        }
    }

    //The following functions are based on code from the MSDN website
    /// <summary>
    /// Compress all files in logging directory with matching logging extension
    /// </summary>
    /// <param name="dirPath">Path to logging directory</param>
    /// <param name="logfileExtension">Logging file extension</param>
    /// <returns="bool">Returns false if compression failure</return>
    private static void compressLogFiles(string dirPath, string logfileExtension)
    {
        DirectoryInfo di = new DirectoryInfo(dirPath);
        string newLogFileName;
        int logFileNameLength;
        string logFilename;

        //Compress all files with the correct logging extension
        foreach (FileInfo fi in di.GetFiles(String.Format("*{0}", logfileExtension)))
        {
            if (compress(fi) == true)
            {
                logFilename = fi.Name;
                logFileNameLength = logFilename.Length;
                newLogFileName = logFilename.Substring(0, logFileNameLength - 1) + "_";
                File.Copy( dirPath + logFilename, dirPath + newLogFileName,true);   //Overwrites files with the same name to avoid file write exceptions
                ConsoleUserInterface.printToConsole(String.Format("Renaming {0} to {1}\n\r", logFilename, newLogFileName));
            }
        }
    }

    /// <summary>
    /// Performs file compression to file fi
    /// </summary>
    /// <param name="fi"></param>
    private static bool compress(FileInfo fi)
    {
        try
        {
            // Get the stream of the source file.
            using (FileStream inFile = fi.OpenRead())
            {
                // Create the compressed file.
                using (FileStream outFile = File.Create(fi.FullName + ".gz"))
                {
                    using (GZipStream Compress = new GZipStream(outFile, CompressionMode.Compress))
                    {
                        ConsoleUserInterface.printToConsole(String.Format("Compressing file: {0}\r", fi.Name.ToString()));
                        // Copy the source file into the compression stream.
                        byte[] buffer = new byte[4096];
                        int numRead;
                        while ((numRead = inFile.Read(buffer, 0, buffer.Length)) != 0)
                        {
                            Compress.Write(buffer, 0, numRead);
                            Thread.Sleep(0);
                        }
                        ConsoleUserInterface.printToConsole(String.Format("Generated compressed file: {0}\r\n", outFile.Name.ToString()));
                        return true;
                    }
                }
            }
        }
        catch (ThreadAbortException) { return false; }
        catch (Exception msg)
        {
            ConsoleUserInterface.printToConsole(String.Format(" Error in LoggingThread.compress(): {0}\r\n", msg.Message.ToString()));
            return false;
        }
    }
}
```

```
//FILENAME: FTPThread.cs
using System;
using System.IO;
using System.Collections.Generic;
using System.Threading;
using System.Text;
using Microsoft.Win32;

namespace GatewayApp
{
    public static class FTPThread
    {
        private static bool FTPthreadRunning_=false;
        private static Win32.SYSTEMTIME fileSendTime_;
        private static string logDirectory_;                    //Logfile directory path
        private static bool interruptFileSend_=false;           //Interrupts current FTP send and closes internet/ FTP connection
        private static bool forceSendLogs_=false;               //Overrides the time check and forces the log files to be sent

        private static string serverIP;                         //Remote server IP address - value stored in Registry
        private static string userName;                         //FTP login username
        private static string passWord;                         //FTP login password
        private static bool FTPinProgress_=false;               //FTP of files in progress, so do not try to write to current log file

        private static Thread FTPlogFiles = new Thread(sendLogFiles);

        #region Class Interface

        /// <summary>
        /// FTP thread is running.  Set to false to exit thread.
        /// </summary>
        public static bool threadRunning
        {
            set { FTPthreadRunning_ = value; }
            get { return FTPthreadRunning_; }
        }
        public static bool interruptFileSend
        {
            set { interruptFileSend_ = value; }
        }

        /// <summary>
        /// Overrides time to send log files and sends them now via selected FTP transport to selected server
        /// </summary>
        public static bool forceFileSend
        {
            set { forceSendLogs_ = value; }
        }

        /// <summary>
        /// True if Log files being transferred.  Do not access log files whilst true.
        /// </summary>
        public static bool FTPinProgress
        {
            get { return FTPinProgress_; }
        }

        #endregion

        /// <summary>
        /// Starts FTP process running
        /// </summary>
        public static void startFTPthread()
        {
            if (FTPthreadRunning_ == true) return;

            FTPlogFiles.IsBackground = true;
            FTPlogFiles.Start();
            Thread.Sleep(500);
        }

        private static void sendLogFiles()
        {
            bool startFileTransfer=false;
            bool sendFiles=false;
            bool internetOpened;
            bool once=false;
            bool timeTestResult;
            string currentFilename;
            UInt32 errorMsg=0;

            string regSendTime;
            string logFileExtension;
            string sentLogFileExtension;
            int FTPtransport;

            fileSendTime_.Second = 0;
            FTPthreadRunning_ = true;
            while (FTPthreadRunning_ == true)
            {
                //Pole registry to take any setting changes into effect
                logDirectory_ = String.Format("\\{0}\\{1}\\", Registry.GetValue(Win32.LoggerKey, "Media", "\\"), Registry.GetValue(Win32.LoggerKey, "Directory", ""));
                regSendTime = Convert.ToString(Registry.GetValue(Win32.LoggerKey, "Send time", "0010"));    // Default time is 00h10

                //Ensure that send time is valid
                if (regSendTime.Trim().Length != 4) regSendTime = "0010";

                //Convert time to shorts
                fileSendTime_.Hour = Convert.ToUInt16(regSendTime.Substring(0,2));
                fileSendTime_.Minute = Convert.ToUInt16(regSendTime.Substring(2,2));

                //Test if a match occurs between system hour/minute and user hour/minute
```

```
            timeTestResult = Win32.TimeMatch(fileSendTime_);
            if(timeTestResult == true && once == false)
            {
                once = true;
                startFileTransfer = true;       //Set once if a time match occurs.
            }
            else if(timeTestResult == false && once == true)once = false;

            //If a time match or force ftp selected, open internet or establish GSM connection with remote server and send all log files older than today's file
            if (startFileTransfer == true || forceSendLogs_ == true)
            {
                forceSendLogs_ = false;
                startFileTransfer = false;
                sendFiles = false;

                int connectAttempts = Convert.ToInt32(Registry.GetValue(Win32.FTPkey, "Attempts", 5));
                bool exitLoop = false;

                int interval = 1000 * Convert.ToInt32(Registry.GetValue(Win32.FTPkey, "Attempts interval s", 30));

                #region Loop2: Establish connection with internet and setup FTP session
                do
                {
                    try
                    {
                        //Check registry setting for FTP transport type (GSM/Ethernet)
                        FTPtransport = Convert.ToInt32(Registry.GetValue(Win32.FTPkey, "Transport", 2));  // Default of 2 = ethernet

                        //Check FTP transport selection, open selected transport for FTP
                        if (FTPtransport == (int)ConsoleUserInterface.FTPtransmitter.GSM)
                        {
                            #region Open GSM internet connection

                            ConsoleUserInterface.printToConsole(String.Format("System time is: {0}\n\r", DateTime.Now));
                            ConsoleUserInterface.printToConsole("Starting FTP of logfiles via GPRS\r");
                            ConsoleUserInterface.printToConsole("Suspending GPRS link until FTP is complete\r");
                            ConsoleUserInterface.printToConsole("Please wait...\n\r");


                            //if (GSM.openModem() == false) throw new Exception("Cannot open GSM modem for FTP link\n\r");
                            if (GSM.openFTPlink() == false) throw new Exception("Cannot open FTP session\n\r");
                            ConsoleUserInterface.printToConsole("FTP session opened\n\r");
                            sendFiles = true;
                            exitLoop = true;
                            continue;

                            #endregion
                        }
                        else    //Ethernet or other transport type
                        {
                            #region Open Ethernet internet connection

                            //Read ETH FTP settigns from registry
                            serverIP = Convert.ToString(Registry.GetValue(Win32.FTPETHkey, "Server IP", ""));
                            userName = Convert.ToString(Registry.GetValue(Win32.FTPETHkey, "Username", ""));
                            passWord = Convert.ToString(Registry.GetValue(Win32.FTPETHkey, "Password", ""));

                            if (serverIP == "" || userName == "" || passWord == "")
                            {
                                throw new Exception("Invalid ETH FTP settings. No connection attempted.\n\r");
                            }
                            else
                            {
                                //Open an internet connection with valid server IP
                                ConsoleUserInterface.printToConsole("Opening Internet via Ethernet connection\r");
                                ConsoleUserInterface.printToConsole(String.Format("System time is: {0}\n\r", DateTime.Now));

                                internetOpened = Win32.OpenInternet(Win32.HTTP_USER, ref errorMsg);
                                if (errorMsg == 0)
                                {
                                    if (internetOpened == true)
                                    {
                                        ConsoleUserInterface.printToConsole("Internet connection opened\r");

                                        connectAttempts = Convert.ToInt32(Registry.GetValue(Win32.FTPkey, "Attempts", 5));
                                        int counter = connectAttempts;
                                        exitLoop = false;   //Attempt to open FTP session
                                        do
                                        {
                                            ConsoleUserInterface.printToConsole(String.Format("{0} attempts to open FTP session\r",counter.ToString()));

                                            Win32.OpenFTPsession(serverIP, userName, passWord, ref errorMsg);
                                            if (errorMsg == 0)
                                            {
                                                ConsoleUserInterface.printToConsole("FTP session opened\n\r");
                                                sendFiles = true;
                                                exitLoop = true;
                                                continue;
                                            }
                                            else
                                            {
                                                ConsoleUserInterface.printToConsole(String.Format("FTP session open failed. Error = {0}\n\r",errorMsg));

                                                counter--;
                                                if (counter > 0)
                                                {
                                                    Thread.Sleep(interval); //Wait for set time before trying again
```

```
                                        }
                                        else
                                        {
                                            ConsoleUserInterface.printToConsole("Closing Internet connection\n\r");
                                            //Try to close the internet connection for next FTP attempt in loop
                                            Win32.CloseInternet(ref errorMsg);
                                            exitLoop = true;
                                            sendFiles = false;
                                            throw new Exception(String.Format("{1} attempts to open FTP session failed. Error = {0}\n\r", errorMsg.ToString(),connectAttempts.ToString()));
                                        }
                                    }
                                    Thread.Sleep(0);
                                }while(exitLoop==false);
                            }
                            else
                            {   //Try to close the connection to the internet
                                Win32.CloseInternet(ref errorMsg);
                                throw new Exception(String.Format("Closing internet connection. Error = {0}\n\r", errorMsg.ToString()));
                            }
                        }
                        else
                        {
                            throw new Exception(String.Format("Cannot open Internet. Error = {0}\n\r", errorMsg.ToString()));
                            //ConsoleUserInterface.printToConsole(String.Format("Cannot open Internet. Error = {0}\n\r", errorMsg.ToString()));
                        }
                    }
                    #endregion
                }
            }
            catch (Exception msg)
            {
                ConsoleUserInterface.printToConsole(String.Format("Exception in sendLogFiles(): {0}\r", msg.ToString()));
                ConsoleUserInterface.printToConsole(String.Format("Remaining attempts = {0} in {1}s\n\r", connectAttempts - 1, interval / 1000));
            }

            if (interruptFileSend_ == true)
            {
                ConsoleUserInterface.printToConsole("Stopping FTP session\n\rClosing Internet connection\n\r");
                interruptFileSend_ = false;
                startFileTransfer = false;
                exitLoop = true;
                sendFiles = false;
                continue;
            }

            connectAttempts--;
            if (connectAttempts == 0)
            {
                exitLoop = true;
                sendFiles = false;
                ConsoleUserInterface.printToConsole("Cannot connect to internet or open FTP session\n\r");
                continue;
            }

            Thread.Sleep(interval);    //Wait a maximum of 60s before attempting connection again
        } while (exitLoop == false);
        #endregion
    }

    //Start file sending
    if (sendFiles == true)
    {
        //Only allow file send if an FTP connection is opened
        sendFiles = false;

        //Check registry setting for FTP transport type (GSM/Ethernet)
        FTPtransport = Convert.ToInt32(Registry.GetValue(Win32.FTPkey, "Transport", 2));  //Default of 2 = ethernet

        //FTP only compressed log files
        logFileExtension = ".gz";

        //Registry stores sent log files extension
        sentLogFileExtension = Convert.ToString(Registry.GetValue(Win32.LoggerKey,"Old Extension","gz_"));
        sentLogFileExtension = String.Format(".{0}", sentLogFileExtension);


        //Get directory listing of logging directory
        DirectoryInfo dirInfo = new DirectoryInfo(logDirectory_);
        FileInfo[] fileList = dirInfo.GetFiles(String.Format("*{0}", logFileExtension));

        #region Loop3: FTP files
        try
        {
            //Send files now that valid internet and FTP connection exist
            foreach (FileInfo fileNext in fileList)
            {
                currentFilename = fileNext.ToString().Trim();

                //Set a flag to ensure that logger does not access current log file until FTP completed
                if(LoggingThread.currentLogFileName == currentFilename) FTPinProgress_ = true;
                ConsoleUserInterface.printToConsole(String.Format("\rSending file {0}\r", fileNext.ToString().Trim()));

                //Check FTP transport selection, open selected transport for FTP
                if (FTPtransport == (int)ConsoleUserInterface.FTPtransmitter.GSM)
```

```
                    {
                        #region Send files via GSM

                        if (GSM.putFTPfile(currentFilename) == false) throw new Exception("Cannot create a new file on FTP server\n\r");
                        if (sendFile(logDirectory_, currentFilename) == false) throw new Exception("\rGSM file transfer failed\n\r");

                        //Change extensions of all log files except current log file
                        string newFilename = currentFilename.Replace(logFileExtension, sentLogFileExtension);
                        if (FTPinProgress_ == false)
                        {
                            try
                            {
                                ConsoleUserInterface.printToConsole(String.Format("\rRenaming file {0} to {1}\n\r", currentFilename, newFilename));
                                File.Move(logDirectory_ + currentFilename, logDirectory_ + newFilename);
                            }
                            catch
                            {
                                ConsoleUserInterface.printToConsole(String.Format("\rCannot rename file {0} to {1}\r", currentFilename, newFilename));
                                ConsoleUserInterface.printToConsole("Check for duplicate filenames\n\r");
                            }
                        }
                        FTPinProgress_ = false;

                        #endregion
                    }
                    else
                    {
                        #region Send files via Ethernet

                        if (Win32.WriteFileByFTP(String.Format("{0}{1}", logDirectory_, currentFilename), currentFilename) == false)
                        {
                            errorMsg = Win32.GetLastError();
                            startFileTransfer = false;
                            throw new Exception("\rFile transfer failed. Error = " + errorMsg.ToString() + "\r\n");
                        }
                        else
                        {
                            ConsoleUserInterface.printToConsole("\rFile transfer successful!\r");

                            //Change extensions of all log files except current log file
                            if (FTPinProgress_ == false)
                            {
                                string newFilename = currentFilename.Replace(logFileExtension, sentLogFileExtension);

                                try
                                {
                                    //Rename the log file so that it is not uploaded again
                                    //string newFilename = currentFilename.Replace(@".LOG", @".LO_");
                                    //string newFilename = currentFilename.Replace(logFileExtension, sentLogFileExtension);

                                    ConsoleUserInterface.printToConsole(String.Format("\rRenaming file {0} to {1}\n\r", currentFilename, newFilename));
                                    File.Move(logDirectory_ + currentFilename, logDirectory_ + newFilename);
                                }
                                catch
                                {
                                    ConsoleUserInterface.printToConsole(String.Format("\rCannot rename file {0} to {1}\r", currentFilename, newFilename));
                                    ConsoleUserInterface.printToConsole("Check for duplicate filenames\n\r");

                                }
                            }
                            FTPinProgress_ = false;
                        }

                        #endregion
                    }
                    if (interruptFileSend_ == true)
                    {
                        ConsoleUserInterface.printToConsole("Stopping FTP session\n\rClosing Internet connection\n\r");
                        interruptFileSend_ = false;
                        startFileTransfer = false;
                        break;  //Leave foreach loop
                    }
                } //End of foreach loop
            }
            catch (Exception msg)
            {
                ConsoleUserInterface.printToConsole(String.Format("Exception in sendLogFiles(): {0}\n\r", msg.ToString()));
            }
            #endregion

            #region Terminate internet connection
            try
            {
                if (FTPtransport == (int)ConsoleUserInterface.FTPtransmitter.GSM)
                {
                    if (GSM.closeFTP()==false)throw new Exception("Cannot close FTP session\n\r");

                }
                else //Transport: ethernet
                {
                    //Close FTP session and internet connection
                    startFileTransfer = false;
                    sendFiles = false;

                    if (Win32.CloseFTPsession(ref errorMsg) == true) ConsoleUserInterface.printToConsole(
```

```
    "FTP session closed = OK\r\n");
                        else throw new Exception("FTP session close failed. Error number = " + errorMsg.ToString
    () + "\r\n");

                        if (Win32.CloseInternet(ref errorMsg) == true) ConsoleUserInterface.printToConsole(
    "Internet connection closed = OK\r\n");
                        else throw new Exception("Internet connection close failed. Error number = " + errorMsg.
    ToString() + "\r\n");
                    }
                }
                catch (Exception msg)
                {
                    ConsoleUserInterface.printToConsole(String.Format("Exception in sendLogFiles(): {0}\n\r",
    msg.ToString()));
                }
#endregion
            }

            //Block the thread for 10s to prevent excessive activity in this thread
            Thread.Sleep(10000);
        }   //ThreadRunning loop
    }

    /// <summary>
    /// Sends a file to the GSM modem for FTP transmission to server
    /// </summary>
    /// <param name="filePath">Path to file to send</param>
    /// <param name="newFileName">Filename to send</param>
    /// <returns>True if file send succesful, else false</returns>
    public static bool sendFile(string filePath, string newFileName)
    {
        byte[] sendBuffer = new byte[300];                      //300 byte buffer
        int bytesRemaining;
        bool commsTimeout;
        int transmitTimer;

        try
        {
            //Time allowed in s for data in module buffer to be sent after last Xon is received by Windows
            transmitTimer = Convert.ToInt32(Registry.GetValue(Win32.FTPGSMkey, "Transmit timeout s", 20));
            if (transmitTimer == 0) transmitTimer = 20; //Default to 20s

            ConsoleUserInterface.printToConsole(String.Format("filePath = {0}{1}\n\r", filePath,newFileName));

            FileStream FS = new FileStream(String.Format("{0}{1}", filePath, newFileName), FileMode.Open);
            BinaryReader newFile = new BinaryReader(FS);

            bytesRemaining = (int)FS.Length;

            UART.startHandshakeRX = true;                               //Allow handshaking signals to be received
            UART.transmitTimerConst = transmitTimer;

            commsTimeout = false;
            GSM.GPRSmodemStatus = GSM.GPRSstatus.FTP_FILE_SENDING;
            while (bytesRemaining > 0)
            {
                if (bytesRemaining < 300)
                {
                    newFile.Read(sendBuffer, 0, bytesRemaining);
                    if (UART.sendBuffer(ref sendBuffer, bytesRemaining) == false)
                    {
                        bytesRemaining = 0;
                        commsTimeout = true;
                        continue;
                    }
                    bytesRemaining = 0;
                }
                else
                {
                    newFile.Read(sendBuffer, 0, 300);
                    if (UART.sendBuffer(ref sendBuffer, 300) == false)
                    {
                        bytesRemaining = 0;
                        commsTimeout = true;
                        continue;
                    }
                    bytesRemaining = bytesRemaining - 300;
                }
                Thread.Sleep(0);
            }

            UART.startHandshakeRX = false;
            FS.Close();
            newFile.Close();

            if (commsTimeout == true)
            {
                ConsoleUserInterface.printToConsole("GSM comms timeout: FTPThread:sendFile()");

                return GSM.sendEscapeSequence();    //Put the modem into command mode
            }

            completeFileTransmission();         //Wait for modem to finish sending buffered data before sending
    escape sequence
            return GSM.sendEscapeSequence();    //Put the modem into command mode to send next file

        }
        catch (Exception msg)
        {
            ConsoleUserInterface.printToConsole(String.Format("Exception in FTPThread.sendFile(): {0}\n\r",
    msg));
            return false;
        }
    }

    /// <summary>
    /// Timer to allow buffered data in modem to be sent to FTP server before sending escape
```

```
        /// sequence to modem.  Timer is set to 15s on every Xon received from modem.
        /// </summary>
        private static void completeFileTransmission()
        {
            while (UART.transmitTimer > 0)
            {
                Thread.Sleep(1000);
                UART.transmitTimer--;
            }
        }
    }
}
```

```
//FILENAME: GPIO.cs
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Threading;
using System.Text;

namespace GatewayApp
{
    public static class GPIO
    {
        #region GPIO methods

        /// <summary>
        /// Sets GPIO bit
        /// </summary>
        /// <param name="PortBank">eg A, Bor C</param>
        /// <param name="PortNumber">eg 0..31 - check that the pin is not used for some other IO function</param>
        /// <param name="state">0..1</param>
        /// <returns>True if successful, else 0</returns>
        public static bool setGPIO(string PortBank, int PortNumber, bool state)
        {
            Process IOprocess = new Process(); //From: http://msdn.microsoft.com/en-us/library/system.diagnostics.process(v=VS.90).aspx
            string IOarguments;
            string PortState;

            try
            {
                IOprocess.StartInfo.UseShellExecute = false;

                if (PortBank.Trim() != "A" && PortBank.Trim() != "B" && PortBank.Trim() != "C") throw new Exception("Invalid Port Bank");
                if (PortNumber < 0 || PortNumber > 31) throw new Exception("Invalid Port Number");

                PortState = state == true ? "1" : "0";
                IOarguments = String.Format("{0} {1} {2} {3}", PortBank, PortNumber / 10, PortNumber % 10, PortState);

                IOprocess.StartInfo.Arguments = IOarguments;
                IOprocess.StartInfo.FileName = "\\Windows\\IO_control.exe";
                IOprocess.Start();
                IOprocess.WaitForExit();
                int ExitCode = IOprocess.ExitCode;

                return true;
            }
            catch (Exception msg)
            {
                ConsoleUserInterface.printToConsole(String.Format("Error in GPIO class: {0}\r\n", msg.Message.ToString()));
                return false;
            }
        }
        #endregion
    }
}
```

```
//FILENAME: SocketStream.cs
using System;
using System.Linq;
using System.Collections.Generic;
using System.Text;
using System.Net;
using System.Net.Sockets;
using Microsoft.Win32;

namespace GatewayApp
{
    class SocketStream
    {
        private static Socket s = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
        private static bool socketActive_;

        /// <summary>
        /// An active TCP/IP socket is available
        /// </summary>
        public static bool socketActive
        {
            get { return socketActive_; }
        }

        public static Socket TCPsocket
        {
            get {return s;}
        }

        /// <summary>
        /// Open socket on Ethernet for user defined port and destination IP address
        /// </summary>
        public static void createNewSocket()
        {
            IPAddress remoteIP = IPAddress.Parse(Registry.GetValue(Win32.StreamKey, "Destination IP", "0.0.0.0").
    ToString());
            int port = Convert.ToInt32(Registry.GetValue(Win32.StreamKey, "Out port", "49500"));
            IPEndPoint IPE = new IPEndPoint(remoteIP, port);

            try
            {
                ConsoleUserInterface.printToConsole(String.Format("Opening socket on port {0}\n\rConnecting to
    destination IP: {1}\n\r", remoteIP, port));
                s.Connect(IPE);
                socketActive_ = s.Connected;
            }
            catch (Exception msg)
            {
                socketActive_ = false;
                ConsoleUserInterface.printToConsole(msg.ToString());
            }
        }

        public static void closeSocket()
        {
            try
            {
                if (s.Connected == true)
                {
                    ConsoleUserInterface.printToConsole("Closing streaming socket\n\r");
                    s.Shutdown(SocketShutdown.Both);
                    s.Close();
                    socketActive_ = false;
                }
            }
            catch (Exception msg)
            {
                ConsoleUserInterface.printToConsole(msg.ToString());
            }
        }

        public static void receiveTCPdata()
        {
            byte[] dataIn = new byte[1024];

            if (socketActive_ == true)
            {

                int bytesRec = s.Receive(dataIn);
                ConsoleUserInterface.printToConsole(Encoding.ASCII.GetString(dataIn,0,bytesRec));
            }
            else return;
        }
    }
}
```

```
//FILENAME: Main.cs
using System;

namespace GatewayApp
{
    public class Program
    {
        public static void Main(string[] controlArgs)
        {
            GatewayEngine newEngineInstance = new GatewayEngine();
            newEngineInstance.mainEngine(controlArgs);
            return;
        }
    }
}
```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace GatewayApp
{
    class MODEM
    {
        /// <summary>
        /// Component number of module
        /// </summary>
        public string type;         //Type of module, Telit GC864
        /// <summary>
        /// Baud rate of modeul COM port
        /// </summary>
        public string baud;         //UART board rate
        /// <summary>
        /// ARM IO port bank
        /// </summary>
        public string powerPortBank;     //IO bank A, B or C
        /// <summary>
        /// IO port number
        /// </summary>
        public int powerIO;      //IO pin number 0..31
        /// <summary>
        /// ARM IO port bank
        /// </summary>
        public string resetPortBank;
        /// <summary>
        /// ARM IO port number
        /// </summary>
        public int resetIO;
        /// <summary>
        /// Number of times to retry a modem command if response negative, or failed
        /// </summary>
        public int commsAttempts;
        /// <summary>
        /// Number of milliseconds between sending AT commands during comms attempts
        /// </summary>
        public int attemptInterval;
        /// <summary>
        /// Number of milliseconds UART waits before timing out with an exception
        /// </summary>
        public int readbyteWait;
        /// <summary>
        /// Number of seconds to wait for buffered data in modem to be sent at end of FTP transfer after ↙
    final Xon is sent by modem
        /// </summary>
        public int transmitTimer;
    }
}
```

```
//FILENAME: IO_CONTROL.CPP
#include "stdafx.h"
#include "stdio.h"
#include "string.h"

#include <windows.h>
#include <commctrl.h>
#include <ceddk.h>

#include "C:\WINCE600\PUBLIC\COMMON\OAK\INC\windev.h"
#include "C:\WINCE600\PLATFORM\COMMON\SRC\SOC\ATMEL\COMMON\DRIVERS\GPIO\GPIO.h"
#include "C:\WINCE600\PLATFORM\COMMON\SRC\SOC\ATMEL\AT91SAM9260\INC\AT91SAM9260.h"

/* these pin numbers double as IRQ numbers, like AT91C_ID_* values */
#define PIO_NB_IO           32 /* Number of IO handled by one PIO controller */
#define AT91C_PIN_PA(io)    (0 * PIO_NB_IO + io)
#define AT91C_PIN_PB(io)    (1 * PIO_NB_IO + io)
#define AT91C_PIN_PC(io)    (2 * PIO_NB_IO + io)

/* I/O attributes */
#define PIO_DEFAULT   (0 << 0)
#define PIO_PULLUP    (1 << 0)
#define PIO_DEGLITCH  (1 << 1)
#define PIO_OPENDRAIN (1 << 2)

struct pio_desc
{
    const char *pin_name;   /* Pin Name */
    unsigned int pin_num;   /* Pin number */
    unsigned int dft_value; /* Default value for outputs */
    unsigned char attribute;
    enum pio_type type;
};

/* I/O type */
enum  pio_type
{
    PIO_PERIPH_A,
    PIO_PERIPH_B,
    PIO_INPUT,
    PIO_OUTPUT,
    PIO_UNDEFINED
};

#define GPIO_CONFIGURE_CMD          (2048 + 1)
#define GPIO_SETSTATE_CMD           (2048 + 2)
#define GPIO_GETSTATE_CMD           (2048 + 3)

#define IOCTL_GPIO_CONFIGURE    CTL_CODE(FILE_DEVICE_UNKNOWN, GPIO_CONFIGURE_CMD, METHOD_BUFFERED,                  ↙
    FILE_WRITE_ACCESS)
#define IOCTL_GPIO_SETSTATE     CTL_CODE(FILE_DEVICE_UNKNOWN, GPIO_SETSTATE_CMD, METHOD_BUFFERED, FILE_WRITE_ACCESS)
#define IOCTL_GPIO_GETSTATE     CTL_CODE(FILE_DEVICE_UNKNOWN, GPIO_GETSTATE_CMD, METHOD_BUFFERED, FILE_READ_ACCESS)

typedef struct
{
    DWORD PinNo;            // Pin number
    DWORD PinState;         // Pin value (0 or 1)
} T_GPIOIOCTL_STATE;


//Useage: IO_control <Port Bank><space><Port number><space><Port number><space><state>
//Parameters: Port Bank <A,B,C>, Port number(tens)<0..9>, Port number(ones)<0..9>, state <0..1>
//Returns: 1 if succesful else 0.
int _tmain(int argc, _TCHAR* argv[] ) //http://www.cplusplus.com/forum/beginner/34155/
{
    WORD portNumber;
    WORD portPin_state;
    WORD portAddress;

    portNumber = (*argv[2] - 48) * 10;  //Convert string to integer
    portNumber += (*argv[3] - 48);
    if(portNumber>31)
    {
        return 0;
    }

    portPin_state = *argv[4] - 48;
    if(portPin_state>1)
    {
        return 0;
    }

    switch(*argv[1])
    {
        case 'A':   portAddress = 0 * PIO_NB_IO + portNumber;
                    break;

        case 'B':   portAddress = 1 * PIO_NB_IO + portNumber;
                    break;

        case 'C':   portAddress = 2 * PIO_NB_IO + portNumber;
                    break;

        default:    return 0;   //Invalid port bank returns null
    }

    // Variables
    HANDLE hGPIO = NULL;
    HANDLE hDev = NULL;
    BOOL bSuccessDevIOC = FALSE;
    TCHAR strFileName[10] = TEXT("PIO1:");

    // Open GPIO driver
    hGPIO = CreateFile(strFileName, 0, 0, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);

    if (hGPIO != INVALID_HANDLE_VALUE)
    {
```

```
        const struct pio_desc hw_pio[] =
        {
            {"", portAddress, 1, PIO_DEFAULT, PIO_OUTPUT},
        };
        T_GPIOIOCTL_STATE * pSetState;

        // Configure PIOs
        bSuccessDevIOC = DeviceIoControl(hGPIO, IOCTL_GPIO_CONFIGURE, (LPBYTE*)hw_pio, sizeof(hw_pio), NULL, 0, NULL ↙
    , NULL);
        if (!bSuccessDevIOC)
        {
            return 0;
        }

        pSetState = (T_GPIOIOCTL_STATE*)LocalAlloc(LMEM_ZEROINIT|LMEM_FIXED,sizeof(T_GPIOIOCTL_STATE));
        pSetState->PinNo = portAddress;
        pSetState->PinState = portPin_state;
        bSuccessDevIOC = DeviceIoControl(hGPIO, IOCTL_GPIO_SETSTATE, pSetState, sizeof(*pSetState), NULL, 0, NULL,  ↙
    NULL);
        LocalFree(pSetState);
        if (!bSuccessDevIOC)
        {
            return 0;
        }

        if (!CloseHandle(hGPIO))
        {
            return 0;
        }
    }
    else
    {
        return 0;
    }
    return 1;
}
```

```
/**
 * PACKETTX sends packets at a defined rate to the serial port
 *
 * @author Gary de Villiers
 **/

enum
{
  AM_UART_ADDRESS = 0x89,
};

configuration PACKETTXApp
{
}
implementation
{
  components MainC, PACKETTXC, LedsC;
  components new TimerMilliC() as Timer0;
  components new Alarm32khz32C() as Alarm0;
  components new AMSenderC(6);
  components ActiveMessageC as Radio;
  components SerialActiveMessageC as UART;

  PACKETTXC -> MainC.Boot;

  PACKETTXC.Timer0 -> Timer0;
  PACKETTXC.Leds -> LedsC;
  PACKETTXC.Alarm0 -> Alarm0;

  PACKETTXC.RadioAMControl -> Radio;
  PACKETTXC.RadioAMSend -> AMSenderC;
  PACKETTXC.RadioPacket -> AMSenderC;

  PACKETTXC.UARTAMControl -> UART;
  PACKETTXC.UARTAMSend -> UART.AMSend[AM_UART_ADDRESS];
  PACKETTXC.UARTPacket -> UART;
}
```

```
/**
 * Implementation for PACKETTX application.  Use timers to generate
 * and transmit packets to the serial port at regular intervals.
 **/

#include <stdio.h>;
#include "Timer.h";
#include "RadioCountToLeds.h";

#define blue_led_on Leds.led2On()
#define blue_led_off Leds.led2Off()

module PACKETTXC @safe()
{
    uses
    {
       interface Alarm<T32khz,uint32_t> as Alarm0;
       interface Timer<TMilli> as Timer0;
       interface Leds;
       interface Boot;

       interface Packet as RadioPacket;
       interface AMPacket as RadioAMPacket;
       interface AMSend as RadioAMSend;
       interface SplitControl as RadioAMControl;

       interface SplitControl as UARTAMControl;
       interface AMSend as UARTAMSend;
       interface Packet as UARTPacket;
    }
}
implementation
{
  uint16_t blue_DC;
  uint16_t green_DC;
  uint16_t red_DC;
  uint16_t packet_number;

  bool UART_busy=TRUE;
  message_t UART_packet;

  message_t packet;
  bool radio_busy=FALSE;

  event void Boot.booted()
  {
    call UARTAMControl.start();

    atomic
    {
        blue_DC=0;
        green_DC=40;
        red_DC=80;
        packet_number=0;
    }
  }
   //*****************************************************************************
   //Packet transmitter timer
   //*****************************************************************************
    event void UARTAMControl.startDone(error_t err)
    {
       if(err==SUCCESS)
        {
            UART_busy=FALSE;
            call Timer0.startPeriodic(10); //<---- Adjust packet frequency
        }
       //else call UARTAMControl.start();
    }

   //*****************************************************************************
   event void UARTAMControl.stopDone(error_t err) {}

   //*****************************************************************************
   //Timer interrupt to transmit packets
   //*****************************************************************************
   event void Timer0.fired()
   {
        int16_t UART_packet_len=20;
        char* UART_rcm;

       atomic
       {
            blue_DC++;
            if(blue_DC<99)blue_DC=0;
```

```
            green_DC++;
            if(green_DC<99)green_DC=0;

            red_DC++;
            if(red_DC<99)red_DC=0;

            if(UART_busy==FALSE)
            {
                UART_rcm = (char*)call UARTAMSend.getPayload(&UART_packet,
    UART_packet_len);

                UART_packet_len=sprintf(UART_rcm,"%c%c%02u,%02u,%02u,CPUT CAPETOWN,ZA%c%c
    ",
                    (uint8_t)((packet_number&0xFF00)/0x100),(uint8_t)(packet_number&
    0x00FF),blue_DC,red_DC,green_DC,0x0A,0x0D);

                //UART_packet_len=sprintf(UART_rcm,"%c%c,%04u,CPUT CAPETOWN,ZA%c%c",
                  //(uint8_t)((packet_number&0xFF00)/0x100),(uint8_t)(packet_number&
    0x00FF),packet_number,0x0A,0x0D);

                call blue_led_on;
                if(call UARTAMSend.send(AM_BROADCAST_ADDR, &UART_packet, UART_packet_len)
    ==SUCCESS)
                {
                    call blue_led_off;
                    UART_busy=TRUE;
                    packet_number++;
                    if(packet_number<65534)packet_number=0;
                }
            }
        }
    }
    //*************************************************************************
    //UART completed packet transfer
    //*************************************************************************
    event void UARTAMSend.sendDone(message_t* bufPtr, error_t error)
    {
        if (&UART_packet == bufPtr)
        {
           UART_busy = FALSE;
        }
    }
}
```

Appendix G

Appendix G: Registry and project settings

Windows CE registry and project settings

The Windows registry required modification during the configuration of the OS to permit connections to its embedded FTP and Telnet servers and to enable the SD/MMC card after its launch. In addition, the FTDI VCP driver and *IO_control.exe* application were required to be included in the OS release.

Two files permit the required additions to the registry and the OS image: *project.reg* and *project.bib* respectively and are found in the *Parameter Files* section of the *Solution Explorer* window in Visual Studio.

G.1 *project.reg* entries

G.1.1 Telnet and FTP server settings

The following entries permit *anonymous* access without authentication to the Telnet and FTP servers on the gateway and were added to the *project.reg* file before the OS build was launched:

[HKEY_LOCAL_MACHINE\Comm\TELNETD]
"UseAuthentication" = DWORD:0
"IsEnabled" = DWORD:1
"UserList" = "@*"

[HKEY_LOCAL_MACHINE\Comm\FTPD]
"DefaultDir" = "\\"
"IsEnabled" = DWORD:1
"UseAuthentication" = DWORD:0
"AllowAnonymous" = DWORD:1
"AllowAnonymousUpload" = DWORD:1
"AllowAnonymousVroots" = DWORD:1

G.1.2 SD/MMC memory card settings

The following registry additions enable the SD card driver and configures the directory that it is bound to:

[HKEY_LOCAL_MACHINE\SOFTWARE\FLASHLOADER]
"SDConnect" = DWORD:1

[HKEY_LOCAL_MACHINE\SYSTEM\StorageManager\SDMemory]
"Folder" = "SDcard"

G.1.3 Gateway application automatic launch

The following registry entry is required to launch the gateway application automatically after a system boot:

```
[HKEY_LOCAL_MACHINE\Init]
"Launch70" = "\NandFlash\Apps\Gatewayapp.exe"
```

G.1.4 FTDI VCP driver settings

The FTDI VCP drivers require the following registry entries:

```
[HKEY_LOCAL_MACHINE\Drivers\USB\ClientDrivers\FTDI_DEVICE]
"Prefix" = "COM"
"Dll" = "ftdi_ser.dll"
"ConfigData" = 01,00,3f,3f,10,27,88,13,c4,09,e2,04,71,02,38,41,9c,80,4e,c0,34,\
    00,1a,00,0d,00,06,40,03,80,00,00,d0,80
"InitialIndex" = DWORD:0
"DeviceArrayIndex" = DWORD:0
"LatencyTimer" = DWORD:2

[HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients\1027_24577\Default\
    Default\FTDI_DEVICE]
"DLL" = "ftdi_ser.dll"

[HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients\Default\Default\
    255\FTDI_DEVICE]
"DLL" = "ftdi_ser.dll"
```

G.2 *project.bib* entries

The entries in *project.bib* ensure that the files listed are included in the OS image during the build process. The VCP drivers and *IO_control.exe* were added to the *FILES* section of *pjoect.bib*. Their presence in the *\Windows* directory was confirmed after the OS was deployed to the gateway platform.

FILES

FTDIPORT.INF C:\WINCE600\OSDesigns\GatewayOS\GatewayOS\GatewayRequiredFiles\ FTDIPORT.INF NK SU

ftdi_ser.dll C:\WINCE600\OSDesigns\GatewayOS\GatewayOS\GatewayRequiredFiles\ ftdi_ser.dll NK SU

IO_control.exe C:\WINCE600\OSDesigns\GatewayOS\GatewayOS\GatewayRequiredFiles\ IO_control.exe NK SU