University College London

Department of Electronic and Electrical Engineering

TeleCommunications
for Industry

kOS

"kind of Operating System"

Hamed Haddadi
MSc Project Thesis
August 2004
Supervisor: Dr Lionel Sacks

Abstract

"kOS" is the name given to a "kind of Operating System" which is the processing heart of the nodes that are used in the SECOAS project. SECOAS is a research project which aims to investigate a range of novel and emerging technologies needed to create self-organizing networks of microcontrollers, integrate the best ideas into a sensor network, and prove that the network can be used by scientists to meet the needs of a dynamic and challenging sensing application.

The operational requirements of the SECOAS project has led to unique features of kOS; it is very simple, it has a very small footprint (16 Kbytes) and yet it allows data distribution and execution of networked algorithms, therefore ensuring a powerful, robust and reliable overall system architecture.

One of the most important requirements from a microcontroller-based sensor network node is efficiency. It is extremely hard, if not impossible, to change batteries of these nodes when deployed in harsh environments. The operating system must be able to manage the resources carefully, using the individual modules only when required and turn them off as soon as their scheduled tasks are completed. It must also be able to recover successfully from unexpected errors and allow re-programmability and distribution of new performance policies through the network.

In the context of this project, various aspects within the operating system of wireless sensor networks are investigated using the kOS operating system. The main objective of this project is to increase the efficiency and power awareness of kOS using short-wake and long-sleep duty cycles, predictive task scheduling of nodes with minimum interruption to the system operation and smart use of different clocking methods to offer the least possible power budget figures.

This report contains a brief introduction to wireless sensor networks and their applications, operating systems deployed in the sensor nodes, development of kOS and its characteristics, efficiency issues in kOS and a summary of improvements that can be achieved using energy-aware algorithms and power-efficiency techniques.

Acknowledgements

My most sincere gratitude goes to Dr. Lionel Sacks for his support and supervision from the day that I started working in the Advanced Communications Systems Engineering (ACSE) group until the completion of this project. His help and advice led to my confidence in the choice of the project's direction and more importantly my future life.

I would like to thank Dr. Matthew Britton who helped me to identify the objectives of the SECOAS research project and guided me in developing the requirements of the kOS architecture.

I am truly grateful of all the practical help and advice given to me by Gerald McBrearty and Andrew Moss of EE laboratories. I also appreciate the cooperation of EE system support team for providing network facilities and hardware specifically for the use of this project. Rabina Choudhry has also provided helpful advice and assistance.

I would also like to appreciate the help, advice and support given to me by my family and all the partners involved in the SECOAS project, including, but not limited to, Dr. John Argirakis of Intelysis, and my friends at the ACSE group, Toks Adebutu, Adrian Ching, Aleksander Lazarevic, Aghileh Marbini, Danish Mistry, Venus Shum, Ibiso Wokoma and all those friends such as Jonas Griem, Jason Spencer and Nikolaos Vardalachos who made my time an unforgettable one in the Department of Electronic and Electrical Engineering, University College London.

Finally I wish to acknowledge the EPSRC for providing the scholarship which made it possible for me to enter the world of telecommunications while experiencing the true joy of student life from a new perspective.

Contents

Abbreviations and Acronyms

ACK: *Acknowledgement*

API: *Application Program Interface*

bps: *bits per second*

CSMA: *Carrie Sense Multiple Access*

CTL: *Control Line*

CTS: *Clear-To-Send*

DSP: *Digital Signal Processing*

EEPROM: *Electrically Erasable Programmable Read-Only Memory*

EMI: *Electro-Magnetic Interference*

GPS: *Global Positioning System*

I^2C: *Inter-Integrated Circuit*

IDE: *Integrated Development environment*

IP: *Internet Protocol*

ISM: *Industrial, Scientific & Medical*

ISO: *International Standards Organization*

MAC: *Medium Access Control*

MCU: *Microcontroller Unit*

MIPS: *Millions of Instructions per Second*

MPEG: *Moving Picture Experts Group*

OSI: *Open systems Interconnect*

PCB: *Printed Circuit Board*

QoS: *Quality of Service*

RISC: *Reduced Instruction Set Computer*

RF: *Radio Frequency*

RFI: *Radio Frequency Interference*

RTOS: *Real Time Operating System*

RTS: *Ready-To-Send*

RXD: *Receive Data*

TCP: *Transmission Control Protocol*

TTL: *Transistor-Transistor Logic*

TXD: *Transmit Data*

UCL: *University College London*

UDP: *User Datagram Protocol*

USART: *Universal Synchronous Asynchronous Receiver Transmitter*

UTP: *Unshielded Twisted Pair*

WDT: *Watchdog Timer*

Figures

Tables

1. Introduction to the project

The field of telecommunication has been experiencing tremendous changes in the last two decades. The universal growth and penetration of the internet and the World Wide Web, need for network technologies as a result of sudden growth in computing power and Moorse Law, the introduction of mobile phones and the sudden increase in their deployments are just a few of the examples of technological and business advancements in this field.

Recently the technology improvements in the area of wireless telecommunications have led to an exciting research field into deployment of wireless sensor networks. A sensor is a device that responds to a physical stimulus, such as thermal energy, electromagnetic energy, acoustic energy, humidity, RF Signal Strength, pressure or motion, by producing a signal, usually electrical. Sensor networks consist of a collection of microcontroller-based nodes that are equipped with sensors and are used for measurement and monitoring activities. Sensor networks are often mistaken for ad-hoc networks of wireless computers. The number of sensor nodes in a sensor network can be many orders of magnitude higher than the nodes in an ad-hoc network and also sensor networks are usually used for broadcast communication, whereas most ad hoc networks are based on point-to-point communications.

Environmental monitoring is one of the greatest areas of deployments of sensor networks. Currently, oceanographic studies frequently involve using large, expensive devices to log data, typically for several months at a time. During each deployment, there are substantial risks that the platform will be damaged or destroyed. This approach also has the disadvantage that the sensors measure environmental phenomenon only at one physical location. An alternative is to use a network of sensors to build a spatial-temporal picture of the environment. This leads to a system that is robust even when nodes are destroyed or the network topology changes. Furthermore, nodes can be added at will or reconfigured for various purposes. The availability of increasingly low-cost microprocessors and radio devices has made this approach feasible from an economic point of view.

Following this approach, SECOAS project was designed to be an autonomous network of simple sensors that are influenced partially through the use of user-forwarded policies, but which primarily coordinate themselves with adaptive

behaviour. In this network, data is efficiently and continuously collected from the network with predictable levels of performance and delay. Moreover, mechanisms for energy savings at foreseeable intervals without serious impact on transport capacity and overhead are required. Data transport is maintained and gracefully reconfigured after individual or collective failure. A summary of the project and related research issues are addressed in [I]. *Figure 1* shows the top-level view of the system components. In the figure, there is one base-station node on the land and a number of floating sensor nodes in the ocean. The base-station node sets receive and transmit timing of nodes in a hierarchy that is formed by its inducing transmissions. In real deployment scenario there may be more base-station nodes, some within the ocean environment, and the current estimate of distance between the nodes is around 1km. This may change in time due to ocean environment and drift on the sand bank. More on this is addressed in [II]. Policies flow to the network from the base-station, whilst data flows back to shore from the sensor nodes [1].

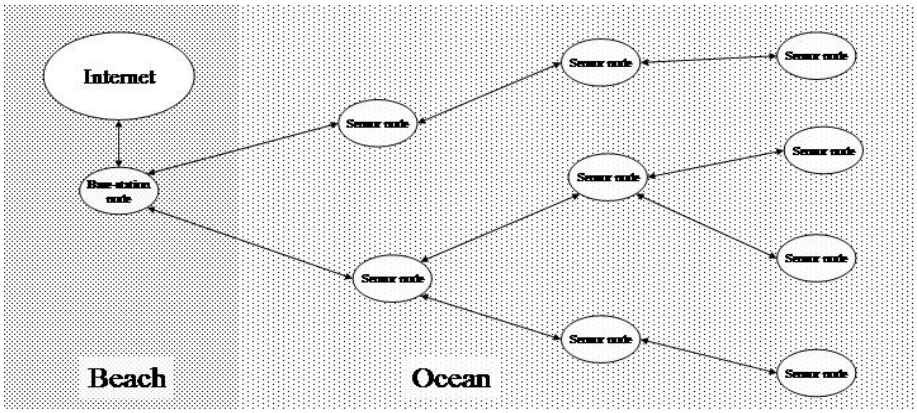


Figure 1: System Components [1]

The kOS operating system is developed at UCL EE laboratories as a unique and innovative Real Time Operating System (RTOS) to support these aims in the context of the SECOAS project. It is a simple, very small footprint (16 Kbytes) operating system designed to support intrinsically networked applications on a basic microcontroller unit (MCU). This project aims to investigate the development and efficiency of kOS.

1.1 The aims of the project

In the context of this project, the aims are:

- To develop certain requirements of the kOS architecture, including data communication, and to look at the competence of kOS when compared with other operating systems developed in this arena.

- To study into power efficiency and implementation of energy saving techniques, effective data communication strategies and maximum resource utilization.

- To design and test methods to minimize the power usage of the sensor nodes and to calculate life on batteries in different configurations.

- To consider the effectiveness of idle and active states and compare the latencies and devise competent methods for compromising between the use of different systems states and complexity of task operations.

1.2 Project context and motivation

Sensor networks are one of the most innovative ideas which researchers across the world are working on. Every day many new papers are published on new technologies deployed in this field and there are increasing number of application-specific projects looking at feasibility of sensor networks for environmental, security and marketing monitoring applications. Distributed Sensor networks are used in many different configurations and varieties, from fixed sensor nodes to mobile nodes, from wired communications to wireless RF devices, from static network topology to dynamically changing topology. However these technological advances have also brought new challenges to processing large amounts of data in a bandwidth-limited, power-constrained, unstable and dynamic environment [6]. In any embedded system design, the power efficiency of the system tends to be one of the most important issues. This is due to the fact that the system may spend days and weeks in idle stage and it is usually hard to change the batteries of these embedded systems. Hence in most sensor

network systems, specific attention should be paid to the design of an optimum operating system for the application.

SECOAS project started in September 2003 in order to explore the employment of self-organizing sensor networks for environmental monitoring applications. The project is DTI funded under the Next Wave Technologies and Markets program. The main objectives of the SECOAS project are:

- To discover and demonstrate decentralized algorithms that enable automated adaptation to failures, upgrades and requirement changes in a distributed network of micro-controllers (smart sensors)
- To investigate and demonstrate novel cooperative adaptive data handling techniques
- Design lightweight, low power, ad-hoc wireless communication protocols that can adapt to a wide range of physical layer media, and support a range of end to end guarantees for network services
- To explore methods for implementing locally intelligent sensors capable of dynamic self-configuration
- To develop and test an appropriate control interface for scientific user communities
- To demonstrate and prove the new technologies in a realistic application context
- To undertake a major re-evaluation of environmental sensing field methodologies, and design new approaches that fully exploit the new technology [4].

The deployment area is the Scroby Sands Wind Farm [5] where the sea bed movements will be monitored. The project partners include University College London, University of East Anglia, BTexact Laboratories, Intelisys, University of Essex, Plextek and Engineering and Physical Sciences Research Council. The outcome of the SECOAS project will benefit the researchers of telecommunications and system design to get more familiar with sensor network systems. On completion, the sensor nodes and their operation will enable a thorough investigation of effects of wind farms and environmental characters of the deployment area and the result will

directly affect future operation of offshore wind farms in areas where they can have an adverse effect on the wild life or the environment.

The development of the kOS Operating System started in summer 2003, with the first version and the user guide being released by the author at August 2003. Since then the development on the kOS has been continued in simulation, with specific emphasis on the networking and data transfer algorithms. However as the project proceeds towards the implementations stage, the analysis of power budget calculation, efficiency of the operating systems, data transfer link utilization and MCU power requirements in terms of sleep cycles becomes critical. This project aims to investigate the power issues within the kOS, while developing the necessary software algorithms and hardware platform to enable task scheduling, sleep cycles and data transfer from the sensor modules to the MCU and from the MCU to the radio units used to transmit the data on Radio Frequency (RF) links to the base station nodes. The deployment of the sensor nodes in ocean means that they should run for as long as possible without any need to change batteries and every care must be taken to use the resources carefully in order to minimize the amount of current drawn from the batteries.

Currently kOS does not support sleep mode and hence the applications can run to completion or they may reschedule themselves for execution while the MCU is awake. Radio communication is usually is the most battery-intensive function in the sensor networks. This includes the energy spent on turning on the radio modules to send data, and also on energy spent on leaving the receivers on in order to listen out for incoming packets so it is not economical to leave the receive channels on idle mode waiting for data to arrive at random times. This brings out the importance of scheduling for applications. Another important issue that rises here is the utilization of communication links and the importance to decrease the time spent on data communications in order to send the MCU to sleep as soon as possible and to turn off the radio module of the MCU.

Similar work has been done before at a few other projects using sensor networks. Researchers at University of California, Berkeley, have proposed flexible power scheduling, a distributed on-demand power-management protocol for collecting data in sensor networks. Flexible Power Scheduling aims to reduce radio power consumption while supporting fluctuating demand in the network. This work has been implemented in the “TinyOS” which is the heart of the Smart Dust project. Recently

the EYES project has addressed problems such as distributed information processing, wireless communications and mobile computing based on battery management and change in operation upon reaching lower energy levels. Some of the related projects will be discussed in more details in the next section. however Many of the related projects have implemented either general Operating System environments, or have used traditional communication protocols, both of which are too heavy for requirements of SECOAS and its goals, which aim to generate a lightweight application specific environment, while allowing for future modification and re-programmability for other applications and potential commercial products.

1.3 Layout of the report

The outline of the report is discussed in this section. Chapter 2 gives a brief overview of sensor networks and their applications within the area of telecommunication. Some of the projects that involve use of sensor networks are discussed in this chapter. It is essential to familiarize the intended readers with the general concept of sensor networks; however this section can be skipped by the specialized readers.

Chapter 3 introduces the Operating Systems of the various important sensor networks developed recently. In this chapter a brief review of the general requirements of the Operating System is collected and they are compared with the requirements of the SECOAS project.

Chapter 4 describes the development of kOS and its architecture. The main concepts behind kOS, the operation of kOS, its unique characteristics, task scheduling and data handling is discussed in this section. This chapter will also introduce the implementation of kOS on software simulation and hardware emulation platform.

Chapter 5 discusses the analysis of efficiency issues within kOS. Suggestions for improvements, energy saving algorithms, their implementation and effects on the overall systems are discussed in this section.

Chapter 6 explains the improvements gained in the operation of kOS by analyzing the result of the experiments performed and it argues the outcomes versus the main goals of the project.

Chapter 7 summarizes the achievements of the project and suggests the future work which can be ensued to improve the concepts behind this project.

2. Sensor networks and their applications

In this chapter a brief review of the recent developments in the field of sensor networks, alongside some example projects is presented. After reading this section the reader will be more familiar with the general architecture of sensor networks and their potential applications. Some of the most important relevant research projects are also explained and referenced in this section.

2.1 Defining sensor networks

Sensor networks are usually defined as a collection of distributed microcontroller-based devices that are used to monitor the environment and produce a measurable response to a change in physical condition (such as temperature) or in chemical condition (such as concentration). Sensor networks have recently been the hub of activity of many telecommunication research groups around the world and their development has led to a rapid growth in the variety of applications, implementations and protocols. They are a major part of new research initiatives into Ambient Intelligence and ubiquitous computing. Recently, the concepts of high data rate (such as imaging or video sensors for security applications) and even mobile sensor nets are being considered for trials. Some of the characteristics of sensor networks are listed here:

- Made up of hundreds or thousands small autonomous nodes
- Use wireless technology to communicate
- Require low data rates, from a few bits per second (bps) to a few kbps
- Subject to energy/power constraints (i.e. battery operated)
- Interfaced into control or monitoring systems
- Necessitate low latency and survivable communications [7]

Sensor networks are always formed by a cluster of cheap, small and low power devices composed by integrating low power analogue, digital and RF electronic devices. Every network is typically consisted of many hundreds of these nodes. Sensor nodes are usually fitted with an on-board microcontroller unit, so Instead of sending the raw data to the nodes responsible for data fusion, the nodes use their processing abilities to locally carry out simple computations and transmit only the

required and partially processed data. Even though the MCU-based nodes may have limited memory and data processing, when collaborated together in form of a large distributed wireless system, they can communicate with each other to produce intelligent data processing and execute intense monitoring, self-organising and location-based algorithms.

Sensor nodes collaborate to be able to cope with the environment: sensor nodes operate completely wireless, and are able to spontaneously assemble themselves into a network, dynamically adapt to device failure and degradation, manage movement of sensor nodes, and react to changes in task and network requirements. Despite these dynamic changes in configuration of the sensor network, critical real-time information must still be disseminated dynamically from mobile sensor data sources through the self-organising network infrastructure to the applications and services.

Sensor network systems will enhance usability of appliances, and provide condition-based maintenance in the home. These devices will enable fundamental changes in applications spanning the home, office, clinic, factory, vehicle, metropolitan area, and the global environment. Sensor node technology enables data collection and processing in a variety of situations, for applications, which include environmental monitoring, context-aware personal assistants (tracking of location, activity, and environment of the user), home security, machine failure diagnosis, medical monitoring, and surveillance and monitoring for security [8]. The main objective of using Sensor Nodes in most situations is to avoid having a single point of failure in monitoring applications which have to run un-corrupted for many months. The sensor nodes are smaller, cheaper and less prone to failure. The packages can be physically designed and embedded in a way that maximum protection from environmental hazards is possible. They are designed to minimise the huge costs of powerful and complex centralised data collection and processing platforms which usually get implemented at hazardous environments or in the middle of sea beds and oceans and oil plants.

Amongst the many bright ideas used in sensor networks is the idea of Self-Organisation. Self-Organisation is a capability achieved by using a set of clever algorithms, which are activated upon loss of sensor nodes are more important master nodes. The effect of Self-Organisation is creation of a new network topology where the duties and responsibilities of the lost nodes are distributed between the

surrounding sensor nodes in their neighbourhood to ensure minimum interruption in data collection and task distribution.

The biologically inspired idea of dynamic network set-up is extremely important in areas where thousands of sensors are randomly distributed, for example those thrown out of an airplane in an ocean [9]. The flexibility and operation of such a network is certainly dependant on how fast a network in an ad-hoc manner can be set up. This approach leads to a set of functional requirements. The nodes must be able to configure themselves for monitoring period in space and time, be able to condense data and minimize communication overhead back to the base station, be able to substitute for each other in case of failure, and be able to locate themselves, relative to other nodes, or by using Global Positioning Systems (GPS).The nodes must respond to changes in the environment (both the monitored environment and the network) as autonomously as possible. There is a need for a general, localised control system at a higher level [4]. Some of these requirements will be discussed in more details in the next sections.

2.2 Applications of sensor networks

The research into sensor network applications was originally initiated for military purposes such as battlefield surveillance and enemy tracking. However, just like the internet, sensor networks quickly found their way into commercial and civil applications. Today researchers across the world are looking into the implementation of sensor networks in areas such as:

- Chemical, biological and nuclear detection and monitoring
- Traffic observation, detection and management
- Industrial control systems
- Portable and mobile site monitors
- Security and video monitoring systems

In this section a brief outline of the current areas of use of sensor networks and examples of the research projects and implementations are given.

2.2.1 Environmental monitoring

Environment Observation and Forecasting Systems (EOFS) are large distributed systems that span large geographic areas and monitor, model and forecast physical processes, such as environmental pollution, flooding etc. Usually they consist of three components: sensor stations, a distribution network, centralized processing farm. Some of the characteristics of EOFS are:

- *Centralized processing*: the environment model is very computational intensive; they usually run on a central server and process data gathered from the sensor network.
- *High data volume*: for example, nautical X-band radar can generate megabytes of data per second.
- *Quality of Service (QoS) sensitivity*: it defines the utility of the data; there is an engineering trade-off between QoS and energy constraint.
- *Extensibility*
- *Autonomous operation*[10]

Some of the examples of environmental monitoring using sensor networks are listed below:

I- CORIE: it is a prototype of environmental monitoring application for Columbia River. 13 stationary sensor nodes are deployed across the Columbia River estuary, 1 mobile sensor station drifts off-shore. Those sensor stations are usually fixed on a pier or a buoy. The stationary stations are powered by power grid, while the mobile station uses solar panel to harness solar energy. Sensor data are transmitted via wireless link toward onshore master stations. The data is then further forwarded to a centralized server and fed into a computationally intensive physical environment model. The output of the model is used to guide vessel transportation and forecasting.

II- ALERT: Automated Local Evaluation in Real-Time is probably the first well-known wireless sensor network being deployed in real world. It was developed by the National Weather Service in the 1970's. ALERT provides important real-time rainfall and water level information to

evaluate the possibility of potential flooding. ALERT sensor sites are usually equipped with meteorological/hydrological sensors, such as water level sensors, temperature sensors, and wind sensors, data are transmitted via light-of-sight radio communication from the sensor site to the base station, a Flood Forecast Model is adopted to process those data and issue automatic warning, web-based query is available. Currently ALERT is deployed across most of the western United States and is heavily used for flood alarming in California and Arizona.

III- Great Duck Island system: In August 2002, researchers from University College Berkeley (UCB) and Intel Research Laboratory deployed a mote-based tiered sensor network on Great Duck Island, Maine, to monitor the behaviour of storm petrel.

IV- PODS-A Remote Ecological Micro-Sensor Network: PODS is a research project in University of Hawaii that built wireless network of environmental sensor to investigate why endangered species of plants will grow in one area but not in neighbouring areas. They deployed camouflaged sensors node, called Pods, in Hawaii Volcanoes National Park. The Pods consist of a computer, radio transceiver and environmental sensors sometimes including a high resolution digital camera, relay sensor data via wireless link back to the Internet. Bluetooth and 802.11b are chosen as Medium Access Control (MAC) layers and data is delivered in Internet Protocol (IP) packets. Energy efficiency is identified as one of the design goals, an ad-hoc routing protocols called Multi-Path On-demand Routing was developed. Two types of sensor data are collected, weather data are collected every ten minutes and image data are collected once per hour, users can use Internet to access the data from a server in University of Hawaii. More information can be found in [III].

2.2.2 Health Applications

Applications in this category include monitoring of human physiological data, tracking and monitoring of doctors and patients inside a hospital. In the Smart Sensors and Integrated Microsystems (SSIM) project, retina prosthesis chips consisting of 100 micro sensors are built and implanted within the human eye. The aim of the SSIM

project is to enable patients with no or limited vision to see distant objects at an acceptable level of clarity. The wireless communication is required to suit the need for feedback control, image identification and validation. Other similar applications include Glucose level monitors, Organ monitors, Cancer detectors and General health monitors. The idea of embedding wireless biomedical sensors inside human body is promising, although many additional challenges exist: the system must be extremely safe and reliable, require minimal maintenance and ideally attract energy from the body heat [10]. The work in the medical area requires clusters of extremely small sensors, with dimensions in orders of few millimetres, which can be embedded under the skin or planted in vital organs. This is an exciting and recent area of research and its scopes are beyond the field of this document.

2.2.3 Commercial applications

Sensor networks are implemented in various home, office and industrial applications. Some of the important projects are listed in this section.

I- Structure Health Monitoring (SHM) System: SHM is another important domain for sensor network application. The widely accepted goals of SHM system include detecting damage, localizing damage, estimating the extent of the damage and predicting the residual life of the structure, as proposed in [IV]. SHM has been an evolving technology since it was first proposed in 1990's, the latest approach, which uses wireless sensor networks, has many advantages: low deployment and maintenance cost, large physical coverage, high spatial resolution etc. One of the barriers is that damage detection is very difficult even for sophisticated sensors, thus breakthrough in damage detection using small MEMS sensors is much needed. So far, a SHM system using wireless sensor network technology is yet to emerge.

II- Smart Energy: Societal-scale sensor network can greatly improve the efficiency of energy-provision chain, which consists of 3 components, the energy-generation, distribution, and consumption infrastructure. It is reported that 1 percent load reduction due to demand response can lead to a 10 percent reduction in wholesale prices, while a 5 percent load response can cut the wholesale price in half. In the wake of recent energy regulation

in California [11], proposes a gradual roll-out plan to make energy supply chain part of an integrated network of monitoring, information processing, controlling, and actuating devices, in a hope to spread the consumption of energy over time reducing peak demand. That would be a complex and long-term project [10].

2.3 Protocols and implementation techniques

There is currently no standards organisation involved in the design and implementation of sensor networks. However, implementation of a distributed sensor network of wireless devices, like any other telecommunication network, needs clear definition of all the levels and protocols in the Open Systems Interconnect (OSI) stack. The International Standards Organisation (ISO) developed the OSI architectural model. *Figure 2* shows the components of the OSI stack and example protocols. The term stack is used to show that in order to get from a layer in system to a layer in another system data needs to traverse through all the lower layers.

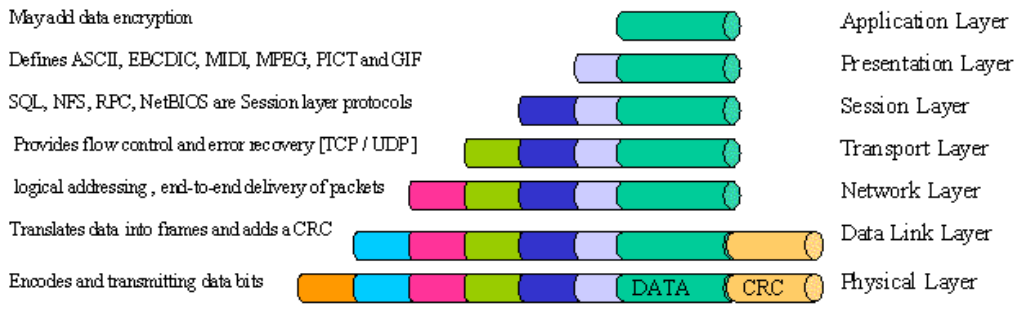


Figure 2: OSI Stack (Figure courtesy of ISO [27])

The physical layer is a largely unexplored area in sensor networks. Open research issues range from power-efficient transceiver design to modulation schemes:

• Modulation schemes: Simple and low-power modulation schemes need to be developed for sensor networks. The modulation scheme can be either base band, as in UWB, or pass band.

• Strategies to overcome signal propagation effects

• Hardware design: Tiny, low-power, low-cost transceiver, sensing, and processing units need to be designed. Power-efficient hardware management strategies are also essential. Some strategies are managing frequencies of operation, reducing switching power, and predicting work load in processors.

In the data-link layer, although some Medium Access Control (MAC) schemes have been proposed for sensor networks, link layer protocol design is still largely open to research. Key open research issues include:

• MAC for mobile sensor networks: The proposed Self-Organizing Medium Access Control for Sensor Networks (SMACS) and the Eavesdrop-And-Register (EAR) algorithms [12] perform well only in mainly static sensor networks. It is assumed in the connection schemes that a mobile node has many static nodes as neighbours. These algorithms must be improved to deal with more extensive mobility in the sensor nodes and targets. Mobility issues, carrier sensing, and back-off mechanisms for the Carrier Sense Multiple Access (CSMA)-based scheme also remain largely unexplored.

• Determination of lower bounds on the energy required for sensor network self-organization

• Error control coding schemes: Error control is extremely important in some sensor network applications like mobile tracking and machine monitoring. The feasibility of other error control schemes in sensor networks needs to be explored.

• Power-saving modes of operation: To prolong network lifetime, a sensor node must enter into periods of reduced activity when running low on battery power. The enumeration and transition management for these nodes is open to research.

The networking layer of sensor networks is usually designed according to the following principles:

• Power efficiency is always an important consideration.

• Sensor networks are mostly data-centric.

• Data aggregation is useful only when it does not hinder the collaborative effort of the sensor nodes.

• An ideal sensor network has attribute-based addressing and location awareness.

Flooding is an old technique that can also be used for routing in sensor networks. In flooding, each node receiving a data or management packet repeats it by

broadcasting, unless the maximum number of hops for the packet is reached or the destination of the packet is the node itself. Flooding is a reactive technique, and it does not require costly topology maintenance and complex route discovery algorithms. A derivation of flooding is gossiping in which nodes do not broadcast but send the incoming packets to a randomly selected neighbour [13]. A sensor node randomly selects one of its neighbours to send the data. Once the neighbour node receives the data, it randomly selects another sensor node. Although this approach avoids the implosion problem by just having one copy of a message at any node, it takes a long time to propagate the message to all sensor nodes.

The development of transport layer protocols is a challenging effort because the sensor nodes are influenced by the hardware constraints such as limited power and memory. As a result, each sensor node cannot store large amounts of data like a server in the Internet, and acknowledgments are too costly for sensor networks. Therefore, new schemes that split the end-to-end communication, probably at the sinks, may be needed where User Datagram Protocol (UDP)-type protocols are used in the sensor network and traditional Transmission Control Protocol (TCP)/UDP protocols in the Internet or satellite network.

Implementation of each stack level protocol must be considered carefully in order to maximise the throughput of the system. For a brief description of some of the implementation architectures refer to [V].

2.4 Research projects on sensor networks

There are currently many research groups across the world working on different aspects of sensor networks, from the physical characteristics to application layer. Some of the most important projects and their context and scopes are briefly described in this section.

2.4.1 Smart Dust Project

Researchers at University of California, Berkeley (UCB) have been working on the Smart Dust project for many years. The goal of the Smart Dust project is to build a self-contained, millimetre-scale sensing and communication platform for a massively distributed sensor network. Smart Dust nodes, called “motes”, are desired to be

around the size of a grain of sand and will contain sensors, computational ability, bi-directional wireless communications, and a power supply, while being inexpensive enough to deploy by the hundreds. The science and engineering goal of the project is to build a complete, complex system in a tiny volume using state-of-the art technologies (as opposed to futuristic technologies), which will require evolutionary and revolutionary advances in integration, miniaturization, and energy management. A mote consists of:

• Processor: Atmel AVR ATmega 128L μcontroller:
– 128KB flash ROM, 4KB RAM, 4KB EEPROM
– Up to 8 MHz
• Radio: Chipcon CC1000:
– UHF transceiver (300 MHz – 1 GHz)
– FSK modulation, up to 76.8 kBaud
• Sensor boards

In the Smart Dust network, three types of nodes are present: Application Nodes, which are in charge of running data processing tasks, Network Nodes, which synchronise the network, and Base Stations, which are the hubs for data communication with the main database. *Figure 3* presents the proposed network architecture of the Smart Dust mote-based network. They envision that this smart dust will be integrated into the physical environment, perhaps even powered by ambient energy [14]. The motes use a main microcontroller and an extra one as a co-processor. The Smart Dust project has now finished, but many interesting ideas were developed as a result of this project, the most important one being the development of the TinyOS operating system, which is discussed in more detail in next chapter.

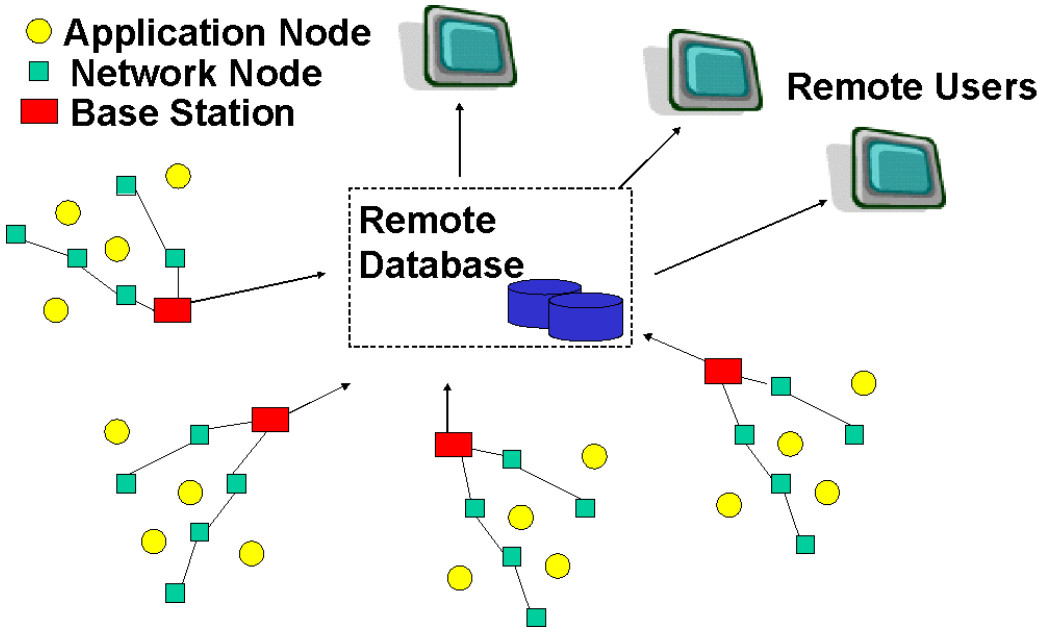


Figure 3: UCB Smart Dust network (Figure courtesy of UCB [11])

2.4.2 EYES Project

The EYES project is a three years European research project, on self-organizing and collaborative energy-efficient sensor networks. It addresses the convergence of distributed information processing, wireless communications, and mobile computing. The project runs from March 2002 till February 2005 [15].

The EYES network has two distinct key system layers of abstraction: the sensor and networking layer, and the distributed services layer. Each layer provides services that may be spontaneously specified and reconfigured.

The sensor and networking layer contains the sensor nodes (the physical sensor and wireless transmission modules) and the network protocols. Ad-hoc routing protocols allow messages to be forwarded through multiple sensor nodes taking into account the mobility of nodes, and the dynamic change of topology. Communication protocols must be energy-efficient since sensor nodes have very limited energy supply. To provide more efficient dissemination of data, some sensors may process data streams, and provide replication and caching.

The distributed services layer contains distributed services for supporting mobile sensor applications. Distributed servers coordinate with each other in order to perform decentralised services. These distributed servers may be replicated for higher

availability, efficiency and robustness. We have identified two major services. The lookup service supports mobility, instantiation, and reconfiguration. The information service deals with aspects of collecting data. This service allows vast quantities of data to be easily and reliably accessed, manipulated, disseminated, and used in a customized fashion by applications [15].

On top of this architecture applications can be built using the sensor network and distributed services. *Figure 4* displays the EYES sensor network architecture.

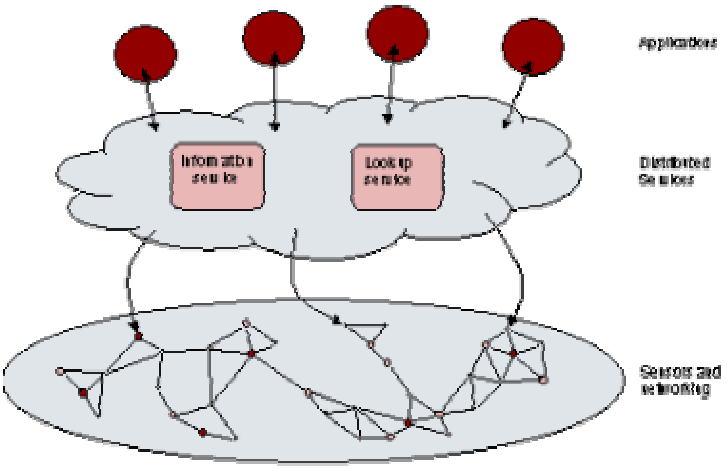


Figure 4: EYES sensor network architecture [15]

2.4.3 Bluetooth-based sensor networks

Bluetooth is a short range (up to around 10 metres) RF connectivity protocol which has been the centre of attention of many manufacturers of home and mobile products for short range wire replacement. Bluetooth communicates on the frequency of 2.45 Gigahertz, which has been set aside by international agreement for the use of Industrial, Scientific and Medical (ISM) devices. *Figure 5* shows the Bluetooth protocol stack.

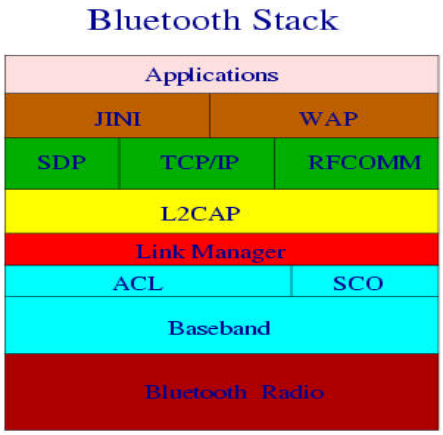


Figure 5: Bluetooth protocol stack [16]

The most important general attributes of Bluetooth are:

- Use of Frequency Hop (FH) spread spectrum, which divides the frequency band into a number of hop channels. During a connection, radio transceivers hop from one channel to another in a pseudo-random fashion.
- Supports up to 8 devices in a Pico-net (two or more Bluetooth units sharing a channel).
- Built-in security.
- Non line-of-sight transmission through walls and briefcases.
- Omni-directional. Supports both isochronous and asynchronous services; easy integration of TCP/IP for networking [16].

An ongoing research project at the Intelligent Systems Group at Malmo University is based on Bluetooth technology. A prototype Wireless Sensor Network consisting of one access point and several sensor point nodes based on Bluetooth technology has been developed and tested. The focus of the design work was the RFCOMM layer of the Bluetooth stack. The structure of the WSN, shown in *Figure 6*, is built up in such a way that an access point, represented by an Ericsson Bluetooth module attached to a PC, served as a controller, allowing sensors to send data on request. The sensors are implemented as true embedded objects, waiting for the access point to wake them up and start sending data [17].

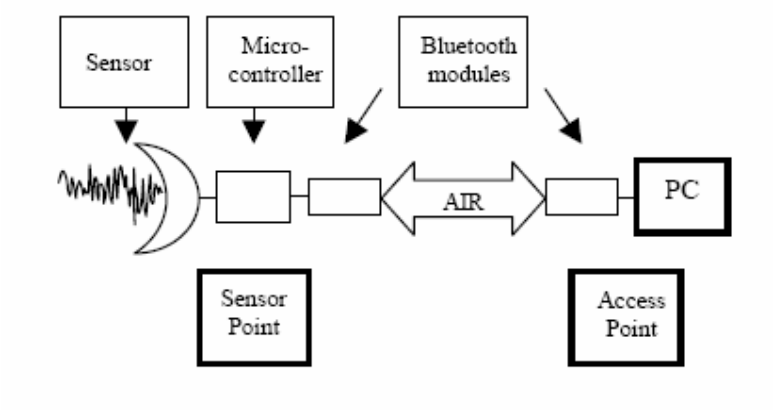


Figure 6: Sensor network using Bluetooth (Figure courtesy of Malmo University [17])

However the biggest disadvantage of Bluetooth is its extremely short range. It is impractical to use Bluetooth for large environmental monitoring projects. Another major obstacle when deploying Bluetooth protocol is its complicated stack and lengthy set-up time, which may interfere with the ideas behind fast synchronisation and self-organisation.

2.5 SECOAS project

SECOAS is a research project, funded by UK department of Trade and Industry, which is aiming to trial a sensor network monitoring offshore sedimentation processes at Scroby Sands offshore wind farm. SECOAS project officially started in summer 2003, with BTexact Technologies, University College London department of Electronic and Electrical Engineering being responsible for the application development and system architecture. The full trial of the systems using approx 50 nodes is going to be deployed in summer 2005. The trial is designed to demonstrate technological solutions to problems in communication and maintenance in a difficult environment, and also to prove the science benefits of the high spatial resolution measurements the network enables. Immediate applications are foreseen in coastal defence and water management.

The wind farm development located on the Scroby Sands sand bank provides the key requirements capture for the SN development. The location of wind turbines on sand banks can produce highly complex turbulent flows both locally, producing sand shift and scouring of the turbine bases; and on a wider scale having impacts on along-shore sand banks and sea barriers. Measuring the ocean in this environment is

currently expensive. Typical "Landers" will cost £100-200K to build including sensors, acoustic releases and etc and have high deployment and retrieval costs (as the lander must be precision located). A single lander can only measure the environment at a single point; it is vulnerable to destruction by storms, burial by moving sand waves and accidents from trawlers – as such it is a 'one shot' proposal. In science terms, the key problem is that the current technologies do not give good coverage in either time or space. For most of each deployment little happens so power and data storage is wasted; operational cycles are always compromises between the need for high resolution measurements during the active periods (usually storms) and the uncertainties in the frequency, timing and severity of the active periods. Typically, most monitoring systems undergo fixed sleep-sample duty cycles which are good enough for monitoring regular features (e.g. tides) but cannot reactively monitor transit effects. Turbulence is far from regular through space and there is clear need for high special sampling to measure features such as sand: drift, scouring and build up.

The initial main goals of the SECOAS project are:

- The sum of the system's physical maintenance, training, node unit and deployment costs must be competitive with current environmental monitoring practices.
- The network must operate in hazardous or remote environmental conditions.
- The project must demonstrate the wide applicability of the techniques and technologies used.
- The project must ensure the demonstration of autonomous collegiate management in a wireless sensor network [19].

Figure 7 shows the Function flow block diagram of the SECOAS operation.

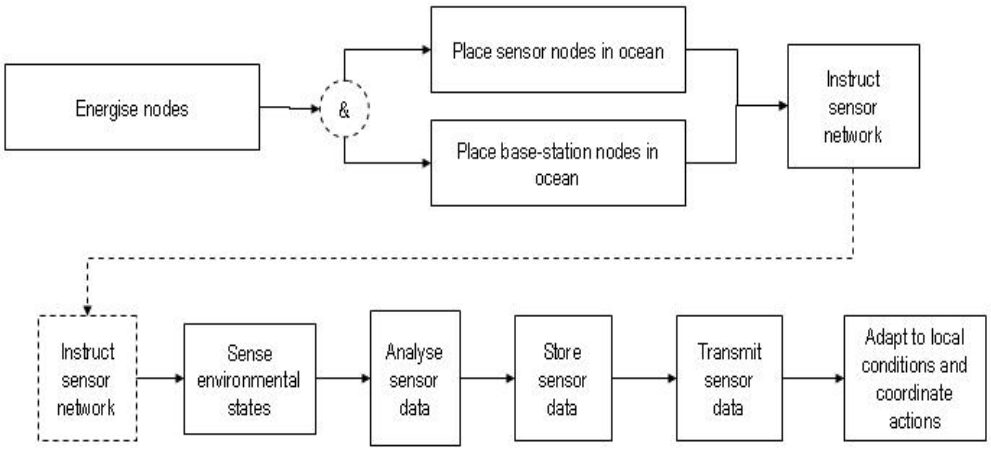


Figure 7: Function flow block diagram of SECOAS system [19]

The approach developed in SECOAS is to develop a SN platform consisting of a large number (30-50) of low cost (<£1000) nodes. Each node has basic functionality; a small μ-processor, low rate communications and sensing capabilities. Sensing capabilities will focus on features such as, optical backscatter –a measure of sediment load in the water – pressure, and temperature. In addition to the general sensor nodes, there will be a few master packages which can measure other parameters (e.g. surface pressure) and for communications with the science base-stations. For deployment the nodes could be scattered from a boat, if they are capable of determining their own location. Key amongst the technical requirements is the need for the nodes to be able to communicate between them selves on a nearest neighbour basis. This can be achieved using surface-floating tethered buoys supporting low cost, low power radio communications such as IEEE802.15.4, operating in the IMS spectrum with low data rates (250 kbps) and high node densities (~250) and low power demands (*e.g.* 30μW for a 1000/1 sleep/transmit cycle). For more general applications sonar communications are a viable alternative. This approach brings its own set of functional requirements. The nodes must be able to configure them selves for monitoring frequency in space and time, to be able to condense data and communication it back to the base station, to be able to substitute for each other in case of failure, and be able to locate them selves. The nodes must respond to local changes in the environment (both the monitored environment and each other) as autonomously as possible. Thus there is a need for a general, localised control system

with the whole system manageable by the scientist / operator. It is not possible for the operator to control the each node individually so the system must be manageable at a high level. More information about the general directions of the SECOAS project can be found in [I].

One of the main requirements of the SECOAS project is a low-power operation. Idle modes in the MCU and low duty cycles will help achieving this. Work on the system architecture and the operating system of the SECOAS project started with development of applications in simulation environment, and test are currently being carried out on a PIC18F452 MCU from Microchip. The most important specifications of this chip are:

- 32 Kbytes flash program memory, allowing up to 16384 single word instructions to be executed.
- 256 Bytes EEPROM memory with 40 year data retention
- C compiler optimized architecture/instruction set
- Linear program memory addressing to 32 Kbytes
- Linear data memory addressing to 1.5 Kbytes
- Up to 10 Millions of Instructions Per second (MIPS) operation
- 16-bit wide instructions, 8-bit wide data path
- Priority levels for interrupts
- 8 x 8 Single Cycle Hardware Multiplier
- Three external interrupt pins
- Three timers with interrupt capability and counting during sleep execution
- Addressable USART module, Supports RS-485 and RS-232
- Inter-Integrated Circuit (I^2C)™ Master and Slave mode
- Watchdog Timer (WDT) with its own On-Chip RC
- Power saving SLEEP mode
- Low power consumption:
  - < 1.6 mA typical @ 5V, 4 MHz
  - 25 µA typical @ 3V, 32 kHz
  - < 0.2 µA typical standby current
- Compatible 10-bit Analogue-to-Digital Converter module (A/D) with:
  - Fast sampling rate
  - Conversion available during SLEEP [18]

The PIC MCU resides in the application board which has other components, such as a crystal, external EEPROM, RS232 interface and regulators. In the context of this report, this collection is called the kOS board and is considered to be designed on a single Printed Circuit Board which is powered by an external battery.

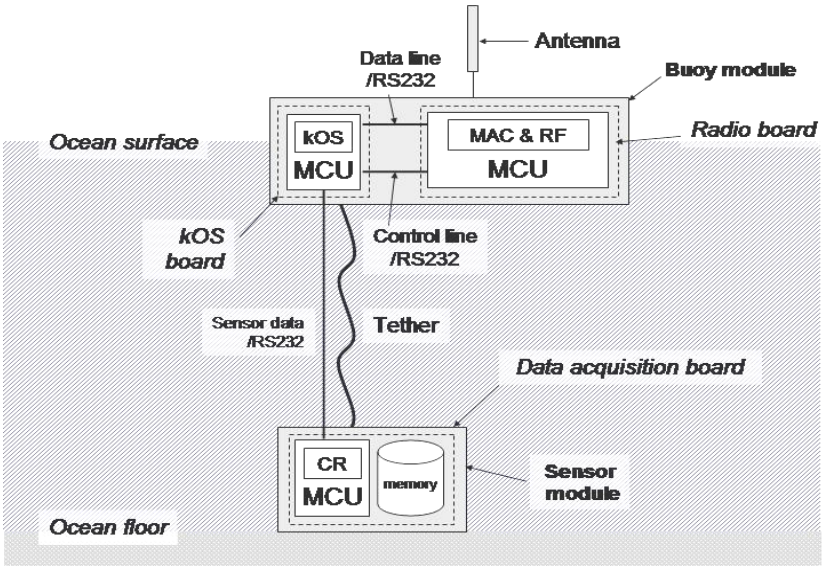


Figure 8: System Configuration of Sensor Node [1]

Figure 8 shows the system configuration of a SECOAS sensor node. The kOS board, which includes the MCU and any external EEPROM memory, resides in a floating buoy together with a radio board. This buoy is connected to a submerged sensor module which logs data locally. The sensor module occasionally passes important data to the kOS board which stores it and forwards it to the base-station. Various applications use this data and communicate with themselves across the network. User polices and application data are forwarded by the radio module to the kOS board. Information may be passed to the sensor module in the case of data re-transmission requests [1].

There are two types of nodes in SECOAS architecture: Sensor Nodes and Base-station Nodes. Figure 9 displays the state machine for a sensor node operation. Baste-Station nodes are the link between the sensor network and the shore-based node. Base station initiates synchronization throughout the network by transmitting “beacons” and collects the data from sensor nodes. Policy distribution and software upgrades are

also done via base station nodes. This project is mainly concerned with the operation of sensor nodes, as they are the heart of the system, collecting data and distributing tasks and applications. *Figure 9* displays the state machine for a sensor node operation.

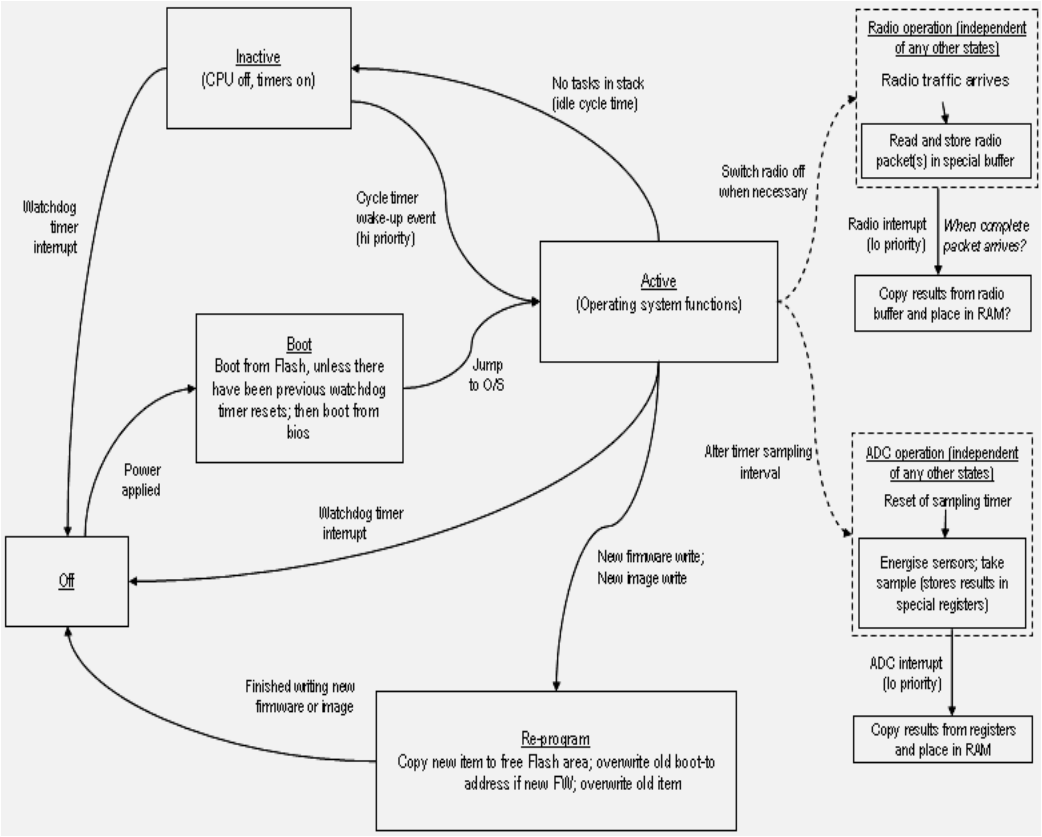


Figure 9: State machine of node operation

3. Operating system in sensor networks

Operating system is defined as a large and complicated computer program that kicks in from the moment a computer, or in smaller devices a microcontroller, is powered and booted up. Commercial operating systems usually allow simultaneous execution of many big complicated computer programs peacefully on one physical computer. The operating system is also responsible for hiding the details of the computer hardware from the application programmers. This last characteristic is an extremely important one for comparison of computer operating systems with sensor network operating systems. This chapter brings a brief introduction to a few of the operating systems used for sensor networks and discusses the reasons that make these operating systems unsuitable for objectives of the SECOAS project.

3.1 Operating system requirements for sensor networks

Operating systems in wireless sensor communication increasingly must satisfy a tight set of constraints, such as power and real time performance, on heterogeneous software and hardware architectures. In this domain, it is well understood that traditional general-purpose operating systems are not efficient or in many cases not sufficient for these types of complex real time, power-critical domain specific systems implemented on advanced heterogeneous architectures. More efficient solutions are obtained with operating systems that are developed to exploit the reactive event-driven nature of the domain and have built-in aggressive power management. General-purpose operating systems do not target low power applications, they have no built-in energy management mechanisms [20].

For many sensor network nodes the identification and implementation of appropriate operating system primitives is still a research issue. In many current projects, applications are executing on the bare hardware without a separate operating system component. In sensor networks, the operating system must have a few specific properties which are listed here:

- Compactness: The operating system is typically running on an extremely small microprocessor with limited on-board memory, so it must be small.
- Simplicity: Even though the operating system is small, it should still allow simple implementation of multi-tasking and memory management.

- Robustness: The operating system running on the nodes must be robust and facilitate the safe execution of reliable distributed applications.
- Reliability: The nodes can not be manually reset, so the operating system must use all the available facilities, such as the Watchdog Timer (WDT), to enable reliable operation and self-maintenance.
- Remote operation: The operating system must be remotely re-programmable to enable remote changes in network operation.
- Power awareness: It is extremely important to make efficient use of limited battery resources available to the sensor nodes and it is the job of the operating system to keep the operation mode to minimum time.

3.2 The TinyOS architecture

TinyOS was initially developed by the U.C. Berkeley EE/CS Department [21]. It is the operating system of the motes used in the Smart Dust project. TinyOS specifically targets event-driven communication systems. Because TinyOS is not designed to support a broad range of general applications, it can cut down on expensive OS services such as dynamic memory allocation, virtual memory, etc. In addition, unnecessary performance degrading polling is eliminated and context switching is minimized and very efficiently implemented. On some implementations, special hardware accelerators are used to help the core pieces of TinyOS run more efficiently. For example, the accelerators allowed data encryption to be performed by the hardware, which is thousands of times more efficient than performing the same function in software. TinyOS provides components such as:

• Analogue to digital conversion
• Cryptography
• Data logging
• File system
• I2C communication
• LED control
• Memory allocation
• Random number generation
• Routing

- Sensor board input
- Serial communication (wired and wireless)
- Timers
- Watchdog timer [26]

TinyOS possess certain qualities that are very attractive for low power heterogeneous systems. Its event-driven asynchronous characteristics can naturally support the interactions and communications between modules of vastly different behaviour and processing speeds in a heterogeneous system. Its simplicity incurs minimal overheads and it has some support for concurrency.

TinyOS has its own limitations and is insufficient to fulfil the ambitious role demanded by low power heterogeneous systems. First of all, TinyOS primitives are microprocessor centric, while advanced system architectures consist of heterogeneous modules of custom logic, programmable logic, memories, Digital Signal Processors (DSP), embedded processors, and other optimized domain specific modules. Furthermore, TinyOS only supports rudimentary power management scheme [20]. The logical next step is to extend TinyOS and establish it as the global management framework that incorporates the heterogeneous architecture modules in the system, as well as devise sophisticated power management mechanisms. More detailed analysis of the TinyOS advantages and disadvantages is followed in the next section. Development of TinyOS is continued at UC Berkley laboratories.

3.3 EYES Operating system architecture

The main processor used in the EYES project is MSP430F149, produced by Texas Instruments. It is a 16-bit processor and it has 60 Kbytes of program memory and 2 Kbytes of data memory. It also has several power saving modes. A node is also equipped with an auxiliary serial EEPROM memory of 8 Megabits (used for application and data storage). Effort is made to enable limited power consumption by use of interrupts for data transmission and receive tasks. A task scheduler maintains the execution of tasks. It can be implemented as a simple FIFO schema or a more advanced one allowing priority and deadline driven executions for real-time applications. The interrupts are seen as tasks that are scheduled to be executed.

EYES operating system has to facilities: resource management and a remote procedure call mechanism. Resource management deals with the operations of resource request, declaration and allocation. Resource requests appear in the case that a node has to perform a specific computation but does not have enough memory, energy, or even speed. A query will be submitted to the system (its neighbours) and the ones that can do the computation will respond (resource declaration and allocation occurs). The remote procedure call will be the mechanism that allows the requester to perform the needed computation using other's node resources.

The use of these levels in the EYES operating systems dictates the use of Application Program interfaces (API). There are two API levels, one specifically designed to deal with operating system tasks. The other API level sits between the application and the Distributed System Layer. The application can get the results of the sensor network (the processed data) or can ask the network to adapt and perform a specific function. The network structure and algorithms used are transparent for the end-user [22].

3.4 Operating system requirements of SECOAS project

SECOAS project requirements are for a slowly-changing topology, gossip-based communications transport over a minimal communications stack and the interchange-ability of node roles. Hence there is little need for network addressing or routing. The EYES project, and its associated EYES operating system is much closer to our goals in that it seeks to support a self-organising sensor network of distributed applications. The EYES operating system is a general-purpose operating system for wireless sensor networks and it also has the advantage of having APIs to enable its operation and interaction with other applications. However its complex structure makes it unsuitable for the SECOAS project which has specific requirements on simplicity, efficiency and low overhead for data communications to be used in environmental monitoring applications.

There are various reasons why TinyOS is unsuitable for SECOAS requirements. Firstly, TinyOS is designed to support concurrency-intensive operations in order that applications and radio-slave devices (that have no buffering and strict hard real-time constraints) be multithreaded to give acceptable levels of service. This differs from

the SECOAS approach of single task execution: SECOAS applications have more relaxed latency requirements and can be pre-empted by higher-priority tasks. These high priority tasks can be changed by adopting new network topologies and applying different policies. Radio module intended to be deployed in SECOAS (and other external hardware devices) has its own buffering and therefore also more relaxed timing constraints. Secondly, TinyOS was designed to be as minimal as possible in order to execute on extremely limited devices; the stated goal of supporting cubic millimetre scale devices with minimal processing (the emphasis is on forwarding to an intelligent node) is vastly different from SECOAS goal which is to execute a small, simple OS on cheap wallet-sized devices. Emphasis of SECOAS is on the support of distributed, networked applications rather than hardware. The data processing algorithms, such as compression of collected data in case of high network utilisation, or lots of collisions leading to data loss, can be done on the sensor nodes to avoid passing raw, unnecessary data on the energy-intensive communication channel.

There are several embedded operating systems that were considered as candidates for the SECOAS project. All these operating systems fail to meet SECOAS requirements because of one or more of the following:

(i) They are specific to various embedded hardware functions suitable only for particular applications and do not easily allow application extensions or have only static task allocations.

(ii) They do not allow easy portability between different platforms and MCU hardware, due to use of application-specific languages or APIs.

(iii) They use multitasking techniques, whereas SECOAS requires a simple single tasking execution model.

(iv) Their memory footprint is excessive for our MCU-based system.

(v) They execute only on powerful Pentiums or ARM processors.

4. Architecture and development of kOS

The development of kOS started by the author in summer 2003 in order to explore the feasibility of design of a unique operating system for the SECOAS project. The first version of kOS, kOS V0.1 Alpha, was written in a mix of assembly language and C, using the MPLAB® Integrated Development Environment (IDE), which is the simulator and programmer software for PIC microcontroller range. First version had a footprint of only 588 bytes, yet it included various functional blocks, task modules and a simple task dispatcher working with timers. Use of SLEEP and WDT facility, EEPROM read/write, timers and task scheduling was experimented primarily in this version.. More information about this version is available in [VI].

Development of kOS has continued in the department of Electronics Engineering, UCL and the current version is V0.13 Alpha. This section is an abstract summary of the guide to the kOS Operating System. For more detailed architectural reference please refer to 82].

4.1 kOS structure

The kOS is divided into *objects* and *methods*. Objects and methods are classified as either system or application—system methods are used to apply control to multiple objects, whilst application methods are specific to individual applications. Task execution is performed by specifying objects, methods and execution times. *Figure 10* Figure 10 shows the hierarchy of these kOS functional components. As can be seen, the main routine has various objects available to execute, while these objects share various methods. The methods share library routines.

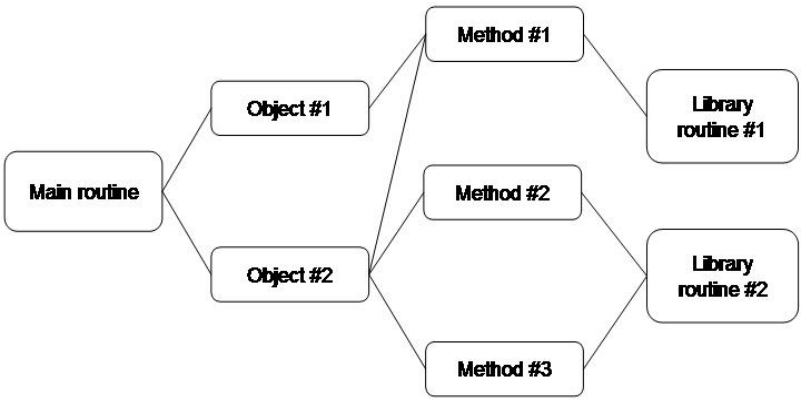


Figure 10: kOS Functional Components Hierarchy

4.1.1 kOS objects

Table 1 shows the full list of kOS objects.

Object #	**Object description**
0	System – kOS
1	System – Radio interface
2	System – Sensor interface
3	System – Messaging
4	System – User interface
5	System – Scheduler
6-98	*System – Reserved for future system objects*
99	System – All application objects (used to call all objects with a common method
100	*System – Reserved for future system object*
101	Application – Adaptive sampling
102	Application – Quorum sensing
103	Application – Auto-location
104	Application – Control-theoretic adaptive sensing
105	Application – Re-programming
106	Application – Data processing
107	Application – Base-station (data processing and re-transmission requests)

108-255	*Reserved for future application objects*

Table 1: kOS Objects

More information on individual kOS can be found in [1]. However the messaging object is of great importance in the context of this project and implementation of algorithms. This object implements the SECOAS Application Messaging (SAM) and SECOAS Data Messaging (SAD) protocols. SAM is used between applications, whilst SAD is used between applications and the sensor module. The Messaging object is scheduled periodically, and is scheduled after any radio or sensor interface bytes have been received. When new data is found, the Messaging object schedules a *new data arrival* method for the destination object.

Note, however, that applications may also check for new data themselves when they finish execution, by invoking the Messaging object's *check for new object data* method. If new data is available, the Messaging object will directly schedule the application's *new data read* method. This will be a rare occurrence, as the periodically scheduled method described in the paragraph above will catch any data arriving over the radio interface. However, this second approach allows objects to share data over SAM or SAD if that is required.

When an application is informed of new data, it is the application's responsibility to read the new data and incorporate the data into its own buffer, as the packet will be deleted at the end of each cycle.

SECOAS Application Messaging (SAM) Protocol

SAM is used by objects for intra- and inter-node communication. The majority of SAM messages contain application information, such as the sharing of parameters or data. However, any object may use SAM to pass data or control information over the radio interface to neighboring nodes or to applications residing on the same node. Applications may message other applications on the same node by simple writing a SAM packet to the receive buffer, with the correct destination object ID specified.

The specification for the SAM protocol is shown in Table 2 [note that header and footer bytes are not actually transmitted or received over radio], these are removed by the radio interface before transmission in order to keep the packets short, to satisfy the radio hardware requirement of a 16-byte maximum packet size. From the table, it can be seen the minimum (radio) overhead for a packet is 6 bytes.

Field label	Field description	Number of bytes used (byte pos)
Header	Hex 3C3C. In ASCII this reads "<<"	2 (+0,+1)
Sender node ID	Used for network or application addressing. See Table 3	1 (+2)
Destination node ID	Used for network or application addressing. See Table 3	1 (+3)
Sender object ID	The ID matches the object number	1 (+4)
Destination object ID	This will usually be the same as the sender ID object	1 (+5)
Length of data field	*in bytes*	1 (+6)
Data payload	This payload is defined by application writers	X
Checksum	The checksum data is taken from the sender node ID field to the end of the data payload field	1 (+7+X)
Footer	Hex 3E3E. In ASCII this reads ">>"	2 (+8+X,+9+X)

Table 2: SAM packet fields, Byte positions are shown in parentheses.

The *SAM packetiser* object examines the checksum, header and footer before informing any applications that new data has arrived. If any of these tests fail, the data is deleted and no further action is taken.

The mechanism for forwarding of packets has not been fully defined yet—it may be done at the networking layer, or done at the application layer. A 1-byte address will be used for this, as shown in Table 3.

Address	Summary
1-200	*Reserved for developer-assigned node IDs*
201	Route packet upstream 1 node-node hop distance

202	Route packet downstream 1 node-node hop distance
203	Route packet upstream the maximum number of node-node hops, used to get user data (possibly from a sensor module) to the base-station
204	Route packet downstream the maximum number of node-node hops (used by the base-station to flood packets to try and reach a specific node destination in the absence of network-level packet routine, or to flood multicasts, possible for application usage)

Table 3: Network Route Addresses

SECOAS Application Data (SAD) Protocol

As with SAM packets, the SAD header and footer bytes are not actually sent over radio. The specification for the SAD protocol is shown in Table 4.

Field label	**Field description**	**Number of bytes used (byte pos)**
Header	Hex 0x5B5B. In ASCII this reads "[["	2 (+0,+1)
Packet type	*invalid = 0*, P-Value = 1, Data = 2, Error = 3, Acknowledgement (ACK) = 4, Data request = 5, *6-15 reserved*	1 (+2)
Data field	This holds either a 5-byte P-Value payload, a 7-byte Data record empty for an Error or ACK packet, or contains a 4-byte Request	5 (+3,+7/+11/+5//)
Checksum	Checksum data taken on the (i) packet type/data type, (ii) sequence number and (iii) data fields	1
Footer	Hex 0x5D5D. In ASCII this reads "]]"	2

Table 4: SAD packet fields

The sensor package sends data only when an Ack packet is received from kOS for the previous transmission (unless it is the first transmission).

4.1.2 kOS methods

The current list of methods is shown in Table 5: *kOS Methods*. The behaviour of each is described in [1].

Method #	Method description
0	System – Default execution of object
1	System – Reset *(note resetting kOS object resets PIC chip and board)*
2	System – Error reporting
3	System – Status reporting
4	System – Initialise
5	System – Test the object
6	System – New data arrived for object
7	System – Enable continuous testing mode
8	System – Disable continuous testing mode
8-99	*Reserved for future shared system methods*
102	Radio interface – Clean buffers
113	Sensor interface – Clean buffers
120	Messaging – Check for new data in radio receive buffer
121	Messaging – Check for new data in radio transmit buffer
122	Messaging – Check for new data in sensor receive buffer
123	Messaging – Check for new data in sensor transmit buffer
140	kOS – end-cycle house-keeping
141-255	*Reserved for future application methods*

Table 5: kOS Methods

4.1.3 Node memory organisation

The microcontroller unit (MCU), the Microchip PIC 18F452, has three main types of memory; FLASH EPROM—a non-volatile memory for program installation; RAM—a volatile memory used for short-term data; and EEPROM—a non-volatile memory used for long-term storage of data. The 18F452 has 32k FLASH, 1.5k RAM and 200 bytes of EEPROM on-board. The kOS object code, object methods and constants are stored in the main *program memory*. In the *RAM* the variables needed by

the kOS, including the high- and low-priority task queues, sensor data storage, and radio receive and transmit data buffers are stored. Data required for long-term storage, including a history of boot-up failures are stored in the *PIC data store*. Sensor data from the sensor module and local data from the Microchip development board (PICDEM2) on-board temperature sensor, and possibly data copied as it passes through the radio upstream to the base-station are stored in the *local sensor data store*. An outline of the usage of the four memory areas is shown in *Figure 11*.

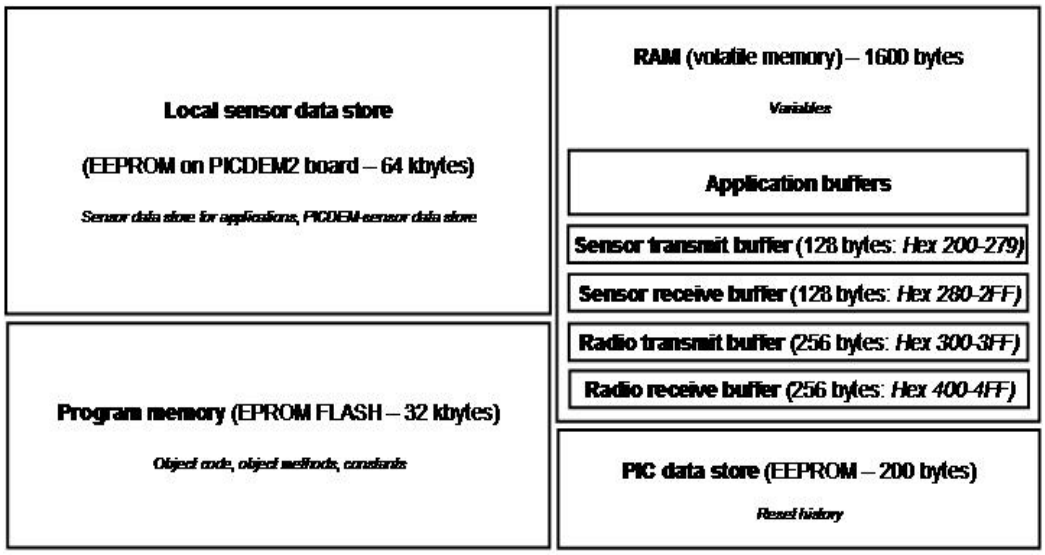


Figure 11: Outline of kOS Memory Organisation

The exact usage of each memory area (except program memory) together with item descriptions is shown in Table 6, Table 7 and Table 8.

Item	Addresses used (Hex)	Bytes used
Sensor transmit buffer	200-279	128
Sensor receive buffer	280-2FF	128
Radio transmit buffer	300-379	128
Radio receive buffer	380-3FF	128
Base-station buffer	400-4FF	256

Table 6: RAM (Volatile) Memory Organisation

Item	Addresses used (Hex)	Bytes used
Number of consecutive resets due to WDT time-out	0	1

Table 7: PIC EEPROM (Non-volatile) Memory Organisation

Item	Addresses used (Hex)	Bytes used
Local sensor data buffer	0000-03E7	1000
Temporally-compressed data buffer	03E8-07CF	1000
Spatially-compressed data buffer	07D0-0BB7	1000

Table 8: PICDEM2 EEPROM (Non-volatile) Memory Organisation

4.2 kOS operation

The kOS executes on both standard and base-station nodes—base-station features are simple enabled or disabled at compile-time for a standard sensor node. The kOS is designed to support a sensing interface and a radio interface, while allowing remote re-programmability and control, and data acquisition.

The basic operation of the kOS involves a short-wake/long-sleep duty cycle, and scheduled executions of high- and low-priority tasks. Most important advantages of kOS are in the fact that it is a simple, interrupt-driven operating system, and allows only 1 level of task preemption. This is adequate, as all low-priority kOS applications do not have strict latency requirements and save their own contexts in case of interruption. The kOS relies on a high degree of autonomy with its applications. Generally, applications will schedule themselves for their next execution and manage themselves. The radio interface is special as it receives a high-priority execution in order to satisfy the low-latency requirements of the radio module. Other applications execute at a lower priority and may be preempted by radio interface functions.

The basic operation of the kOS revolves around the sleep/activity/sleep cycle. The device is woken by a high or low priority interrupt. These interrupts cause the high or low priority scheduling routines to execute. The appropriate task is serviced, and then

the service routine returns the device to sleep. If a second task was due during the execution, the second task is executed immediately on going to sleep, even though its own execution time has passed. This basic operation is shown in a simple state diagram in *Figure 12*.

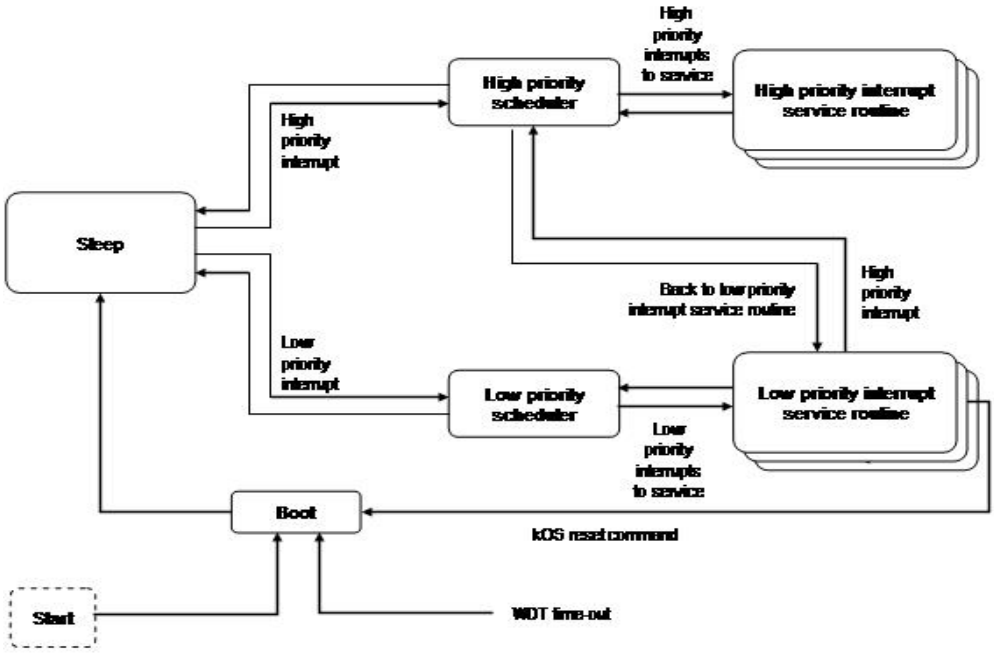


Figure 12: State Transitions in kOS Operation

Figure 13 shows how data flows between the radio and sensor interfaces, kOS buffers and the kOS applications. The received radio data is either sent straight to the sensor transmit buffer, in the case of a SAD packet, or sent to an application, depending on the destination object ID specified. Any data stored in the sensor receive buffer is sent straight to the radio transmit buffer. Applications may send data to either transmit buffer, and may also write packets to the radio receive buffer, to enable communication between applications. Data stored in either of the transmit buffers is sent to the respective module.

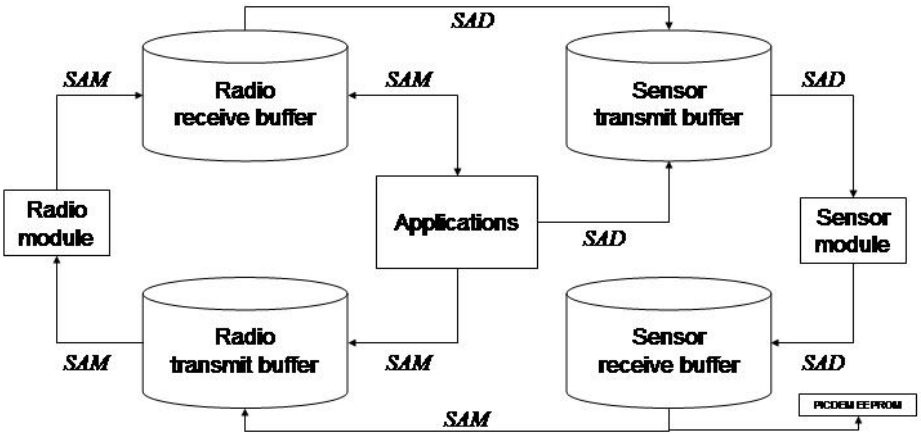


Figure 13: Data flow between modules, buffers and applications

Communication between the kOS board and the radio buffer, and between the actual sensors and the kOS board takes place using the RS232 interface protocol. The RS232 interface operates at 9600 baud with 8 data bits, 1 stop bits, 1 parity bit and Clear-To-Send (CTS) hardware flow control. The kOS board multiplexes its RS232 connection between the radio and sensor boards regularly, and waits for arriving bytes to fill the USART buffer. When this occurs, the USART causes a high-priority interrupt, and the byte is read into the appropriate buffer. The CTS pin is used to signal to the appropriate module that it is free to transmit. It is expected that both modules use their CTS pins in the much the same way to control the kOS transmissions. *Figure 14* shows the configuration of the RS232 interface between the kOS and the two modules. As shown, the PIC uses a line driver to convert to RS232 levels, and then uses a Control Line (CTL) to multiplex and de-multiplex the RS232 signals between the two modules.

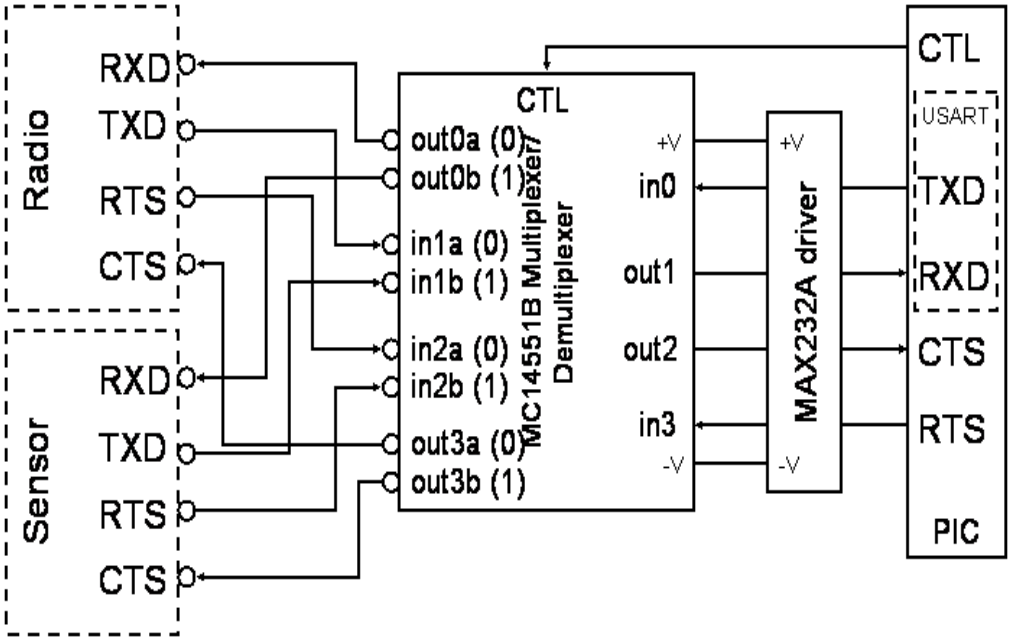


Figure 14: RS232 Hardware Interface

This chapter briefly described the characteristics of kOS V0.18 Alpha. More information is available in [1]. In the next chapter, efficiency issues within the kOS are discusses. Methods of achieving longer operation using batteries and efficient use of the SLEEP facility of the PIC MCU and higher utilisation of the communication links will be investigated.

5. Analysis of the kOS communication efficiency

Power consumption is an important factor in design and use of sensor networks. Sensor devices may have to be deployed in remote and dangerous areas and be capable of collecting data and communicating with other sensors and the base-station for weeks or even months, without the possibility of changing the batteries. Another issue which arises here is the fact that the sensor nodes are usually designed to be embedded within other modules, so it is not possible to connect them to power lines or attach large batteries to them. The nodes can be powered by use of a battery or by energy harvesting from the environment. Power is a scarce resource in a sensor network node and must be consumed wisely.

Energy harvesting techniques, using vibration, wind energy and solar batteries are a way of creating additional energy on board. However the gathered energy may be enough for the microcontrollers to extend their life, but it will certainly not be enough for radio communications between the devices. This section aims to address some of the important efficiency factors which are investigated using the kOS as an example of a light-weight operating system in a sensor network environment. The first field trial of the SECOAS project is designed around the Microchip PICDEM2 Plus board. *Figure 15* shows the Microchip PICDEM2 Plus board.

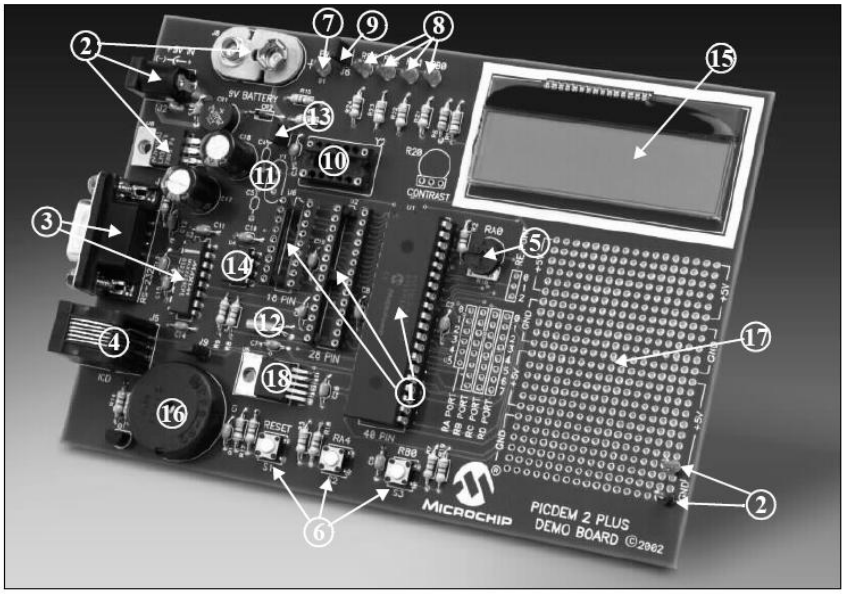


Figure 15: Microchip PICDEM2 Plus evaluation board (Figure courtesy of Microchip® Inc [24])

The Main features of the board are:

1: 18, 28 and 40-pin DIP sockets.

2: On-board +5V regulator for direct input from 9V, 100 mA AC/DC wall adapter or 9V battery, or hooks for a +5V, 100 mA regulated DC supply.

3: RS-232 socket and associated hardware for direct connection to an RS-232 interface.

4: In-Circuit Debugger (ICD) connector.

5: 5 KΩ pot for devices with analogue inputs.

6: Three push button switches for external stimulus and Reset.

7: Green power-on indicator LED.

8: Four red LEDs connected to PORTB.

9: Jumper J6 to disconnect LEDs from PORTB.

10: 4 MHz canned crystal oscillator.

11: Unpopulated holes provided for crystal connection.

12: 32768 Hz crystal for Timer1 clock operation.

13: Jumper J7 to disconnect on-board RC oscillator (approximately 2 MHz).

14: 32K x 8 Serial EEPROM.

15: LCD display.

16: Piezo buzzer.

17: Prototype area for user hardware.

18: Microchip TC74 thermal sensor [83].

In the context of this chapter the communication and power saving techniques are investigated using the PICDEM2 Plus board in order to demonstrate the energy savings which are achieved through the course of this project.

5.1 Communication between the kOS and the sensors

In the SECOAS project, the need for reliable and powerful communication in the ocean environment dictates that the radio modules are powered by a battery source located in a floating buoy. The kOS board resides in this floating buoy and it communicates to the sensor node at the bottom layer of the ocean, via an RS232 USART interface of cable length of around 30 metres. There are 5 data lines present in the RS232 interface:

- Receive data
- Transmit data
- Ready-to-send
- Clear-to-send
- Ground

This will indicate the use of a 6 core twisted pair, shielded, waterproof copper cable. The shielding is required as any long wire will act as an antenna and pick up interference, Even though these interferences may be minimal at the bottom of the ocean and with data rates as low as 9.6 kbps, the temperature difference between the wire core and the water temperature can result in development of copper oxide. Shielding the wire will delay this process and stop harmful interference.

An important problem in the communication between the sensors and the kOS board is the resistance of the wire. The RS232 is operating at voltages of -8V to 8V. The resistance of the copper wire can result in a voltage drop, as a result of which the power consumption will be increased. The normal Un-shielded twisted Pair (UTP) cables have an impedance of around 100-120 Ohms/km according to manufacturer's and experiments conducted by the author. The cost of cables increase greatly as they are shielded and the core impedance is decreased and can be as high as £10 per metre for industry standard versions (e.g. 0.5 mm^2 gate copper for low temperature, with resistance of around 30 Ω/km) seen in public component catalogues such as Farnell and RS.

5.2 RS232 cable experiment

An experiment is designed in order to analyse the possible effects of using the UART RS232 standard as the communication interface. In this experiment a 30 metre, 6-core UTP cable is used to provide the 5 wires required for the RS232 communications, and the joint are soldered to the appropriate pins of the male and female (DB9) connectors. For the purpose of the experiment, the serial port of a PC is connected to the kOS board by the RS232 cable. The PC acts as a sensor module, receiving and sending data to the kOS board using the serial port and the RealTerm terminal emulator freeware software. The data is received and sent successfully,

however the current consumption on the cable is around 0.2 mA, which is considerably high. *Figure 16* shows the equipment set-up for the RS232 experiment.

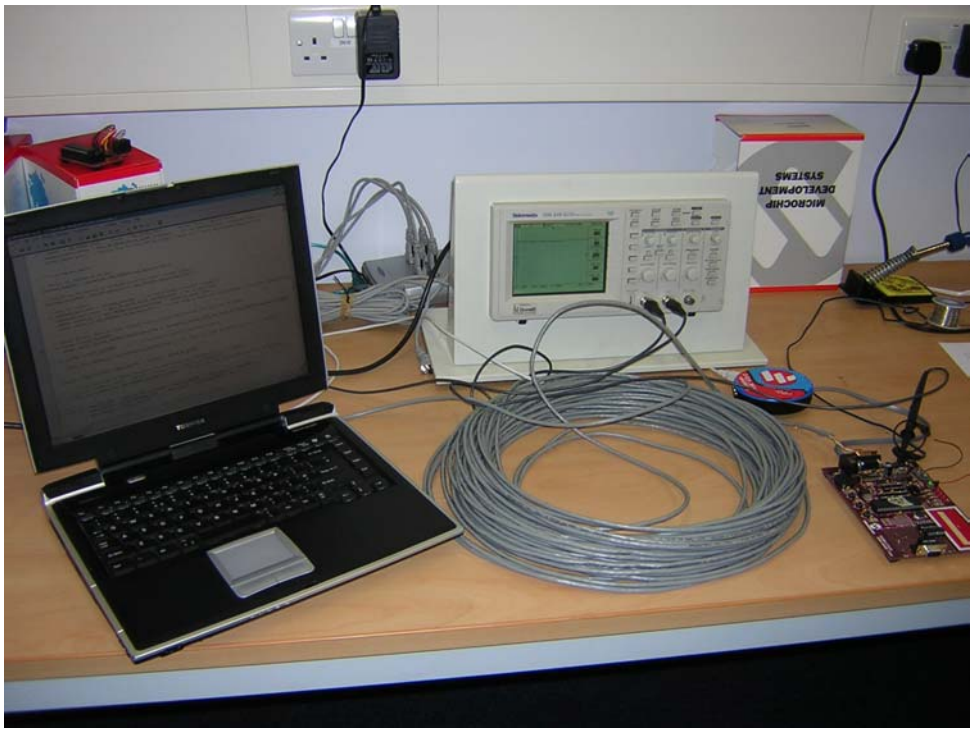


Figure 16: Equipment set-up for the RS232 experiment

The Transmit Data (TRD) and Receive Data (RXD) PIC18F452 has its UART module outputs on pins 25 and 26. RTS and CTS are also hardwired to pins 2 and 3. The MAX232A acts as the RS232 driver. The RS232 signal has a positive 8V voltage for the HIGH state and 0V for the LOW state. Even though in the RS232 standard the LOW state is required to be at -8V, use of 0V does not cause any problems for the low data rates used on the kOS board. However the main issue with the MAX232A is its constant current consumption of around 8 mA, which is a major strain on the current consumption of the board as a whole. Since neither the radio nor the sensors are not transmitting data, or listening out for incoming data apart form their dedicated timeslots, the operation of the RS232 driver is not required most of the time and a method has to be devised to shut down the device while at idle state. *Figure 17* brings an overview graph of the practical current measurements, consumed by the PIC and the MAX232A for transmitting a data message. The reading on all of the graphs have

been taken at discrete times and not on a continuous basis, as the equipment to enable such readings was not available to the author, hence the data points are not connected.

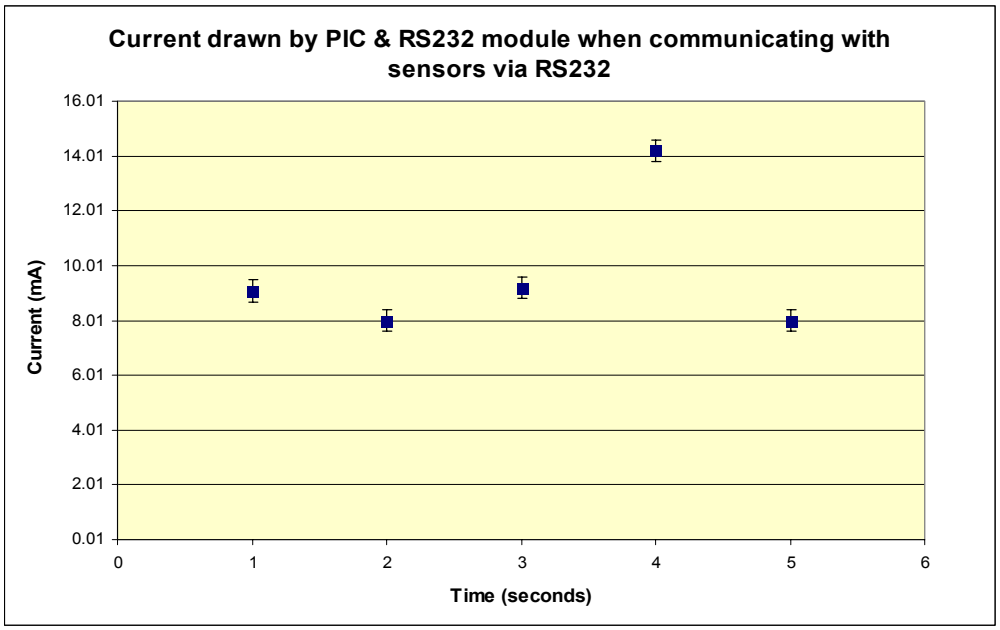


Figure 17: Current consumption of the PIC + RS232 driver

The most convenient way of overcoming the high-current problem is to use a more efficient driver chip for the communication between the boards. An alternative which has been devised is to use a more sophisticated RS232 driver circuitry which enables a shut-down mode when there is no data being sent or received. The author's suggested chip is the Maxim MAX242 [23]. The MAX242 driver has three operating modes: active, low power shut-down mode and it is one of the few which has the option of an active receiver whilst in shut-down mode. The current consumption of MAX242 is 0.1 μA at 25°C in the shut-down mode, which is virtually 90% of the time that the radio or sensor modules are on. When active, the MAX242 only draws 4 mA at +5V, which is still 50% less then the MAX232A IC which is used on the PICDEM2 Plus board. This will also lead to a lower current consumption by the resistors at the RS232 interface, which are currently 4.7 kΩ resistors, so each of them will consume 1.7 mA at 8V RS232 voltage. *Figure 18* displays the theoretical achievement of this method.

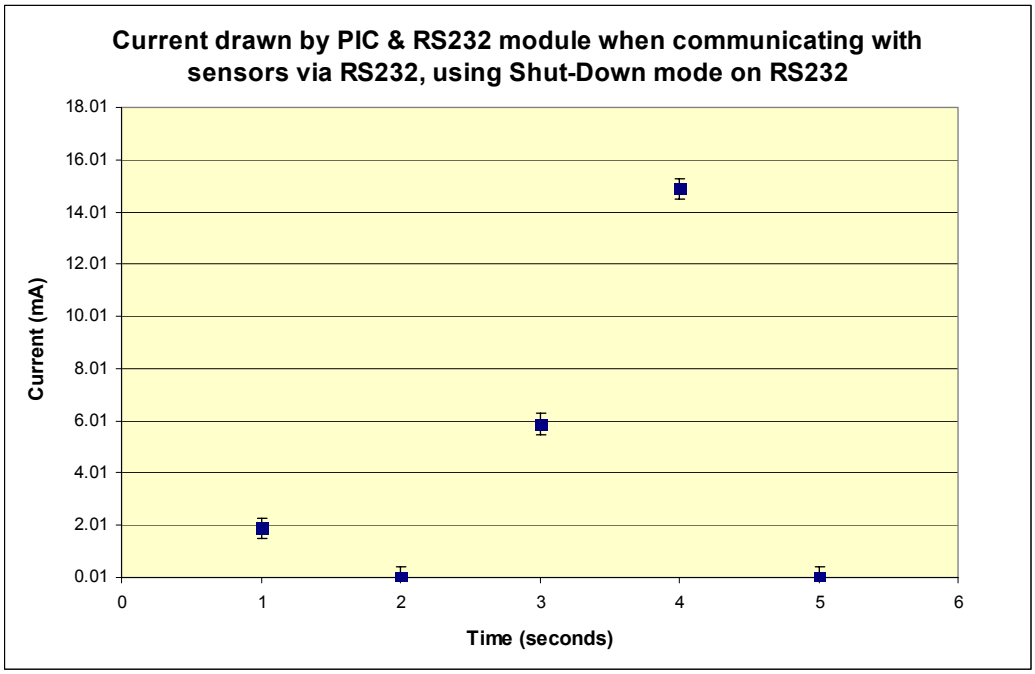


Figure 18: Current consumption of the PIC + RS232 using shut-down mode

The use of shut-down modes on the driver further complicates the PCB design and system architecture but due to the fact that there is already a CTS and RTS hardware flow control system implemented it is more feasible to use these as the controller for the RS232 driver. *Figure 19* displays a possible method for controlling the power on the circuit. Note that this is only possible in the transmit cycles, as in receive cycles the MCU has no control over the timing of the data arrivals and the path should be already set-up by the RS232 driver chip.

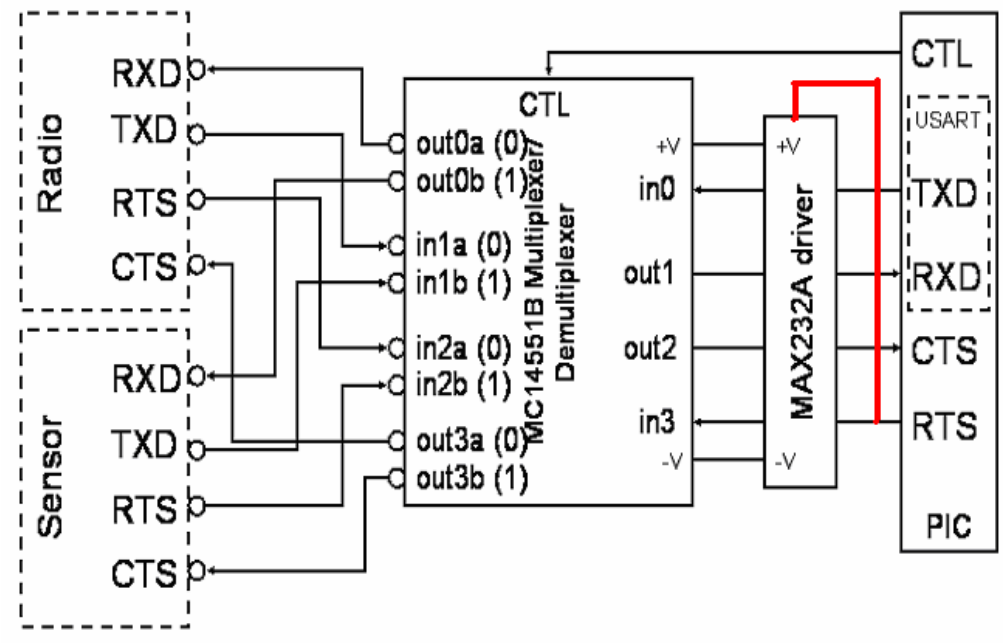


Figure 19: Modified RS232 controller circuit

5.3 Practical energy savings on the kOS board

For the 1st field trials of the SECOAS project, Microchip PICDEM2 Plus boards will be used as the kOS board. These boards have certain features that can be eliminated in order to minimise the on-board current consumption, which normally exceeds 40 mA. The most important items that can be immediately made redundant are the debug LEDs and the LCD display. LEDs and the LCD consume around 20 mA when powered. The on-chip buzzer and temperature sensor can also be removed, even though the temperature sensor, Microchip's TC74, can be used as an emergency temperature reader in case of failure of the main temperature sensor at the bottom of the ocean.

Another improvement which can lead to lower power consumption is the use of the on-chip EEPROM as opposed to the external EEPROM module provided on the kOS board. The EEPROM on the kOS board is the Microchip 24LC256. This EEPROM module operates at 5.5V, with a maximum draw current of 3 mA, a maximum read current of 400 μA, and a standby current of 0.1 μA which is equivalent to the PIC18F452 sleep mode current consumption. The 24LC256 offers 32 Kbytes of memory with data retention of 40 years, while the on-chip EEPROM of the PIC18F452 offers only 256 bytes. As most of the applications are continuously

sending and receiving data with periods of orders of seconds, it is more feasible to use the on-chip EEPROM for essential information and perhaps back-ups of data means that have not been sent due to storm or radio channel interference.

Since the kOS board consumes only around 10 mA while the MCU is working, it is feasible for it to get power from the radio module's batteries which are 6V, 12 Ah NiCad batteries. However another issue that arises here is that the PICDEM2 Plus on-board regulator, the LM7805, operates as voltages above 7V with a quiescent current charge of around 5 mA to 1A at 5V output. This range is far too large for the requirements of the kOS board. An alternative regulator which is much more suitable to the applications is the LM2936. It outputs up to 50 mA and has a low quiescent current of 0.2 mA. The output is 5V as long as the input is at least 5.5V and it also has a power shutdown mode which is extremely useful if all the parts, apart from the PIC MCU, are moved to a separate board, so that they can be shut down when not required by the PIC. However turning the regulator and the RS232 driver on and off can only be beneficial if the frequency of this is less than a second, as the discharge capacitors take a fraction of a second to get fully charged and stabilise.

5.4 Alternative communication techniques

The deployment location of the sensor nodes makes them susceptible to natural phenomena such as ocean storms, damage by the local fish species and interference by various wireless devices present on the telecommunication devices used by the fishermen and ships. The fact that sensors board is located at the bottom of the ocean makes it necessary for them to transmit the data to the kOS board on the sea level. However there are alternative methods that can be used for the transmission of data. A normal RS232 connection over cable is an extremely expensive way of sending the sensor data across as the shielded, waterproof and low resistance 5-core cables are costly.

An alternative scheme is to use a simple optical fibre and LED system. Cheap plastic optical fibres are not immune to Electro-Magnetic Interference (EMI) and Radio Frequency Interference (RFI), water will not harm them and they can be used in places where voltage isolation and insulation is an issue. Typical operating temperatures are 0-70°C. The 600 nm LED transmitters and receivers can cover distances of up to 120 metres with data rates of up to 40 kBaud using plastic optical

fibre, operating in the low current mode. The average output collector current of the receiver is 5 mA and the receivers and transmitters (Transmitter: HFBR1524, Receiver: HFBR1524) are readily available from manufacturers, with a complete kit costing less than £20 [24].

Experimental work has also been carried out in the EE department to investigate the use of ultra-sonic links for the data transmission under the water.

In receive mode the radio consumes around 30 mA of current and in receive mode it draws around 10 mA current. In order to minimise the radio communication, the use of ideas behind some audio and video compression technique such as MPEG is favourable. Each sensor data reading has a maximum of 10 bytes that it can send to the kOS board as a SAD packet, which is then wrapped in a SAM packet and sent over the radio. As there is a field to indicate the number of data bytes, there is room for decreasing the actual data bytes and transmitting smaller packets. This is done by comparing every sensor data reading in a SAD packet, and comparing it to the previous value, and only sending the differential value inside the SAM packet over the radio. In this way, many measurements, such as temperature, which only differ by a small change on each reading, can be sent in a minimal format. As a demonstration, consider the field below as an 8 byte temperature reading at t_0:

0x BEEF FACE

And this is the next reading at time t_1:

0x BEEF FADE

Now clearly the difference is only in the last two bytes, which can be gained by deducting the second value from the first, and is equivalent to 0x10. So in this case only 2 bytes need to be retransmitted, with one bit, dedicated in the SAM packet, to indicate whether the value is a positive or negative difference. With a sensing frequency of around 10 readings an hour, this can result to more than 80% decrease in communication at stable climate times.

6. Power saving methods for kOS

The central issue of power efficiency is one of the most important assets of any sensor network deployed nowadays. The sensor nodes will need to remain in the environment for weeks or months, and their efficient battery life management is extremely important for an extended operation period. As previously discussed, some wireless sensor nodes use the ambient energy to charge up their batteries, however for a heavy-duty application such as ocean monitoring, adding photocells and other equipment may further increase the size and complexity of the sensor nodes and defeat the objective of "cheap and simple" modules.

The kOS nodes need to be competitive with other types of sensor nodes developed round the world. The most famous of these is the Tiny "motes" developed at the University of California, Berkeley. *Figure 20* displays the measured current consumption for transmitting a single radio message at maximum transmit power on the motes running TinyOS.

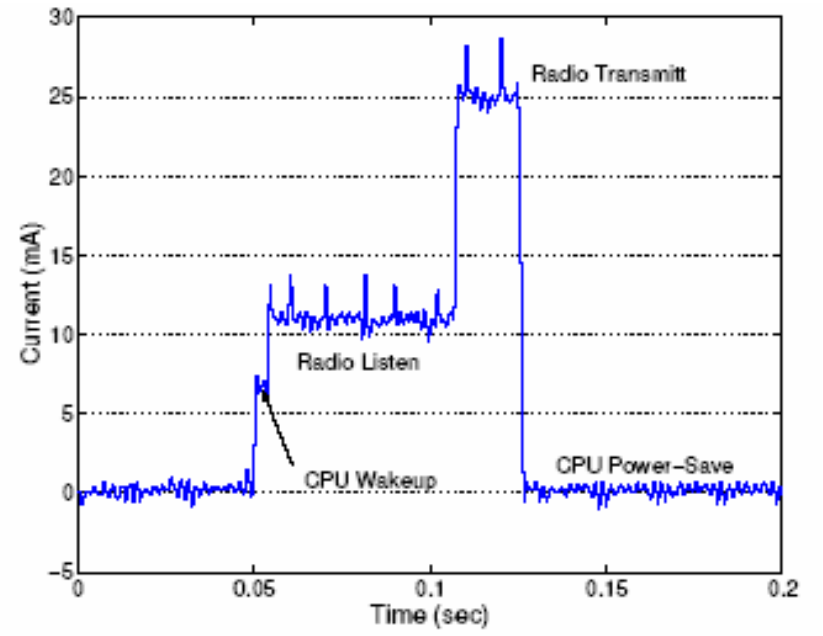


Figure 20: Measured current consumption for transmitting a radio message at maximum transmission power on the motes. (Figure courtesy of Harvard University [28])

In this chapter various methods and schemes are proposed and tested in order to minimise the power consumption of the kOS boards. The main emphasis of this chapter is on use of various operational modes of the PIC18F452 MCU to minimise its activity period. Alternative task scheduling schemes are also discussed in this chapter.

6.1 Effective use of the SLEEP mode

The PIC18F452 MCU has two main modes of operation, Active mode and SLEEP mode. In the active mode, the MCU on the kOS board is running of a crystal oscillator speeding 4 MHz. The MCU can run off a few different clock sources, the most important ones being the crystal oscillator and the Timer1 oscillator. A crystal oscillator circuit is built-in between pins T1OSI (input) and T1OSO (amplifier output). It is enabled by setting control bit T1OSCEN (T1CON<3>) [18]. The oscillator is a low power oscillator rated up to 200 kHz. It will continue to run during SLEEP. It is primarily intended for a 32 kHz crystal. On the kOS board, the value of this crystal is 32768 Hz. This figure has particular importance in scheduling tasks which will be explained in this section. Two 33pF capacitors are required to provide charge-up for the Timer1 oscillator. The start-up time can be a software hazard for applications and state machines so the software must provide a time delay to ensure proper start-up of the Timer1 oscillator. When running on the 32 kHz Timer1 Oscillator, the typical current consumption of the PIC is 0.2 mA, at V_{DD} = 4.2 V and temperature range of 40°C to 85°C.

The standard operation of the MCU requires a crystal oscillator to be connected between the OSC1 and OSC2 pins. This crystal is usually a quartz resonator, supported by two charge-up capacitors of 33 pF and can speed as high as 40 MHz for up to 10 MIPS operation. When the PIC is running of the crystal operator, the typical current drawn is around 1.2 mA at V_{DD} = 4.2 V and operating temperature 25°C. Clearly there is a huge advantage in using the high speed crystal for quick operation when speed is needed, however this comes at a cost of 6 times higher current drawn by the PIC, and the discharge capacitor current for crystal operation, which is around 11 mA for a canned oscillator clock module (used on the PICDEM2 Plus board) producing Transistor-Transistor Logic (TTL) output, and 9 mA for a quartz cut crystal (replaced on the board for kOS operation).

SLEEP is a feature which is available on most modern MCUs. It means that the operation of the programs are halted, and the MCU enters a low-power, idle mode, drawing only enough current to keep it alive. In PIC MCUs, SLEEP (Power-down) mode is the low current consumption state and is entered by executing a SLEEP instruction. The main device oscillator is turned off, so no system clocks are occurring in the device, with the exception of the optional Timer1 oscillator and the Watch-Dog timer (WDT), if enabled.

For lowest current consumption in this mode, all I/O pins must be placed at either V_{DD} or V_{SS}, external circuitry must be drawing no current from the I/O pins, ADC module must be powered-down and external clocks are disabled. User must also pull all I/O pins that are hi-impedance inputs, high or low externally, to avoid switching currents caused by floating inputs. The T0CKI input should also be at V_{DD} or V_{SS} for lowest current consumption. The contribution from on-chip pull-ups on PORTB should be considered. The MCLR pin must be at a logic high level.

The interrupts need to be enabled in order to be able to wake the device up from SLEEP:

```
// configure interrupts
    INTCONbits.GIE   = 1;    // Global Interrupt Enable
    INTCONbits.PEIE  = 1;    // PEripheral Interrupt Enable
    RCONbits.IPEN    = 1;    // Interrupt Priority level Enable
```

When the device executes a SLEEP instruction, the on-chip clocks and oscillator are turned off and the device is held at the beginning of an instruction cycle. With the oscillator off, the OSC1 and OSC2 signals will stop oscillating. Since all the transistor switching currents have been removed, SLEEP mode achieves the lowest current consumption of the device (only leakage currents). Enabling any on-chip feature that will operate during SLEEP will increase the current consumed during SLEEP. Some examples of these features include the WDT (7 μA) and Timer1 oscillator (6.5 μA).

The device can wake-up from SLEEP through a list of events, from which the most relevant interrupts are mentioned here:

1. External RESET input on MCLR pin
2. Watchdog Timer Wake-up (if WDT was enabled)

3. Interrupt from INT pin, RB port change or a Peripheral Interrupt
4. TMR1 interrupt. Timer1 must be operating as an asynchronous counter
5. TMR3 interrupt. Timer3 must be operating as an asynchronous counter
6. USART RX or TX (Synchronous Slave mode)
7. Low Voltage Detect (LVD) interrupts

Other peripherals cannot generate interrupts, since during SLEEP no on-chip clocks are present. The most important reason to use the SLEEP mode is to save energy. Current drawn by PIC18F452 during SLEEP mode is 0.1 μA which is 12000 times less than the normal operation mode. The current version of kOS does not support SLEEP mode and this is an extremely attractive way of introducing power efficiency into kOS. The current Time slot transition of kOS operates by using the Timer1 as a counter. The Timer1 module timer/counter has the following features:

- 16-bit timer/counter (two 8-bit registers; TMR1H and TMR1L)
- Readable and writable (both registers)
- Internal or external clock select
- Interrupt-on-overflow from 0xFFFF to 0x0000

Timer1 has a pre-scale value of 1:8 which can be set by setting bits 5-4 in the Timer1 Control register (T1CON <T1CKPS1:T1CKPS0>). The pre-scale enables the Timer value to be extended by a multiple of 8 for each count. Using Timer1 as time-slot transition will lead to time-slot durations of around 0.52 seconds while in active mode:

$$(4/4*10^{6})*8*(2^{16}-1)=0.524\ \text{Seconds}$$

The period of each clock tick has to be multiplied by 4, as in the counter mode, Timer1 counts the instructions, and each instruction is 4 clock cycles, equating to 1 μs using a 4 MHz clocking rate. If SLEEP is enabled in this configuration, the average simulated SLEEP/Active duty cycle ratio can be found. *Figure 21* displays the SLEEP/AWAKE ratio using one task in time-slot number 4.

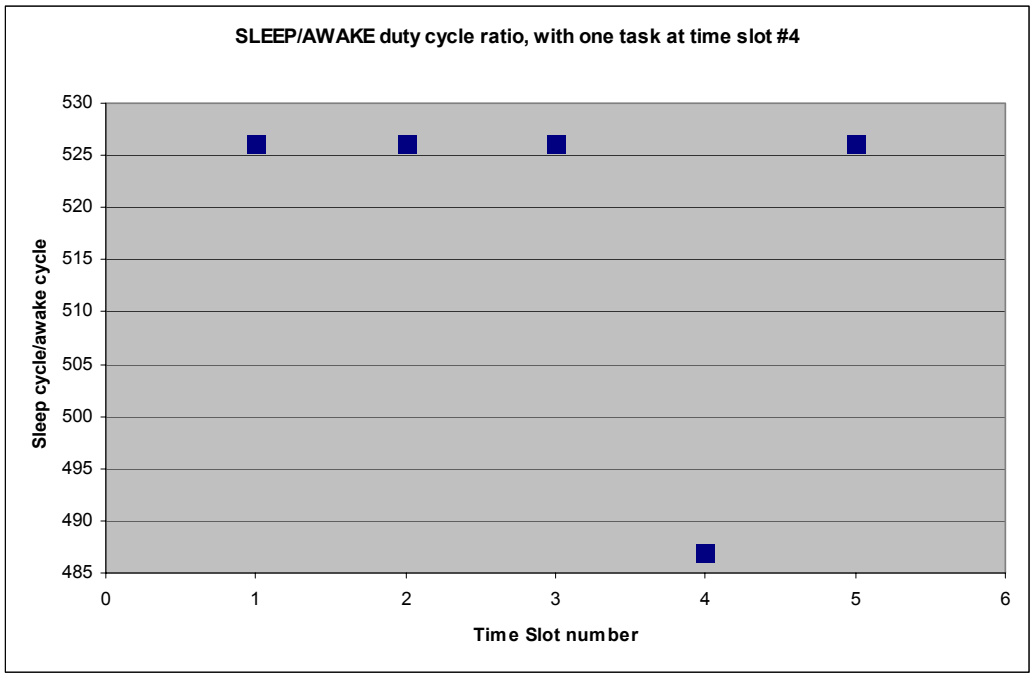


Figure 21: SLEEP/Active duty cycle ratio

It can be seen that the average is more than 500, indicating the fact that the processor is in idle mode 99.8% of the time when only one task is scheduled per cycle (10 time-slots). This indicates that perhaps the processor can save nearly 1.2 mA of current 99.8 % of the time, which at V_{DD}= 4.2 V it results in 4.4 mW less power at 99.8% of the time.

When tasks are scheduled to be run in different time-slots, there is a variation in length of activity period in different time slots. In some time slots, the only operations are kOS house keeping tasks. In others, there are scheduled, period applications being executed. *Figure 22* displays an example of the time slot activity lengths, with time slot one being the longest, as a result of tasks being scheduled and all the initial set-up and application executions. Some time slots have certain tasks running, like messaging in time slot 4 and gossiping in time slot 11. This clearly shows that not all time slots are of equal value and importance.

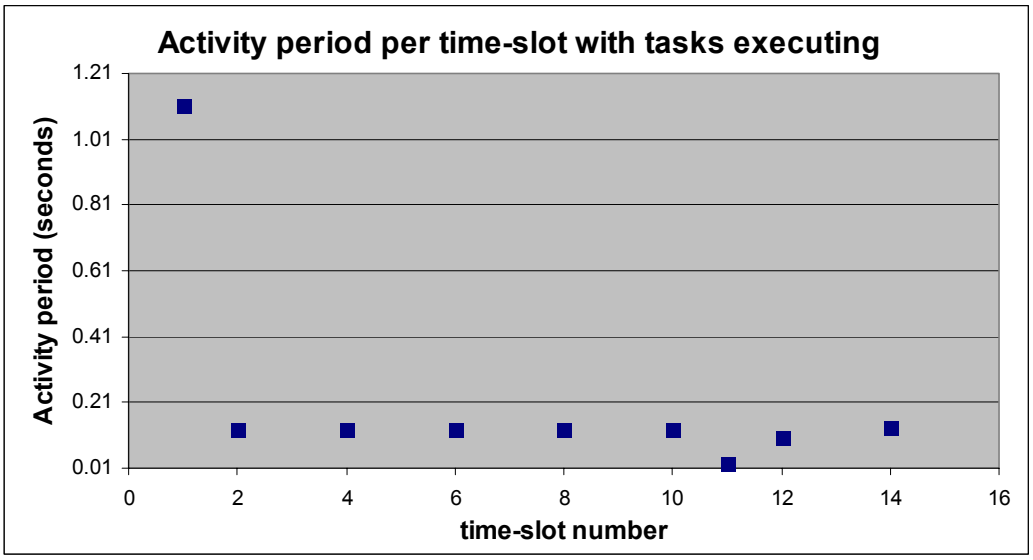


Figure 22: Activity period in time slots

With the current configuration of the kOS board, Timer1 runs off the main crystal oscillator, with configuration such as its frequency is equal to $f_{osc}/4$ which results in 1 MHz operation of Timer1. However while in sleep, Timer1 can not run off the main oscillator, and the WDT generates a wake-up interrupt. For prompt operation of Timer1, its oscillator must be enabled and it must be operating as an asynchronous counter. When Timer1 oscillator is enabled, the device can be taken to sleep safely and it can wake up successfully upon the Timer1 counter resetting. The timer 1 can then be changed to the "timer" operation mode, in order to keep track of the time-slots. However this may lead to events being delayed due to SLEEP and active modes using two different clocks. The best way of overcoming this issue is to set the Timer1 to an asynchronous counter indefinitely. In this way the tasks have a global view towards the timing of their execution and there will be no confusion as to when a task must be executed next. When the pre-scale is set to zero, and Timer1 is operating of the attached 32768 Hz oscillator, the time-slot length can be calculated based on Timer1 reset from 0xFFFF to 0x0000 as:

$$(1/32678)*2^{16} = 2 \text{ seconds!!!!!!}$$

This is an extremely convenient choice for tracking the time-slots and even the global time of the network as a whole and it is tested by the author on the current kOS

board. It is recommended that the 2nd field trial of the SECOAS project must consider replacing the always-on Timer1 with the above method to achieve nearly 99% sleep time when only a few tasks are running. The configuration bits of Timer1 are set as shown below:

```
// TIMER1 configuration: timer1 is used for hi-priority and lo-priority interrupts


T1CONbits.RD16 = 1;             // 16 bit mode
T1CONbits.TMR1ON = 1;           // Turn timer on

T1CONbits.TMR1CS = 1;           // Use external clock source.
T1CONbits.T1SYNC = 1;           // Do not synchronise external clock input

T1CONbits.T1CKPS1 = 0;
T1CONbits.T1CKPS0 = 0;          // 1:1 pre-scale

T1CONbits.T1OSCEN = 1;          // Enable timer1's internal oscillator

IPR1bits.TMR1IP = 1;            // IPR=Interrupt Priority Register -> set timer1 to
                                                        High-priority interrupt
PIE1bits.TMR1IE = 1;            // Enable an interrupt when timer1 overflows
```

Power-up delays are controlled by two timers, so that no external RESET circuitry is required for most applications. The delays ensure that the device is kept in RESET, until the device power supply and clock are stable. The first timer is the Power-up Timer (PWRT), which optionally provides a fixed delay of 72 ms (nominal) on power-up only. The second timer is the Oscillator Start-up Timer (OST), intended to keep the chip in RESET until the crystal oscillator is stable. When the device wakes up from the SLEEP mode, the Oscillator Start-up Timer (OST) provides 1024 oscillator cycle delay [18].

In the current kOS configuration, this delay would be equal to:

$$(1024 / 4*10^{6}) = 2.56 * 10^{-4} \text{ seconds}$$

For a Timer1 oscillator of 32768 Hz, the period of each time-slot is equal to:

$$(1/32768)= 3.051 * 10^{-5} \text{ seconds}$$

The ratio of the delay value to the time-slot length is therefore equal to:

$$((2.56 * 10^{-4} / (3.051 * 10^{-5})) = 8.39$$

The delay value is therefore just over 8 time-slots. This means that for a SLEEP instruction to be worth executing, the value of the remaining cycles on Timer1 until execution of the task should really be greater than 9, otherwise more time is wasted in going to SLEEP and wake-up than it would have been spent if the device was simply left running:

```
    // MAIN LOOP
while (1) {

    {_asm CLRWDT _endasm}        // clear the Watch-Dog Timer (WDT). This is
                                 recommended before executing a Sleep() instruction.


    timer1_value = ReadTimer1();

    if (timer1_value < 65526){   // 65535 -9 = 65526  if TMR1 is nearly there no point
                                 sleeping!
    Sleep();                     // TIMER1 usually wakes the PIC up upon reset
}
    else
{

    kOS_status_record &= ~(isWDTWU() << WDTWU); // Check if we woke up due to the WDT
                                                and Set or reset the bit flag accordingly
}
```

Intelligent use of the SLEEP facility enables huge power cuts (factor of 12000) at most of the operation times. As the number of task increases the SLEEP-Active ratio decreases, however by using the Timer1 oscillator, and considering the fact that in the SECOAS 1st and 2nd filed trial there will be less than 10 tasks running, with simple, short cycle operations, the use of the SLEEP is extremely important. *Figure 23* displays a graphical display of the number of active instructions, i.e. those in which useful processing is done, per each operation time-slot. Operational time-slot is one in

which there is at least one task scheduled. For the purpose of these measurements, tasks were scheduled on each single even time-slot and the number of cycles that CPU was not in SLEEP mode was counted. The most important factor to consider in these result is that with such heavy task scheduling, executing processor intensive tasks such as data gossiping and quorum sensing, the number of active instruction cycles, where the CPU is not in SLEEP mode, is yet well below 30% of the number of instruction cycles in a time-slot, which is 131072 cycles in the current version of kOS (v18.2). An important factor to consider here is that the scheduler object takes around 26616 cycles on each operational timeslot where it has to update the scheduled tasks. This is a large overhead compared with the simplicity objective of kOS and virtually most of the MCU instruction cycles are taken by this. Efficiency of the scheduler has to be taken into consideration and some suggestions are given in the next chapter. As the tasks never need to be scheduled so intensively in an ocean monitoring application, there is no doubt in benefits of SLEEP mode.

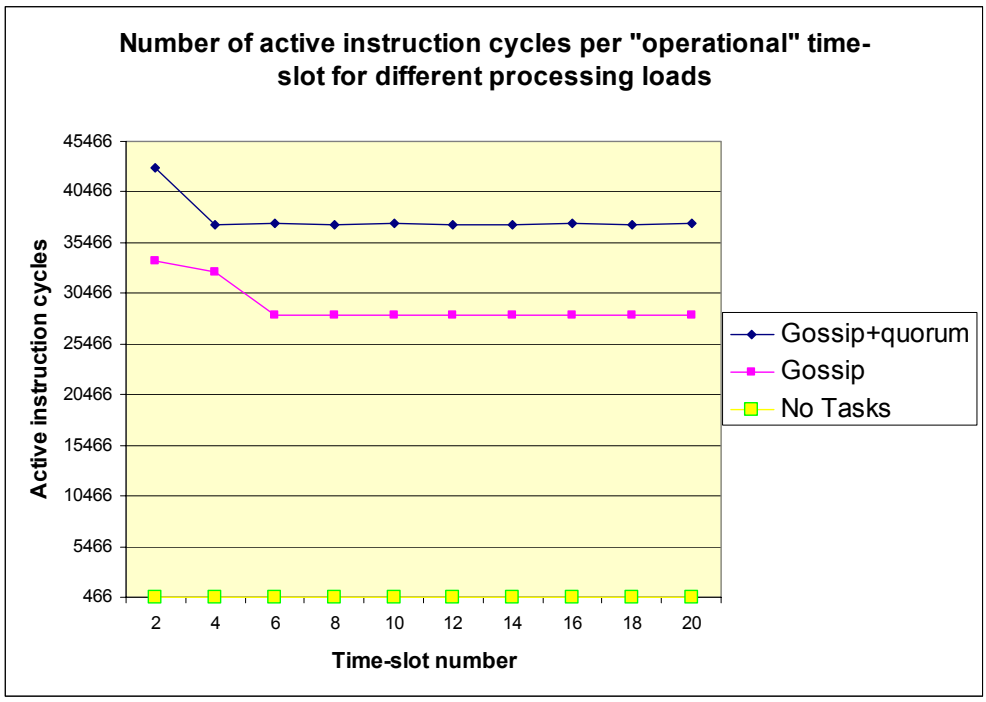


Figure 23: Active instruction cycles in operational time-slots

In terms of the overall current consumption of the system, currently the kOS board consumes around 30 mA, using some of the modification mentioned and implemented by author previously, such as replacement of the crystal clock module by a crystal

oscillator. The overall system, when adding the radio module consumes around 35 mA when the radio is not transmitting or receiving any data. Even though the current consumption of the PIC is only a small fraction of the whole system, introducing sleep for 99% of the time increases the average life of the kOS board by 5.31% and the life of the overall system by 4.4%, over the estimated 14 day trial period, which is in important achievement. The implementation of the SLEEP facility is recommended to be implemented and tested in the 1st field trial of the SECOAS project.

6.2 Smart-clocking the MCU

One of the largest currents on the kOS board is due to the use of a 4 MHz oscillator crystal, drawing around 8.9 mA of current. There is also a current decrease factor of 6 when the PIC is running off the Timer1 oscillator, from 1.2 mA to 0.2 mA. In order to minimize the current drawn from the PIC during the normal operation and house-keeping, where operational speed is of minimum importance, it is worth considering this great power difference between the power consumption when compromising speed.

The PIC18F452 device includes a feature that allows the system clock source to be switched from the main oscillator to an alternate low frequency clock source. This alternate clock source is the Timer1 oscillator. If a low frequency crystal (32 kHz, for example) has been attached to the Timer1 oscillator pins and the Timer1 oscillator has been enabled, the device can switch to a Low Power Execution mode. The clock switching feature is enabled by programming the Oscillator Switching Enable (OSCSEN) bit in Configuration Register 1H to a ’0’. The system clock source switching is performed under software control. The system clock switch bit, SCS (OSCCON<0>) controls the clock switching. When the SCS bit is ’0’, the system clock source comes from the main oscillator that is selected by the FOSC configuration bits in Configuration Register 1H. When the SCS bit is set, the system clock source will come from the Timer1 oscillator. The SCS bit is cleared on all forms of RESET.

__CONFIG _CONFIG1H, _OSCS_ON_1H & _XT_OSC_1H *; oscillator switch enable on & setup for an XT oscillator across OSC1 and OSC2*

The Timer1 oscillator must be enabled and operating to switch the system clock source. The Timer1 oscillator is enabled by setting the T1OSCEN bit in the Timer1 control register (T1CON). If the Timer1 oscillator is not enabled, then any write to the SCS bit will be ignored (SCS bit forced cleared) and the main oscillator will continue to be the system clock source [18].

```
T1CONbits.T1OSCEN = 1;          // Enable timer1's internal oscillator
```

Figure 24 displays a visual estimate for the activity of the MCU. Initially, there is a lot of house-keeping and settings and task scheduling activities to be done, so the big black period indicates high processing tasks, then the device can be taken to sleep, indicated by blue line. Upon Timer1 reset, the device wakes up, clears the interrupt, checks the scheduler to find the next task which is to be executed and executes the task if it happens to be scheduled to run in the current time-slot. The scheduled is then updated and device is taken to sleep again. However as the time-slots have short durations, in many of them there may not be any tasks running.

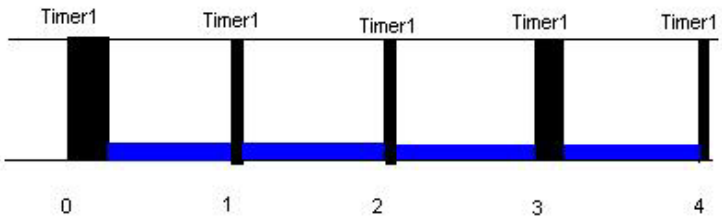


Figure 24: Time-slot transitions and task executions

Figure 25 displays the current drawn by the PIC MCU when executing the messaging tasks in time-slot 3. It can be seen that initially the current is 1.2 mA, indicating that the PIC is running of the main crystal oscillator. In the time-slots where no tasks are scheduled to be run, the MCU keeps running of the Timer1 oscillator, hence drawing 0.2 mA. When there is a task scheduled in the time-slots, like time-slot 4 in this example, the oscillator is switched to the main crystal oscillator and the current draw increases to full rate. The use of a histogram in this chart indicates that the current stays the same until the next transition in the oscillator configuration or the next SLEEP cycle.

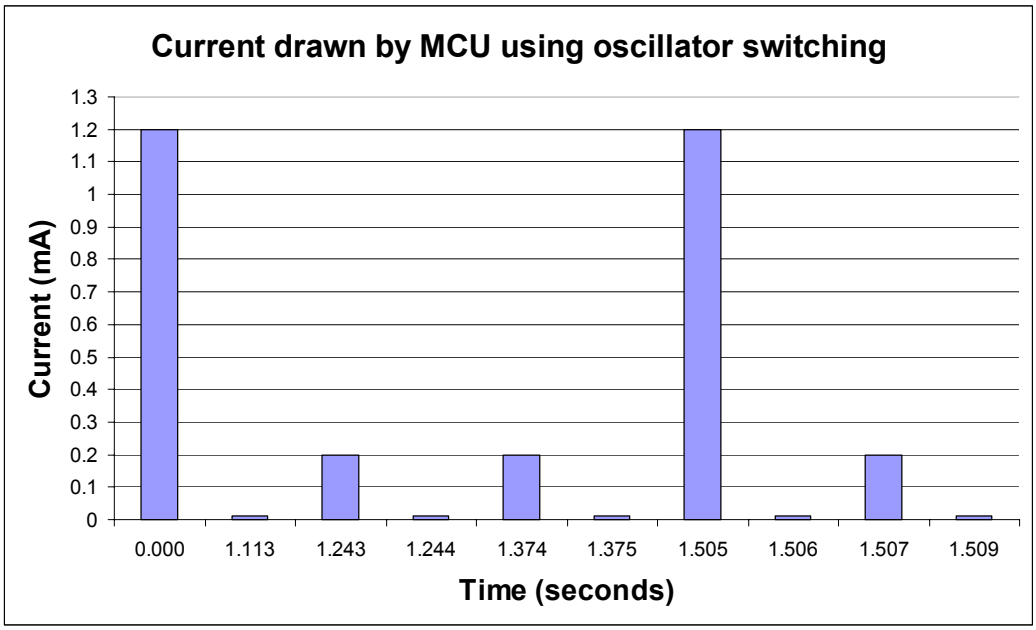


Figure 25: Current drawn by MCU using oscillator switching

It is beneficial to operate the device using the Timer1 oscillator, and when it is required to have high-speed operations, such as data gossiping and compression algorithms, the system clock can be switched to the high speed crystal:

```
if (schedule_lo.slotflags != 0) {      // are there any tasks in the schedule?
                                       // We have tasks in the schedule.
        OSCCONbits.SCS = 0;            // Switch to the crystal oscillator for fast execution
```

When the tasks are executed, the system can be restored to normal operating condition:

```
        OSCCONbits.SCS = 1;            // Switch to timer1 oscillator for low-power operation
```

There are four main applications currently running on kOS: Gossiping, data fusion, auto-location and messaging. If each task is scheduled once in every cycle, remaining 6 time-slots in the cycle can simply run using the timer1 oscillator. This is a saving equal to 83.3% at the 6 time-slots that are not executing any tasks and overall (100- (0.833 * 6) = 50 % reduction in the cycle operation. On the current SECOAS node configuration, this is equivalent of just over 2.1% increase in the overall system life over the 14 day 1st trial period.

In summary, oscillator switching has advantage of power saving, and disadvantage of the start-up delay of the oscillator, which is 1024 cycles. Some of the house keeping applications will never need as many as 500 cycles, so for such applications, use of oscillator-switching brings an unnecessary overhead. The best configuration for kOS is:

- The default oscillator is the Timer1 slow oscillator.
- For high speed applications and heavy algorithms, use Crystal clock.
- For slow radio and RS232 send and receive, stay on Timer1 oscillator.

6.3 Comparison of different power schemes

In terms of overall power-budget planning for the SECOAS project, *Table 9* displays some configurations that can be used for the complete kOS board:

	PIC MCU & regulator & clock current (mA)	***SLEEP & oscillator switching***	***Oscillator switching current***	***RS232 ACTIVE & Shut-down current (mA)***	***Total kOS board, Active & SLEEP current (mA)***
Current kOS board (tested)	PIC18F452, LM340T, Canned crystal clock	Neither implemented	None, 1.2 mA constant current	MAX232 7.9 mA active, 12.8 mA TX/RX	31.87
1st Field trial (partially tested)	PIC18F452, LM2936, Crystal module	SLEEP implemented & Tested	Not implemented, 1.2 mA active	MAX242, 4mA Active	13.89

2nd Field Trial (theoretical)	PIC18F452, LM2936, Crystal module	BOTH implemented	Implemented, 1.2 mA full power, 0.2 mA Timer1 oscillator	MAX242, 4mA ACTIVE, 0.1 mA SLEEP	10.17
Final deployment (theoretical)	PICL18F452, Timer1 oscillator	SLEEP, RC and Timer1 oscillator	Constant 0.2 mA ACTIVE operation	MAX242, 4mA ACTIVE, 0.1 mA SLEEP	0.97

Table 9: Total current drawn by the kOS board in different configurations

The author believes that by implementing the suggested applications and methodologies in this thesis the life of the kOS board as a whole increases by at least 70%, hence the average expected life-time on a 12 Amp-Hour battery will increase from the current estimated 16.6 days to 49.1 days. This is assuming 10% activity on RS232 sensor communication, and assuming the current heavy utilization of the PIC MCU, which is around 30% for experimental purposes.

The main point that arises here is the issue of using a crystal oscillator, which currently draws around 8.9 mA. Elimination of crystal oscillator would have many advantages, and this can lead to the whole system running off the Timer1 oscillator. This requires the devices to initially start off an RC oscillator and then switching to the Timer1 oscillator. This will lead to an extremely long life of 514 days or 1 year and 3 months. *Figure 26* and *Figure 27* display the current draw and lifetime of the kOS board in different configurations.

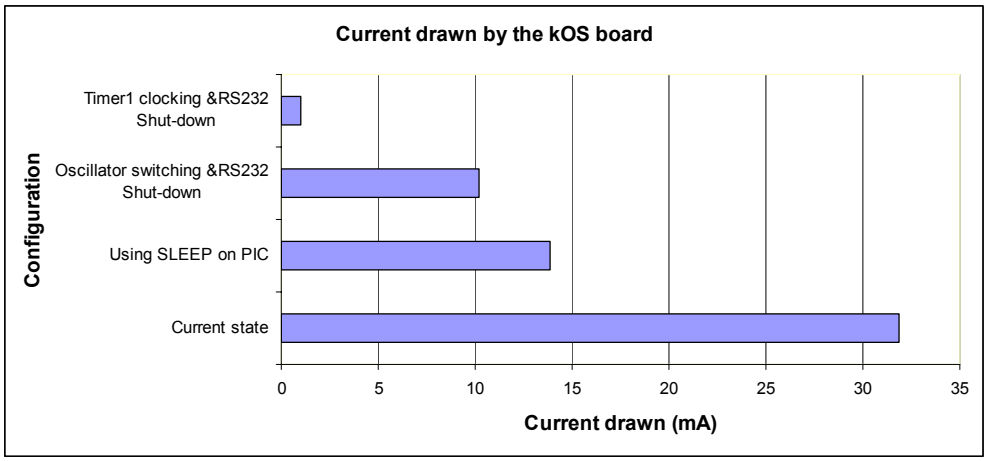


Figure 26: Current drawn by the kOS board in different configurations

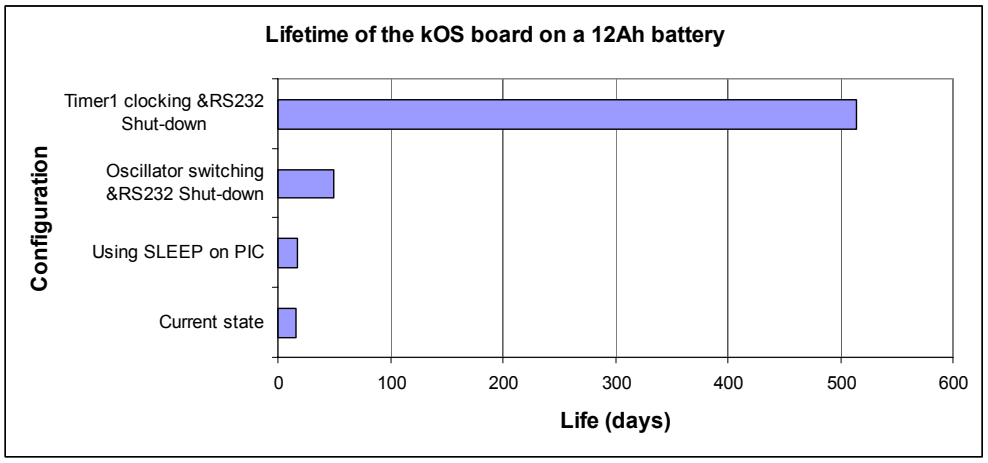


Figure 27: Lifetime of the kOS board in different configurations

6.4 Dynamic resource allocation

Successful deployment of the sensor nodes in the ocean will require careful analysis of their software and hardware performance in the laboratory in order to ensure that care has been taken for all sorts of unpredictable natural phenomena and their effects on the sensors and the radio modules. This analysis requires intensive research into the specifications of all the components used in the hardware platform and comprehensive testing and debugging of the software to ensure reliability. One of the most important facilities available on the MCU is the Watchdog Timer.

The Watchdog Timer on PIC18F452 is a free running on-chip RC oscillator, which does not require any external components. This RC oscillator is separate from

the RC oscillator of the OSC1/CLKI pin. That means that the WDT will run, even if the clock on the OSC1/CLKI and OSC2/CLKO/RA6 pins of the device has been stopped, for example, by execution of a SLEEP instruction. During normal operation, a WDT time-out generates a device RESET (Watchdog Timer Reset). If the device is in SLEEP mode, a WDT time-out causes the device to wake-up and continue with normal operation (Watchdog Timer Wake-up). The TO bit in the RCON register will be cleared upon a WDT time-out so that the programmer can find out if a reset has been due to the WDT, by examining this bit. The Watchdog Timer is enabled and disabled by a device configuration bit. If the WDT is enabled, software execution may not disable this function. When the WDTEN configuration bit is cleared, the SWDTEN bit enables/ disables the operation of the WDT.WDT can be enabled at the time of programming the PIC, and with a post-scale value of 1:128 max, it can cater for all sorts of software failures and state machine transition errors.

However the hardware operation does not have such a convenient facility for fault finding. It is intended to include many probing points for the PCB designed for kOS. The probe points are not available on the PICDEM2 Plus board, especially as it is a 3-layer PCB. This entails comprehensive operation and power analysis before the field trials to ensure prompt operation, and most importantly to be able to measure the system life time. One of the most important ways of controlling the overall system behaviour is to use the scheduler effectively. *Figure 28* displays the way scheduler controls the communication and inter-application messaging. The scheduler has complete control over the execution and setting the task times and hence it is the central point that can be used to control the behaviour of applications.

It was shown in the previous section that with only two applications, i.e. data gossiping and quorum sensing, when scheduled at high rates, as high as 30% of the MCU instruction cycles are utilized. As more applications are added, this situation can lead to the MCU being 100% utilised most of the time, hence if some applications require higher periodicity, there is no resources to be allocated to them, in terms of CPU cycles and EEPROM and data memory.

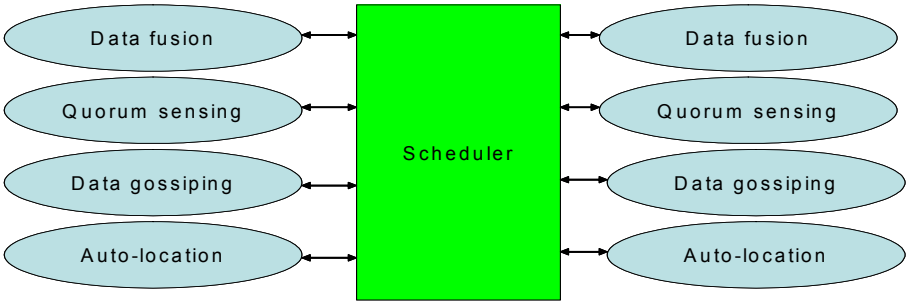


Figure 28: Central role of the Scheduler

A practical way of controlling the behaviour of objects is to control their CPU instruction cycles. Application can be assigned a maximum percentage of a time-slot's instruction cycles, and if it exceeds that limit, its periodicity can be reduced. A prototype variable is already implemented in the kOS to measure the instruction cycles that an object, such as messaging object, uses on each execution. This variable is called IC (Instruction Cycles):

```
unsigned long messaging_total_ICs = 0;      // count the total number of instruction cycles used by the
                                            Messaging object

void messaging__set_IC(void) {
        messaging_timer3_start = ReadTimer3();
}

void messaging__update_IC(void) {
        if (messaging_total_ICs < 0xFFFFFFFF)
                messaging_total_ICs += (ReadTimer3()-messaging_timer3_start);
}
```

If an application exceeds its maximum number of allocated instruction cycles, this can be detected and dealt with accordingly. When there is not much pressure on resources, applications may be able to execute above their limits. However when the system resources such as CPU cycles are scarce, resources may be allocated dynamically to the applications.

```
If ((messaging_total_IC > messaging_max_IC) & (utilisation > max_utilisation)){
    messaging_period--;
```

```
update_messaging(messaging period);
}
```

The deployment of this scheme requires two initial stages:

1. Assigning priorities to objects and applications: Each object and method in kOS holds a unique ID number.

```
// object definitions
#define OBJ_SYS_KOS                 1
#define OBJ_SYS_RADIO_INTERFACE     2
.
.
// method definitions
#define METHOD_SYS_DEFAULT_EXEC     1
#define METHOD_SYS_RESET            2
.
.
```

These ID numbers can be used to assign priorities to applications. In this way the scheduler can simply check the ID number of scheduled tasks versus their priority, and if a higher priority application requires to be run it can simply skip execution of lower priority tasks if enough CPU instruction cycles are not available.

```
#define GOSSIP_SEND_PRI     1     // Send priority for scheduler
#define GOSSIP_ADD_PRI      2     // Add priority for scheduler
#define QS_PRI              3     // Quorum priority for scheduler
```

2. Measuring the resources required by each task on execution: This is crucial as it will directly affect on the resources (memory, instruction cycles) required for each task. Some task like messaging, need a very large number of time slots and if there is too many tasks scheduled, they may have to be postponed or cancelled in favour of more important tasks such as scheduling and radio buffer management. In order to take measurements of instruction cycles used by each task, a special function was used to read the value of timer3 (working as asynchronous counter) at start and end of each task execution. The

difference, multiplied by 8 to compensate for the timer3 pre-scale value, would indicate the number of used instruction cycles.

```
void messaging__set_IC(void) {
        messaging_timer3_start = ReadTimer3();
}
// ************************
void messaging__update_IC(void) {

        if (messaging_total_ICs < 0x0FFFFFF)
                messaging_total_ICs += (ReadTimer3()-messaging_timer3_start) << 4;

        if (messaging_total_ICs > 100)
                messaging_total_ICs -= 100;
}
```

After these measurements there is need for a table to determine the overall properties of applications, in order to judge their priority and execution period. *Table 10* displays an example of such table needed to analyse the current status of the system tasks and their scheduling priority.

OBJECT/METHOD	Priority	Average iterative ICs	Period
A	1	400	τ_a
B	0	500	τ_b
C	0	10000	τ_c

Table 10: System methods and their execution properties

When such table is formed completely for all of the tasks, the periodicity can be queried from the task scheduler. This will enable us to count the duty cycle of active instruction cycles occupied by the tasks.

```
                GOSSIP_ADD_PERIOD=
        find_task_period_lo(OBJ_APPLIC_GOSSIP, METHOD_GOSSIP_ADD);
                GOSSIP_SEND_PERIOD =
        find_task_period_lo(OBJ_APPLIC_GOSSIP, METHOD_GOSSIP_SEND);
```

```
QUORUM_QS_PERIOD =
find_task_period_lo(OBJ_APPLIC_QUORUM, METHOD_QS_PERFORM);
```

The total duty cycle of the MCU application per cycle can be found by using the formula below:

Current average duty cycle= ∑ (period* Average IC)/number of timeslots

An example of such application is the GOSSIP_SEND method, when queried it has an average instruction cycle count of 2833 including a call to the MESSAGING object's CONSTRUCT_SAM_PACKET method. When the GOSSIP_SEND has a periodic execution of every other timeslot, it will have an overall weight of 5*2833 instruction cycles per kOS cycle, considering the current configuration of 10 timeslots per cycle. GOSSIP_ADD method has 2399 instruction cycles and when executed with a periodic execution of every other timeslot, it has 5*2399 instruction cycles per kOS cycle. The current average duty cycle instructions can hence be calculated as:

Duty cycles per kOS cycle= ((2833+2399)*5) = 26160 instructions per cycle

If the result of this calculation is bigger than the maximum allowed, the low priority applications can be forced to reduce their period in order to accommodate tasks with higher priority or higher number of instruction cycles. Tasks can even be removed from the scheduler queue if necessary. For example if the GOSSIP_SEND method is exceeding its maximum allowed period and we have a high priority task:

```
If ((priority =0) && (GOSSIP_SEND_PERIOD > MAX_GOSSIP_SEND_PERIOD))
    remove_task_lo(OBJ_APPLIC_GOSSIP, METHOD_GOSSIP_SEND);
```

Dynamic resource allocation can be extended to control the memory usage and EEPROM usage of application methods and it will enable total control of the scheduler over the applications' behaviour.

7. Conclusions and future work

In this section, the achievements of the project are compared to the objectives, and future improvements are suggested. The initial objectives are listed below, with the respective implementation and achievements explained.

- Objective 1: *To develop certain requirements of the kOS architecture, including data communication, and to look at the competence of kOS when compared with other operating systems developed in this arena.*

This objective was partially met. Comparison with other operating systems was done carefully with other kOS developers and effort is taken to improve the competence of kOS. However kOS is still under-going development and work is yet needed to improve the requirements. The major reason for failure in total completion of the study into kOS communication study was the need for a radio module and sensor module and neither of them were available at the time.

- Objective 2: *To study into power efficiency and implementation of energy saving techniques, effective data communication strategies and maximum resource utilization.*

This objective was fully met. Extensive test and measurements were carried out on the kOS boards, and many practical improvements were made on the boards. Study was carried out on the resource utilization and dynamic scheduling.

- Objective 3: *To design and test methods to minimize the power usage of the sensor nodes and to calculate life on batteries in different configurations.*

This objective was fully met. SLEEP facility was added to the kOS main body, taking the MCU to idle state for most of the time, offering great power savings. Life of the sensor nodes on batteries were calculated in different configuration using practical measurements on the actual hardware which is going to be used for the sensor nodes.

- Objective 4: *To consider the effectiveness of idle and active states and compare the latencies and devise competent methods for compromising between the use of different systems states and complexity of task operations.*

This objective was fully met. SLEEP states for the MCU and communication system were investigated and methods of smart-clocking were deployed and tested, enabling low power operation of the MCU for house-keeping tasks and full speed, high-power mode for the high-priority applications.

This project has covered many aspects of communication, battery management and task management of the kOS. Implementation of the techniques investigated in this work can lead to great improvement in operation of the kOS. However the author believes that an extremely important improvement that can be made to the kOS board and their cost is to design an application-specific PCB for the kOS boards as opposed to using the Microchip PICDEM2 Plus boards. This work can lead to use of low-voltage surface-mount MCU and components. The current cost of Microchip PICDEM2 Plus boards is £68 from Farnell. The total cost of a PCB required for the kOS board is calculated in *Table 11: Estimated cost of the kOS board PCB*.

Component	Price (£)
PIC18F452	7.47
MAX242 TX-RX	3.16
LM2936 regulator	1.68
MC256 EEPROM	3.32
TC74A Temperature sensor	0.58
Miscellaneous components	5
TOTAL	**£21.21**

Table 11: Estimated cost of the kOS board PCB

The design of PCB will decrease the cost of the boards by around 70% as there is no need for the LCD module and many components that are provided on the PICDEM2 Plus board for evaluation purposes. Considering the fact that 100s of these nodes will be required in future, this is a great saving both for cost and more importantly for energy saving on batteries.

Appendix A: Time plan and project management

Date	Person	Subject
24/11/03	Dr L Sacks	Initial project plan and agreement reached.
22/12/03	Dr M Britton	Discussion regarding the kOS status and requirements.
03/02/04	Dr L Sacks	Project objectives defined.
05/02/04	Dr L Sacks	Discussions on services available on Sensor networks.
28/04/04	SECOAS	RS232 communication and ADC use.
06/05/04	Dr L Sacks	Practical implementation of sleep and power usage.
10/05/04	SECOAS	Location and power requirements of sensor nodes.
20/05/04	Dr M Britton	Use of interrupts for RS232 and power usage
07/06/04	Lab Support	Use of RS232 UTP and fibre optics
11/06/04	Dr J Argirakis	Characteristics of UTP cable used for sensor nodes
16/06/04	Dr M Britton	LCD use, Timer1 use, low power clocks
23/07/04	Dr L Sacks	Dynamic resource allocation, CPU cycles
16/08/04	Dr L Sacks	Thesis write-up

Table 12: Project meetings

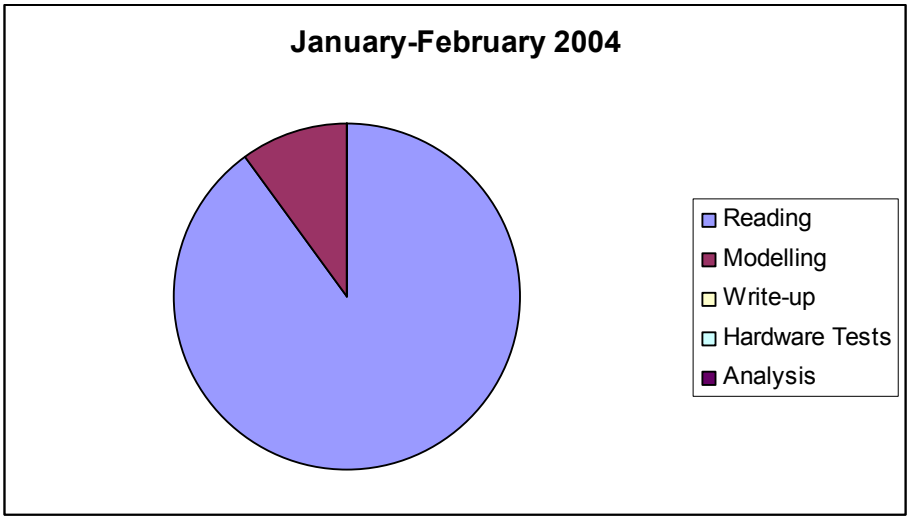


Figure 29: January-February activities

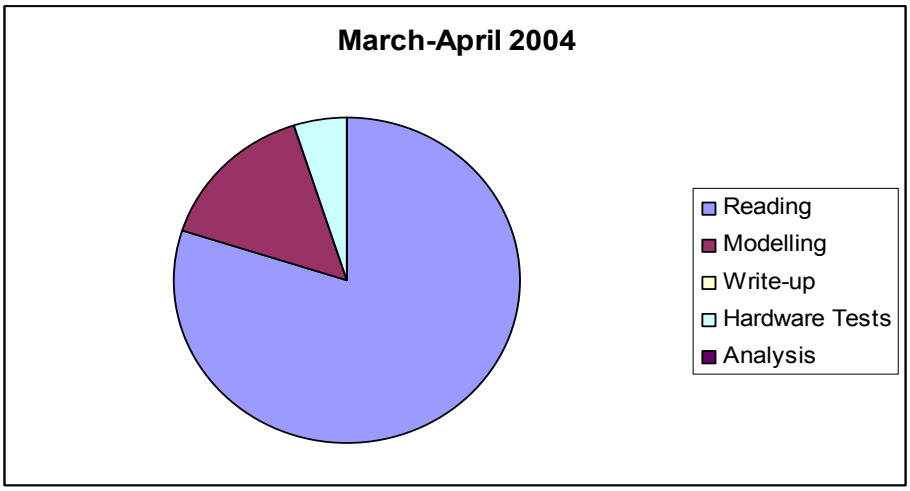


Figure 30: March-April activities

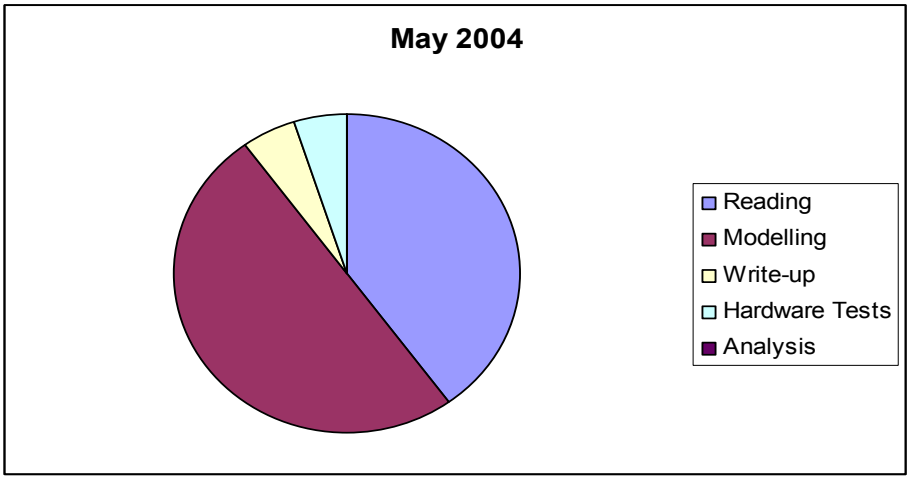


Figure 31: May activities

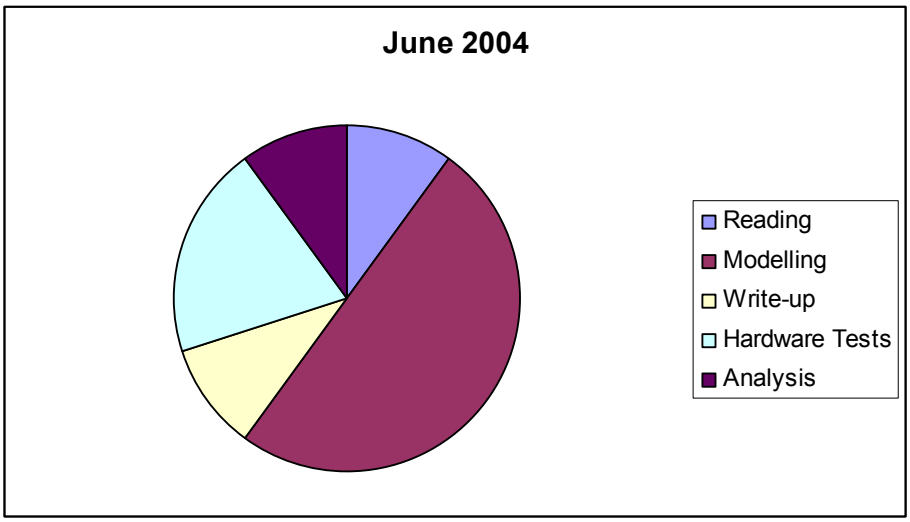


Figure 32: June activities

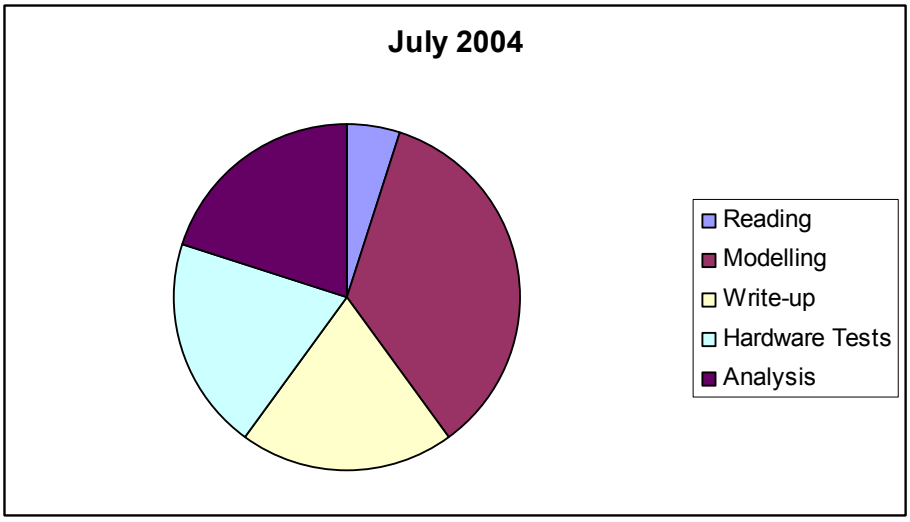


Figure 33: July activities

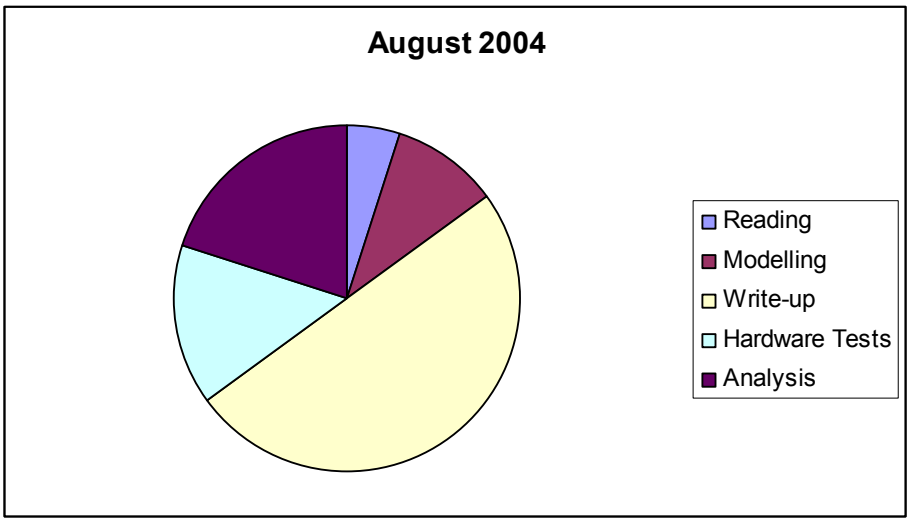


Figure 34: August activities

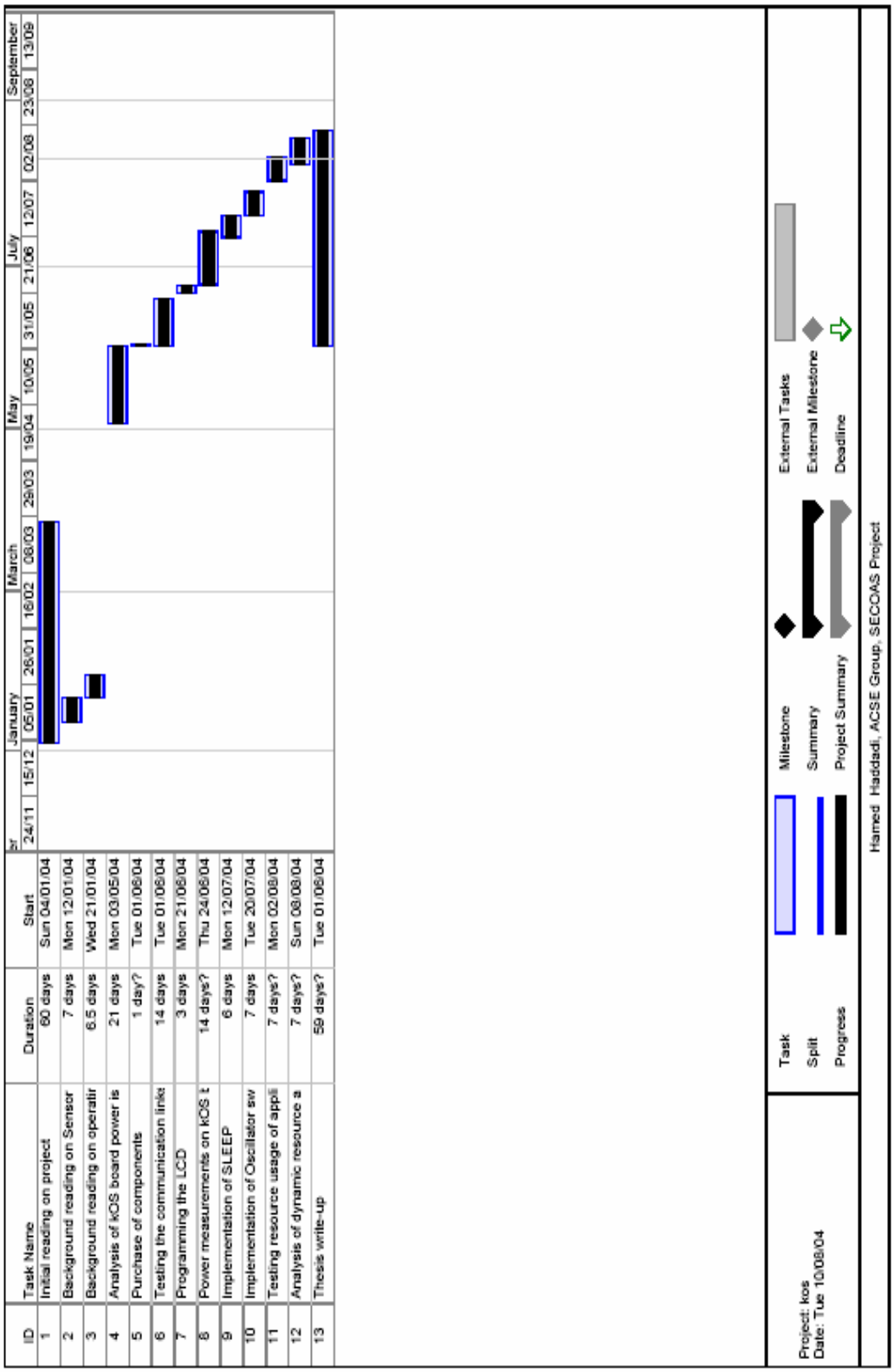


Figure 35: Project tasks

Appendix B: Software and components

MICROCHIP

PIC18FXX2

28/40-pin High Performance, Enhanced FLASH Microcontrollers with 10-Bit A/D

High Performance RISC CPU:

- C compiler optimized architecture/instruction set
  - Source code compatible with the PIC16 and PIC17 instruction sets
- Linear program memory addressing to 32 Kbytes
- Linear data memory addressing to 1.5 Kbytes

Device	On-Chip Program Memory		On-Chip RAM (bytes)	Data EEPROM (bytes)
	FLASH (bytes)	# Single Word Instructions		
PIC18F242	16K	8192	768	256
PIC18F252	32K	16384	1536	256
PIC18F442	16K	8192	768	256
PIC18F452	32K	16384	1536	256

- Up to 10 MIPs operation:
  - DC - 40 MHz osc./clock input
  - 4 MHz - 10 MHz osc./clock input with PLL active
- 16-bit wide instructions, 8-bit wide data path
- Priority levels for interrupts
- 8 x 8 Single Cycle Hardware Multiplier

Peripheral Features:

- High current sink/source 25 mA/25 mA
- Three external interrupt pins
- Timer0 module: 8-bit/16-bit timer/counter with 8-bit programmable prescaler
- Timer1 module: 16-bit timer/counter
- Timer2 module: 8-bit timer/counter with 8-bit period register (time-base for PWM)
- Timer3 module: 16-bit timer/counter
- Secondary oscillator clock option - Timer1/Timer3
- Two Capture/Compare/PWM (CCP) modules. CCP pins that can be configured as:
  - Capture input: capture is 16-bit, max. resolution 6.25 ns (TCY/16)
  - Compare is 16-bit, max. resolution 100 ns (TCY)
  - PWM output: PWM resolution is 1- to 10-bit, max. PWM freq. @: 8-bit resolution = 156 kHz, 10-bit resolution = 39 kHz
- Master Synchronous Serial Port (MSSP) module, Two modes of operation:
  - 3-wire SPI™ (supports all 4 SPI modes)
  - I^2C™ Master and Slave mode

Peripheral Features (Continued):

- Addressable USART module:
  - Supports RS-485 and RS-232
- Parallel Slave Port (PSP) module

Analog Features:

- Compatible 10-bit Analog-to-Digital Converter module (A/D) with:
  - Fast sampling rate
  - Conversion available during SLEEP
  - Linearity ≤ 1 LSb
- Programmable Low Voltage Detection (PLVD)
  - Supports interrupt on-Low Voltage Detection
- Programmable Brown-out Reset (BOR)

Special Microcontroller Features:

- 100,000 erase/write cycle Enhanced FLASH program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory
- FLASH/Data EEPROM Retention: > 40 years
- Self-reprogrammable under software control
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own On-Chip RC Oscillator for reliable operation
- Programmable code protection
- Power saving SLEEP mode
- Selectable oscillator options including:
  - 4X Phase Lock Loop (of primary oscillator)
  - Secondary Oscillator (32 kHz) clock input
- Single supply 5V In-Circuit Serial Programming™ (ICSP™) via two pins
- In-Circuit Debug (ICD) via two pins

CMOS Technology:

- Low power, high speed FLASH/EEPROM technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Industrial and Extended temperature ranges
- Low power consumption:
  - < 1.6 mA typical @ 5V, 4 MHz
  - 25 µA typical @ 3V, 32 kHz
  - < 0.2 µA typical standby current

Figure 36: PIC18F452 datasheet[18]

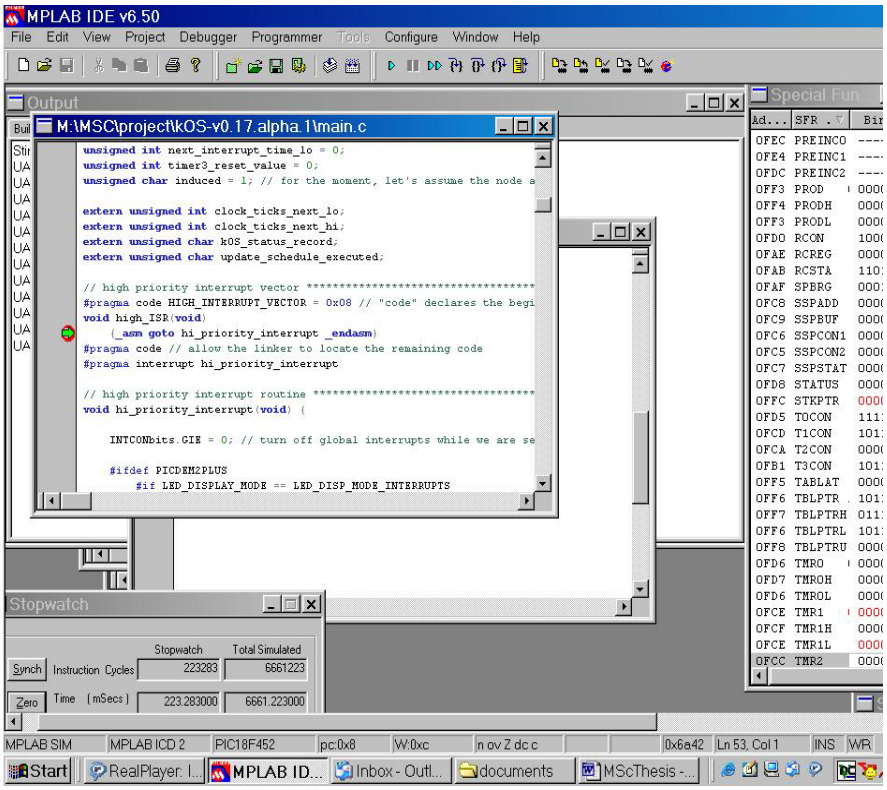


Figure 37: MPLAB® IDE Software [VI]

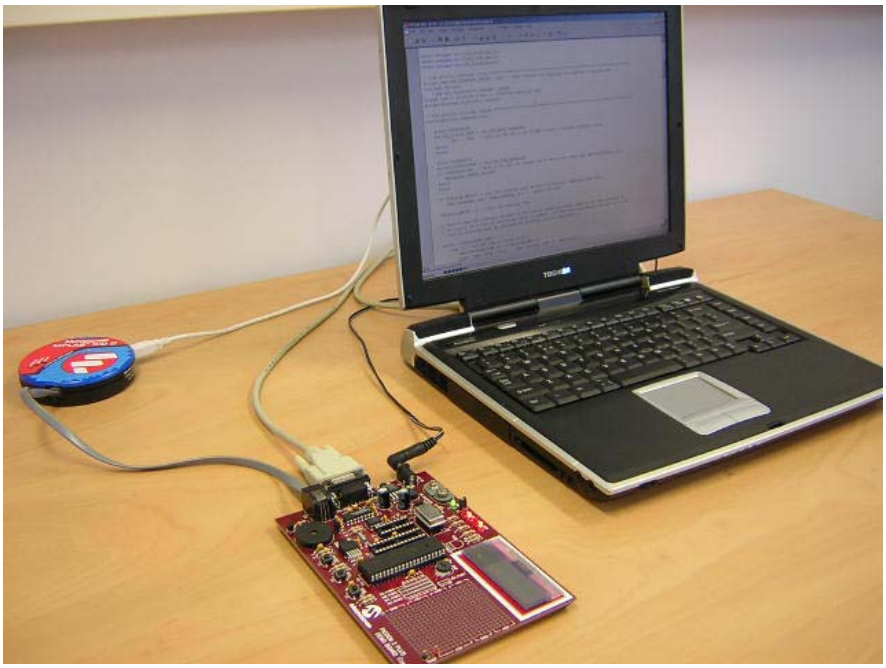


Figure 38: Microchip ICD2 and PICDEM™2 Plus board [24]

19-4323; Rev 8; 11/99

MAXIM

+5V-Powered, Multichannel RS-232 Drivers/Receivers

MAX220–MAX249

General Description

The MAX220–MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where ±12V is not available.

These parts are especially useful in battery-powered systems, since their low-power shutdown mode reduces power dissipation to less than 5µW. The MAX225, MAX233, MAX235, and MAX245/MAX246/MAX247 use no external components and are recommended for applications where printed circuit board space is critical.

Applications

Portable Computers

Low-Power Modems

Interface Translation

Battery-Powered RS-232 Systems

Multi-Drop RS-232 Networks

Features

Superior to Bipolar

- Operate from Single +5V Power Supply (+5V and +12V—MAX231/MAX239)
- Low-Power Receive Mode in Shutdown (MAX223/MAX242)
- Meet All EIA/TIA-232E and V.28 Specifications
- Multiple Drivers and Receivers
- 3-State Driver and Receiver Outputs
- Open-Line Detection (MAX243)

Ordering Information

PART	TEMP. RANGE	PIN-PACKAGE
MAX220CPE	0°C to +70°C	16 Plastic DIP
MAX220CSE	0°C to +70°C	16 Narrow SO
MAX220CWE	0°C to +70°C	16 Wide SO
MAX220C/D	0°C to +70°C	Dice*
MAX220EPE	-40°C to +85°C	16 Plastic DIP
MAX220ESE	-40°C to +85°C	16 Narrow SO
MAX220EWE	-40°C to +85°C	16 Wide SO
MAX220EJE	-40°C to +85°C	16 CERDIP
MAX220MJE	-55°C to +125°C	16 CERDIP

Ordering Information continued at end of data sheet.

*Contact factory for dice specifications.

Selection Table

Part Number	Power Supply (V)	No. of RS-232 Drivers/Rx	No. of Ext. Caps	Nominal Cap. Value (µF)	SHDN & Three-State	Rx Active in SHDN	Data Rate (kbps)	Features
MAX220	+5	2/2	4	4.7/10	No	—	120	Ultra-low-power, industry-standard pinout
MAX222	+5	2/2	4	0.1	Yes	—	200	Low-power shutdown
MAX223 (MAX213)	+5	4/5	4	1.0 (0.1)	Yes	✓	120	MAX241 and receivers active in shutdown
MAX225	+5	5/5	0	—	Yes	✓	120	Available in SO
MAX230 (MAX200)	+5	5/0	4	1.0 (0.1)	Yes	—	120	5 drivers with shutdown
MAX231 (MAX201)	+5 and +7.5 to +13.2	2/2	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; same functions as MAX232
MAX232 (MAX202)	+5	2/2	4	1.0 (0.1)	No	—	120 (64)	Industry standard
MAX232A	+5	2/2	4	0.1	No	—	200	Higher slew rate, small caps
MAX233 (MAX203)	+5	2/2	0	—	No	—	120	No external caps
MAX233A	+5	2/2	0	—	No	—	200	No external caps, high slew rate
MAX234 (MAX204)	+5	4/0	4	1.0 (0.1)	No	—	120	Replaces 1488
MAX235 (MAX205)	+5	5/5	0	—	Yes	—	120	No external caps
MAX236 (MAX206)	+5	4/3	4	1.0 (0.1)	Yes	—	120	Shutdown, three state
MAX237 (MAX207)	+5	5/3	4	1.0 (0.1)	No	—	120	Complements IBM PC serial port
MAX238 (MAX208)	+5	4/4	4	1.0 (0.1)	No	—	120	Replaces 1488 and 1489
MAX239 (MAX209)	+5 and +7.5 to +13.2	3/5	2	1.0 (0.1)	No	—	120	Standard +5/+12V or battery supplies; single-package solution for IBM PC serial port
MAX240	+5	5/5	4	1.0	Yes	—	120	DIP or flatpack package
MAX241 (MAX211)	+5	4/5	4	1.0 (0.1)	Yes	—	120	Complete IBM PC serial port
MAX242	+5	2/2	4	0.1	Yes	✓	200	Separate shutdown and enable
MAX243	+5	2/2	4	0.1	No	—	200	Open-line detection simplifies cabling
MAX244	+5	8/10	4	1.0	No	—	120	High slew rate
MAX245	+5	8/10	0	—	Yes	✓	120	High slew rate, int. caps, two shutdown modes
MAX246	+5	8/10	0	—	Yes	✓	120	High slew rate, int. caps, three shutdown modes
MAX247	+5	8/9	0	—	Yes	✓	120	High slew rate, int. caps, nine operating modes
MAX248	+5	8/8	4	1.0	Yes	✓	120	High slew rate, selective half-chip enables
MAX249	+5	6/10	4	1.0	Yes	✓	120	Available in quad flatpack package

Figure 39: MAX242 Datasheet 23

National Semiconductor

February 1995

LM2936 Ultra-Low Quiescent Current 5V Regulator

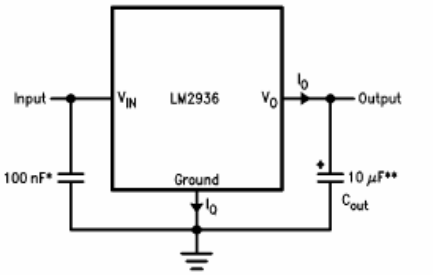
General Description

The LM2936 ultra-low quiescent current regulator features low dropout voltage and low current in the standby mode. With less than 15 μA quiescent current at a 100 μA load, the LM2936 is ideally suited for automotive and other battery operated systems. The LM2936 retains all of the features that are common to low dropout regulators including a low dropout PNP pass device, short circuit protection, reverse battery protection, and thermal shutdown. The LM2936 has a 40V operating voltage limit, −40°C to +125°C operating temperature range, and ±3% output voltage tolerance over the entire output current, input voltage, and temperature range. The LM2936 is available in both a TO-92 package and an 8-pin surface mount package with a fixed 5V output.

Features

- Ultra low quiescent current ($I_Q \leq 15$ μA for $I_O \leq 100$ μA)
- Fixed 5V, 50 mA output
- Output tolerance ±3% over line, load, and temperature
- Dropout voltage typically 200 mV @ I_O = 50 mA
- Reverse battery protection
- −50V reverse transient protection
- Internal short circuit current limit
- Internal thermal shutdown protection
- 40V operating voltage limit

Typical Application



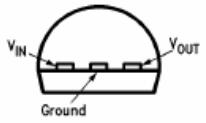
TL/H/9759–1

* Required if regulator is located more than 2" from power supply filter capacitor.

** Required for stability. Must be rated for 10 μF minimum over intended operating temperature range. Effective series resistance (ESR) is critical, see curve. Locate capacitor as close as possible to the regulator output and ground pins. Capacitance may be increased without bound.

Connection Diagrams

TO-92 Plastic Package (Z)

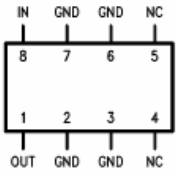


TL/H/9759–2

Bottom View

Order Number LM2936Z-5.0
See NS Package Number Z03A

8-Pin SO (M)



TL/H/9759–6

Top View

Order Number LM2936M-5.0
See NS Package Number M08A

RRD-B30M75/Printed in U. S. A.

Figure 40: LM2936 Datasheet (Courtesy of National Semiconductor)

Appendix C: References

1. M. Britton: “Guide to the kOS Operation System, V 0.18 Alpha, UCL Internal report, June 2005
2. M. Britton, L. Sacks, H. Haddadi: “kOS - A Robust, Stateless Operating System Supporting a Self-Organising Wireless Sensor Network”, Submitted to The First IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks
3. B. Hohlt, L. Doherty, E. Brewer: “Flexible Power Scheduling for Sensor Networks”, IEEE and ACM Third International Symposium on Information Processing in Sensor Networks April 2004
4. SECOAS homepage: http://www.adastral.ucl.ac.uk/sensornets/secoas/
5. Offshore Wind Farms: http://www.offshorewindfarms.co.uk/sites/scroby-sands.html
6. H. Qi, S. S. Iyengar, K. Chakrabarty: “Distributed Sensor Networks - a review of recent research”, Journal of the Franklin Institute, vol. 338, pp. 655-668, 2001
7. Mesh-Enabled Solutions for Sensor Networks, http://www.meshnetworks.com/pages/applications/sensor_networks.htm
8. T. Nieberg, S. Dulman, P. Havinga, L. v. Hoesel, J. Wu: “Collaborative Algorithms for Communications in Wireless Sensor Networks”, in (Ambient Intelligence: Impact on Embedded Systems Design), edited by T. Basten, M. Geilen, H. de Groot, Kluwer Academic Publishers, November 2003
9. I. Wokoma, L. Sacks, I. Marshall: “Biologically Inspired Models for Sensor Network Design”, London Communication Symposium, September 2002
10. Ning Xu: “A Survey of Sensor Network Applications”, Computer Science Department, University of Southern California
11. UCB Smart Energy project: http://www.citris.berkeley.edu/smartenergy/smartenergy.html
12. K. Sohrabi, “Protocols for Self-Organization of a Wireless Sensor Network” IEEE Personal Communications, Oct. 2000
13. Ibiso Wokoma, Ioannis Liabotis, Ognjen Prnjat, Lionel Sacks, Ian Marshall: “A Weakly Coupled Adaptive Gossip Protocol for Application Level Active Networks”, 3rd International Workshop on Policies for Distributed Systems and Networks (POLICY'02), June 05 - 07, 2002, Monterey, California
14. Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister: “System architecture directions for networked sensors”, 9th international conference

on Architectural support for programming languages and operating systems, Cambridge, Massachusetts, United States, 2000

15. Eyes Project webpage: http://eyes.eu.org/
16. Bluetooth specifications: http://www.bluetooth.org
17. A. Rodzevski, J. Forsberg, I. Kruzela, "Wireless sensor network with Bluetooth", In Proceedings for Smart Objects Conference (SOC'03), May 2003, France
18. Microchip PIC18F452 Data sheet, http://www.microchip.com
19. Systems Engineering Master Document for the SECOAS Project, DTI Next Wave Technologies and Markets, UCL Internal report, June 2003
20. Suet-Fei Li, Roy Sutton and Jan Rabaey, "Low Power Operating System for Heterogeneous Wireless Communication Systems", PACT 01 Conference, Barcelona, Spain September 8-12, 2001
21. TinyOS home page: http://webs.cs.berkeley.edu/tos/
22. S.Dulman, P.Havinga: "Operating System Fundamentals for the EYES Distributed sensor Network ", PROGRESS Workshop, Utrecht, the Netherlands, October 2002
23. Maxim +5V-Powered, Multi-channel RS-232 Drivers/Receivers Datasheet
24. Microchip PICDEM™2 Plus User's Guide, http://www.microchip.com
25. HP, The Versatile Fibre Optic Connection, available from http://rswww.com
26. Gary Wong: "Motes, nesC, and TinyOS" December 9, 2003
27. International Organization for Standards: http://www.iso.org
28. Philip Levis, Nelson Lee, Matt Welsh, and David Culler: "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications", In Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys) 2003, November 2003

Appendix D: Bibliography

[I] L. Sacks, M. Britton, I. Wokoma, A. Marbini, T. Adebutu, I. Marshall, C. Roadknight, J. Tateson, D. Robinson and A. Gonzalez-Velazquez, *The development of a robust, autonomous sensor network platform for environmental monitoring*, Sensors and their Applications XXII, Limerick, Ireland, 2nd-4th September 2003

[II] A. Gonzalez-Velazquez, M. Britton, L. Sacks and I. Marshall, *Energy savings in wireless ad hoc sensor networks as a result of network synchronisation,* London Communication Symposium, University College London, 8th-9th September 2003

[III] Edoardo Biagioni and Kent Bridges, *The application of remote sensor technology to assist the recovery of rare and endangered species*. In Special issue on Distributed Sensor Networks for the International Journal of High Performance Computing Applications, Vol. 16, N. 3, August 2002.

[IV] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. *Wireless sensor networks for habitat monitoring*. In ACM International Workshop on Wireless Sensor Networks and Applications (WSNA'02), Atlanta, GA, September 2002.

[V] I.F.Akyildiz, , Su Weilian, Y Sankarasubramaniam, and E Cayirci, *A survey on sensor networks*, IEEE Communications magazine, August 2002

[VI] H. Haddadi, *MPLAB IDE for SECOAS, Object Development Environment Users Guide*, UCL internal report, August 2003