Preface

This report describes project work carried out in the Faculty of Arts, Computing, Engineering and Sciences at Sheffield Hallam University between June 2006 till September 2006.

The submission of the report is in accordance with the requirements for the award of the degree of MSc in Electronics and Information Technology under the auspices of the University.

Acknowledgement

This work would not be completed without the guidance and advice of Dr Bala Amavasai. His experience and knowledge had helped resolved many of the difficulties encountered, particularly in the programming of the evaluation boards. His diligence and patience has been an inspiration and encouragement.

Thanks are also in order to Kim Chuan in the MMVL group who had also volunteered his ideas in the steps leading to the completion of this project.

My gratitude to Mr Adrian Jeffries who has helped in providing the components needed in assembling the circuits needed in the project.

I also extend my gratitude to my colleague, Daren Lee, who has worked along me, though on a separate project, whose presence throughout this project has helped me in persevering whenever I faced difficulties.

My gratitude also to Chee Ming and Yu Chen who have graciously provided many delicious and satisfying lunches.

My fellow housemates in Sharrow Street who have to bear with the constant gripes and grouses. Kevin, Kok Seng, Kok Khim, Tissa and Wen Yuan, thank you for the patience and encouragement.

I am also grateful to my fellow brothers and sisters in the Sheffield Chinese Christian Church who have shown much concern and care throughout the duration of my time in Sheffield.

My parents too have played an important part, though not directly, in the completion of this project. Thank you for your prayers and kind words of encouragement.

Last but definitely not least, I would like to acknowledge God for whom everything that has seemed impossible has only been made possible by the grace He has shown.

Abstract

The advent of wireless networks have greatly simplified the transmission of data and setting up communication networks. However an increasing need for a wireless network that is suitable for home automation and industrial control has led to the development of the IEEE 802.15.4 standard that emphasizes on low complexity with multi-month or multi-year battery life.

The implementation of the IEEE 802.15.4 standard is investigated using the Jennic JN5121-EK000 Evaluation Kit. A wireless voice control system was built to represent a simple remote wireless voice control system.

A signal is to be fed in real-time to the system which would then be correlated with a previously sampled signal.

Various forms of speech recognition systems would be investigated to implement a simple yet effective voice control system.

The system would have to be fast and the algorithm to correlate the signals would have to be simple within the constraints of the memory space of the JN5121 evaluation kit and yet effective.

Table of Contents

.List of Figures

Chapter 5:

Chapter 1
Introduction

The aim of the thesis is to develop a wireless voice control system based on the IEEE802.15.4 standard. This was to be achieved using the Jennic JN5121-EK000 Evaluation Kit which is based on the IEEE802.15.4 standard including ZigBee. The evaluation kit was to be integrated with a voice capture system. Then a suitable method of correlating the signals was to be implemented in order to match signals.

1.1 Background

The rapid growth of wireless communication in recent years have led to numerous forms of wireless devices based on IEEE standards such as 802.11, Bluetooth and infra-red. Especially in the Wi-Fi and Bluetooth circle, numerous applications have found their way into home communication, wireless Internet, PC-to-phone communication, wireless VOIP, digital cameras, digital presentations, etc. The high bandwidths of these standards have provided a wireless form of communication that has provided much accessibility to information and technology. However there was an increasing need for a technology that would be suited for low power consumption applications yet

providing a wireless communication that would enable simple home automation. The IEEE802.15.4 was introduced to provide a low-cost, low-power solution that would enable multi-year battery life in applications suitable for home automation and industrial control.

1.2 Motivation

Speech recognition is a very complex process of converting speech contained in a signal into words. The use of Hidden Markov Models and Artificial Neural Networks in speech recognition software is widespread and complex. This project utilised a simple approach of correlating signals due to the limitations of the onboard memory and time constraints of the project itself. The goal was to achieve a speech recognition system that would be able to work within the guidelines set by the IEEE 802.15.4 standard without compromising efficiency and effectiveness.

1.3 Time Schedule

Task	**Time Period**
Literature review; investigation of IEEE 802.15.4 standard and speech recognition	June 2006
Contruction of voice capture system	June 2006
Familiarisation of the Jennic J5121 evaluation kit	July 2006
Programming of voice control system	August 2006
Writing of dissertation	September 2006

Figure 1.1 Time Schedule of project

1.4 Potential Hazards

Working long straight hours in the project room posed a few dangers such as fume poisoning whilst soldering and repetitive strain injury from working long hours with the keyboard and mouse. These injuries were avoided by taking a few precautionary methods:

- Ensuring sufficient ventilation whilst soldering
- Use of fume extraction fan whilst soldering
- Positioning of monitor and chair to a comfortable position
- Resting every hour for 10 minutes to avoid strain injuries
- Ensuring plugs are working by visual inspection

1.5 Technical limitations

Due to the technology being used in this project is a relatively young technology (The IEEE 802.15.4 standard was formed in 2003), there were few resources to be tapped into for the implementation various functions needed in the report. Little references could be made to any other research work being investigated on the voice control implementation which was based on the IEEE 802.15.4 standard. The project had also a time constraint of three months in implementing a simple speech recognition algorithm and there was a shortage of time in fine tuning the algorithm.

1.6 Report Guideline

Chapter 2 discusses the IEEE 802.15.4 standard and its comparison with other wireless technologies. A brief discussion on the speech recognition models is included. Chapter 3 briefs on the JN5121 microcontroller and the microphone pre-amplifier circuit used in this project. Chapter 4 describes the configuration and parameter settings to sample the voice signals. Chapter 5 details the results and observations as well as a critical analysis of the results obtained. Chapter 6 recommends further work to improve the functionalities of the system as well as the conclusion.

1.7 Summary

This chapter gives the background and motivation behind the project. It briefly accounts the timeline schedule of the project and gives a brief outline of the next chapters.

Chapter 2

Relevant Theory and Analysis

This chapter contains a basic introduction of the IEEE 802.15.4 standard and the basics of speech recognition.

2.1 The 802.15 Task Group Number 4

The IEEE 802.15.4 standard is part of four task groups of the 15th group of the IEEE802 which specialises in Wireless Personal Area Networks (WPANs). It is a low-rate WPAN to enable multi-year battery life. It has a lower complexity compared to other wireless networks facilitating low-cost implementations. Its first edition was released in May 2003.

There are two PHY options defined which allows data rates flexibility. DSSS is deployed in both options. The 868/915 MHz PHY option operates within Europe at 860.0-868.6 MHz. In the US the 902-928MHz range is selected. At 868MHz, there is only one channel available offering a data rate of 20kbit/s. At 915MHz, there are 10 channels with 40kbit/s per channel available. At these lower frequencies, though there are better propagation conditions but there are also interference from other analogue bands. Increasing the frequency to the free ISM

band of 2.4GHz which operates at 2.4-2.4835 GHz, the number of channels is increased to 16 with 250kbit/s per channel. Operating in the ISM band allows worldwide operation but suffers from higher propagation loss and also interference from other devices working in the 2.4GHz ISM band. These devices are expected to cover a range of 10-20 m with output power of 1mW.

2.1.1 Configuration of a LR-WPAN

There are two different types of devices in a LR-WPAN network; a full-function device (FFD) and a reduced-function device (RFD). The FFD is able to operate in three different modes; PAN coordinator, a coordinator or a device. The FFD can communicate with RFDs or other FFDs while a RFD can only communicate with a FFD. Thus, the RFD is suitable for simple applications such as light switches or a passive humidity sensor where no large amounts of data are transmitted and communication with a single FFD is sufficient. The RFD is thus configured using minimal resources and memory to reduce costs.

2.1.2 Network Topologies

There are two topologies available depending on the needs of the application; star topology or the peer-to-peer topology.

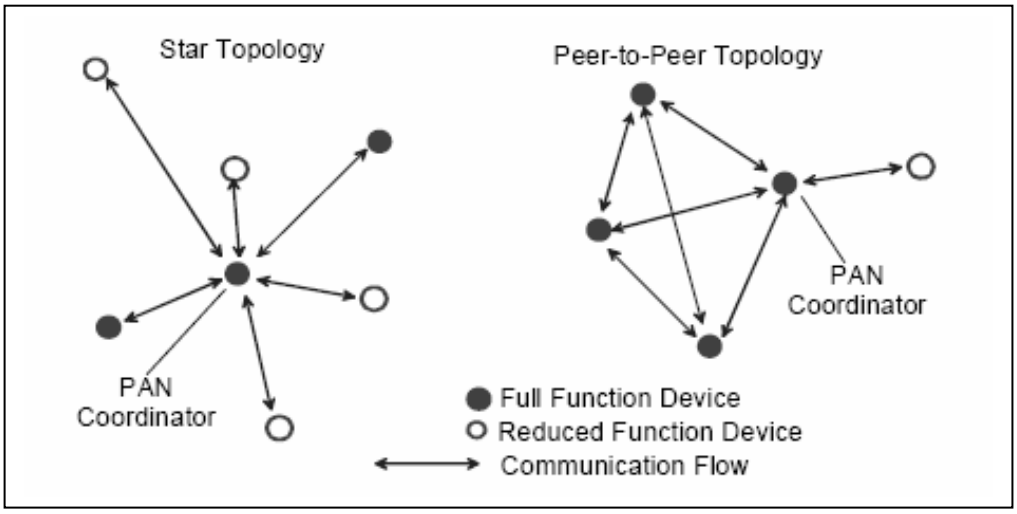

Fig 2.1 Different possible topologies in a IEEE 802.1.5.4 network (Diagram taken from [1]IEEE Std 802.15.4, p14)

In the star topology, the communication within the LR-WPAN is between devices and a single controller, the designated PAN coordinator. A device would have some application which associates itself as either the initiation point or termination point. The PAN coordinator could be used to start, end or transmit communication routes around the network. Each device has a unique 64 extended address on either topology. This unique address is used for direct communication within the PAN or it can be exchanged for a short address allocated by the PAN when the device associates with controller.

While the peer-to-peer topology also contains a PAN coordinator, any device may communicate with another device within range of one another. This allows more complex network formations such as a mesh networking topology, allowing messages to have a multiple hop routing to another device on the

network. Such networks would benefit applications in wireless sensor networks, security, smart farming, and inventory assessments.

2.1.3 Models of Data Transfer

There are three types of data to be transmitted;

i) data transfer from a device to a coordinator
ii) data transfer from a coordinator to a device
iii) data transfer between two peer devices

In the star topology only two types of data are present as there are no peer devices; data is transferred between a coordinator and a device. In a peer-to-peer topology, all three types of data transfer may be present. The availability of beacon transmissions would affect the transmission modes of the data. Through the use of beacons, low-latency devices such as computer peripherals would be able to communicate with the WPAN. The beacon is also used to associate neighbouring networks.

2.1.3.1 Data transfer to a coordinator

The device that wishes to communicate with the coordinator, listens for the beacon within the network. Two possible modes are possible; one with a beacon-enabled network and one beaconless.

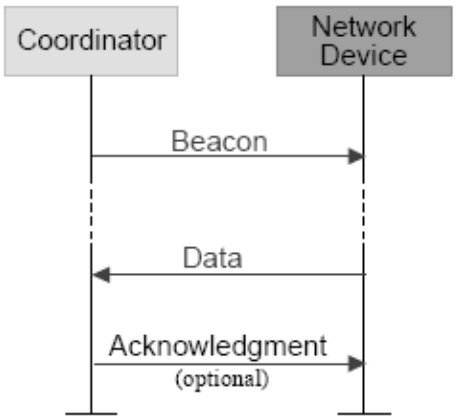

Fig 2.2 Data transfer to a coordinator in a beacon-enabled network (Diagram from [1]IEEE Std 802.15.4, p19)

In a beacon-enabled network; after finding the beacon, the device will synchronise to the superframe structure. This structure is defined by the coordinator. The superframe is divided into 16 equally sized slots. If the superframe structure is not used, the coordinator may turn off beacon transmissions. Thus, beacons are used to synchronise devices within the WPAN, by identifying the PAN and describing the structure of the defined superframe. When a device intends to communicate during the contention access period between two beacons, the slotted CSMA-CA mechanism is used to provide appropriate access. This superframe could have active and inactive slots. During the inactive zone, the coordinator will not interact with the network and may enter a low-power mode to preserve battery life. Once the coordinator has received the data, it could send an optional acknowledgement frame.

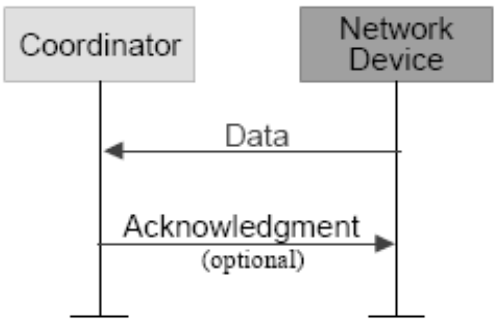

Fig 2.3 Data transfer to a coordinator in a beaconless network (Diagram from [1]IEEE Std 802.15.4, p19)

In a beaconless network, the device simply transmits the data frame using slotted CSMA-CA to the coordinator. The coordinator could send an optional acknowledgement frame if the data frame is received successfully.

2.1.3.2 Data transfer from a coordinator

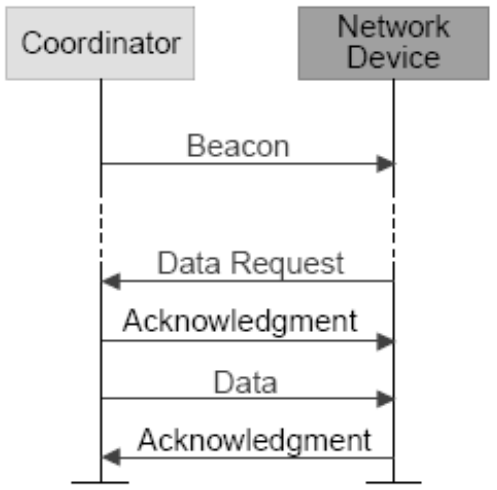

Fig 2.4 Data transfer from a coordinator in a beacon-enabled network (Diagram from [1]IEEE Std 802.15.4, p20)

In a network with beacons enabled, the coordinator will set the beacon to indicate a pending message. A listening device would then transmit a MAC frame to receive the message. The coordinator could then send an optional

acknowledgement frame and sends the message using CSMA-CA. Upon receiving successfully the data, the device would then send an acknowledgement frame. The message would then be removed from a list of pending messages in the beacon.

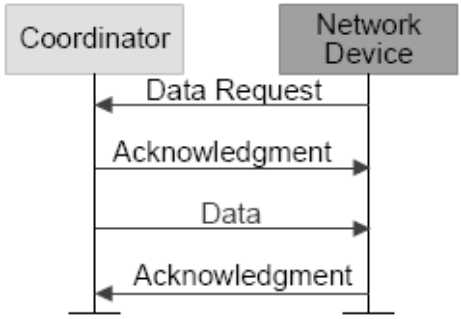

Fig 2.5 Data transfer from a coordinator in a beaconless network (Diagram from [1]IEEE Std 802.15.4, p20)

In a beaconless network, the data is stored in the coordinator until the device makes contact with the coordinator and requests the data. This communication is executed using unslotted CSMA-CA. The coordinator would then send an acknowledgement frame and the data is sent. If there is no data pending, the coordinator would transmit a data frame with a zero-length payload. The device would send an acknowledgement upon receiving successfully the data.

2.1.3.3 Data transfer in a peer-to-peer network

In a peer-to-peer network each device that wants to communicate must receive constantly to maintain connectivity with each other within its radio sphere. This is done using unslotted CSMA-CA. An alternative method but more complicated is to synchronise with each other constantly.

2.1.3.4 The CSMA-CA mechanism

The difference between the beacon-enabled and beaconless network is further discussed here, analysing the different CSMA-CA mechanism deployed in both configurations.

In a beaconless network, the unslotted CSMA-CA mechanism is used. Every time a device wishes to transmit a data frame or MAC command, it waits for a random period of time. The device would transmit the data if the network is found to be free. If busy, the device would then again wait for a random period of time before trying again. In this network, acknowledgement frames are sent without using CSMA-CA mechanism.

In a beacon-enabled network, a slotted CSMA-CA mechanism is used. The backoff slots are synchronised with the start of the beacon transmission. A device that wishes to transmit must wait till the next backoff slot and then wait for a random number of backoff slots. If the channel is found to be busy, it again waits for a random number of backoff slots. If the channel is not busy, the device would begin transmitting on the next free backoff slot boundary. Acknowledgement and beacon frames however are sent without using a CSMA-CA mechanism.

2.1.4. MAC frame of 802.15.4

In comparison, the MAC layer of 802.15.4 is much simpler than other WPAN technologies such as Bluetooth. There is no support for a synchronous voice link. Each MAC frame begins with a 2-byte frame control field, describing the content of the rest of the frame. The next 1-byte sequence number is used to match acknowledgements with the previous data transmitted. The variable address field which is 10-20 bytes long may contain source and/or destination addresses in various formats. While the payload is variable in length, the whole MAC

frame has a maximum length of 127 bytes. A 16-bit FCS protects the frame. Four different MAC frames have been defined; acknowledgement, data, beacon and MAC command.

2.1.4.1 The MAC sublayer of 802.15.4

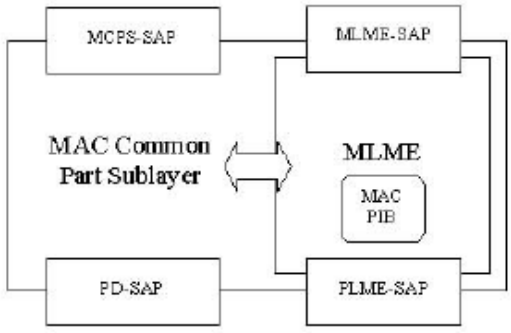

Fig 2.6 The MAC sublayer
(Diagram from [1]IEEE Std 802.15.4, p55)

The MAC sublayer is analogous to the OSI reference model of the data link layer. It is responsible for handling access to the physical radio channel such as generating and synchronising network beacons in coordinators, supporting PAN association and disassociation, controlling network security, setting the CSMA-CA mechanism for data transfer and maintaining a link between two MAC devices.

The MAC sublayer is an interface located between the SSCS (Service Specific Convergence Sublayer) and the PHY (Physical Layer). It includes two main subcomponents; the MLME (MAC Layer Management Entity) and the MCPS (MAC Common Part Sublayer). The MLME provides the communication interface which controls the layer management functions and maintains a database of managed objects known as the MAC sublayer PIB (PAN Information Base). The two SAPs(Service Access Points) in the MAC layer provides two services; the MAC data service which is linked through the MAC common part

sublayer (MCPS) data SAP (MCPS-SAP); and the MAC management service accessed through the MLME-SAP.

2.1.5 Primitives

The services defined in the IEEE802.15.4 standard make use of 4 primitives: Request, Indication, Response and Confirm.

Using an example of a device that wishes to connect to an existing PAN; the Request primitive is used to indicate a service is being initiated. The Request is directed from the MAC to the PHY layer where it is modulated and transmitted. The coordinator would then receive the signal at the PHY layer, demodulates it and directs it to the MAC.

The Indication primitive is then generated to the application layer of the coordinator which would then decide if the device is allowed into the PAN. The Response primitive would then travel the same path and translated into a Confirm primitive, telling the device if the request has been approved.

2.1.6 Transmitting Data

The MCPS layer controls the transmission of data between any two devices in the PAN network. A device that wishes to transmit a message will issue a Request to the MCPS-Data service. Among the parameters that need to be defined in the Request primitive is the addressing mode of the source (SrcAddrMode) where the address could be a 16-bit address, an IEEE 64 bit extended address or omitted altogether; the 16 bit PAN identifier (SrcPANId) of the device that it is being transmitted; the address of the source device (SrcAddr); the addressing mode of the destination which could again take a normal 16-bit address, an IEEE 64 bit extended address or omitted altogether; the destination PAN identifier (DstPANId); the destination address (DstAddr); the MAC Sublayer Data Unit

Length (msduLength) which indicates the number of octets contained in the MSDU to be transmitted; the number of octets forming the MSDU (msdu); the handle of the MSDU (msduHandle); and the transmission options (TxOptions) such as the receiving of acknowledgements after transmission of messages, the transmission of the message during a reserved GTS slot, sending of an indirect transmission whereby the transmission data is stored in a queue until the coordinator is able to receive it and the setting of security features.

The use of a message queue enables the transmission of messages to a RFD (Reduced Function Device) in sleep mode. The message will be in queue until the device is in running mode. Though slower, this method helps to reduce lost transmissions. If the devices were to continue to send messages, there is a probability that the message queue would be occupied. As yet there is no specification on this problem and it would be up to the discretion of the programmer to ensure that such an overflow does not occur. The messages in this indirect queue can be purged using the MCPS-Purge Request.

2.1.7 Transmission Range

The transmission range if built specified around a transmitter power of 0.5 mW allows a line-of-sight range of 10 m to 100 m. If a mesh network is to be constructed and routing capabilities added, it is theoretically possible to extend the range to several thousands of meters at the cost of high latency.

2.1.8 Reliability

There are different data checking mechanisms within the different layers to ensure the reliability of data received. At the Physical layer, the offset quadrature phase shift keying(O-QPSK) is used for the 2.4GHz band. The PSK technique deployed allows a more robust and reliable network compared with FSK (Frequency Shift Keying) deployed in Bluetooth. The channels used in the IEEE

802.15.4 standard are different from the channels used by WiFi channels. This prevents interference between the networks although they operate within the same bandwidth. At the MAC level, a 16 bit frame check sequence (FCS) ensures that errors are removed in the first layers of the MAC. If the FFDs are set to send acknowledgment frames upon receiving the data, it would further increase the reliability of sent data in a network.

2.1.9 Security

IEEE802.15.4 specifies three levels of security: no security, access control lists, and symmetric encryption using AES-128. Key distribution is not specified further. Security is a must for home automation or industry control applications. Up to now, the success of this standard is unclear as it is squeezed between Bluetooth and enhanced RFID/RF controllers.

2.1.10 ZigBee

A popular implementation of the 802.15.4 standard is ZigBee which sits on the 802.15.4 standard and handles the network and application layers. Comparing ZigBee and other wireless network configurations:

	Zigbee	**Bluetooth**	**WiFi**
Freq. range (GHz)	2.4-2.4835	2.4-2.4835	2.4-2.835
Standard	802.15.4	802.15.1	802.11g
Data rate	250 kbps	1 Mbps	54 Mbps
Bandwidth (MHz)	83.5	83.5	83.5
Access	CSMA/CA	TD	
Modulation	BPSK,OQPSK,DSSS	GFSK,FHSS	BPSK,QPSK,MQAM,OFDM
Range (m)	30	10(100)	30
Power consumption	Tx : 35 mA	Tx : 40 mA	Tx : 400+ mA
Standby Power	3 uA	200 uA	20 mA
Memory	32 – 60 kB	100+ kB	100+ kB
Topologies	Mesh, Point-Multipoint	Point-Multipoint	Point-Multipoint

Fig 2.7 Comparison of Zigbee and other wireless network configurations

Thus, the advantages of ZigBee are the low power consumption and mesh networking capabilities (which theoretically allows an unlimited number of devices). With the lower data rate, ZigBee is suitable for non-speed critical applications.

2.2 Speech recognition

2.2.1 Introduction

Speech recognition in this project is defined as the process of translating a signal containing speech into words using computer based algorithms. It is used in mobile phones in applications such as voice dialling, voice-assisted customer service hotlines and simple data entry such as the input of a string of numbers such as a credit card number. It is different from voice recognition which is the process of identifying the person who is speaking and not just what is being said.

2.2.2 Factors in speech recognition

The concern in any speech recognition algorithm is the performance of the system measured by the word error rate. It is influenced by many factors such as the environment such as the background noise and the rate of speech of the speaker.

Many speaker-dependent dictation systems today, claim of a high accuracy of between 98% and 99% in recognising words when operating in a controlled environment. The optimal conditions usually include that the speaker have matched the speaker characteristics using the training approach usually contained within the system, a proper rate of speech and a quiet environment.

Systems which do not include training could recognise a smaller number of words from the speaker and is often used in mobile phone applications or voice-assisted call routing in large organisations.

2.2.3 Methods employed in speech recognition

There are a few modern approaches widely used in speech recognition such as the HMM(Hidden Markov Model) approach and the ANN(Artificial Neural Network) approach. They are both briefly described in this chapter.

2.2.3.1 Hidden Markov Model

The Hidden Markov Model (HMM) is based on the Markov process (a process dependent on past states given the present state) where the system has unknown parameters that needs to be ascertained by determining the hidden parameters using the known parameters. This enables the systems to be used widely in systems that require pattern recognition such as the speech recognition system being discussed. The HMM process is considered as a simple dynamic Bayesian network.

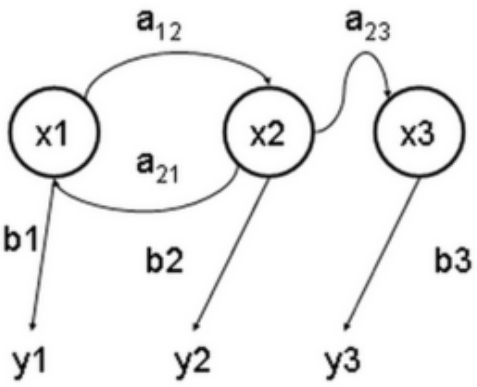

Fig 2.8 State Transitions in a Hidden Markov Model
(Diagram taken from http://en.wikipedia.org/wiki/Hidden_Markov_model)

The diagram above describes the possible state transitions in a Hidden Markov Model :-

x - hidden states

y – observable outputs

a – transition probabilities

b - output probabilities

In a normal Markov model, each state in the model is directly observable and the probabilities involved in the process are with regards to the state transitions. However in a hidden Markov model, each state is not directly visible but influential variables are made known.

Every state contains a distribution probability across the possible output of tokens. A generated token sequence from the HMM would be used to predict the sequence of states.

2.2.3.1.1 Architecture of a HMM

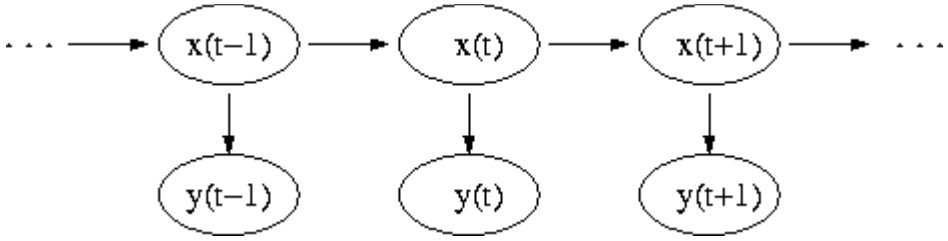

Fig 2.9 General architecture of a HMM (Diagram taken from http://en.wikipedia.org/wiki/Hidden_Markov_model)

The diagram illustrates a general architecture of the HMM. The oval shapes are representations of random variables. Variable x(t) is the hidden variable at time t while y(t) is the value of the observed variable. Each hidden variable is dependent only on the value of the hidden variable x(t-1), the time at t-1. The observable variables are only dependent on the value of the hidden variable at any particular point of time.

Therefore the probability of an observed sequence Y=y(0),y(1),…,y(L-1) of length L is given by:

P(Y)= $\sum_{X} P(Y \mid X)P(X)$ where the sum includes all possible hidden node sequences for X=x(0),x(1),…,x(L-1). As the number of variables increases substantially, the use of a dynamic programming algorithm, the forward algorithm is deployed.

2.2.3.1.2 Common problems with Hidden Markov Models

If the parameters of the model are given, the probability of a particular output sequence is normally solved by the forward algorithm to accommodate large numbers of variables.

If the parameters of the model and the generated output sequence are given, the possible sequence of states hidden is found using the Viterbi algorithm.

If the output sequence is given, and the possible set of state transitions and output probabilities are normally found by training the parameters of the HMM based on a dataset of sequences. This is possible using the Baum-Welch algorithm.

2.2.3.1.3 Example to illustrate the use of HMM

Assuming that we would like find out over the telephone what a particular person who lives far away has done in a day. It is given that there are three activities that he is interested in; walking his dog, shopping for groceries and cleaning his house. Each of these activities are dependent only on the weather on that particular day. Although there is no dataset on the weather information, a probability sequence could be determined from the details of what he has done each day.

Two possible states of the weather are then 'Rainy' and 'Sunny', although they are not observable, i.e. they are hidden. The activities of 'walking', 'shopping' and 'cleaning' are the weather-dependent activities which are 'observable'. Hence the Hidden Markov Model could be used to simulate this scenario.

A pseudo-code implementation of it could be:

```
States = ('Rainy', 'Sunny')

Observables = ('walk', 'shop','clean')

Start probability = ('Rainy':0.6, 'Sunny':0.4)

Transition probability : {
'Rainy':{'Rainy': 0.7, 'Sunny':0.3},
'Sunny': {'Rainy': 0.4, 'Sunny': 0.6},}
Emission_probability = {
'Rainy' : {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},
'Sunny' : {'walk': 0.6, 'shop': 0.3, 'clean': 0.1},}
```

Assuming the person lives in an area where it rains often, the start probability gives a heavier weight on it being rainy. The transition probability is the Markov chain of the change in weather. From the example, there is a 40% chance that the tomorrow would be rainy if today is sunny. The emission probability represents the likelihood of a certain activity being performed. If it is rainy, there is only a 40% chance he would be shopping for groceries. If it is sunny, it is most probable he would be walking his dog at 60% than cleaning the house.

2.2.3.1.4 The use of HMM in statistical speech recognition in a noisy channel

In many modern techniques of speech recognition systems, the system has to sort out the most possible word sequence $\tilde{W}$ from possible word sequences W* in an acoustic signal A.

$$\tilde{W} = \arg\max_{W \in W^*} \Pr(W \mid A)$$

Rewriting the above using Bayes' rule;

$$\tilde{W} = \arg\max_{W \in W^*} \frac{\Pr(A \mid W)\Pr(W)}{\Pr(A)}$$

Since the acoustic signal is common no matter which word sequence is selected, thus the equation is simplified

$$\tilde{W} = \arg\max_{W \in W^*} \Pr(A \mid W)\Pr(W)$$

The term Pr(A | W) is widely known as the acoustic model while the term Pr(W) is known as the language model.

Speech signals could be represented as piece-wise stationary signals or short-time stationary signals. In this project the Analogue-to-Digital converter reads every 10ms, thus, any signal which is being input could be viewed as a stationary process. This is one of the reason speech could be represented by a Markov model of stochastic processes known as states.

Speech recognition systems based on Hidden Markov Models could also be trained easily and fast. A simple approach to the model would be to output n-dimensional real-valued vectors every 10 ms. The vectors consist of cepstral coefficients, obtained through Fourier transforming a short speech window and decorrelating the spectrum of the speech window using a cosine transform and selecting first coefficients. A mixture of diagonal covariance Gaussians would be present in each state, hence giving a probability for each observed vector. Each word or phenome will thus have a different output distribution. As a result, a Hidden Markov model of a sequence of words would have been strung from individually trained hidden Markov models for each phenome or word.

Chapter 3
The Jennic Microcontroller and Microphone Preamplifier

3.1 About the JN5121 microcontroller

The JN5121 microcontroller was developed to provide an integrated solution for applications based on the IEEE 802.15.4 standard, working in the 2.4-2.5GHz frequency band.

The JN5121 microcontroller consists of an on chip 32-bit RISC core, a 2.4 GHz IEEE802.15.4 transceiver, 64Kb of ROM and 96Kb of RAM, two application timers, three system timers, 4-input 12-bit 100ksps ADCs(Analogue-to-Digital Converters), 2 11-bit DACs(Digital-to-Analogue Converters, 2 UARTs, SPI port with 5 selects, 2 wire serial interface and 21 GPIO(General Purpose Input/Output).

The transceiver is IEEE 802.15.4 standard compliant, has a 128-bit AES security processor, a MAC accelerator with packet formatting, CRCs, address checks, auto-acknowledgements, timers, an integrated power management system and

low power oscillator for sleep mode which runs at below 5uA, has a receiver sensitivity of -93dBm and a transmit power of 1dBm.

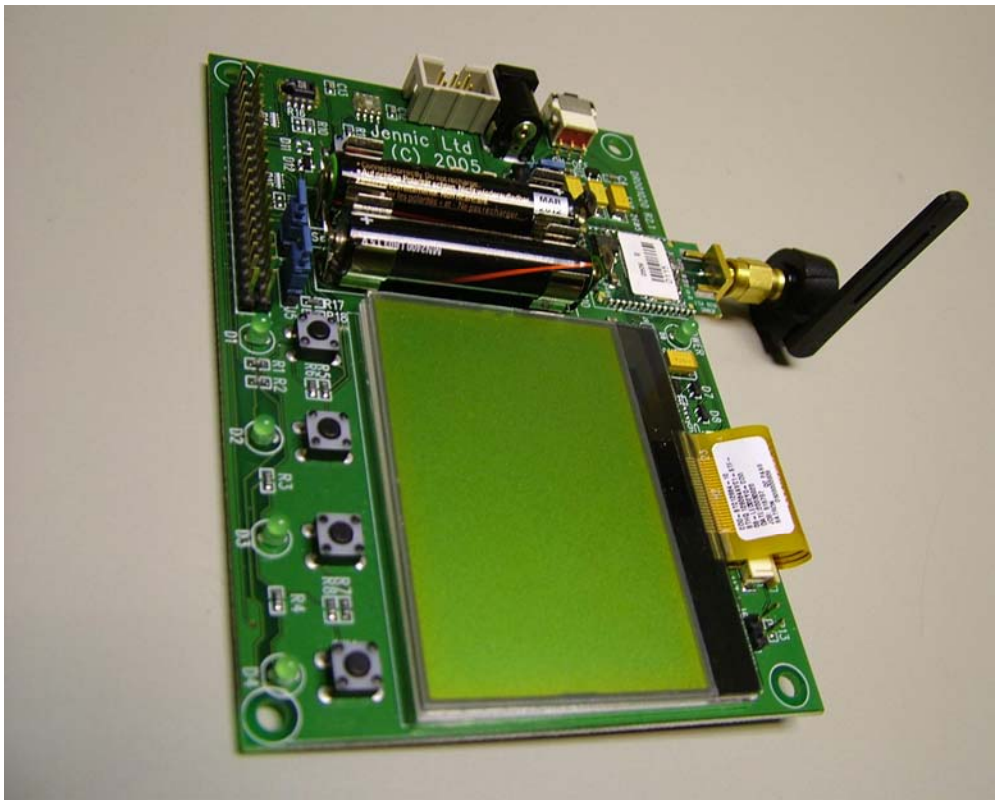

Fig 3.1 Photo of the Full Function Device Controller of the JN5121 evaluation kit

Jennic, the developer of the JN5121 microcontroller, has provided a set of library functions which control the transceiver and peripherals of the JN5121. Coupled with an Application Programming Interface, it has simplified the programming complexities and is done on the C language platform and debugged using the JN5 series software developer kit. These software libraries, Application Programming Interface and software development kit were provided in the CD that came with the evaluation board kit. Descriptions of how to program these were documented in the Hardware Peripheral Library and examples of the code usage were demonstrated in the JN5121 microcontroller manual.

3.2 About the Jennic Evaluation Kit (JN51210MO00x Module)

The Jennic Evaluation Kit consist of two types of boards, designated the Controller and Endpoint. The Controller Board has a LCD panel, 4 buttons and 4 LEDs while the Endpoint has two buttons and two LEDs. In all other aspects the two boards have the same capabilities. Both the boards contain the JN5121 microcontroller.

The Controller Board contains a 128x64 pixel LCD display. Using the provided application software, access to this LCD panel is achieved using a set of configured library functions.

Both the Controller and Endpoint Boards contain a RS-232 port which is used to provide communications with the JN5121 as well as programming the JN5121-MO00x module flash memory (1Mbit).

The boards also have three sensors which measure light, temperature and humidity. Again, the use of a set of library functions will provide access to the necessary controls and status of each sensor.

3.3 The microphone preamplifier

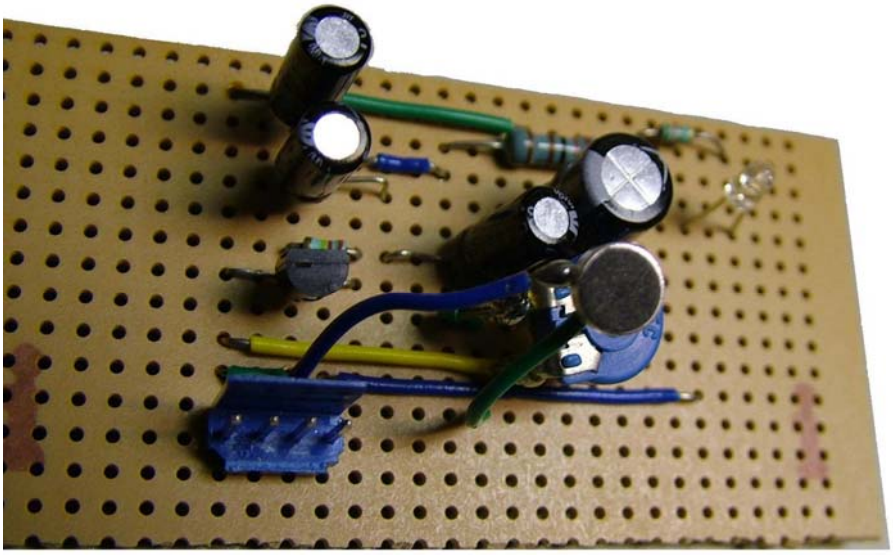

Fig 3.2 Photo of the microphone preamplifier

The microphone preamplifier circuit was designed by Tomi Engdahl (http://www.epanorama.net/circuits/micamp.html , 1996) and uses only one transistor. The amplification of the circuit was 35dB and has a flat frequency response from 20 Hz to 20 kHz. The diagram of the circuit is as follows:

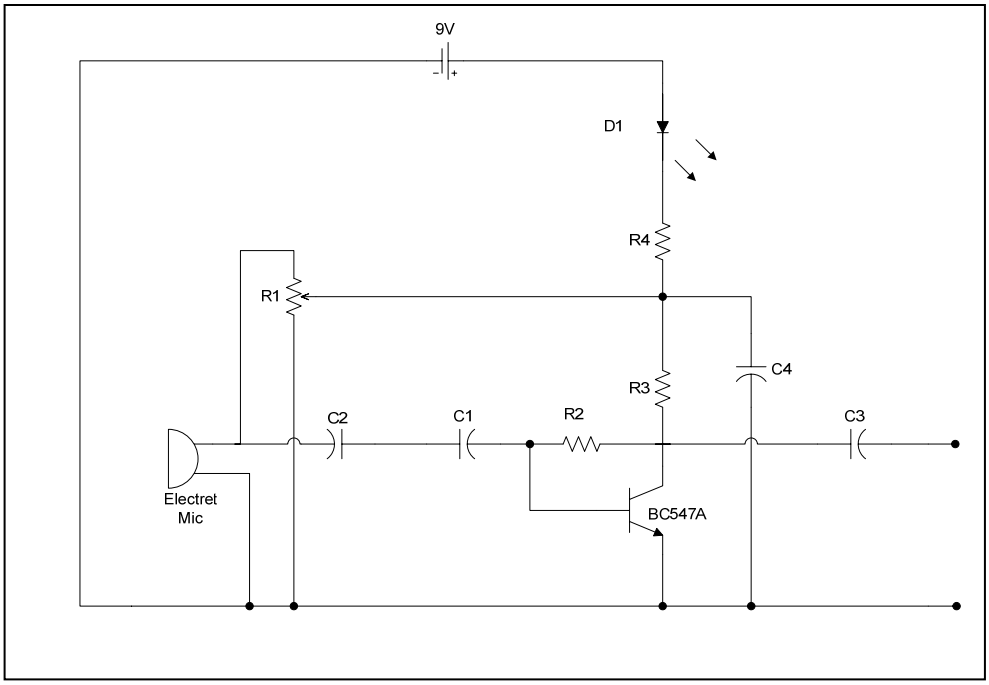

Fig 3.3 Diagram of pre-amplifier circuit

The components used in this circuit are listed below:

R1	1 MOhm potentiometer
R2	220 kOhm
R3	2.2 kOhm
R4	120 Ohm
C1 – C3	10 uF 16V electrolytic
C4	100 uF 16V electrolytic
D1	Red LED
Q1	BC547A

Resistor R4 and capacitor C5 filters out noise from the battery which powers the circuit. Capacitors C1 and C2 preempts the DC bias of the electret microphone input. The electret microphone input is connected to resistor R1 which feeds the current of about 1mA. The red LED is used to indicate the operation of the circuit when the battery is connected. There exists a voltage drop of about 1.8V across the LED.

Chapter 4
Configuration of the microcontroller

This chapter details the required files for the configuration of the microcontroller, the steps taken to build the code and load into the flash programmer and the various configurations of the peripherals on the microcontroller.

4.1 Required files

The files used to make the demonstrator are as follows:

In Developer/DemoApplication/Source:

wuart_c.c	Source for the demonstrator application that runs as a coordinator on an FFD
wuart_e.c	Source for the demonstrator application that runs as an endpoint on an RFD (can also run on an FFD)
DemoConfig.h	Configuration parameters for the demonstrator applications
AppQueueApi.c AppQueueApi.h	Application Queue API

SimpleFifo.c SimpleFifo.h	A simple FIFO manager used by the Application Queue API
uart.c, serialq.c, serial.c	used to support serial communications using the on-chip UART

In Developer/DemoApplication/Build:

wuart_c.mk	Make file for the coordinator wireless UART application
wuart_e.mk	Make file for the endpoint wireless UART application
Config.mk	Make file for features common to the other make files
ExBuild.ld	Link definition file, used to tell the linker where in memory to place the various code and data blocks
FlashHeader.S	Header that appears at the beginning of a flash image, containing 9 32-bit words that define the size and position of the application in memory,provide the entry points for the code, and specify the size and speed of the flash device itself.

In Developer/BoardAPI/Source or Developer/BoardAPI/Public:

LcdDriver.c LcdDriver.h	LCD panel driver
LcdFont.c LcdFont.h	Font used by LCD driver
Button.h	Button read functions
Math.h	Mathematical functions such as Abs and Sqrt

In Developer/HardwareAPI/Source or Developer/HardwareAPI/Public:

AppHardwareApi.c AppHardwareApi.h	Application hardware API

In Developer/Public:

jendefs.h	General type definitions used as a standard throughout Jennic code
mac_sap.h	Definitions for the structures, enumerations and types used for accessing the 802.15.4 stack through the Mac interface
nbo_pub.h	Network byte order utility, to translate words between little- and big-endian systems

In Developer/Stack/Public:

AppApi.h	Application API used to access the stack libraries. For the demonstrator, this is partly accessed through the Application Queue API.

In Developer/Stack/Library:

LibStack.a	Library, full stack capable of GTS, beaconing and other optional features of 802.15.4

4.2 Building The Code and Flash Programming

To make the file needed for the flash program to load, the following steps were taken:

1. Cygwin is started and is navigated to the folder Developer/Wuart/Build.
2. To generate the bin file, 'make –f wuart_c.mk' was typed which would utilize the wuart_c.mk file which will compile the list of required files for the coordinator. Similarly for the endpoint, 'make –f wuart_e.mk' was typed to create the bin file for the endpoint.

3. Typing 'make –f wuart_c.mk clean' followed by 'make –f wuart_c.mk' would execute a full rebuild for the coordinator and 'make –f wuart_e.mk clean' followed by 'make –f wuart_e.mk' for the endpoint.

4. The appropriate device; coordinator or endpoint device, is plugged into the serial programmer and switched off.

5. The flash programmer was started and navigated to the folder 'Developer/Wuart/Build' and linked to the file 'wuart_c.bin' for the coordinator or 'wuart_e.bin' for the endpoint.

6. The device is switched on and would automatically be set to program mode.

7. Pressing the 'Start' button will execute the flash program function.

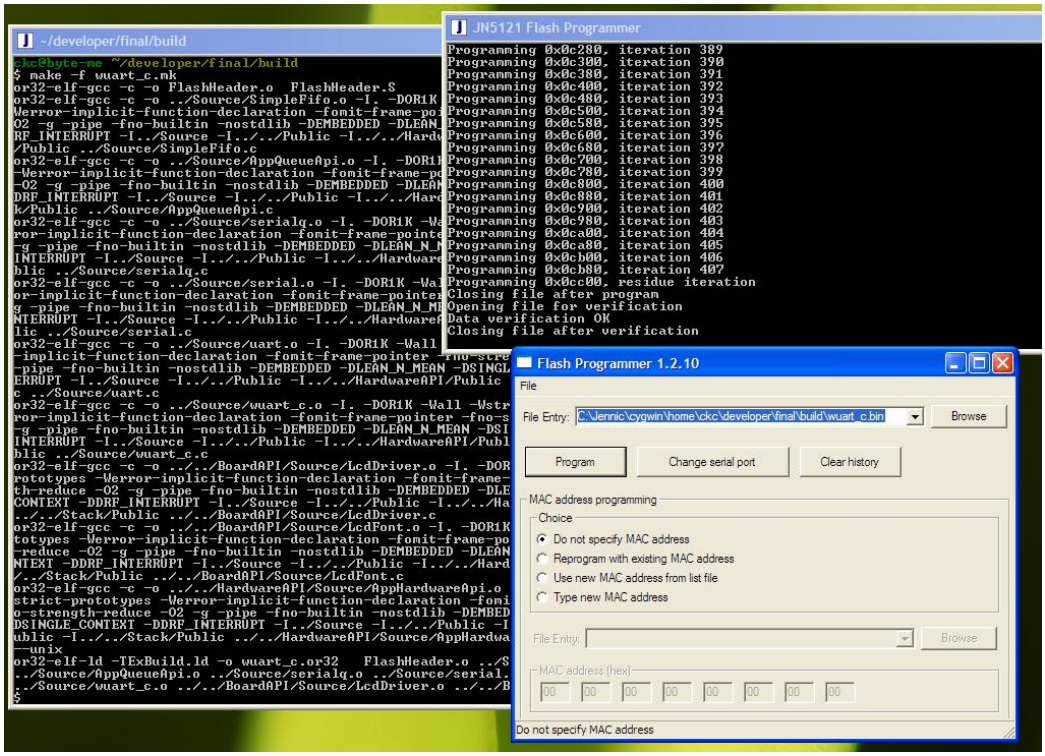

Fig 4.1 Photo of Cygwin and Flash Programmer

4.3 Function descriptions

Both the coordinator and endpoint functions are similar in their functions and hence the descriptions for the various functions are described here.
The following are descriptions of the various functions used in the program:

4.3.1 AppColdStart

The main entry point for the program; called when the ROM resident boot loader has been loaded. It would initialise the various variables needed such as arrays, temporary counters, pointers to arrays and call functions to initialise peripherals such as the Digital-to-Analogue Converter and Timer 1 set to interrupt every 10 ms. It would run in a continuous while loop which will process any button pressed.

4.3.2 AppWarmStart

This would be the main entry point after a warm start, i.e. a restart of the CPU from a sleep mode while the RAM contents are still retained. However this mode is not used throughout the program and acts as a fail safe mode.

4.3.3 vAdcInit

Sets and initialises the Analogue-to-Digital Converter 1 by configuring the *ApConfigure* and *AdcEnable* functions.

4.3.4 vValtoDec

Converts an 8-bit value to a string of textual decimal representation, which would be used to display text on the LCD.

Chapter 5
Implementation and Results

This chapter details the implementation of the project and the outcomes of it. Due to the size of RAM available (90k) in the memory of the microcontroller, the objective was to create a simple form of speech recognition within the limits of the microcontroller as well as within the time constraints of the project. The ADC conversion and the algorithm to match signals should be fast and accurate.

5.1 Outline of program

Given the many peripherals within the JN5121 evaluation kit and the Application Programming Interface that is resident within the microcontroller, the project utilised various peripherals as detailed further on. The program described in this chapter were mostly performed in the controller board as the use of the LCD proved helpful in ascertaining the results of the implementations.

The following flowchart gives a simple description of the main loop of the program in the coordinator:

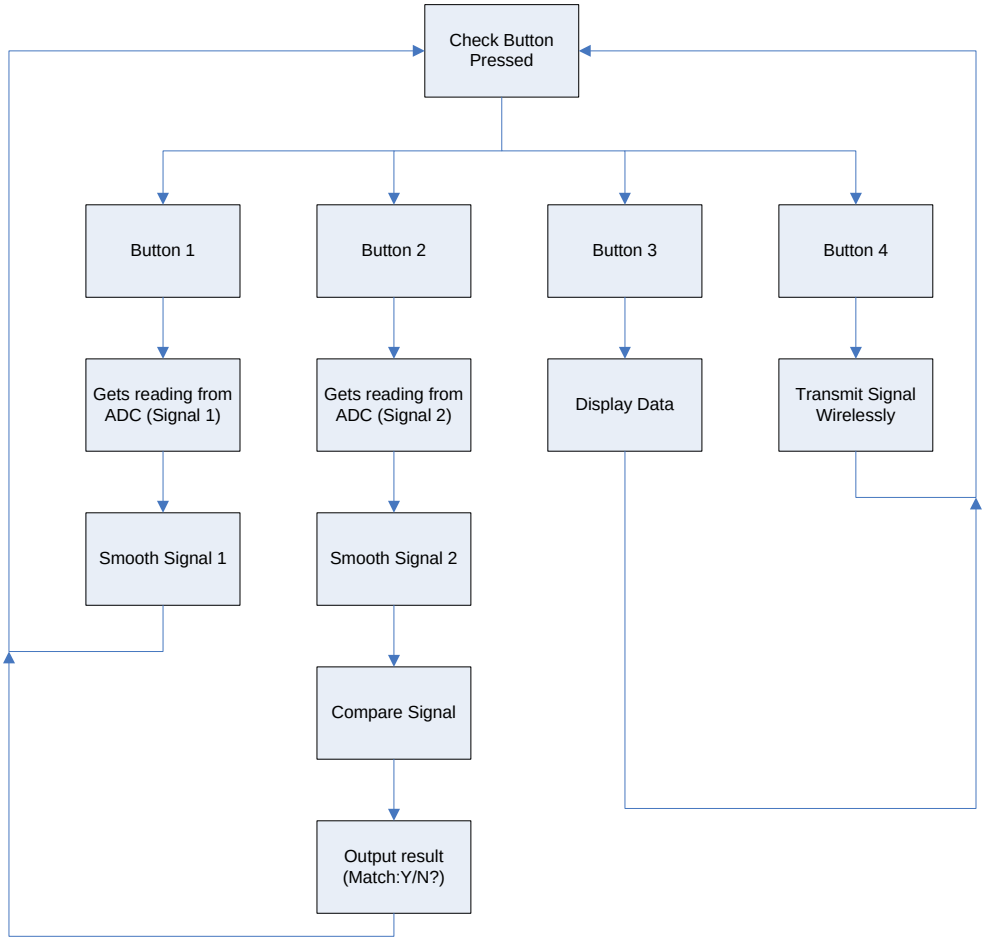

Fig 5.1 The main loop of the program in the coordinator

The main loops runs in a continuous while function that will check if the key of any one of the four buttons have been pressed.

The functions of the button input modules are defined in Button.h. Button 1 and Button 2 are used to input the signals for the voice control part of the project, Button 3 is used to display the saved data in the memory on the LCD panel and

Button 4 is used to send a signal wirelessly upon a successful matching of signals.

5.2 Configuration of the ADC

The project utilised Analogue to Digital input (ADC1) which is accessed on the expansion connector on pin 34. The ADC uses a successive approximation design in order to achieve a high accuracy conversion required in wireless sensor network applications. The input range of the ADC was set between 0V to the reference voltage. The reference voltage is taken from the internal voltage reference. The ADC has programmable clock periods to allow a trade-off between conversion speed and resolution with the full 12-bit resolution achieved with the 250kHz clock rate. The input clock to the ADC is the internal 16Mhz clock and is divided down to either 2Mhz, 1Mhz, 500kHz or 250kHz with a programmable divider. During an ADC conversion, the selected input channel (ADC1) is sampled for a fixed period and held.

The ADC clock and sampling period are set with the *vAHI_ApConfigure()* command. The declaration of the *vAHI_ApConfigure() command is as follows:-*

```
vAHI_ApConfigure ( E_AHI_AP_INT_DISABLE,
          E_AHI_AP_SAMPLE_2,
          E_AHI_AP_CLOCKDIV_500KHZ,
          E_AHI_AP_INTREF);
```

The boolean *E_AHI_AP_INT_DISABLE* disables the interrupt after a conversion is completed; unsigned 8-bit integer *E_AHI_AP_SAMPLE_2* selects a division of 2 of the internal clock of 16MHz, unsigned 8-bit integer *E_AHI_AP_CLOCKDIV_500KHZ* sets the clock divide ratio to 500kHz, and the boolean *E_AHI_AP_INTREF* sets the reference voltage to an internal reference voltage.

The *vAHI_AdcEnable()* function was configured as follows:

```
/* configure & enable ADC1 */
    vAHI_AdcEnable (E_AHI_ADC_CONVERT_ENABLE,
           E_AHI_AP_GAIN_2, //set input range 0V - Vref
           E_AHI_ADC_SRC_ADC_1);
```

This sets the ADC to enable continuous conversion, with a voltage ref of between 0V to Vref (internal reference voltage as set by the *ApConfigure* function) and sets the ADC 1 to be active.

5.3 Configuration of Timer 0

Timer 0 is configured by the following parameters and functions;

```
vAHI_TimerEnable3Param(E_AHI_TIMER_0, 32, TRUE);
vAHI_TimerClockSelect(E_AHI_TIMER_0, FALSE, TRUE);
vAHI_TimerStartRepeat(E_AHI_TIMER_0, 30, 60);
```

TimerEnable3Param enables Timer 0 with a prescale value of 32 and the boolean True enables interrupts from the timer when the output goes high. Timer 0 uses DIO 11-13 of the microcontroller. DIO 11 is the clock/gate input, DIO 12 the capture input and DIO 13 the PWM output.

TimerClockSelect configures Timer 0 to the internal 16 MHz clock and gates the output pin when the gate input is high.

TimerStartRepeat configures Timer 0 that sets 30 clock periods after the timer starts before the output goes high and 60 clock periods before the output goes low. The process repeats until it is stopped, since the interrupt is enabled at TimerEnable3Param, it is triggered at the low-high transition and again at the high-low transition.

The time period of one whole cycle for Timer 0 is thus :

$$\text{Time Period} = \underbrace{\frac{1}{16MHz}}_{\text{(internal clock speed)}} \quad \underbrace{(32)}_{\text{(prescale value)}} \quad \underbrace{(60)}_{\text{(clock cycle)}} = 120us$$

5.4 Sampling of Voice Signal

The use of Timer 0 and the Analogue-to-Digital Converter Channel 1 enables us to sample the voice signals over a specific time. A normal human speech voice is usually between 3kHz to 4kHz. According to the Nyquist Theorem, the sampling rate should be twice the bandwidth of the voice signal in order to achieve a good resolution of the sampled signal. Hence the sampling rate should be at least 8kHz. As was previously described the time period of a clock cycle of Timer 0 is 120 us. If the ADC was to take a reading every interrupt from Timer 0, the sampling rate would thus be:

$$\text{Sampling rate} = \frac{1}{120us} = 8.33kHz$$

Hence the Nyquist theorem is satisfied since the sampling rate is more than 8kHz. Given the limited amount of memory to work with in the microcontroller, the project suffers from a limited sampling time. If an array of 2048 samples were to be collected, the sampling time would be ;

Sampling time of 2048 samples = 2048 x 120us = 0.246 sec

Hence during testing of the algorithm, the sampling rate was lowered to enable a longer speech signal being sampled.

The usage of the external RAM in the evaluation board was considered as there was 1MB of Flash EEPROM available. However the slow write speed hindered the fast conversion needed and as such the idea was discarded.

5.5 Smoothing of voice signal

Two methods were investigated and implemented to smooth the incoming voice signals. The signals were first converted to digital values using the Analogue-to-Digital Converter and saved to an array of memory in the RAM memory block in the microcontroller. Three arrays made up of 2048 blocks of 32-bit unsigned integers were allocated to store the two incoming signals to be matched and a temporary array was created to store the arithmetic algorithms to be performed on the incoming signals.

The first smoothing function was to take an average of five blocks of memory at any time by summing the total values of the sampled signals and dividing the sum by the total number of samples being considered:

```
while(counter<windowsize)
{
   if(counter<=2)
   {
      temp[counter]=(sample[counter]+sample[counter+1]+
            sample[counter+2])/3;
   }
   if(counter>2)
   {
      temp[counter]=(sample[counter-2]+sample[counter-1]
            +sample[counter]+sample[counter+1]+sample[counter+2]
      )/5;
   }
   if(counter>=(newwindowsize-2))
   {
      temp[counter]=(sample[counter]+sample[counter-1]
            +sample[counter-2])/3;
   }
   counter++;
}
```

In effect, a temporary array is used to store the averaged values of five sampled signals. The central signal is summed with two previous signals and two forward signals, producing a rough smoothing effect.

The second method investigated was the Gaussian smoothing function. The Gaussian function is defined as

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ is the mean and σ the standard deviation. The Gaussian function is thus a probability function of the normal distribution of the set of data. The function is used in a one-dimensional array with the mean assumed to be 0 so as to have the function centered about the line x=0.

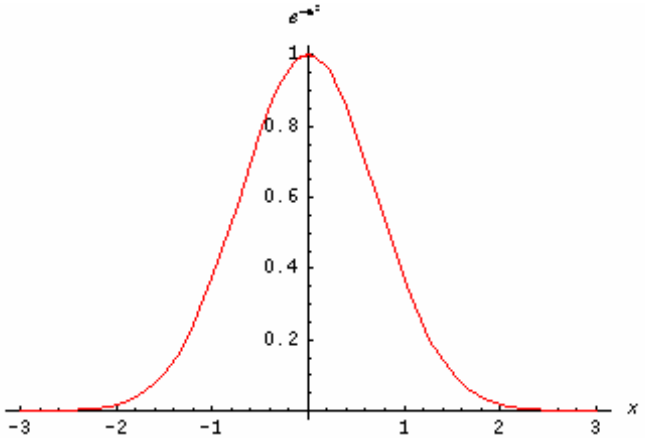

Fig 5.2 Gaussian function with a zero mean.

The function is to be multiplied with blocks of 5 samples to give a smoothing function. The values of the Gaussian function is shifted three decimal places to

avoid working with floating point numbers as the values were to be worked in unsigned integers.

The code for the implementation of the Gaussian smoothing is as follows:

```
for(counter=0;counter<windowsize;counter++)
{
      if(counter<=2)
      {
      temp[counter]=(window[counter]*1000+window[counter+1]*
      422+window[counter+2]*14)/1436;
      }
      if(counter>2)
      {
            temp[counter]=(window[counter-2]*14+window[counter-
            1]
            *422+window[counter]*1000+window[counter+1]*422+win
            dow
            [counter+2]*14)/1872;
      }
      if(counter>=(windowsize-2))
      {
            temp[counter]=(window[counter]*1000+window[counter-
            1]
            *422+window[counter-2]*14)/1436;
      }
```

This smoothing function gives a gentler smoothing function and preserves the edges better than the simpler averaging method. The degree of the smoothing is determined by the standard deviation being used.

5.6 Comparison of signals

Having smoothed both incoming signals, the signals are compared by taking the absolute sum of the subtraction of the signals. This is in effect calculating the distance between the two signals in Euclidean space and the difference between the two signals would be reflected in the sum total of the subtraction.

```
for(i=0; i<windowsize; i++)
{
      *tmp_ptr = abs((*sam_ptr) - (*w_ptr));
      sam_ptr++;
      w_ptr++;
      tmp_ptr++;
}
```

5.7 Display of data on LCD

A function was implemented so that when Button 3 on the controller board is pressed, the data saved on all three arrays of data, window, sample, and temp array. The window array was primarily used to store the first signal that would be matched. The sample array was primarily used to store the second converted signal. This could be updated constantly to correlate against the first saved signal. The temp array provides a temporary array for the calculation needed in performing the smoothing function. After the smoothing function is performed on both signals, the temp array would be used to store the absolute values of the subtraction of both signals; adding the sum of the total values in the temp array would give the distance of the two signals in Euclidean space.

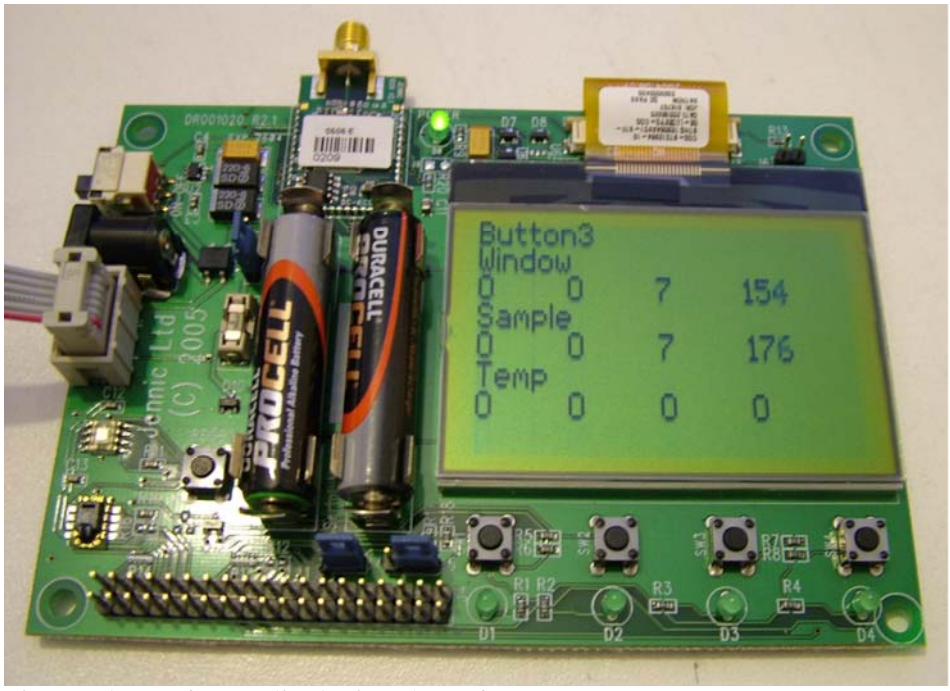

Fig 5.3 Photo of LCD displaying data of arrays

The JN5121 has a 128x64 pixels resolution LCD which could be used to display both text and graphics. The driver for the LCD makes use of a shadow of the content to be shown on the LCD. Information to be displayed are stored in the shadow memory and the *vLcdRefreshAll()* command updates the entire shadow memory to the LCD. The *vLcdRefreshAll()* command takes approximately 4.5 ms to execute, hence no real-time update of the incoming data of the Analogue-to-Digital conversion could be displayed at the sampling rate of 120us.

Font sizes for the display on the LCD are specified in 'Lcdfont.h'; characters are 8 pixels high and have varying width of up till 7 pixels. Functions for the display of text and graphics are defined in LcdDriver.h. The LCD was initialised using the *vLcdResetDefault* command which sets the bias and gain for the LCD panel with the default settings, giving a good contrast and clearing the LCD panel.

The display of stored data in the program was coded as below:

```
while(counter<windowsize)
{
      vLcdWriteText("Window -", 1 , 0);
      pu8Payload = (uint8)((*w_ptr) & 0xff);
      vValToDec(acString, pu8Payload, "  ");
      vLcdWriteText(acString, 2 , 90);
      pu8Payload = (uint8)((*w_ptr >> 8) & 0xff );
      vValToDec(acString, pu8Payload, "  ");
      vLcdWriteText(acString, 2 , 60);
      pu8Payload = (uint8)((*w_ptr >> 16) & 0xff );
      vValToDec(acString, pu8Payload, "  ");
      vLcdWriteText(acString, 2 , 30);
      pu8Payload = (uint8)((*w_ptr >> 24) & 0xff );
      vValToDec(acString, pu8Payload, "  ");
      vLcdWriteText(acString, 2 , 0);
      vLcdWriteText("Sample -", 3 , 0);
      pu8Payload = (uint8)((*sam_ptr) & 0xff);
      vValToDec(acString, pu8Payload, "  ");
      vLcdWriteText(acString, 4 , 90);
      pu8Payload = (uint8)((*sam_ptr >> 8) & 0xff );
      vValToDec(acString, pu8Payload, "  ");
      vLcdWriteText(acString, 4 , 60);
      pu8Payload = (uint8)((*sam_ptr >> 16) & 0xff );
      vValToDec(acString, pu8Payload, "  ");
      vLcdWriteText(acString, 4 , 30);
      pu8Payload = (uint8)((*sam_ptr >> 24) & 0xff );
      vValToDec(acString, pu8Payload, "  ");
      vLcdWriteText(acString, 4 , 0);
      vLcdWriteText("Temp -", 5 , 0);
      pu8Payload = (uint8)((*tmp_ptr) & 0xff);
      vValToDec(acString, pu8Payload, "  ");
      vLcdWriteText(acString, 6 , 90);
      pu8Payload = (uint8)((*tmp_ptr >> 8) & 0xff );
      vValToDec(acString, pu8Payload, "  ");
      vLcdWriteText(acString, 6 , 60);
      pu8Payload = (uint8)((*tmp_ptr >> 16) & 0xff );
      vValToDec(acString, pu8Payload, "  ");
      vLcdWriteText(acString, 6 , 30);
      pu8Payload = (uint8)((*tmp_ptr >> 24) & 0xff );
      vValToDec(acString, pu8Payload, "  ");
      vLcdWriteText(acString, 6 , 0);
      vLcdRefreshAll();
      for(i=0; i<55000; i++);
      counter++;
      w_ptr++;
      sam_ptr++;
      tmp_ptr++;
}
```

5.8 Results

The distance in the Euclidean space of the two sampled signals is to be measured using the following equation

$$\text{Euclidean distance between two signals} = \sum_{i=1}^{n} \sqrt{(p_i - q_i)^2}$$

where p_i is the second signal to be correlated with the first signal q_i and n is the length of the array, i.e. 2048 in this project.

5.8.1 Inherent System Noise

The resolution of the ADC is of maximum 12-bits. Without connecting any signals into the ADC port on pin 34 of the expansion port of the Controller Board, readouts were taken to check the system's inherent noise.

The following details the distance readings taken at different times when the pin to ADC port is not connected:

Reading #	Hex Value of Euclidean distance	Decimal Value of Euclidean Distance
1	0x00010822	67618
2	0x0000899C	35228
3	0x00006AA2	27298
4	0x00009425	37925
5	0x000097DC	38876
6	0x00006959	26969
7	0x0000AA19	43545

Fig 5.4 Euclidean distance with no inputs (stray values)

The length of the arrays were 2048, hence the first value of 67618, the average distance between two arrays is 67618/2048 = 33.02.

The readings of this value even without any sampled signals being fed into the system could be due to noise within the microcontroller.

5.8.2 Background Noise and Microphone Preamplifier Noise

The microphone preamplifier was connected to the ADC pin and again multiple measurements of the Euclidean distance between two signals were taken, without any voice input to measure the background noise and the microphone preamplifier noise, assuming that the noise from the microcontroller is negligible.

Reading #	Hex Value of Euclidean distance	Decimal Value of Euclidean Distance
1	0x0001862F	99887
2	0x0004718	18200
3	0x0002871	10353
4	0x0002394	9108
5	0x0002548	9544
6	0x000AC0C	44044

Fig 5.5 Euclidean distance with background noise

As we could see from the results, it is similar to the measured Euclidean distance of the microcontroller noise.

5.8.3 Speech Signal Sampling

A speech signal of the words 'Hello' was recorded and fed into the system. Using Matlab the frequency spectrum of the signal is generated:

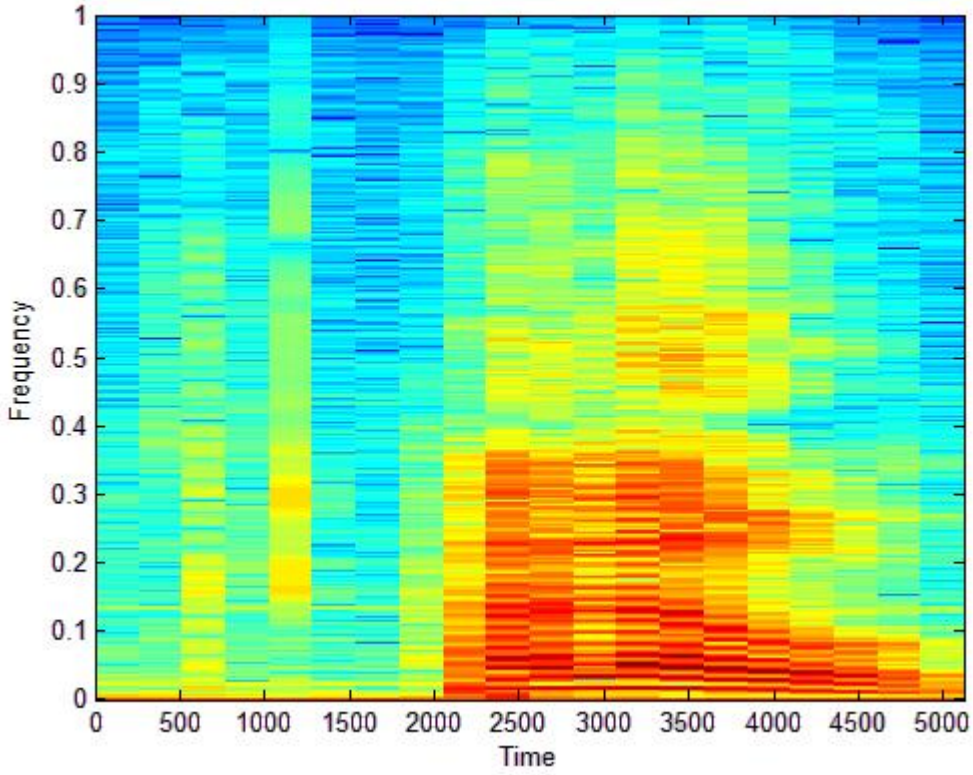

Fig 5.6 Spectrogram of the 'Hello' signal

The entire length of the signal is slightly over 5 seconds long. The spoken 'Hello' was intentionally heavily emphasized at the beginning; hence it could be seen from the frequency spectrum that it starts at around 2 seconds of the time frame.

The following is a collection of readings of the measured Euclidean distance between the stored signal and subsequent samplings:

Reading #	Hex Value of Euclidean distance	Decimal Value of Euclidean Distance
1	0x000321A2	205218
2	0x000ACB25	707365
3	0x0013A26A	1286762
4	0x0001B01E1	1769953
5	0x00043FED	278509
6	0x00058859	362585

Fig 5.7 Euclidean distance of the same speech signal sampled at different times

A filter process was applied to the program that would reduce the 'noise' effects on the calculation of the Euclidean distance.

The following list the process:

```
tmp_ptr=&temp[0];

for(i=0; i<windowsize; i++)
{
      if(*tmp_ptr<50)
      {
            *tmp_ptr = 0;
      }
      tmp_ptr++;
}
```

In effect if the absolute of the subtraction of the two arrays is less than a value of 50, the temp[array] value is reset to 0. This removes the effects of noise but will also affect the correlation of the signal. However the value of 50 is small compared to the maximum resolution of the ADC (2^12), hence it is assumed negligible.

Once the filter process was put into effect, subsequent measurements when no signal is fed gives zero Euclidean distance.

The 'Hello' waveform was again fed into the system and the following results were obtained:

Reading #	Hex Value of Euclidean distance	Decimal Value of Euclidean Distance
1	0x0001F357	127831
2	0x00016ADA	92890
3	0x0001A990	108944
4	0x000192CE	103118
5	0x000197A3	104355
6	0x00017F45	98117
7	0x0001FEFC	130812

Fig 5.8 Euclidean distance of the same speech signal sampled at different times with filter

The results show a more consistent reading and this better reflects the real Euclidean distance of the two signals.

5.9 WUART implementation

If the correlated signal was found to be matching, a signal was to be sent wirelessly either from the coordinator to the endpoint or from the endpoint to the coordinator depending on which board the correlation was done. Modifying the WUART implementation from Jennic, there was a successful receive and sent signals using the IEEE 802.15.4 standard. Using the Hyperterminal windows on a PC, the communication between the coordinator and endpoint could be simulated by the sending of data through the serial port and transmitted wirelessly across to the other device.

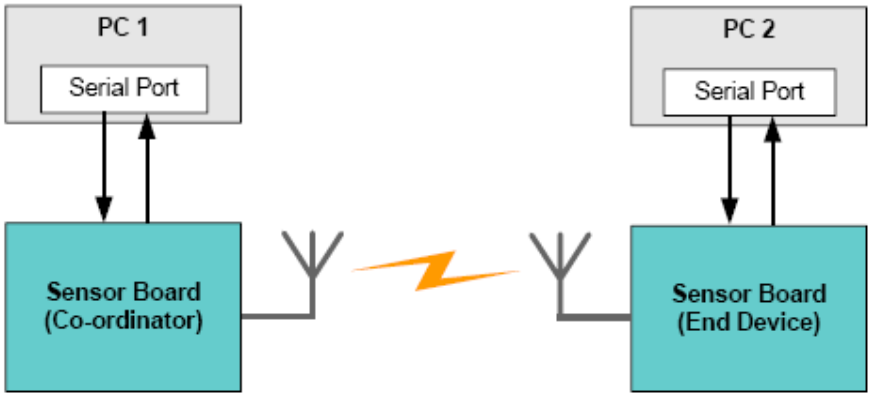

Fig 5.9 Wireless Serial Link (Diagram taken from *Application Note: JN5121 Wireless Microcontroller:Serial Cable Replacement using 802.15.4*, pg 4)

5.10 Costing

The Jennic JN5121 Evaluation Board was used on loan from Sheffield Hallam University. The cost used in this project was for the construction of the microphone preamplifier circuit and the USB-to-serial converter needed to communicate between the PC and the JN5121 microcontroller.

Most of the components were available from the Stores in Sheffield Hallam University. The list below shows the estimated cost of building the microphone preamplifier circuit:

Component	**Units**	**Cost**	**Sub Total**
Resistors	4	£0.15	£0.60
Capacitors	4	£0.20	£0.80
LED	1	£0.20	£0.60
BC547A	1	£0.10	£0.10
Electret microphone	1	£3.85	£3.85
1 MOhm varactor	1	£0.10	£0.10
		Total	£6.05

Fig 5.10 List of component costs

5.11 Discussion of Results

The effects of background noise and noise from the microcontroller were eliminated using a software filter. During the course of sampling the voice signals, measurements using the oscilloscope and signal generator were also used, and the following discussion of the results are both from the previous published results as well as results which were not able to be recorded in the report.

The ADC function in the microcontroller gives a maximum of a 12-bit resolution conversion of an analogue signal to a digital signal from 0V to Vref. Hence if a signal contains negative values, the ADC will only read a value of 0 and the sampling is inaccurate. Thus there is a need to set a offset bias voltage to compensate this problem by about 1V to enable the readings of negative values of a signal.

The amount of memory in the JN5121 is limited to 90k bytes of RAM, organised as 24x32-bit words. The use of three arrays, each the size of 2048 x 32-bit unsigned integers uses up 8192 bytes of RAM. When the size of arrays was increased to 8196, the program halted and was not able to run. This could be due to the memory allocation needed for the program itself which contains many header files.

Running the ADC at the speed of 8 kHz gives a total sampling time of only 0.246 seconds. This was sufficient for detecting short burst of sounds like clapping but insufficient for speech recognition.

From the various result tables of the Euclidean distance measurements of two signals, there were inconsistencies of the readings although the same signal is fed in both times. This is due to time-shifting as the input of the signals was not sampled at the same time frame. A windowing technique was deployed in order to window out only the speech signal and this would be correlated with another signal in order to minimise the time-shifting problem. However as the ADC was not able to convert negative values of the signal, hence the windowing function was not accurate.

The limited amount of memory also prevents the use of a proper correlation technique. Using a proper correlation technique would be better if the ADC was able to convert negative signals at a fast speed and a sizeable amount of temporary array available to store calculations as the correlation of two signals is time-invariant.

Chapter 6
Conclusion and Further Work

6.1 Further Work

There were limitations as to how much better a voice control system could be built using the IEEE 802.15.4 standard. With a maximum transfer rate of 250 kbps, transmission of a voice signal over the air was not possible as it would have required a wireless standard with a higher transfer rate such as the IEEE 802.11g standard. Capturing the voice signal with a sampling rate of about 8kHz was feasible using the built-in ADC and library functions, however the WUART(Wireless UART) function has a latency of about 10ms, therefore the sampled voice signals could not be transferred in real-time from the RFD to the FFD.

There are a few areas of improvement that would have a better result in achieving a voice recognition system within the limits of the IEEE802.15.4 standard. The IEEE802.15.4 is suitable for lower transfer rate networks and thus is suitable for home automation with capabilities such as controlling the lighting or central heating. It requires much more lower power consumption and have the capabilities of mesh networking enabling hundreds of different Reduced

Function Devices to be connected to a single coordinator. The number of RFDs available were limited only to four units and thus further work in building a larger network of inter-communicating devices would require more RFD units. This would enable routing algorithms to be tested and the maximum distance between a FFD and a RFD to be analysed.

While sampling the voice signals using the built microphone preamplifier circuit, there was inherent background noise picked up. Achieving a better signal-to-noise ratio would require using a band pass filter circuit to eliminate the white noise.

Continuous development on the Jennic Evaluation Board by Jennic Ltd has led to newer library functions and the constant updates on their website reflect this. Some of these new functions were not easily implemented on the current Evaluation Board JN5121 and a newer Evaluation Board could result in a better sampling of the voice signal using a regulating function before each reading from the ADC.

In order to circumvent the problem of the ADC not being able to sample negative values, a positive bias voltage offset needs to be set at the pre-amplifier circuit which would enable it to read negative values.

To reduce the measured Euclidean distance between the two signals, a multi-resolution process could be performed on the sampled signals.
Changing the standard deviation value in the Gaussian function to be multiplied across the sampled signals would affect the smoothing function. More analysis and sampling would need to be done in order to find the optimal value of the standard deviation.

6.2 Conclusion

The main objectives of analysing the IEEE802.15.4 standard and constructing a remote voice control system using the Jennic JN5121 Evaluation Kit have been achieved. The system was able to convert analogue to digital signals at a fast speed and the algorithms for correlating signals were simple and fast.
System limitations have been identified and recommended solutions have been suggested.

Overall, the system could be implemented in a non-critical home-based application but needs further work in any critical applications.

Bibliography

[1] IEEE (2006), IEEE Std 802.15.4; Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs) – 2003. [online]. Last accessed 5 September 2006 at URL: http://standards.ieee.org/getieee802/download/ 802.15.4-2003.pdf

[2] Hidden Markov Model [online]. Last accessed 5 September 2006 at URL: http://en.wikipedia.org/wiki/Hidden_Markov_ model

[3] Greg's guide to Hungarian Notation [online]. Last accessed 5 September 2006 at URL: http://www.gregleg.com/oldHome/ hungarian.html

[4] Simple microphone preamplifier [online]. Last accessed 5 September 2006 at URL: http://www.epanorama.net/circuits/micamp.html

[5] Schiller, Jochen, Mobile Communications 2003, Second Edition, Pearson Education Limited. ISBN 0-321-12381-6

[6] H.M. Deitel, P.J. Deitel, Small C++ How To Program, 2005, Fifth Edition, Pearson Education, Inc. ISBN 0-13-185758-4

Acronyms and Abbreviations

ADC	Analogue to Digital Converter
AES	Advanced Encryption Standard
API	Application Programming Interface
CPU	Central Processing Unit
DSSS	Direct Sequence Spread Spectrum
FCS	Frame Check Sequence
FFD	Full Function Device
FIFO	First-In, First-Out queue
FHSS	Frequency Hopping Spread Spectrum
GSFK	Gaussian Frequency Shift Keying
HMM	Hidden Markov Model
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LR-WPAN	Low-Rate Wireless Personal Area Network
MAC	Media Access Control
MCPS	MAC common part sublayer
MCPS-SAP	MAC common part sublayer-service access point
MFR	MAC footer
MHR	MAC header
MIC	Message integrity code
MISO	Master-In Slave-Out
MLME	MAC sublayer management entity
MLME-SAP	MAC sublayer management entity-service access point
MSB	Most significant bit
MSC	Message sequence chart
MPDU	MAC protocol data unit
MSDU	MAC service data unit
O-QPSK	Offset Quadrature Phase Shift Keying
PAN	Personal Area Network
PANPC	Personal Area Network Computer
PD-SAP	PHY data service access point
PDU	Protocol Data Unit
PER	Packet error rate
PHY	Physical layer
PSU	Power Supply Unit
QPSK	Quadrature Phase Shift Keying
RAM	Random Access Memory
RFD	Reduced Function Device
RISC	Reduced Instruction Set Computer
SPI	Serial Peripheral Interface
SSCS	Service Specific Convergence Sublayer
UART	Universal Asynchronous Receive Transmit
WUART	Wireless Universal Asynchronous Receive Transmit

Appendix A: Source Code of the Coordinator

```
/****************************************************************************
*
* MODULE:             wuart_c.c
*
* COMPONENT:          $RCSfile: $
*
* VERSION:            $Name: $
*
* REVISION:           $Revision: $
*
* DATED:              $Date: $
*
* STATUS:             $State: $
*
* AUTHOR:             Ian Morris
*
* DESCRIPTION
*
* CHANGE HISTORY:
*
* $Log: $
*
*
* LAST MODIFIED BY:   $Author:  $
*                     $Modtime: $
*
*
****************************************************************************
*
*  (c) Copyright 2000 JENNIC Ltd
*
****************************************************************************/

/****************************************************************************/
/***        Include files                                                 ***/
/****************************************************************************/

#include <jendefs.h>
#include <AppHardwareApi.h>
#include <AppQueueApi.h>
#include <mac_sap.h>
#include <nbo_pub.h>
#include <string.h>
#include <stdlib.h>

#include <LcdDriver.h>
#include "Button.h"
//#include <math.h>
#include "serialq.h"
#include "uart.h"
#include "serial.h"

/****************************************************************************/
/***        Macro Definitions                                             ***/
/****************************************************************************/
#define LED_OUTPUTS_MASK          0x0000C000UL
#define LED1_MASK                 0x00008000UL
#define LED2_MASK                 0x00004000UL

/* Network parameters */
#define PAN_ID                    0x0401U
#define COORD_ADDR                0x0502U

/* Wireless UART device data */
#define MAX_UART_NODES            1
#define UART_NODE_ADDR_BASE       0x1000U
#define MAX_DATA_PER_FRAME        64
#define UART_CONT_EXT_ADDR_LO     0x00001010UL
#define UART_CONT_EXT_ADDR_HI     0x00000101UL
#define UART_NODE_EXT_ADDR_LO     0x10101010UL
#define UART_NODE_EXT_ADDR_HI     0x01010101UL
```

```
/****************************************************************************/
/***        Type Definitions                                              ***/
/****************************************************************************/

/* Button values */
typedef enum
{
    E_KEY_0 = BUTTON_0_MASK,
              E_KEY_1 = BUTTON_1_MASK,
              E_KEY_2 = BUTTON_2_MASK,
              E_KEY_3 = BUTTON_3_MASK,
              E_KEYS_1_AND_3 = (BUTTON_1_MASK | BUTTON_3_MASK),
              E_KEYS_0_AND_3 = (BUTTON_0_MASK | BUTTON_3_MASK)
} teKeyValues;

/* System states with respect to screen display being shown */
typedef enum
{
    E_STATE_INIT,
    E_STATE_START_COORDINATOR,
    E_STATE_RUNNING_UART_APP

} teState;

/* Used to track an association between extended address and short address */
typedef struct
{
    uint32 u32ExtAddrLo;
    uint32 u32ExtAddrHi;
    uint16 u16ShortAddr;
} tsAssocNodes;


/* All application data with scope within the entire file is kept here,
   including all stored node data, GUI settings and current state */
typedef struct
{
    struct
        {
            tsAssocNodes asAssocNodes[MAX_UART_NODES];
            uint8        u8AssociatedNodes;
        } sNode;

    struct
    {
        teState eState;
        uint8   u8Channel;
    } sSystem;
} tsCoordData;

/****************************************************************************/
/***        Local Function Prototypes                                     ***/
/****************************************************************************/

PRIVATE void vValToDec(char *pcOutString, uint8 u8Value, char *pcLabel);
PRIVATE void vStringCopy(char *pcFrom,char *pcTo);
PRIVATE void vAdcInit(void);


/****************************************************************************/
/***        Exported Variables                                            ***/
/****************************************************************************/

/****************************************************************************/
/***        Local Variables                                               ***/
/****************************************************************************/

PRIVATE tsCoordData sCoordData;
uint8 u8TxFrameHandle = 0;
uint8 u8RxFrameHandle = 0;
uint32 windowsize=2048;


/****************************************************************************/
```

```
/***        Exported Functions                                            ***/
/*****************************************************************************/

/*****************************************************************************/
/***        Local Functions                                               ***/
/*****************************************************************************/
PRIVATE void vWUART_Init(void);
PRIVATE void vWUART_TxData(void);
PRIVATE void vProcessEventQueues(void);
PRIVATE bool_t bStartCoordinator(void);
PRIVATE void vHandleNodeAssociation(MAC_MlmeDcfmInd_s *psMlmeInd);
PRIVATE void vProcessIncomingMlme(MAC_MlmeDcfmInd_s *psMlmeInd);
PRIVATE void vProcessIncomingData(MAC_McpsDcfmInd_s *psMcpsInd);
PRIVATE void vProcessIncomingHwEvent(AppQApiHwInd_s *psAHI_Ind);

/*****************************************************************************
*
* NAME: AppColdStart
*
* DESCRIPTION:
*
* PARAMETERS:      Name            RW  Usage
* None.
*
* RETURNS:
* None.
*
* NOTES:
* Entry point for a power on reset or wake from sleep mode.
*****************************************************************************/
PUBLIC void AppColdStart(void)
{
       vAdcInit();


       //vAHI_TimerStop(E_AHI_TIMER_1);
       //starts timer
       vAHI_TimerEnable3Param(E_AHI_TIMER_0, 64, TRUE);
       vAHI_TimerClockSelect(E_AHI_TIMER_0, FALSE, TRUE);
       vAHI_TimerStartRepeat(E_AHI_TIMER_0, 50, 100);
       //initialise LCD
       vLcdResetDefault(); // LCD Default settings
    vLcdClear();        // Clear Shadow memory
    vLcdRefreshAll();   // Copy Shadowe mem to Lcd
       int lcdrowpos = 0;


       uint32 window[windowsize];
       uint32 sample[windowsize];
       uint32 temp[windowsize];

       long int sum=0;
       long int sum_int = 0;

       uint32 *w_ptr;
       uint32 *sam_ptr;
       uint32 *tmp_ptr;
       char acString[8];

       int i =0;

       uint32 counter=0;

       uint8 pu8LcdLoad;
       uint8 u8KeysDown=0;
    uint8 u8NewKeysDown;



       w_ptr=&window[0];
       *w_ptr = counter;
       sam_ptr=&sample[windowsize];
       *sam_ptr = counter;
```

```
tmp_ptr=&window[counter];

for(counter=0;counter<(windowsize);counter++)
{
	window[counter] = 0;
}
for(counter=0;counter<(windowsize);counter++)
{
	sample[counter] = 0;
}
for(counter=0;counter<(windowsize);counter++)
{
	temp[counter] = 0;
}
vLcdWriteText("Init", lcdrowpos , 0);
vLcdRefreshAll();
while(1)
{
	/* Process key press */
	u8NewKeysDown = u8ButtonReadFfd();

	if (u8NewKeysDown != 0)
	{
		if ((u8NewKeysDown | u8KeysDown) != u8KeysDown)
		{
			u8KeysDown |= u8NewKeysDown;
			/* Key presses depend on mode */
			switch(u8NewKeysDown)
			{
			case E_KEY_0:

			//start sampling from ADC. take 'windowsize' samples.

			 vLcdClear();
			 w_ptr=&window[0];
			 *w_ptr = counter;
			 counter = 0;
			 vLcdWriteText("Button1", lcdrowpos , 0);
			 vLcdRefreshAll();

			 vLcdWriteText("StartSampling", 1 , 0);
			 vLcdRefreshAll();

			 w_ptr=&window[0];
			 counter=0;



			 while(counter<windowsize)
			 {
				// If this is an event from timer 0
				if (bAHI_TimerFired(E_AHI_TIMER_0))
				{
					vAHI_AdcStartSample();
					//read from ADC
					while (bAHI_AdcPoll());
					*w_ptr=u16AHI_AdcRead();
					w_ptr++;
					counter++;
				}
			 }
			 vLcdWriteText("SamplingDone", 7 , 0);
			 vLcdRefreshAll();

			 w_ptr=&window[0];
			 tmp_ptr=&temp[0];
			 counter = 0;

			 for(counter=0;counter<windowsize;counter++)
			 {
				if(counter<=2)
				{
```

```
	temp[counter]=(window[counter]*1000+window[counter+1]*422+window[counter+2]*14)/14
36;
                                        }
                                        if(counter>2)
                                        {
                                                temp[counter]=(window[counter-
2]*14+window[counter-
1]*422+window[counter]*1000+window[counter+1]*422+window[counter+2]*14)/1872;
                                        }
                                        if(counter>=(windowsize-2))
                                        {

	temp[counter]=(window[counter]*1000+window[counter-1]*422+window[counter-
2]*14)/1436;
                                        }

                                }

                                counter = 0;


                                for(counter=0; counter<windowsize; counter++)
                                {
                                        window[counter]=temp[counter];
                                }


                                counter = 0;
                                w_ptr=&window[0];




                                break;

                        case E_KEY_1:


                                vLcdClear();
                                vLcdWriteText("Button2", lcdrowpos , 0);
                                vLcdRefreshAll();

                                vLcdWriteText("StartSampling", 1 , 0);
                                vLcdRefreshAll();

                                counter = 0;
                                w_ptr=&window[0];
                                sam_ptr=&sample[0];


                                while(counter<windowsize)
                                {
                                        // If this is an event from timer 0
                                        if (bAHI_TimerFired(E_AHI_TIMER_0))
                                        {
                                                vAHI_AdcStartSample();
                                                //read from ADC
                                                while (bAHI_AdcPoll());
                                                *sam_ptr=u16AHI_AdcRead();
                                                sam_ptr++;
                                                counter++;
                                        }
                                }


                                vLcdWriteText("SamplingDone", 2 , 0);
                                vLcdRefreshAll();


                                counter = 0;
```

```
                                        for(counter=0;counter<windowsize;counter++)
                                        {
                                                if(counter<=2)
                                                {

        temp[counter]=(sample[counter]*1000+sample[counter+1]*422+sample[counter+2]*14)/14
36;
                                                }
                                                if(counter>2)
                                                {
                                                        temp[counter]=(sample[counter-
2]*14+sample[counter-
1]*422+sample[counter]*1000+sample[counter+1]*422+sample[counter+2]*14)/1872;
                                                }
                                                if(counter>=(windowsize-2))
                                                {

        temp[counter]=(sample[counter]*1000+sample[counter-1]*422+sample[counter-
2]*14)/1436;
                                                }

                                        }


                                        counter = 0;


                                        for(counter=0; counter<windowsize; counter++)
                                        {
                                                sample[counter]=temp[counter];

                                        }

                                        sam_ptr=&sample[0];
                                        w_ptr=&window[0];

                                        tmp_ptr=&temp[0];


                                        for(i=0; i<windowsize; i++)
                                        {

                                                *tmp_ptr = abs((*sam_ptr) - (*w_ptr));
                                                sam_ptr++;
                                                w_ptr++;
                                                tmp_ptr++;
                                        }

                                        vLcdWriteText("AbsDone", 3 , 0);
                                        vLcdRefreshAll();



                                        tmp_ptr=&temp[0];

                                        for(i=0; i<windowsize; i++)
                                        {
                                                if(*tmp_ptr<50)
                                                {
                                                        *tmp_ptr = 0;
                                                }

                                                tmp_ptr++;
                                        }



                                        tmp_ptr=&temp[0];

                                        for(i=0; i<windowsize; i++)
                                        {
```

```
                sum = sum+ *tmp_ptr;
                tmp_ptr++;
            }
            vLcdWriteText("SumDone", 4 , 0);
            vLcdRefreshAll();


            sum_int = sum;

            pu8LcdLoad = (uint8)((sum_int) & 0xff );
            vValToDec(acString, pu8LcdLoad, "  ");
            vLcdWriteText(acString, 6 , 50);
            pu8LcdLoad = (uint8)((sum_int >> 8) & 0xff );
            vValToDec(acString, pu8LcdLoad, "  ");
            vLcdWriteText(acString, 6 , 0);
            pu8LcdLoad = (uint8)((sum_int >> 16) & 0xff );
            vValToDec(acString, pu8LcdLoad, "  ");
            vLcdWriteText(acString, 5 , 50);
            pu8LcdLoad = (uint8)((sum_int >> 24) & 0xff );
            vValToDec(acString, pu8LcdLoad, "  ");
            vLcdWriteText(acString, 5 , 0);

            vLcdRefreshAll();

            sum = 0;
            break;

        case E_KEY_2:

            vLcdClear();
            vLcdWriteText("Button3", lcdrowpos , 0);
            vLcdRefreshAll();
            counter = 0;

            w_ptr = &window[0];
            sam_ptr = &sample[0];
            tmp_ptr = &temp[0];

            while(counter<windowsize)
            {
                vLcdWriteText("Window -", 1 , 0);
                pu8LcdLoad = (uint8)((*w_ptr) & 0xff);
                vValToDec(acString, pu8LcdLoad, "  ");
                vLcdWriteText(acString, 2 , 90);
                pu8LcdLoad = (uint8)((*w_ptr >> 8) & 0xff );
                vValToDec(acString, pu8LcdLoad, "  ");
                vLcdWriteText(acString, 2 , 60);
               pu8LcdLoad = (uint8)((*w_ptr >> 16) & 0xff );
                vValToDec(acString, pu8LcdLoad, "  ");
                vLcdWriteText(acString, 2 , 30);
               pu8LcdLoad = (uint8)((*w_ptr >> 24) & 0xff );
                vValToDec(acString, pu8LcdLoad, "  ");
                vLcdWriteText(acString, 2 , 0);

                vLcdWriteText("Sample -", 3 , 0);
                pu8LcdLoad = (uint8)((*sam_ptr) & 0xff);
                vValToDec(acString, pu8LcdLoad, "  ");
                vLcdWriteText(acString, 4 , 90);
                pu8LcdLoad = (uint8)((*sam_ptr >> 8) &
0xff );
                vValToDec(acString, pu8LcdLoad, "  ");
                vLcdWriteText(acString, 4 , 60);
                pu8LcdLoad = (uint8)((*sam_ptr >> 16) &
0xff );
                vValToDec(acString, pu8LcdLoad, "  ");
                vLcdWriteText(acString, 4 , 30);
                pu8LcdLoad = (uint8)((*sam_ptr >> 24) &
0xff );
                vValToDec(acString, pu8LcdLoad, "  ");
                vLcdWriteText(acString, 4 , 0);


                vLcdWriteText("Temp -", 5 , 0);
                pu8LcdLoad = (uint8)((*tmp_ptr) & 0xff);
```

```
                                        vValToDec(acString, pu8LcdLoad, "  ");
                                        vLcdWriteText(acString, 6 , 90);
                                        pu8LcdLoad = (uint8)((*tmp_ptr >> 8) &
0xff );
                                        vValToDec(acString, pu8LcdLoad, "  ");
                                        vLcdWriteText(acString, 6 , 60);
                                        pu8LcdLoad = (uint8)((*tmp_ptr >> 16) &
0xff );
                                        vValToDec(acString, pu8LcdLoad, "  ");
                                        vLcdWriteText(acString, 6 , 30);
                                        pu8LcdLoad = (uint8)((*tmp_ptr >> 24) &
0xff );
                                        vValToDec(acString, pu8LcdLoad, "  ");
                                        vLcdWriteText(acString, 6 , 0);


                                        vLcdRefreshAll();
                                        for(i=0; i<2500000; i++);
                                        counter++;
                                        w_ptr++;
                                        sam_ptr++;
                                        tmp_ptr++;

                                }


                                break;

                        case E_KEY_3:

                                vLcdClear();
                                vLcdWriteText("Button4", 0 , 0);
                                vLcdRefreshAll();


                                vWUART_Init();


                                vProcessEventQueues();

                                switch (sCoordData.sSystem.eState)
                                {
                                case E_STATE_INIT:
                                        sCoordData.sSystem.u8Channel = 0;
                                        sCoordData.sSystem.eState =
E_STATE_START_COORDINATOR;
                                        break;

                                case E_STATE_START_COORDINATOR:
                                        if(bStartCoordinator())
                                        {
                                                sCoordData.sSystem.eState =
E_STATE_RUNNING_UART_APP;
                                        }
                                        break;

                                case E_STATE_RUNNING_UART_APP:
                                        break;
                                }




                                break;

                        default:

                                break;

                        }
                }
        }
        else
        {
```

```
                    u8KeysDown = 0;
                }

        }
}

/*****************************************************************************
*
* NAME: AppWarmStart
*
* DESCRIPTION:
* Entry point for a wake from sleep mode with the memory contents held. We
* are not using this mode and so should never get here.
*
* PARAMETERS:      Name            RW  Usage
* None.
*
* RETURNS:
* None.
*
* NOTES:
* None.
*****************************************************************************/
PUBLIC void AppWarmStart(void)
{
    AppColdStart();
}

/*****************************************************************************
*
* NAME: vValToDec
*
* DESCRIPTION:
* Converts an 8-bit value to a string of the textual decimal representation.
* Adds a text string after the text.
*
* PARAMETERS:      Name            RW  Usage
*                  pcOutString     R   Location for new string
*                  u8Value         R   Value to convert
*                  pcLabel         R   Label to append to string
*
* RETURNS:
* void
*
*****************************************************************************/
PRIVATE void vValToDec(char *pcOutString, uint8 u8Value, char *pcLabel)
{
    const uint8 au8Digits[3] = {100, 10, 1};
    uint8 u8Digit;
    uint8 u8DigitIndex;
    uint8 u8Count;
    bool_t boPreviousDigitPrinted = FALSE;

    for (u8DigitIndex = 0; u8DigitIndex < 3; u8DigitIndex++)
    {
        u8Count = 0;
        u8Digit = au8Digits[u8DigitIndex];
        while (u8Value >= u8Digit)
        {
            u8Value -= u8Digit;
            u8Count++;
        }

        if ((u8Count != 0) || (boPreviousDigitPrinted == TRUE)
            || (u8DigitIndex == 2))
        {
            *pcOutString = '0' + u8Count;
            boPreviousDigitPrinted = TRUE;
            pcOutString++;
        }
    }

    vStringCopy(pcLabel, pcOutString);
}
```

```
/****************************************************************************
*
* NAME: vStringCopy
*
* DESCRIPTION:
* Simple string copy as standard libraries not available.
*
* PARAMETERS:      Name    RW  Usage
*                  pcFrom  R   Pointer to string to copy
*                  pcTo    W   Pointer to store for new string
*
* RETURNS:
* void
*
****************************************************************************/
PRIVATE void vStringCopy(char *pcFrom, char *pcTo)
{
    while (*pcFrom != '\0')
    {
        *pcTo = *pcFrom;
        pcTo++;
        pcFrom++;
    }
    *pcTo = '\0';
}

/****************************************************************************
*
* NAME: vAdcInit
*
* DESCRIPTION:
* Initialise the ADC.
*
* PARAMETERS:      Name    RW  Usage
* None
*
* RETURNS:
* void
*
****************************************************************************/
PRIVATE void vAdcInit(void) //, uint32 u32Length)
{
    vAHI_ApConfigure ( E_AHI_AP_INT_DISABLE,
               E_AHI_AP_SAMPLE_2,
               E_AHI_AP_CLOCKDIV_500KHZ,
               E_AHI_AP_INTREF);



    /* configure & enable ADC1 */
    vAHI_AdcEnable (E_AHI_ADC_CONVERT_ENABLE,
               E_AHI_AP_GAIN_2, //set input range 0V - Vref
               E_AHI_ADC_SRC_ADC_1);


}

/****************************************************************************
 *
 * NAME: vWUART_Init
 *
 * DESCRIPTION:
 * Initialises stack and hardware, sets non-default values in the 802.15.4
 * PIB.
 *
 * PARAMETERS:      Name            RW  Usage
 * None.
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
```

```
 ****************************************************************************/
PRIVATE void vWUART_Init(void)
{
    MAC_MlmeReqRsp_s   sMlmeReqRsp;
    MAC_MlmeSyncCfm_s  sMlmeSyncCfm;

    sCoordData.sNode.u8AssociatedNodes = 0;

    /* Initialise stack and hardware interfaces. We aren't using callbacks
       at all, just monitoring the upward queues in a loop */
    (void)u32AppQApiInit(NULL, NULL, NULL);
    (void)u32AHI_Init();

    /* Set Pan ID and short address in PIB (also sets match registers in hardware) */
    sMlmeReqRsp.u8Type = MAC_MLME_REQ_SET;
    sMlmeReqRsp.u8ParamLength = sizeof(MAC_MlmeReqSet_s);
    sMlmeReqRsp.uParam.sReqSet.u8PibAttribute = MAC_PIB_ATTR_PAN_ID;
    NBO_vFromU16(PAN_ID, sMlmeReqRsp.uParam.sReqSet.uPibAttributeValue.au8PanId);
    vAppApiMlmeRequest(&sMlmeReqRsp, &sMlmeSyncCfm);

    sMlmeReqRsp.uParam.sReqSet.u8PibAttribute = MAC_PIB_ATTR_SHORT_ADDRESS;
    NBO_vFromU16(COORD_ADDR, sMlmeReqRsp.uParam.sReqSet.uPibAttributeValue.au8ShortAddr);
    vAppApiMlmeRequest(&sMlmeReqRsp, &sMlmeSyncCfm);

    /* Allow nodes to associate */
    sMlmeReqRsp.u8Type = MAC_MLME_REQ_SET;
    sMlmeReqRsp.u8ParamLength = sizeof(MAC_MlmeReqSet_s);
    sMlmeReqRsp.uParam.sReqSet.u8PibAttribute = MAC_PIB_ATTR_ASSOCIATION_PERMIT;
    sMlmeReqRsp.uParam.sReqSet.uPibAttributeValue.u8AssociationPermit = 1;
    vAppApiMlmeRequest(&sMlmeReqRsp, &sMlmeSyncCfm);

    /* Enable receiver to be on when idle */
    sMlmeReqRsp.u8Type = MAC_MLME_REQ_SET;
    sMlmeReqRsp.u8ParamLength = sizeof(MAC_MlmeReqSet_s);
    sMlmeReqRsp.uParam.sReqSet.u8PibAttribute = MAC_PIB_ATTR_RX_ON_WHEN_IDLE;
    sMlmeReqRsp.uParam.sReqSet.uPibAttributeValue.u8RxOnWhenIdle = 1;
    vAppApiMlmeRequest(&sMlmeReqRsp, &sMlmeSyncCfm);

    /* Set LED IO's to outputs */
    vAHI_DioSetDirection(0, LED_OUTPUTS_MASK);
    vAHI_DioSetOutput(LED1_MASK, 0);
    vAHI_DioSetOutput(0, LED2_MASK);

    /* Initialise the serial port and rx/tx queues */
    vSerial_Init();

    /* Use wake timer to give 10ms tick period */
    //vAHI_TimerEnable(E_AHI_TIMER_1, 128, TRUE);
    //vAHI_TimerEnable(E_AHI_TIMER_1, 128, FALSE, TRUE);
       /*
    vAHI_TimerEnable3Param(E_AHI_TIMER_1, 128, TRUE);
    vAHI_TimerClockSelect(E_AHI_TIMER_1, FALSE, TRUE);
    vAHI_TimerStartRepeat(E_AHI_TIMER_1, 625, 1250);
       */
}

/****************************************************************************
 *
 * NAME: vProcessEventQueues
 *
 * DESCRIPTION:
 * Check each of the three event queues and process and items found.
 *
 * PARAMETERS:      Name            RW  Usage
 * None.
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vProcessEventQueues(void)
{
```

```
    MAC_MlmeDcfmInd_s *psMlmeInd;
        MAC_McpsDcfmInd_s *psMcpsInd;
    AppQApiHwInd_s    *psAHI_Ind;

    /* Check for anything on the MCPS upward queue */
    do
    {
        psMcpsInd = psAppQApiReadMcpsInd();
        if (psMcpsInd != NULL)
        {
            vProcessIncomingData(psMcpsInd);
            vAppQApiReturnMcpsIndBuffer(psMcpsInd);
        }
    } while (psMcpsInd != NULL);

    /* Check for anything on the MLME upward queue */
    do
    {
        psMlmeInd = psAppQApiReadMlmeInd();
        if (psMlmeInd != NULL)
        {
            vProcessIncomingMlme(psMlmeInd);
            vAppQApiReturnMlmeIndBuffer(psMlmeInd);
        }
    } while (psMlmeInd != NULL);

    /* Check for anything on the AHI upward queue */
    do
    {
        psAHI_Ind = psAppQApiReadHwInd();
        if (psAHI_Ind != NULL)
        {
            vProcessIncomingHwEvent(psAHI_Ind);
            vAppQApiReturnHwIndBuffer(psAHI_Ind);
        }
    } while (psAHI_Ind != NULL);
}

/****************************************************************************
 *
 * NAME: vProcessIncomingMlme
 *
 * DESCRIPTION:
 * Process any incoming managment events from the stack.
 *
 * PARAMETERS:      Name            RW  Usage
 *                  psMlmeInd
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vProcessIncomingMlme(MAC_MlmeDcfmInd_s *psMlmeInd)
{
    switch(psMlmeInd->u8Type)
    {
    case MAC_MLME_IND_ASSOCIATE:
        /* Only allow nodes to associate if network has been started */
        if (sCoordData.sSystem.eState == E_STATE_RUNNING_UART_APP)
        {
            vHandleNodeAssociation(psMlmeInd);
        }
        break;
    }
}

/****************************************************************************
 *
 * NAME: vProcessIncomingData
 *
 * DESCRIPTION:
 * Process incoming data events from the stack.
```

```
 *
 * PARAMETERS:      Name            RW  Usage
 *                  psMcpsInd
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/

 uint8 toHex(uint8 x)
 {
     x=x & 0x0f;

     if (x>9)
        x=x-10+'A';
    else
        x=x+'0';

     return  x;
 }

PRIVATE void vProcessIncomingData(MAC_McpsDcfmInd_s *psMcpsInd)
{
    MAC_RxFrameData_s *psFrame;
    MAC_Addr_s *psAddr;
    uint16 u16NodeAddr;
    uint8 i;

    psFrame = &psMcpsInd->uParam.sIndData.sFrame;
    psAddr = &psFrame->sAddrPair.sSrc;

    /* Check that this is a data frame */
    if (psMcpsInd->u8Type == MAC_MCPS_IND_DATA)
    {
        /* Check that data is from UART node */
        u16NodeAddr = NBO_u16To(psAddr->uAddr.au8Short);

        if (u16NodeAddr == sCoordData.sNode.asAssocNodes[0].u16ShortAddr)
        {
            if (psFrame->au8Sdu[0] == u8RxFrameHandle)
            {
                u8RxFrameHandle++;

                /* Copy frame data to serial buffer for output on UART */
                for (i = 1; i < psFrame->u8SduLength; i++)
                {
                                vSerial_TxChar(toHex(psFrame->au8Sdu[i] >> 4));
                    vSerial_TxChar(psFrame->au8Sdu[i]);
                }
                            vSerial_TxChar((uint8)' ');
            }
            /* Must have missed a frame */
            else if (psFrame->au8Sdu[0] > u8RxFrameHandle)
            {
                u8RxFrameHandle = psFrame->au8Sdu[0] + 1;

                /* Copy frame data to serial buffer for output on UART */
                for (i = 1; i < psFrame->u8SduLength; i++)
                {
                                vSerial_TxChar(toHex(psFrame->au8Sdu[i] >> 4));
                    vSerial_TxChar(psFrame->au8Sdu[i]);
                }
                            vSerial_TxChar((uint8)' ');
            }
            /* Must be the same frame as last time */
            else if (psFrame->au8Sdu[0] < u8RxFrameHandle)
            {
                /* Dont do anything as we already have the data */
            }
        }
    }
}
```

```
/****************************************************************************
 *
 * NAME: vProcessIncomingHwEvent
 *
 * DESCRIPTION:
 * Process any hardware events.
 *
 * PARAMETERS:      Name            RW  Usage
 *                  psAHI_Ind
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vProcessIncomingHwEvent(AppQApiHwInd_s *psAHI_Ind)
{
    /* If this is an event from UART0 */
    if (psAHI_Ind->u32DeviceId == E_AHI_DEVICE_UART0)
    {
        /* If data has been received */
        if ((psAHI_Ind->u32ItemBitmap & 0x000000FF) == E_AHI_UART_INT_RXDATA)
        {
            /* Process UART0 RX interrupt */
            vUART_RxCharISR((psAHI_Ind->u32ItemBitmap & 0x0000FF00) >> 8);
        }
        else if (psAHI_Ind->u32ItemBitmap == E_AHI_UART_INT_TX)
        {

            /* Process UART0 TX interrupt */
            vUART_TxCharISR();
        }
    }

    /* If this is an event from timer 1 */
    if (psAHI_Ind->u32DeviceId == E_AHI_DEVICE_TIMER1)
    {
        if (bAHI_TimerFired(E_AHI_TIMER_1))
        {
            if (sCoordData.sSystem.eState == E_STATE_RUNNING_UART_APP)
            {
                vWUART_TxData();
            }
        }
    }
}

/****************************************************************************
 *
 * NAME: vHandleNodeAssociation
 *
 * DESCRIPTION:
 * Handle request by node to join the network. If the nodes address matches
 * the address of a light switch then it is assumed to be a light switch and
 * is allowed to join the network.
 *
 * PARAMETERS:      Name            RW  Usage
 *                  psMlmeInd
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vHandleNodeAssociation(MAC_MlmeDcfmInd_s *psMlmeInd)
{
    MAC_MlmeReqRsp_s   sMlmeReqRsp;
    MAC_MlmeSyncCfm_s  sMlmeSyncCfm;

    uint16 u16ShortAddress;
```

```
        /*edited
        uint32 u32AddrLo;
    uint32 u32AddrHi;
        */

    /* Default to PAN access denied */
    uint8 u8AssocStatus = 2;

    /* Default short address */
    u16ShortAddress = 0xffff;


    /* Check that the device only wants to use a short address */
    if (psMlmeInd->uParam.sIndAssociate.u8Capability & 0x80)
    {
        if (sCoordData.sNode.u8AssociatedNodes < MAX_UART_NODES)
        {
            /* Allocate short address as next in list */
            u16ShortAddress = UART_NODE_ADDR_BASE + sCoordData.sNode.u8AssociatedNodes;

            /* Store details for future use */

sCoordData.sNode.asAssocNodes[sCoordData.sNode.u8AssociatedNodes].u16ShortAddr =
u16ShortAddress;
            sCoordData.sNode.u8AssociatedNodes++;

            /* Assume association succeeded */
            u8AssocStatus = 0;

            /* Turn on LED to show node has assocaited */
            vAHI_DioSetOutput(0, LED1_MASK);
        }
    }

    /* Create association response */
    sMlmeReqRsp.u8Type = MAC_MLME_RSP_ASSOCIATE;
    sMlmeReqRsp.u8ParamLength = sizeof(MAC_MlmeRspAssociate_s);
    memcpy(sMlmeReqRsp.uParam.sRspAssociate.au8DeviceAddr,
           psMlmeInd->uParam.sIndAssociate.au8DeviceAddr,
           MAC_EXT_ADDR_LEN);
    NBO_vFromU16(u16ShortAddress, sMlmeReqRsp.uParam.sRspAssociate.au8AssocShortAddr);
    sMlmeReqRsp.uParam.sRspAssociate.u8Status = u8AssocStatus;
    sMlmeReqRsp.uParam.sRspAssociate.u8SecurityEnable = FALSE;

    /* Send association response. There is no confirmation for an association
       response, hence no need to check */
    vAppApiMlmeRequest(&sMlmeReqRsp, &sMlmeSyncCfm);
}

/****************************************************************************
 *
 * NAME: bStartCoordinator
 *
 * DESCRIPTION:
 * Starts the network by configuring the controller board to act as the PAN
 * coordinator.
 *
 * PARAMETERS:      Name            RW  Usage
 * None.
 *
 * RETURNS:
 * TRUE if network was started successfully otherwise FALSE
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE bool_t bStartCoordinator(void)
{
    /* Structures used to hold data for MLME request and response */
    MAC_MlmeReqRsp_s   sMlmeReqRsp;
    MAC_MlmeSyncCfm_s  sMlmeSyncCfm;

    /* Start Pan */
    sMlmeReqRsp.u8Type = MAC_MLME_REQ_START;
```

```
    sMlmeReqRsp.u8ParamLength = sizeof(MAC_MlmeReqStart_s);
    NBO_vFromU16(PAN_ID, sMlmeReqRsp.uParam.sReqStart.au8PanId);

    //sMlmeReqRsp.uParam.sReqStart.u8Channel = sCoordData.sSystem.u8Channel;
    sMlmeReqRsp.uParam.sReqStart.u8Channel = 0x0B;

    sMlmeReqRsp.uParam.sReqStart.u8BeaconOrder = 0x0f; /* No beacons */
    sMlmeReqRsp.uParam.sReqStart.u8SuperframeOrder = 0x0f;
    sMlmeReqRsp.uParam.sReqStart.u8PanCoordinator = TRUE;
    sMlmeReqRsp.uParam.sReqStart.u8BatteryLifeExt = FALSE;
    sMlmeReqRsp.uParam.sReqStart.u8Realignment = FALSE;
    sMlmeReqRsp.uParam.sReqStart.u8SecurityEnable = FALSE;
    vAppApiMlmeRequest(&sMlmeReqRsp, &sMlmeSyncCfm);

    if (sMlmeSyncCfm.u8Status != MAC_MLME_CFM_OK)
    {
        /* Error during MLME-Start */
        return(FALSE);
    }

    return(TRUE);
}

 /****************************************************************************
 *
 * NAME: vWUART_TxData
 *
 * DESCRIPTION:
 *
 * PARAMETERS:      Name            RW  Usage
 * None.
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vWUART_TxData(void)
{
    MAC_McpsReqRsp_s  sMcpsReqRsp;
    MAC_McpsSyncCfm_s sMcpsSyncCfm;
    uint8 *pu8Payload, i = 0;
    int16 i16RxChar;

    i16RxChar = i16Serial_RxChar();

    if (i16RxChar >= 0)
    {
        /* Create frame transmission request */
        sMcpsReqRsp.u8Type = MAC_MCPS_REQ_DATA;
        sMcpsReqRsp.u8ParamLength = sizeof(MAC_McpsReqData_s);
        /* Set handle so we can match confirmation to request */
        sMcpsReqRsp.uParam.sReqData.u8Handle = 1;
        /* Use short address for source */
        sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sSrc.u8AddrMode = 2;
        NBO_vFromU16(PAN_ID, sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sSrc.au8PanId);
        NBO_vFromU16(COORD_ADDR,
sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sSrc.uAddr.au8Short);
        /* Use short address for destination */
        sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sDst.u8AddrMode = 2;
        NBO_vFromU16(PAN_ID, sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sDst.au8PanId);
        NBO_vFromU16(sCoordData.sNode.asAssocNodes[0].u16ShortAddr,
sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sDst.uAddr.au8Short);
        /* Frame requires ack but not security, indirect transmit or GTS */
        sMcpsReqRsp.uParam.sReqData.sFrame.u8TxOptions = MAC_TX_OPTION_ACK;

        pu8Payload = sMcpsReqRsp.uParam.sReqData.sFrame.au8Sdu;

        pu8Payload[i++] = u8TxFrameHandle++;
        pu8Payload[i++] = (uint8)i16RxChar;

        while (((i16RxChar = i16Serial_RxChar()) >= 0) && (i < MAX_DATA_PER_FRAME))
        {
```

```
            /* Set payload data */
            pu8Payload[i++] = (uint8)i16RxChar;
        }

        /* Set frame length */
        sMcpsReqRsp.uParam.sReqData.sFrame.u8SduLength = i;

        /* Request transmit */
        vAppApiMcpsRequest(&sMcpsReqRsp, &sMcpsSyncCfm);
    }
}

/****************************************************************************/
/***        END OF FILE                                                   ***/
/****************************************************************************/
```

Appendix B : Source Code of the Endpoint

```
/****************************************************************************
 *
 * MODULE:             wuart_e.c
 *
 * COMPONENT:          $RCSfile: $
 *
 * VERSION:            $Name: $
 *
 * REVISION:           $Revision: $
 *
 * DATED:              $Date: $
 *
 * STATUS:             $State: $
 *
 * AUTHOR:             Ian Morris
 *
 * DESCRIPTION
 *
 * CHANGE HISTORY:
 *
 * $Log: $
 *
 *
 * LAST MODIFIED BY:   $Author: pc1 $
 *                     $Modtime: $
 *
 *
 ****************************************************************************
 *
 *  (c) Copyright 2000 JENNIC Ltd
 *
 ****************************************************************************/

/****************************************************************************/
/***        Include files                                                 ***/
/****************************************************************************/

#include <jendefs.h>
#include <AppHardwareApi.h>
#include <AppQueueApi.h>
#include <mac_sap.h>
#include <nbo_pub.h>

#include "serialq.h"
#include "uart.h"
#include "serial.h"

/****************************************************************************/
/***        Macro Definitions                                             ***/
/****************************************************************************/
#define LED_OUTPUTS_MASK   0x0000C000UL
#define LED1_MASK          0x00008000UL
#define LED2_MASK          0x00004000UL

/* Defines the channels to scan. Each bit represents one channel. All channels
   in the channels (11-26) in the 2.4GHz band are scanned. */
#define SCAN_CHANNELS 0x07FFF800UL

/* Network parameters */
#define PAN_ID                    0x0401U
#define COORD_ADDR                0x0502U

/* Wireless UART device data */
#define MAX_UART_NODES            1

//edited by kc
#define UART_CONT_EXT_ADDR_LO     0x00001010UL
#define UART_CONT_EXT_ADDR_HI     0x00000101UL
#define UART_NODE_EXT_ADDR_LO     0x10101010UL
#define UART_NODE_EXT_ADDR_HI     0x01010101UL
```

```
#define UART_NODE_ADDR_BASE       0x1000U
#define MAX_DATA_PER_FRAME        64

/****************************************************************************/
/***        Type Definitions                                              ***/
/****************************************************************************/
/* State machine states */
typedef enum
{
    E_STATE_OFF,
    E_STATE_SCANNING,
    E_STATE_ASSOCIATING,
    E_STATE_RUNNING,
} teState;

/* All application data with scope within the entire file is kept here,
   including all stored node data */
typedef struct
{
    struct
    {
        teState eState;
        uint8   u8Channel;
        uint16  u16ShortAddr;
    } sSystem;
} tsDeviceData;

/****************************************************************************/
/***        Local Function Prototypes                                     ***/
/****************************************************************************/

/****************************************************************************/
/***        Exported Variables                                            ***/
/****************************************************************************/

/****************************************************************************/
/***        Local Variables                                               ***/
/****************************************************************************/
PRIVATE tsDeviceData sDeviceData;
uint8 u8TxFrameHandle = 0;
uint8 u8RxFrameHandle = 0;

/****************************************************************************/
/***        Exported Functions                                            ***/
/****************************************************************************/

/****************************************************************************/
/***        Local Functions                                               ***/
/****************************************************************************/
PRIVATE void vWUART_Init(void);
PRIVATE void vStartScan(void);
PRIVATE void vStartAssociate(void);
PRIVATE void vProcessEventQueues(void);
PRIVATE void vProcessIncomingMlme(MAC_MlmeDcfmInd_s *psMlmeInd);
PRIVATE void vProcessIncomingData(MAC_McpsDcfmInd_s *psMcpsInd);
PRIVATE void vProcessIncomingHwEvent(AppQApiHwInd_s *psAHI_Ind);
PRIVATE void vHandleActiveScanResponse(MAC_MlmeDcfmInd_s *psMlmeInd);
PRIVATE void vHandleAssociateResponse(MAC_MlmeDcfmInd_s *psMlmeInd);
PRIVATE void vWUART_TxData(void);

// edited by kc

PRIVATE void vWUART_TxData_ADC(void);
//PRIVATE void vWUART_TxData_2(void);
//PRIVATE void vWUART_TxData_3(void);
//PRIVATE void vAdcDataLogger(uint16 *pau16DataBuffer);


/****************************************************************************
 *
 * NAME: AppColdStart
 *
 * DESCRIPTION:
 *
```

```
 * PARAMETERS:      Name            RW  Usage
 * None.
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * Entry point for a power on reset or wake from sleep mode.
 ****************************************************************************/
PUBLIC void AppColdStart(void)
{
    vWUART_Init();

    vStartScan();

    while(1)
    {
        vProcessEventQueues();
    }
}

/****************************************************************************
 *
 * NAME: AppWarmStart
 *
 * DESCRIPTION:
 * Entry point for a wake from sleep mode with the memory contents held. We
 * are not using this mode and so should never get here.
 *
 * PARAMETERS:      Name            RW  Usage
 * None.
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PUBLIC void AppWarmStart(void)
{
    AppColdStart();
}

/****************************************************************************
 *
 * NAME: vWUART_Init
 *
 * DESCRIPTION:
 * Initialises stack and hardware, sets non-default values in the 802.15.4
 * PIB.
 *
 * PARAMETERS:      Name            RW  Usage
 * None.
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vWUART_Init(void)
{
    MAC_MlmeReqRsp_s   sMlmeReqRsp;
    MAC_MlmeSyncCfm_s  sMlmeSyncCfm;

    sDeviceData.sSystem.eState = E_STATE_OFF;

    /* Initialise stack and hardware interfaces. We aren't using callbacks
       at all, just monitoring the upward queues in a loop */
    (void)u32AppQApiInit(NULL, NULL, NULL);
    (void)u32AHI_Init();

    /* Set Pan ID in PIB (also sets match register in hardware) */
    sMlmeReqRsp.u8Type = MAC_MLME_REQ_SET;
```

```
    sMlmeReqRsp.u8ParamLength = sizeof(MAC_MlmeReqSet_s);
    sMlmeReqRsp.uParam.sReqSet.u8PibAttribute = MAC_PIB_ATTR_PAN_ID;
    NBO_vFromU16(PAN_ID, sMlmeReqRsp.uParam.sReqSet.uPibAttributeValue.au8PanId);
    vAppApiMlmeRequest(&sMlmeReqRsp, &sMlmeSyncCfm);

    /* Enable receiver to be on when idle */
    sMlmeReqRsp.u8Type = MAC_MLME_REQ_SET;
    sMlmeReqRsp.u8ParamLength = sizeof(MAC_MlmeReqSet_s);
    sMlmeReqRsp.uParam.sReqSet.u8PibAttribute = MAC_PIB_ATTR_RX_ON_WHEN_IDLE;
    sMlmeReqRsp.uParam.sReqSet.uPibAttributeValue.u8RxOnWhenIdle = 1;
    vAppApiMlmeRequest(&sMlmeReqRsp, &sMlmeSyncCfm);

    /* Set LED IO's to outputs */
    vAHI_DioSetDirection(0, LED_OUTPUTS_MASK);
    vAHI_DioSetOutput(LED1_MASK, 0);
    vAHI_DioSetOutput(0, LED2_MASK);

    /* Initialise the serial port and rx/tx queues */
    vSerial_Init();

    /* Use wake timer to give 10ms tick period */
    vAHI_TimerEnable(E_AHI_TIMER_1,4, 128, TRUE);
    vAHI_TimerClockSelect(E_AHI_TIMER_1, FALSE, TRUE);
    vAHI_TimerStartRepeat(E_AHI_TIMER_1, 625, 1250);
}

/****************************************************************************
 *
 * NAME: vProcessEventQueues
 *
 * DESCRIPTION:
 * Check each of the three event queues and process and items found.
 *
 * PARAMETERS:      Name            RW  Usage
 * None.
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vProcessEventQueues(void)
{
    MAC_MlmeDcfmInd_s *psMlmeInd;
        MAC_McpsDcfmInd_s *psMcpsInd;
    AppQApiHwInd_s    *psAHI_Ind;

    /* Check for anything on the MCPS upward queue */
    do
    {
        psMcpsInd = psAppQApiReadMcpsInd();
        if (psMcpsInd != NULL)
        {
            vProcessIncomingData(psMcpsInd);
            vAppQApiReturnMcpsIndBuffer(psMcpsInd);
        }
    } while (psMcpsInd != NULL);

    /* Check for anything on the MLME upward queue */
    do
    {
        psMlmeInd = psAppQApiReadMlmeInd();
        if (psMlmeInd != NULL)
        {
            vProcessIncomingMlme(psMlmeInd);
            vAppQApiReturnMlmeIndBuffer(psMlmeInd);
        }
    } while (psMlmeInd != NULL);

    /* Check for anything on the AHI upward queue */
    do
    {
        psAHI_Ind = psAppQApiReadHwInd();
```

```
        if (psAHI_Ind != NULL)
        {
            vProcessIncomingHwEvent(psAHI_Ind);
            vAppQApiReturnHwIndBuffer(psAHI_Ind);
        }
    } while (psAHI_Ind != NULL);
}

/****************************************************************************
 *
 * NAME: vProcessIncomingMlme
 *
 * DESCRIPTION:
 * Process any incoming managment events from the stack.
 *
 * PARAMETERS:      Name            RW  Usage
 *                  psMlmeInd
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vProcessIncomingMlme(MAC_MlmeDcfmInd_s *psMlmeInd)
{
    /* We respond to several MLME indications and confirmations, depending
       on mode */
    switch (psMlmeInd->u8Type)
    {
    /* Deferred confirmation that the scan is complete */
    case MAC_MLME_DCFM_SCAN:
        /* Only respond to this if scanning */
        if (sDeviceData.sSystem.eState == E_STATE_SCANNING)
        {
            vHandleActiveScanResponse(psMlmeInd);
        }
        break;

    /* Deferred confirmation that the association process is complete */
    case MAC_MLME_DCFM_ASSOCIATE:
        /* Only respond to this if associating */
        if (sDeviceData.sSystem.eState == E_STATE_ASSOCIATING)
        {
            vHandleAssociateResponse(psMlmeInd);
        }
        break;

    default:
        break;
    }
}

/****************************************************************************
 *
 * NAME: vProcessIncomingData
 *
 * DESCRIPTION:
 * Process incoming data events from the stack.
 *
 * PARAMETERS:      Name            RW  Usage
 *                  psMcpsInd
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vProcessIncomingData(MAC_McpsDcfmInd_s *psMcpsInd)
{
    MAC_RxFrameData_s *psFrame;
    MAC_Addr_s *psAddr;
    uint16 u16NodeAddr;
```

```
    uint8 i;

    psFrame = &psMcpsInd->uParam.sIndData.sFrame;
    psAddr = &psFrame->sAddrPair.sSrc;

    /* Check that this is a data frame */
    if (psMcpsInd->u8Type == MAC_MCPS_IND_DATA)
    {
        /* Check that data is from UART node */
        u16NodeAddr = NBO_u16To(psAddr->uAddr.au8Short);

        if (u16NodeAddr == COORD_ADDR)
        {
            if (psFrame->au8Sdu[0] == u8RxFrameHandle)
            {
                u8RxFrameHandle++;

                /* Copy frame data to serial buffer for output on UART */
                for (i = 1; i < psFrame->u8SduLength; i++)
                {
                    vSerial_TxChar(psFrame->au8Sdu[i]);
                }
            }
            /* Must have missed a frame */
            else if (psFrame->au8Sdu[0] > u8RxFrameHandle)
            {
                u8RxFrameHandle = psFrame->au8Sdu[0] + 1;

                /* Copy frame data to serial buffer for output on UART */
                for (i = 1; i < psFrame->u8SduLength; i++)
                {
                    vSerial_TxChar(psFrame->au8Sdu[i]);
                }
            }
            /* Must be the same frame as last time */
            else if (psFrame->au8Sdu[0] < u8RxFrameHandle)
            {
                /* Dont do anything as we already have the data */
            }
        }
    }
}

/****************************************************************************
 *
 * NAME: vProcessIncomingHwEvent
 *
 * DESCRIPTION:
 * Process any hardware events.
 *
 * PARAMETERS:      Name            RW  Usage
 *                  psAHI_Ind
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vProcessIncomingHwEvent(AppQApiHwInd_s *psAHI_Ind)
{
    /* If this is an event from UART0 */
    if (psAHI_Ind->u32DeviceId == E_AHI_DEVICE_UART0)
    {
        /* If data has been received */
        if ((psAHI_Ind->u32ItemBitmap & 0x000000FF) == E_AHI_UART_INT_RXDATA)
        {
            /* Process UART0 RX interrupt */
            vUART_RxCharISR((psAHI_Ind->u32ItemBitmap & 0x0000FF00) >> 8);
        }
        else if (psAHI_Ind->u32ItemBitmap == E_AHI_UART_INT_TX)
        {

            /* Process UART0 TX interrupt */
```

```
            vUART_TxCharISR();
        }
    }

    /* If this is an event from timer 1 */
    if (psAHI_Ind->u32DeviceId == E_AHI_DEVICE_TIMER1)
    {
        if (bAHI_TimerFired(E_AHI_TIMER_1))
        {
            /* Do this every 10ms to tx data received on hardware UART */
            if (sDeviceData.sSystem.eState == E_STATE_RUNNING)
            {
                vWUART_TxData_ADC();
            }
        }
    }
}

/****************************************************************************
 *
 * NAME: vStartScan
 *
 * DESCRIPTION:
 * Start a scan to search for a network to join.
 *
 * PARAMETERS:      Name            RW  Usage
 * None.
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vStartScan(void)
{
    MAC_MlmeReqRsp_s  sMlmeReqRsp;
    MAC_MlmeSyncCfm_s sMlmeSyncCfm;

    sDeviceData.sSystem.eState = E_STATE_SCANNING;

    /* Request scan */
    sMlmeReqRsp.u8Type = MAC_MLME_REQ_SCAN;
    sMlmeReqRsp.u8ParamLength = sizeof(MAC_MlmeReqScan_s);
    sMlmeReqRsp.uParam.sReqScan.u8ScanType = MAC_MLME_SCAN_TYPE_ACTIVE;
    NBO_vFromU32(SCAN_CHANNELS, sMlmeReqRsp.uParam.sReqScan.au8ScanChannels);
    sMlmeReqRsp.uParam.sReqScan.u8ScanDuration = 3;
    vAppApiMlmeRequest(&sMlmeReqRsp, &sMlmeSyncCfm);

    /* Check immediate response */
    if (sMlmeSyncCfm.u8Status != MAC_MLME_CFM_DEFERRED)
    {
        /* Unexpected result: scan request should result in a deferred
           confirmation (i.e. we will receive it later) */
    }
}

/****************************************************************************
 *
 * NAME: vHandleActiveScanResponse
 *
 * DESCRIPTION:
 * Handle the reponse generated by the stack as a result of the network scan.
 *
 * PARAMETERS:      Name            RW  Usage
 *                  psMlmeInd
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vHandleActiveScanResponse(MAC_MlmeDcfmInd_s *psMlmeInd)
```

```
{
    MAC_PanDescr_s *psPanDesc;
    int i;

    /* Make sure it is what we're after */
    if ((psMlmeInd->uParam.sDcfmScan.u8Status == MAC_ENUM_SUCCESS)
        && (psMlmeInd->uParam.sDcfmScan.u8ScanType == MAC_MLME_SCAN_TYPE_ACTIVE))
    {
        /* Determine which, if any, network contains demo coordinator.
           Algorithm for determining which network to connect to is
           beyond the scope of 802.15.4, and we use a simple approach
           of matching the required PAN ID and short address, both of
           which we already know */

        i = 0;
        while (i < psMlmeInd->uParam.sDcfmScan.u8ResultListSize)
        {
            psPanDesc = &psMlmeInd->uParam.sDcfmScan.uList.asPanDescr[i];

            if ((NBO_u16To(psPanDesc->sCoord.au8PanId) == PAN_ID)
                && (psPanDesc->sCoord.u8AddrMode == 2)
                && (NBO_u16To(psPanDesc->sCoord.uAddr.au8Short) == COORD_ADDR))
            {
                /* Matched so start to synchronise and associate */
                sDeviceData.sSystem.u8Channel = psPanDesc->u8LogicalChan;
                vStartAssociate();
                return;
            }
            i++;
        }
    }
    /* Failed to find coordinator: keep trying */
    vStartScan();
}

/****************************************************************************
 *
 * NAME: vStartAssociate
 *
 * DESCRIPTION:
 * Start the association process with the network coordinator.
 *
 * PARAMETERS:      Name            RW  Usage
 * None.
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * Assumes that a network has been found during the network scan.
 ****************************************************************************/
PRIVATE void vStartAssociate(void)
{
    MAC_MlmeReqRsp_s  sMlmeReqRsp;
    MAC_MlmeSyncCfm_s sMlmeSyncCfm;

    sDeviceData.sSystem.eState = E_STATE_ASSOCIATING;

    /* Create associate request. We know short address and PAN ID of
       coordinator as this is preset and we have checked that received
       beacon matched this */
    sMlmeReqRsp.u8Type = MAC_MLME_REQ_ASSOCIATE;
    sMlmeReqRsp.u8ParamLength = sizeof(MAC_MlmeReqAssociate_s);
    sMlmeReqRsp.uParam.sReqAssociate.u8LogicalChan = sDeviceData.sSystem.u8Channel;
    sMlmeReqRsp.uParam.sReqAssociate.u8Capability = 0x80; /* We want short address, other
features off */
    sMlmeReqRsp.uParam.sReqAssociate.u8SecurityEnable = FALSE;
    sMlmeReqRsp.uParam.sReqAssociate.sCoord.u8AddrMode = 2;
    NBO_vFromU16(PAN_ID, sMlmeReqRsp.uParam.sReqAssociate.sCoord.au8PanId);
    NBO_vFromU16(COORD_ADDR, sMlmeReqRsp.uParam.sReqAssociate.sCoord.uAddr.au8Short);

    /* Put in associate request and check immediate confirm. Should be
       deferred, in which case response is handled by event handler */
    vAppApiMlmeRequest(&sMlmeReqRsp, &sMlmeSyncCfm);
```

```
    if (sMlmeSyncCfm.u8Status != MAC_MLME_CFM_DEFERRED)
    {
        /* Unexpected result */
    }
}

/****************************************************************************
 *
 * NAME: vHandleAssociateResponse
 *
 * DESCRIPTION:
 * Handle the response generated by the stack as a result of the associate
 * start request.
 *
 * PARAMETERS:      Name            RW  Usage
 *                  psMlmeInd
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vHandleAssociateResponse(MAC_MlmeDcfmInd_s *psMlmeInd)
{
    /* If successfully associated with network coordinator */
    if (psMlmeInd->uParam.sDcfmAssociate.u8Status == MAC_ENUM_SUCCESS)
    {
        /* Store short address that we have been assigned */
        sDeviceData.sSystem.u16ShortAddr = NBO_u16To(psMlmeInd-
>uParam.sDcfmAssociate.au8AssocShortAddr);
        /* We are now in the running state */
        sDeviceData.sSystem.eState = E_STATE_RUNNING;

        /* Turn on LED to indicate association is complete */
        vAHI_DioSetOutput(0, LED1_MASK);
    }
    else
    {
        /* Try, try again */
        vStartScan();
    }
}


// edited by kc. Read from ADC1
PRIVATE void vAdcDataLogger(uint16 *pau16DataBuffer)
{
//      int i;
        /* configure Analogue Peripheral timings, interrupt & ref voltage */
        vAHI_ApConfigure( //E_AHI_AP_REGULATOR_ENABLE,
        E_AHI_AP_INT_DISABLE,
        E_AHI_AP_SAMPLE_2,
        E_AHI_AP_CLOCKDIV_500KHZ,
        E_AHI_AP_INTREF);
//      while (!bAHI_APRegulatorEnabled);
        /* configure & enable DAC */
        /*
        vAHI_AdcEnable(E_AHI_ADC_CONVERT_ENABLE,
        E_AHI_AP_INPUT_RANGE_1,
        E_AHI_ADC_SRC_ADC_1);
        */

    vAHI_AdcEnable (E_AHI_ADC_CONVERT_ENABLE,
                    E_AHI_AP_GAIN_2, /*set input range 0V - Vref */
                    E_AHI_ADC_SRC_ADC_1);



    while (bAHI_AdcPoll());
    /*busy wait until capture complete */
    *pau16DataBuffer=u16AHI_AdcRead();
```

```
//      while(TRUE)
//      {
//              for (i=0;i<u32Length;i++)
//              {
//                      vAHI_AdcStartSample();
                        /* start capture */
//                      while(bAHI_AdcPoll());
                        /* busy wait until capture complete */
//                      pau16DataBuffer[i] = u16AHI_AdcRead();
                        /* store in buffer */
//              }
//      }


}

/****************************************************************************
 *
 * NAME: vTxUARTData
 *
 * DESCRIPTION:
 *
 * PARAMETERS:      Name            RW  Usage
 * None.
 *
 * RETURNS:
 * None.
 *
 * NOTES:
 * None.
 ****************************************************************************/
PRIVATE void vWUART_TxData_ADC(void)
{
    MAC_McpsReqRsp_s  sMcpsReqRsp;
    MAC_McpsSyncCfm_s sMcpsSyncCfm;
    uint8 *pu8Payload, i = 0;
//    int16 i16RxChar;

    //i16RxChar = i16Serial_RxChar();


    uint16 u16ADCBuffer;
    vAdcDataLogger(&u16ADCBuffer);


    //if (i16RxChar >= 0)
    {
        /* Create frame transmission request */
        sMcpsReqRsp.u8Type = MAC_MCPS_REQ_DATA;
        sMcpsReqRsp.u8ParamLength = sizeof(MAC_McpsReqData_s);
        /* Set handle so we can match confirmation to request */
        sMcpsReqRsp.uParam.sReqData.u8Handle = 1;
        /* Use short address for source */
        sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sSrc.u8AddrMode = 2;
        NBO_vFromU16(PAN_ID, sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sSrc.au8PanId);
        NBO_vFromU16(sDeviceData.sSystem.u16ShortAddr,
sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sSrc.uAddr.au8Short);
        /* Use short address for destination */
        sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sDst.u8AddrMode = 2;
        NBO_vFromU16(PAN_ID, sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sDst.au8PanId);
        NBO_vFromU16(COORD_ADDR,
sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sDst.uAddr.au8Short);
        /* Frame requires ack but not security, indirect transmit or GTS */
        sMcpsReqRsp.uParam.sReqData.sFrame.u8TxOptions = MAC_TX_OPTION_ACK;

        pu8Payload = sMcpsReqRsp.uParam.sReqData.sFrame.au8Sdu; /* bpa - make pu8PayLoad
point to payload bytes in frame */
        pu8Payload[i++] = u8TxFrameHandle++;


        pu8Payload[i++] = (uint8)((u16ADCBuffer >> 4) & 0xff ); // Get the top 8 bits of
the 12-bit ADC

        /* Set frame length */
```

```
        sMcpsReqRsp.uParam.sReqData.sFrame.u8SduLength = i;

        /* Request transmit */
        vAppApiMcpsRequest(&sMcpsReqRsp, &sMcpsSyncCfm);
    }
}

PRIVATE void vWUART_TxData(void)
{
    MAC_McpsReqRsp_s  sMcpsReqRsp;
    MAC_McpsSyncCfm_s sMcpsSyncCfm;
    uint8 *pu8Payload, i = 0;
    int16 i16RxChar;

    i16RxChar = i16Serial_RxChar();

    if (i16RxChar >= 0)
    {
        /* Create frame transmission request */
        sMcpsReqRsp.u8Type = MAC_MCPS_REQ_DATA;
        sMcpsReqRsp.u8ParamLength = sizeof(MAC_McpsReqData_s);
        /* Set handle so we can match confirmation to request */
        sMcpsReqRsp.uParam.sReqData.u8Handle = 1;
        /* Use short address for source */
        sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sSrc.u8AddrMode = 2;
        NBO_vFromU16(PAN_ID, sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sSrc.au8PanId);
        NBO_vFromU16(sDeviceData.sSystem.u16ShortAddr,
sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sSrc.uAddr.au8Short);
        /* Use short address for destination */
        sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sDst.u8AddrMode = 2;
        NBO_vFromU16(PAN_ID, sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sDst.au8PanId);
        NBO_vFromU16(COORD_ADDR,
sMcpsReqRsp.uParam.sReqData.sFrame.sAddr.sDst.uAddr.au8Short);
        /* Frame requires ack but not security, indirect transmit or GTS */
        sMcpsReqRsp.uParam.sReqData.sFrame.u8TxOptions = MAC_TX_OPTION_ACK;

        pu8Payload = sMcpsReqRsp.uParam.sReqData.sFrame.au8Sdu; /* bpa - make pu8PayLoad
point to payload bytes in frame */
        pu8Payload[i++] = u8TxFrameHandle++;
        pu8Payload[i++] = (uint8)i16RxChar;

        while (((i16RxChar = i16Serial_RxChar()) >= 0) && (i < MAX_DATA_PER_FRAME))
        {
            /* Set payload data */
            pu8Payload[i++] = (uint8)i16RxChar;

        }

        /* Set frame length */
        sMcpsReqRsp.uParam.sReqData.sFrame.u8SduLength = i;

        /* Request transmit */
        vAppApiMcpsRequest(&sMcpsReqRsp, &sMcpsSyncCfm);
    }
}



/****************************************************************************/
/***        END OF FILE                                                   ***/
/****************************************************************************/
```