School of Electrical and Electronic Engineering, University of Newcastle

MSc Wireless Embedded Systems 2013-2014, Semester 1

EEE8092: Wireless Sensor Network Project

Formal Project Report

Students:

- Nitisha Garole (105861025)
- Abhijeet Patil (130580339)
- Guanqiao Ren (130621045)
- Michael Walker (130623935)

Introduction

Wireless sensor networks (WSN) are the successful, fast evolving technology that is used for the monitoring of physical and environmental conditions such as sound, pressure, temperature and light. Military applications, mainly in battlefield surveillance, have encouraged the development of this field and over the years research in this field has led to a wide range of application of WSNs. Today the application of WSNs ranges from industrial applications to consumers applications. In industry it is used for the monitoring and control of CNC machines. In the environmental domain it is also used for the detection of landslides, forest fires, air pollutants, volcanic activity [1].

The technical focus of wireless sensor networking is mainly on network issues such as MAC (Media Access Control), routing protocols and energy saving. MAC is the data communication protocol layer which provides for channel and addressing access control mechanisms by which multiple terminals or network nodes can communicate within a multiple access medium.

This report explains how a network of mobile sensor nodes for an indoor environment was developed. These nodes communicated with a 2.4 GHz wireless link (IEEE 802.15.4). Each sensor node is comprised of two sensors and was interfaced to a RISC microcontroller. The role of microcontroller would be to gather information from sensor data, communicate with other sensor nodes, implement a network protocol and provide a PC interface via a serial input/output port for data display [3].

Background

Hardware

IEEE 802.15.4 & Zigbee

ZigBee is widely used in wireless networks because of its features of low rate and low power consumption, high fault tolerance, flexibility and autonomy [4]. The foundation of ZigBee led on the development of the standard IEEE 802.15.4 [5], which is a packet based radio protocol. The communication needs for wireless applications was specified in IEEE 802.15.4 standard. ZigBee is built upon the two layers specified in IEEE 802.15.4 standard for low rate WPANs (Wireless Personal Area Networks) those are the Physical layer and MAC (Media Access Control) layer.

The physical and the electrical characteristics of the network are defined by the Physical layer and it is the first (lowest) layer in the seven layer OSI model of computing networking. It involves techniques such as modulation and spectrum spreading that map the bits of information so that they can propagate through earth's atmosphere efficiently with as high a bandwidth and as low power draw as possible. The power requirements of the transmitter and the sensitivity of the receiver are specified in the Physical layer.

The MAC layer defines how the airwaves will be shared when multiple radios are operating in the same area at the same time. There are network association and dissociation functions embedded in MAC layer that supports the self-configuration and peer-to-peer communication features of a ZigBee network.

ZigBee technology is considered to be less expensive and simpler than other WPANs (for example Bluetooth or Wi-Fi). ZigBee has a wide range of application like wireless light switches, traffic management systems, home and office automation, industrial automation, medical monitoring etc. that have short range wireless transfer of data.

Figure 1: A ZigBee chip:

PICDEM Z

The PICDEM Z development board which is developed by Microchip Technology Inc is the principle hardware used in this project. The PICDEM Z kit comes with the main board and a 2.4 GHz wireless transceiver daughter board, the MRF24J40. This can be used to form a simple two nodes wireless network [6]. The PICDEM Z main board consist a 40 pin DIP socket and this socket can be used for a variety of 40 pin DIP package PIC microcontrollers, but the board comes along with a PIC18LF4620 microcontroller. There are two LEDs, D1 and D2 are driven by microcontroller ports RA0 and RA1 respectively, which can also be used for general digital output (to an oscilloscope for example). The motherboard can be connected via serial port for interactive control and debugging using an RS-232 connector. There are two switches S2 and S3 connected to the microcontroller ports RB5 and RB 4 respectively and there is a S1 button which is used to reset the board. The power applied on board is 9V which can be supplied via battery or an appropriate mains power adaptor.

PIC18LF4620

This is a microcontroller which is manufactured by Microchip Technology Inc. This microcontroller offers high computational performance at a very economical price and enhanced flash program memory [7]. This microcontroller has an operating frequency of 4 MHz and is internally an 8 bit system with a Flash programmable processor [3]

The ICD3 programmer can used for programming the microcontroller and the memory used by this microcontroller is EEPROM data memory for a 1 million erase/write cycle. During the code execution the power consumption can be reduced up to 90% by setting the controller into the internal oscillator block, and there are several other features that can reduce the power consumption. Different oscillator options in this microcontroller gives users wide range in developing application hardware [7], and it uses 8x8 single hardware multiplier.

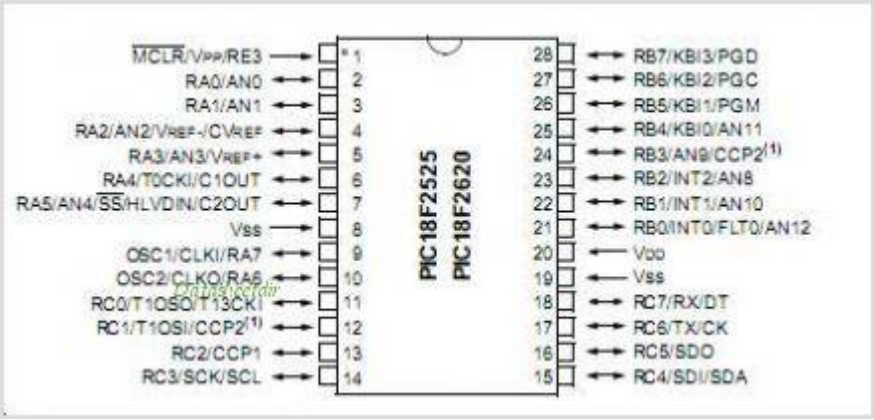

Figure 2: PIC18F4620 Microcontroller [7]

Sensors

Several sensors with minimal circuitry can be interfaced to the PICDEM Z board. The there are header/expansion sockets available on the PICDEM Z board which give access to the processor pins. Out of 13 analog-digital convertor channels on the processor many are assigned for digital input/output to and from other peripherals. There are two sensors used in this project, one is the temperature sensor (thermistor) and the other is a light sensor (photoconductive cell).

Thermistor

Thermistors are made up of semiconductor materials which are sensitive to temperature, so thermistors are temperature sensitive resistors. The resistance of the thermistor varies rapidly with the change in temperature. The temperature sensor used in this project is TC77 digital sensor. The typical resistance of this sensor is 5K at 25^0C and the maximum operating temperature is 10mA and the minimum operating temperature is 1mA. The temperature differences of two plates made up of different elements were measured and the communication is done via SPI/microwire compatible interface.

Photoconductive Cell

Photocells are thin film devices made by depositing a layer of a photoconductive material on a ceramic substrate [8]. Changes in light intensity cause a corresponding change in resistance.

Software

CRC

To ensure the data that arrives at sensor has not been corrupted through collision with the data from another node - or any other interference - it is useful to use a form of error detection. This allows the reliable transmission of data over an otherwise unreliable channel. One of the simplest forms of error detecting codes is the parity bit, where the parity (if an even number of bits of a certain value are in the binary sequence) is calculated and attached to the message. If any single bit of the data is corrupted through interference that can be detected through checking the parity of the sent data if the parity of it has changed the data has been corrupted during transmission. However, if two or any other even number of bits is corrupted during transmission the parity bit will matched that expect and the corruption will go undetected, this is also known as a collision. Even parity checking turns out to be the minimal case of the cyclic redundancy check (CRC) method, where a 1-bit CRC code is generated by the polynomial x+1[9]. For the purposes of this project a more resilient version of CRC should be used rather than a 1-bit CRC.

CRC codes, also called 'polynomial codes' are based on polynomial arithmetic whereby there are no 'carries' used in addition or 'borrows' in subtraction, i.e. everything is carried out modulo 2. Thus in logical terms the XOR gate can be used for both addition and subtraction. To generate a checksum of some binary data, the data is in effect divided by some predetermined polynomial coefficient (some coefficients result in more even distribution of the resulting checksum and are preferred). The division is carried out using an algorithm analogous to standard long division where through continual subtraction (actually XOR'ing) of the message, the remainder of the division is produced and that is used as the checksum.

While normal division is a costly function for a microcontroller to perform, this algorithm implementation is particularly efficient as the whole algorithm requires only two simple CPU instructions; XOR and Shift Left, both available on the PIC18LF4620. The advantage of the CRC code over other forms of error checking is the low likelihood of collisions. These collisions - not to be confused with wireless collisions - occur when the algorithm still generates a valid CRC on a corrupted data packet.

Wireless Networking Protocols

While the CRC ensures that bad data is ignored by the system, it cannot prevent the node from causing bad data itself. Wireless by its nature requires the implementation of protocols for communicating over shared channels to avoid collisions that would result in scrambled data. Due to the project requirement of using only one frequency for all communications by the sensor network, only time based protocols will be discussed, although frequency based solutions to the packet collision problem such as FDMA can also be used and is particularly relevant in satellite communication applications.

Time Division Multiple Access (TDMA)

The techniques by which a finite amount of radio spectrum is shared by many users without any severe degradation in the performance of the system are known as Multiple Access schemes. There are different methods for allowing Multiple Access to a medium, but the one used in this project is TDMA. TDMA produces a shared network medium, where the same channel is shared by several users so the signal is divided into a repetitive structure called TDMA frame which consist of numerous time slots. The user uses the time slots allocated to them while transmitting the data into the traffic channel and it is in order of one after the other [10].

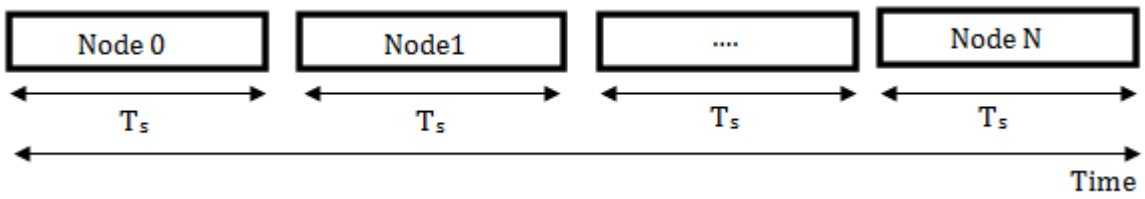

Figure 3: TDMA Frame Structure

TDMA is a simple yet reasonably efficient form of MAC protocol. However, TDMA also comes along with disadvantages like being prone to multipath losses; problems caused due to echo, and radio wave scattering in outdoor environments.

Timing synchronisation in each node is necessary to successfully implement TDMA [3]. There are different methods by which one can achieve this:

Master Mode TDMA
When a TDMA frame begins a beacon message is send out periodically which signals all the nodes in the network. This beacon message is the concern of a single 'master' node that is responsible for the coordination of timing. Before starting the transmission each node waits for an appropriate delay i.e. kTs where k is the node number and T is the time slot duration [3].

Token Passing
In this method one does not need to use a coordinator node. The nodes in the network operate in the sequential order so the node can wait and listen to the transmission of the node previous to it in the cycle before it can transmit its own data. It requires a fixed number of nodes to operate efficiently and is very easy to implement. The system can be made more robust if the time of the node is calculated from any other node rather than the previous node. Any node can also trigger the start of the network cycle by itself if it doesn't receive a transmission from a previous node in an allotted time.

Carrier Sense Multiple Access (CSMA)

CSMA is related but different technique whereby certain terminal capabilities are exploited to obtained better performance. 'Carrier sense' refers to the feedback from the receiver which is used by the transmitter to find if another transmission is in progress before it starts its own transmission. CSMA is based on the principle of *sense before transmit* because of this necessary feature to detect the presence of another wave in the carrier before it transmits it own signal. If a signal in the carrier is detected then it waits a random time period before reattempting to send its packet.

Hidden Node Problem

One of the important aspects of the wireless environment is that the transmitter and the receiver should be within a specified distance from each other so that the information delivered to the receiver is reliable. Assume there are three terminals *a, b, c* where *a* can deliver to *b* but not *c*, similarly *c* can deliver to *b* but not *a*. This may be due to distance and range limitations, or a physical block to signal transmission. If *b* is transmitting to *a*, *c* can sense the transmission, likewise if *b* transmits to *c*, *a* can sense the transmission. However, if *a* sends to *b* even if *c* tries to listen, it will not hear *a's* transmission. If at the same time *c* tries to transmit a packet there will be a collision if standard CSMA protocol is implemented, this is known as *hidden node problem*. This will lead to a dramatic decrease in the throughput when there are large numbers of packages. The hidden node problem can be solved using a variety of techniques such as polling and token passing [11].

System Design

Sensory Input

Sensor Circuit Design

The two sensor components – the thermistor, for measuring temperature and the photoconductive cell, for measuring light intensity – were integrated into an analogue electrical circuit on a stripboard substrate. The circuit utilised the principals of voltage division such that the voltages on the two sensory output wires of the circuit were around half the supply voltage of 3.3v during normal conditions of temperature and lighting. Care was taken to ensure the voltages varied over as broad a range as possible when the respective stimuli of each sensor was changed.

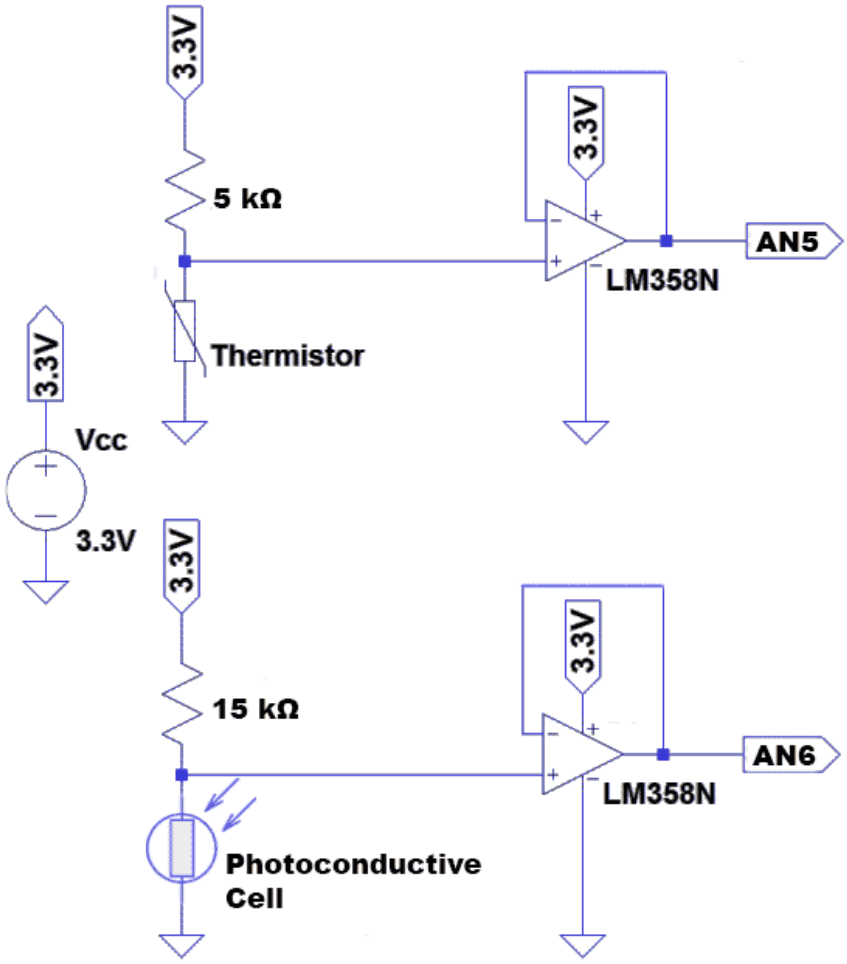

Figure 4: Circuit Schematic for the Sensor Package

For the purposes of energy efficiency, and to avoid damaging the components, both of the sensory devices were decoupled from the rest of the system through the use of an Op-Amp as a buffer. The Op-Amp was used in its unity gain configuration whereby its output was fed-back into the inverting input without a separate resistance and with no path to ground, as there was no need for any amplification of the signal.

The NTC TTC502A disc thermistor supplied has a specified resistance of 5kΩ at a temperature of 25°C [12], which is a good approximate value for average room temperature. To achieve an output of half the supply voltage a voltage divider linear circuit element was used with a matching 5kΩ fixed resistor on the input side of the divider. This means that at 25°C the output of the circuit is half the supply voltage of 3.3v, or 1.65v.

At higher temperatures the resistance across the thermistor decreases so that the voltage on the output drops, and the reverse is true for lower temperatures. Both changes are quite significant enough to be easily detectable by a voltmeter or ADC. The thermistor has a mostly linear response across the temperature range that the sensor node as a whole can withstand so any gain from the op-amp would be unhelpful as an increase to gain would make the output vary with increased non-linearity. For this reason, ultimately no attempt was made to 'boost' the signal strength through an Op-Amp gain, although this was investigated[13], and given the Op-Amp package used was not capable of 'rail-to-rail' voltage swing, full use of the 0-3.3v range was not possible, only a maximum of 0.75v – 2.25v was, which fitted well with the range of voltage division output without amplification quite well.

The TruOpto CE5-20AP09 Photoconductive cell has a specified resistance of 50-100kΩ at a light level of 75 lux[14], this is quite a dark level of light, and through further investigation into the properties of the sensor a mid-point voltage was found to occur under daytime laboratory lighting conditions when a 15 kΩ fixed resistor was used in a similar voltage divider circuit as used for the thermistor. However, unlike the thermistor, the response of the cell was not linear but exponential. After investigation and experimentation it was not deemed appropriate to give a lux value output as the component chosen is not sufficiently accurate enough to benefit from the extra circuitry and/or processing time/power required to linearize the voltage response. Therefore it was deemed acceptable to have the output non-linearly scaled, as all that was required realistically from a basic light sensor was to detect 3 possible conditions; darkness, normal room light conditions and the bright conditions of outdoor sunshine or a flashlight being aimed at the sensor.

The Op-Amp package used in the circuit is the ST LM358N, containing two independent operational amplifiers [15]. Each of the sensor circuits used one as a buffer for the purposes of decoupling the sensor. Unity gain on both amplifiers was used together with usage of the standard supply voltage of 3.3v.

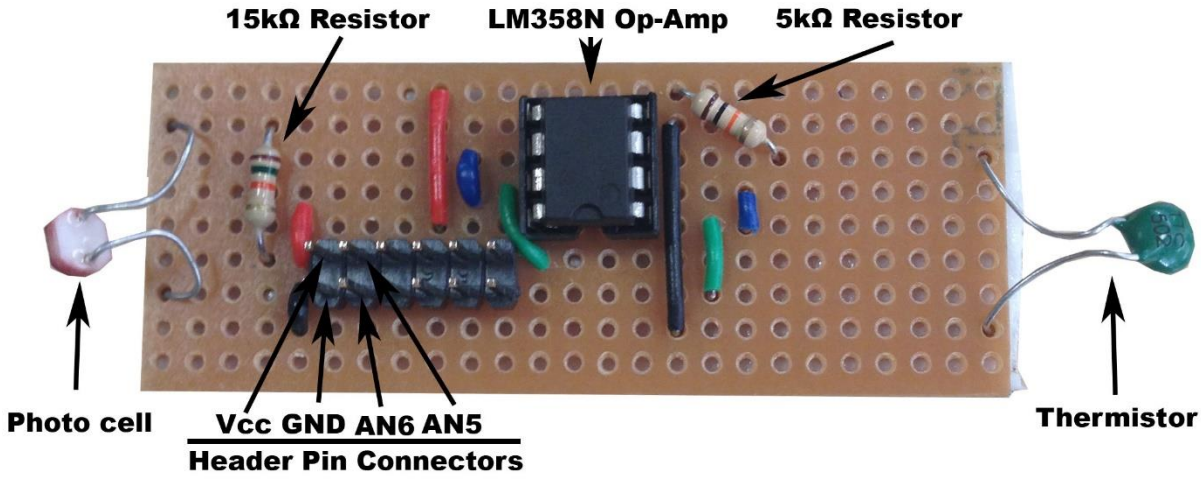

Figure 5: Implementation of Sensor Package

ADC Interfacing & Calibration

The 2 sensors outputs of each sensor package (4 were made, 1 for each sensor node) were connected to the Analogue-Digital Converter on the PIC chip via the AN6 (for temperature) and AN5 (for light) pins on the J7 connector of the main board [6]. As the two pins and the ground and VCC supply pins are all available on this one smaller header it was deemed suitable to create a design in which a small 6x2 header strip was soldered to the stripboard such that the sensor package could be securely attached the main board. The solder side was face up and so for protection it was covered in a white, padded anti-static sheet. This arrangement worked well and also allowed for much easier handling of the device during testing.

Figure 6: Main board with sensor package unattached (left) and attached (right)

The ADC conversion process was separated into two functions in the code for the microcontroller. The function for Temperature acquisition; `GetTemp()` used a standard four line sequence to open the ADC with parameters suitable to sensor input. Fast operation was explored, found possible and tested, but not considered necessary, so the ADC used a conservative clock source 8 times slower than the internal clock with a longer 16 Tad data acquisition time, and right justification.

```
OpenADC(ADC_FOSC_16 & ADC_RIGHT_JUST & ADC_16_TAD,
            ADC_CH6 & ADC_INT_ON & ADC_VREFPLUS_VDD & ADC_VREFMINUS_VSS,
            7);
Delay10TCYx(5);
ConvertADC();
while(BusyADC());
```

With the 10-bit digital value generated, the conversion process into the corresponding temperate or light value had to be calibrated. Using the line equation $y = mx + c$ the temperature (y) for a particular voltage reading (x) was found by finding the ADC value reported against two known temperatures. The temperatures used were ambient room temperature and that caused by holding the thermistor between human fingers. Both the temperatures were measured by appling a small temperature probe (N84FR) to the thermistor.

Figure 7: Calibration of the thermistor by measuring ADC value reported at two known temperatures.

By using the known temperatures two fixed points of the line could be found by solving the two derived simultaneous equations. Doing this for all nodes showed that to report an accurate temperature the value reported by the ADC needed to be multiplied by a slope (m) of 0.55 (±0.3). As this value varied remarkably little between the different sensor packages, all nodes used the same

mean value 0.55. The offset of the line equation (c) varied slightly more for each sensor package, but was found to be 78(±5). The variances in the offset are therefore stored in a small array and used to tailor the calculation for each node. A 10-bit value was stored as an integer type (16-bit) by the ADC function. For output it was deemed expedient to shorten this value to a char type of 8 bits as the extra precision was not required. This was achieved by left shifting the contents of the integer value by two to remove the least significant bits and then recasting the small integer value (16-bit) to char type (8-bit).

The function used for light values; `GetLight()` followed an identical process for ADC data acquisition (expect using Channel 5 instead of Channel 6). However the calibration was simplified as truly accurate lux readings were not possible with the sensor component used. It was found through experimentation that for all sensor packages a digital value for the ADC converter lower than 16 (2^4) was a good indication of bright light conditions. Conversely a value higher than 64 (2^6) was a good indication that the sensor was in shadow. Note the use of powers of two, to account for the fact that the sensor reading vary with the exponent of the digital value.

The two sensor values were then used to update a global variable `svPacket` used for recording the latest reading. This was then used in other parts of the system, specifically when preparing the data packet sent to the other nodes.

Transmission Reliability

CRC generation

The CRC generation algorithm used in the system is based on a lookup-table approach for increased speed. The algorithm uses an array of 16 pre-computed values as a look up table and for each nibble (4-bits) of the data the lookup is used in the calculation of the current checksum/remainder of the long division process. By using a 4-bit approach a smaller lookup table is required, an 8 bit table would result in a quicker CRC calculation, and would theoretical fit (taking up 2 kilobytes of the chips 4k of memory), but the performance improvement is not required. Whereas data memory is a more precious resource and must be used for other things. The algorithm uses the XMODEM poly (0x8408) to generate the CRC, as this has good coverage, and we used the 'reflected' variation of the CRC algorithm [16].

CRC checking

The CRC is first generated by the sender running the algorithm against the transmission packet that is about to be sent, but with a zero value stored in the packets '`CRC`' entry. The results of generation are then stored in the '`tCRC`' ('t' for 'transmitted') variable and sent. When received the transmitted CRC is stored in a separate variable used to store the '`tCRC`' first. Then '`CRC`' in the packet can be reset to zero and another generated CRC is produced called the '`gCRC`' for 'generated CRC'. The generated and transmitted CRC values are compared. Only if they are equal is the packet considered valid and stored to be used in the terminal output. Otherwise it is dropped and the system simply returns to its sleep state.

Networking Protocols

Timers

The PIC microcontroller used contains 4 timer modules. These can be used to periodically generate interrupts that can wake the microcontroller up from the sleep state the system returns to after every action. The Template given to us used one timer, however to both allow the system more flexibility so that it can process the sending of wireless packets, the receiving of wireless packets and the writing results to the terminal interface as separate distinct tasks another timer is required.

Timer0 was set to provide a 1 second interrupt for the transmission the latest result received to the serial terminal. This was done by increasing the pre-scale of a 16-bit count to the maximum 16 to give a maximum count of approximately 1.05 seconds based on an internal clock frequency of 4 MHz. By

starting the timers counts not at zero but at a value of 3035, a relatively accurate 1 sec period timer can be setup to control the terminal output.

The Wireless MRF24J40 board also sends an interrupt when a packet is received, so this functionality doesn't need a timer, but timing the sending of packets does. For this purpose Timer1 is set to provide a much quicker clock than Timer1. It uses no pre-scale (1:1) of a 16-bit count, such that it 'fires' roughly every 65 ms. Timer1 has a maximum prescale of 8, so it is not suitable for the longer timer, this is why it is used for the shorter duration and not the other way around.

TDMA protocol

The TDMA protocol was implemented using a token passing algorithm in the submitted form of the project, while 'Beacon Node' functionality was also investigated during development. Each interrupt by Timer1 causes the system to generate a packet, but the system then tests whether it should send it wirelessly. The function used to check whether to send is `TDMAReady()` and consists of code to check three conditions. First it checks if the node has just received a packet from the previous node in the cycle sometime since it last checked then it simply transmits its data. The cycle being: node 1 then node 2 then node 3 then node 4 then back to node 1. If it hasn't received a packet from the previous node it mostly waits. However it only waits until it has counted the same number of interrupts as the maximum number of nodes that can be running (4). If this occurs all other alternatives are disregarded, including what, if any, node transmitted most recently and the node simply transmits its packet. In this way it is ensured that every node will transmit a packet at some point.

Further development of the protocol was carried out to make the system more robust by having a further wait counters based on when other nodes in the cycle sent data rather than the previous node. However this implementation caused a hard to track intermittent time-bug and so was removed from the final system prior to full testing.

CSMA protocol

The strength of the channel is tested using the `PHYGetRSSI()` function, if this is lower than 10 (value found through experimentation to be a good dividing line on the presence of a free channel) then a packet is sent. The node now uses similar code to that in TDMA to wait until all other nodes have had a chance to send their packets before checking to send again. If the channel is detected to be being used at that point, the node it set to wait a random time. This random value is generated using the function `rand()` modulo the number of nodes in the cycle. This value is used for the wait counter so that, in the case of 4 nodes in the cycle, the node waits either 1, 2, or 3 timeslots rather than 4.

Serial Terminal Output

When a packet is received from another node in the system and it passes the CRC, its contents are replicated in an element of `SvPacket[4]` a storage array of C structs with the same type as the packet used in transmission that is as large as the number of nodes in the system (4). Also the node that the packet was received from is marked as active by storing a positive value ('1') in another associated array `ActiveNode[4]` with the same number of elements. In both cases the element of the array updated is that with the Node ID (Minus one due to c arrays starting at 0) of the node that the packet was received from.

Every one second the Timer0 interrupt for terminal output will occur. At this point the `TerminalOut()` function is called and will loop through the `SvPacket` array and output the latest values stored to the terminal using the `USARTOut()` function. After it has finished it calls the `ResetActiveNodes()` function, which will store a zero to all the elements in the `ActiveNode` array. This indicates that the contents is no longer up to date. If a packet from a node is not received within one second and the interrupt fires again, that node will still be marked as a zero in the `ActiveNode` array. In this case, instead of listing the most recent temperature and light readings stored, the `TerminalOut()` function lists it as '`INACTIVE NODE`' in its output.

Results

Sensor Calibration

The terminal output reports the contents of the `svPacket` variable where results from the nodes own sensors and those received wirelessly from other sensors are stored.

The terminal output shows the results of the sensor calibration. Here nodes 1, 2, and 3 are reporting similar values (although not the same – this disparity seems somewhat unavoidable as leaving the sensors for a few hours seems to allow their resistances to drift somewhat), whereas node 4 is reporting a higher temperature due to being held in between a person finger and thumb.

```
GroupID = 23 NodeID = 1 Temp = 20°C Light = Normal RSSI = 152
GroupID = 23 NodeID = 2 Temp = 22°C Light = Normal RSSI = 168
GroupID = 23 NodeID = 3 Temp = 24°C Light = Normal RSSI = 165
GroupID = 23 NodeID = 4 Temp = 35°C Light = Normal RSSI = 0

GroupID = 23 NodeID = 1 Temp = 20°C Light = Normal RSSI = 160
GroupID = 23 NodeID = 2 Temp = 22°C Light = Normal RSSI = 166
GroupID = 23 NodeID = 3 Temp = 24°C Light = Normal RSSI = 161
GroupID = 23 NodeID = 4 Temp = 35°C Light = Normal RSSI = 0

GroupID = 23 NodeID = 1 Temp = 20°C Light = Normal RSSI = 158
GroupID = 23 NodeID = 2 Temp = 22°C Light = Normal RSSI = 166
GroupID = 23 NodeID = 3 Temp = 24°C Light = Normal RSSI = 168
GroupID = 23 NodeID = 4 Temp = 35°C Light = Normal RSSI = 0

GroupID = 23 NodeID = 1 Temp = 20°C Light = Normal RSSI = 155
GroupID = 23 NodeID = 2 Temp = 22°C Light = Normal RSSI = 165
GroupID = 23 NodeID = 3 Temp = 24°C Light = Normal RSSI = 173
GroupID = 23 NodeID = 4 Temp = 35°C Light = Normal RSSI = 0
```

Figure 8: Terminal output, node 4 thermistor held

The terminal output is given in groups of 4 lines, one line per node. With an empty line to separate each serial output that occurs once per second.

The Light Sensor is also capable of reading whether it is in shadow, under normal lighting or in bright conditions:

```
GroupID = 23 NodeID = 1 Temp = 20°C Light = Bright RSSI = 147
GroupID = 23 NodeID = 2 Temp = 22°C Light = Normal RSSI = 93
GroupID = 23 NodeID = 3 Temp = 24°C Light = Shadow RSSI = 138
```

Figure 9: Light sensor readings

Here can be seen the effect of shining a bright light source (the flash light of a mobile phones camera) on node 1, while simultaneously holding a cupped hand over node 3 so that is was in shadow. Both conditions are reported correctly.

When the nodes are turned off, the system reports them as inactive nodes if it does not receive new data from them within the one second period between sending results to the serial output.

```
GroupID = 23 NodeID = 1 Node Inactive
GroupID = 23 NodeID = 2 Node Inactive
GroupID = 23 NodeID = 3 Node Inactive
GroupID = 23 NodeID = 4 Temp = 35°C Light = Shadow RSSI = 0
```

Figure 10: Inactivity reporting

In figure 10 we can see the results of turning off nodes 1,2 and 3 when the terminal cable is connected to node 4. (note: the anomalous high temperature is a result of only just being used to show the temperature calibration diagram in figure 8.)

Transmission Timing

The first thing investigated during development was the timing of a single node transmitting with no TDMA or CSMA protocols and then using the serial port to communicate results. The following graphs where produces using PicoScope6, which was attached to the header of the main board such that it probed the same wire as the 2 LED outputs on the device, which were used in the code to show packet transmission and packet reception respectively. In the diagrams, transmission of a packet is shown in Blue and reception of one is shown in red.

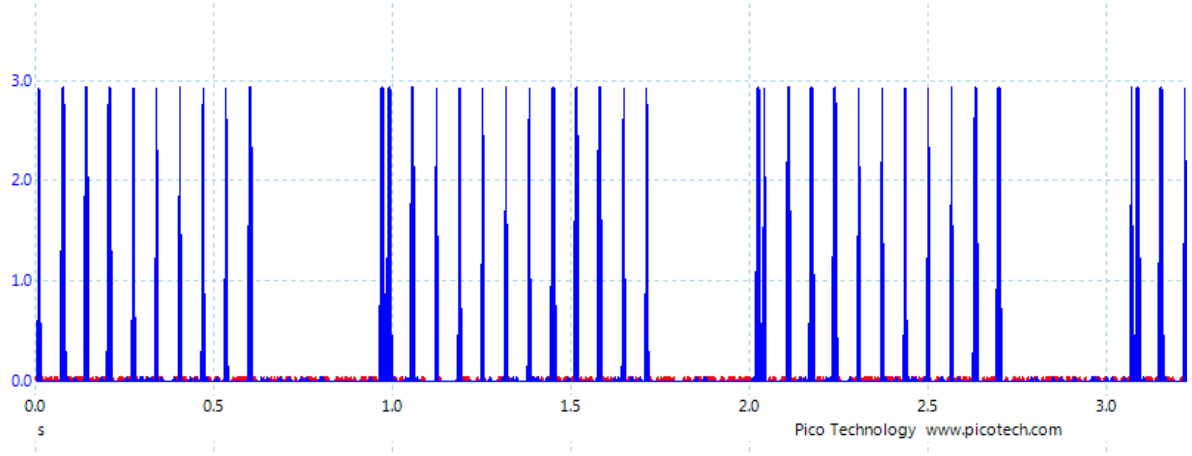

Figure 11: 1 Node, no protocol, 1:1 prescalar

Here we can see the pause in sending caused by the terminal out routine that runs once per second, however the node is still able to send 10 results in this time, which is adequate for the TDMA protocol. The timer used fires every 65ms, which means the ten sends correspond to 0.65 seconds, this is evident in the oscilloscope reading above. The terminal output seems to therefore take the remainder of the time, i.e. 0.35 seconds.

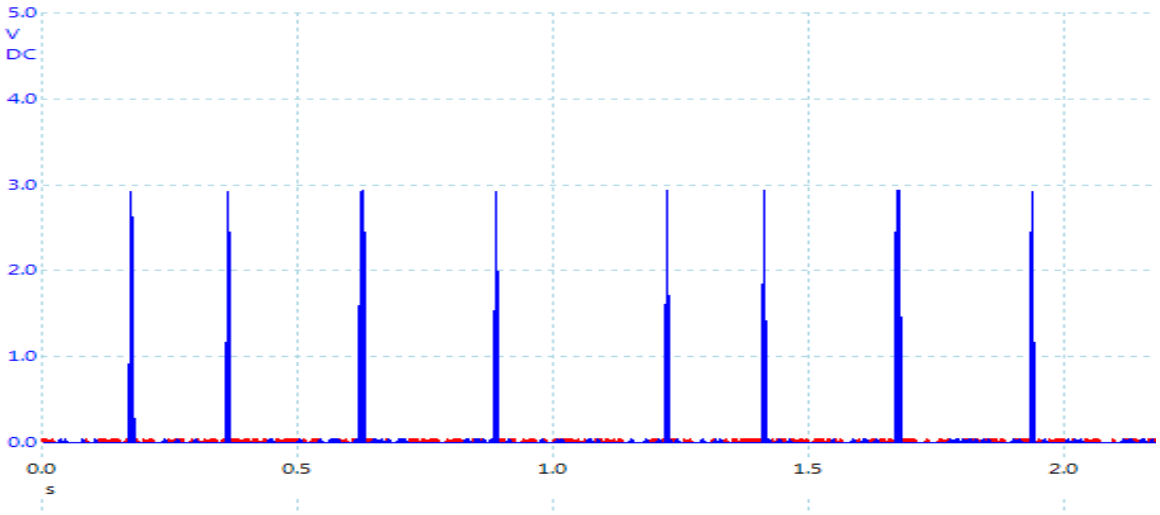

Figure 12: 1 node, no protocol, 1:4 prescaler

We can see that the node is sending only 4 results in the time between the serial output function is run. This is the bare minimum required for accurate use of a cyclic time division with four nodes, and is unlikely to be very robust, therefore the 1:1 prescaler was used for all protocol testing.

When four nodes try to communicate over the same channel the following was found to result:

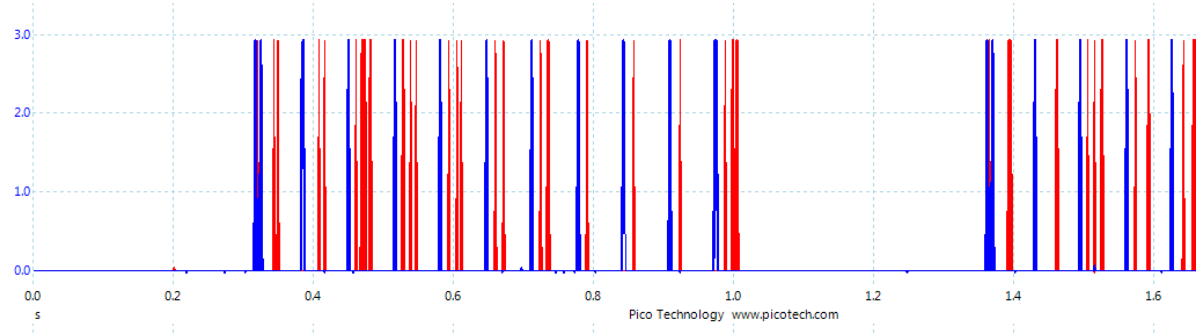

Figure 13: 4 nodes, no protocol, 1:1 prescaler

Here we can see that at the start three packets were received for every one sent from 0.3 to 0.6 seconds. After a while, signal skew due to slightly different node times cause collisions such that only one other signal is received during the period from 0.8 to 1.0 seconds. Without a protocol, packet collisions are unavoidable; hence we now look at testing the functionality of the TDMA protocol implementation:

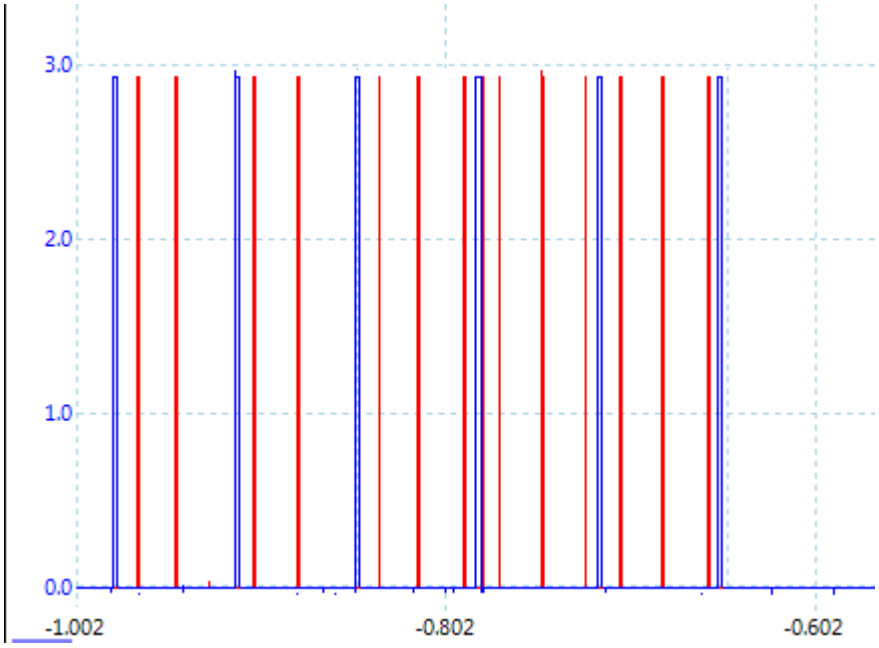

Figure 14: 4 nodes, TDMA

Here we can see a reading of the output using TDMA protocol. This is quite interesting as it shows the node being added into the cycle just after the point marked -0.802. This is the normal way the protocol works, however it is unclear exactly which node is which in this diagram. For that reason the code was modified to add an extra spacer to the cycle (a non-existent 5th node), thus we can better see the cycle:

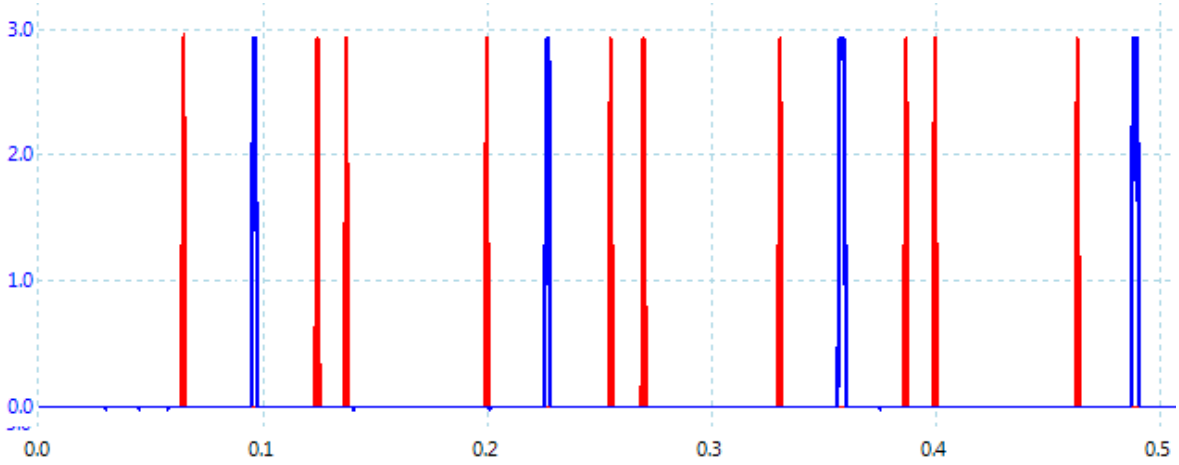

Figure 15: 4 nodes, TDMA with spacer timer, Node 2 output

In this graph we can see the result of running the TDMA protocol on all four nodes, but with 5 time slots, to distinguish the nodes, and tell which the first node is. The output of the second node in the cycle is show. It can be seen that the node received packets from the first, third and fourth nodes in the correct order, with the second node itself transmitting at the correct time. This shows that correct timing is observed when using the TDMA protocol.

CMSA uses a different protocol to attempt to ensure collisions are avoided. Below is a graph of our implementation of CSMA working with 4 nodes:

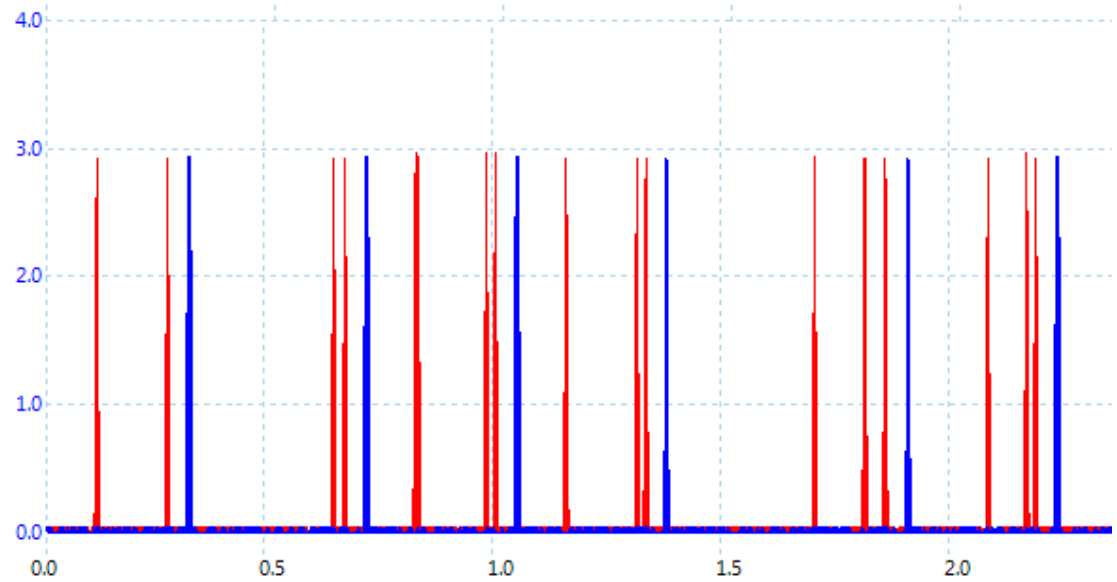

Figure 16: CSMA

After the first half second (where a missed packet is evident) the cycle of 4 nodes sending sequentially can be seen, with three red 'receive' signals for every one blue 'transmit' signal. Thus under CSMA the network can be seen to be working as intended.

Discussion

The sensor package seems to have worked adequately for general, relatively imprecise needs. However other sensor types could have been used to ensure greater accuracy in the results. By limiting the values outputted by the light sensor to only 3 possible values the result was greater reliability and repeatability at the expense of precision and accuracy.

The temperatures reported by the 4 different thermistors were often found to be out by one or two degrees when testing a day or two after calibrating them with their own offsets to report the same temperature. It is believed that more accurate results may be possible with thermistors, however if greater accuracy was required it would be better to implement the sensor using a different component, such as a resistance thermometer (RTD).

From the results section we can see the importance of MAC protocols in the prevention of collisions and the ensuring of reliability of the communication channel. Figure 13 shows that without a protocol collisions cause the loss of many data packets in an intermittent and unpredictable way. By adhering to a scheme that rationally splits the shared media up into separate 'slots' (be they time slots as is the case here, or frequency bands etc.) greater reliability can be assured. This comes at the cost of bandwidth as there is wasted time associated with the idle times nodes wait through for available time slots in TDMA or waiting for a random time in CSMA.

By using either TDMA or CSMA the problems of collision seem to be mostly avoided. When the protocols are running, the nodes gradually fall into a fairly efficient pattern. However, the terminal output routine disrupts the cycle too often for much larger testing to be done.

It would be better to have a shorter time period dedicated to terminal output as it takes up over a third of systems processing time. However the requirement stated that the system should report the latest reading at least once per second and so this was the maximum idle period it could wait. Further investigation in shortening the terminal output run time would be of interest.

The ability to avoid the hidden node problem [17] was explored, but functionality to support this was not added due to timing constraints.

Conclusions

In creating a 4 node sensor network using wireless PAN for data transmission a great deal was learnt about microcontroller software development using the C programming language. The C language is a low-level system programming language suitable to microcontroller development. The use of strong types and pointer manipulation was of considerable interest in comparison to higher level languages that come with features such as garbage collection and duck typing. The rigour required to ensure code correctness was enlightening, as was the techniques of microcontroller debugging using the MP Lab IDE.

By interfacing analogue sensor devices to the microcontroller via an ADC we discovered the fundamentals of how the digital sensing equipment used in industry and society in general is designed and produced. While the soldering and wiring tasks were fairly basic, the functional design problem of making sensor packages that fitted to the main board cleanly was instructive.

In investigating two different multiple access protocols an appreciation for Medium Access Control methods was gained. The issues of timing and protocol robustness where explored and found illuminating and instructive to the needs of real world MAC protocol design.

References

1. Hejlová V. & Voženílek V., "Wireless Sensor Network Components for Air Pollution Monitoring in the Urban Environment: Criteria and Analysis for Their Selection" Scientific Research 2013, 5, 229-240
2. Krco S. & Tsiatsis V., "Mobile Network Supported Wireless Sensor Network Services" IEEE 2007
3. Notes
4. Yu chengbo & Liu Yanfei "An Improved Design of ZigBee Wireless Sensor Network" IEEE 2009
5. "An Introduction to ZigBee" www.rabbit.com , 2008
6. "PICDEM™ Z Demonstration Kit User's Guide", www.microchip.com
7. "PIC18F2525/2620/4525/4620 Data Sheet", www.microchip.com
8. "Photoconductive Cells", Ada Fruit Industries (learn.adafruit.com)
9. Tanenbaum, A.S., "Computer Networks 3rd Edition" pg 187
10. Cooper D., "Multi-user diversity in a TDMA cellular system" ,IEEE 2010
11. Georgiadis L. "Carrier-Sense Multiple Access (CSMA) Protocols", FEB 2002
12. "NTC Disc Thermistors" Datasheet, (www.rapidonline.com)
13. "Scaling Sensor Voltages", Electrotap (electrotap.com/2004/11/01/op-amps-part-1-scaling-sensor-voltages)
14. "TruOpto CE5-20AP09 Photoconductive cell" Datasheet, , (www.rapidonline.com)
15. "LM 158, 258, 358 Low-power dual operational amplifiers" Datasheet, , (www.rapidonline.com)
16. Williams Ross N., "A Painless Guide to CRC Error Detection Algorithms" (repairfaq.org)
17. Gast M., "802.11 Wireless Networks: The Definitive Guide", O'Reilly Press (page 35)

Appendix

Note the use of a separate header file '`picconfig.h`'. In this file is placed the large section of 'DEFINE' lines previously in the master file to increase readability.

```
#include <xc.h>
#include <stdlib.h> //standard library
#include <stdio.h>
#include <plib/spi.h> //serial peripheral interface functions
#include <plib/delays.h> //time delay functions
#include <plib/usart.h> //USART functions
#include <string.h> //string functions
#include <plib/timers.h> //timer functions
#include <plib/adc.h> // Analog to Digital Converter functions
#include "picconfig.h" // Removed Defines from main program
#include "MRF24J40.h" //driver function definitions for MRF24J40 RF transceiver

#define NUMNODES 4
//*****************************************************************************
// Constants
const unsigned char ThisNode = 4; // Change when compiling for each node (1,2,3,4)
const unsigned char NumNodes = NUMNODES;
const unsigned char OurGroup = 23;
const unsigned char OurChann = CHANNEL_17;

const unsigned int  OneSecDelay = 3035;

enum protocol {None, TDMA, CSMA};
const unsigned int  OurProto = CSMA;

enum light {Bright, Normal, Null, Shadow};
const char *lightstr[] = {"Bright\0","Normal\0","Nullch\0","Shadow\0"};

//*****************************************************************************

//*****************************************************************************
//** Function to initialise all I/O resources used by Processor ************
//*** THIS MAY BE MODIFIED AND EXPANDED WITH STUDENTS' CODE            ********
//*****************************************************************************
void Init_IO(void) {
    PORTA = 0x04; //PORTA initially all zeros except RA2 (TC77 chip select)
    TRISA = 0xF8; //RA0 and RA1 outputs (LEDs), RA2 Output (TC77 CS), rest inputs
    TRISB = 0xFF; //PORTB all inputs (RB0 is interrupt, RB4 and RB5 are push
buttons)
    INTCON2bits.RBPU = 0; //enable pull up resistors on PORTB
    PORTCbits.RC0 = 1; //Chip select (/CS) initially set high (MRF24J40)
    TRISCbits.TRISC0 = 0; //Output: /CS
    PORTCbits.RC1 = 1; //WAKE initially set high (MRF24J40)
    TRISCbits.TRISC0 = 0; //Output: WAKE
    PORTCbits.RC2 = 1; //RESETn initially set high (MRF24J40)
    TRISCbits.TRISC2 = 0; //output: RESETn
    ADCON0 = 0x1C; //turn off analog input

    // timer 0 setting
    INTCONbits.INT0IF = 0; //clear the interrupt flag (INT0 = RB0)
    INTCONbits.INT0IE = 1; //enable INT0
    // timer 1 settings
    PIR1bits.TMR1IF = 0;
    PIE1bits.TMR1IE = 1;
```

```
    RCONbits.IPEN = 1; //enable interrupt priorities
    INTCONbits.GIEH = 1; //global interrupt enable
    OSCCONbits.IDLEN = 1; //enable idle mode (when Sleep() is executed)

    OpenSPI(SPI_FOSC_4,MODE_00,SMPMID); //setup SPI bus (SPI mode 00, 1MHz SCLK)
(MRF24J40)
    OpenUSART(USART_TX_INT_OFF & USART_RX_INT_OFF & USART_ASYNCH_MODE &
                          USART_EIGHT_BIT & USART_CONT_RX & USART_BRGH_HIGH,25 );
//setup USART @ 9600 Baud
}
//******************************************************************************
//** Function to output an array of bytes to the USART (RS232 port) ********
//** MUST NOT BE CHANGED BY STUDENTS                                 ********
//******************************************************************************
void USARTOut(char *data, char bytes) {
    int i;
    for(i=0; i<bytes; i++) {
        while(BusyUSART());
        WriteUSART(data[i]);
    }
}

//******************************************************************************
//** Example of simple data packet structure                         ********
//******************************************************************************
typedef struct {
    unsigned char GroupID; // Group Number (should be 23)
    unsigned char NodeID;  // Unique Identifier for this device
    unsigned char Temp;    // Reading on the thermistor
    unsigned char Light;   // Reading on the light sensor
    unsigned int  CRC;     // Cyclic Redundancy Check
    unsigned char Strength;// Keep record of how strong received signal was
} PacketType;

//******************************************************************************
//** Global Variables                                                ********
//******************************************************************************
PacketType TxPacket,RxPacket; //transmitted and received packets
PacketType SvPacket[NUMNODES]; // Storage for received packets
char Text[128]; //buffer for temporary strings
unsigned char Strength;
unsigned char LastReceived = 0;
unsigned char WaitCounter = 0;
unsigned char ActiveNode[NUMNODES];
// Sub Function Declarations (see beneath main for implementation)
void SetupPacket();
void SendPacket();
unsigned char ReceivePacket();
void TerminalOut();
void InitDataStore();
void NoneLoop();
void TDMALoop();
char TDMAReady();
void CSMALoop();
char CSMAReady();
void CSMAWaitRandom();
void ResetActiveNodes();
unsigned char getTemp();
```

```
unsigned char getLight();
unsigned int  getCRC(PacketType packet);

//*********************************************************************
//*** MAIN function of program - transmits or receives text message ********
//*** THIS WILL BE MODIFIED AND EXPANDED WITH STUDENTS' CODE           ********
//*********************************************************************
void main (void) {
    Init_IO(); //initialise all processor I/O resources used by the program
    MRF24J40Init(); //initialise IEEE 802.15.4 transceiver
    SetChannel(OurChann); //set RF channel CHANNEL_11-CHANNEL_26 (EACH GROUP MUST
HAVE UNIQUE CHANNEL)

    // Intialise Timers
    OpenTimer0( TIMER_INT_ON & T0_16BIT & T0_SOURCE_INT & T0_PS_1_16); //setup
timer 0 with prescaler x8
    OpenTimer1( TIMER_INT_ON & T1_16BIT_RW & T1_SOURCE_INT & T1_PS_1_1 &
T1_OSC1EN_OFF & T1_SYNC_EXT_OFF);
    InitDataStore();

    switch (OurProto) {
        case TDMA: TDMALoop();
        case CSMA: CSMALoop();
        default: NoneLoop();
    }
}

void InitDataStore() {
    char i;
    for (i=0;i<NumNodes;i++) {
        SvPacket[i].GroupID = 23;
        SvPacket[i].NodeID  = i + 1;
    }
}

void NoneLoop() {
    while(1) { //infinite loop
        if (INTCONbits.TMR0IF) {
            INTCONbits.TMR0IF = 0;
            WriteTimer1(OneSecDelay);
            TerminalOut();
        }
        if (PIR1bits.TMR1IF) {
            PIR1bits.TMR1IF = 0;
            SetupPacket();
            SendPacket();
        }
        if (PHYReceive((char *)&RxPacket,&Strength) == sizeof(PacketType)) {
//check for received RF packet and validate size
            ReceivePacket();
        }
        Sleep(); //put processor into idle mode and wait for interrupt
    }
}

void TDMALoop() {
    ResetActiveNodes();
    while(1) { //infinite loop
        if (INTCONbits.TMR0IF) {
```

```
            INTCONbits.TMR0IF = 0;
            WriteTimer1(OneSecDelay);
            TerminalOut();
        }
        if (PIR1bits.TMR1IF) {
            PIR1bits.TMR1IF = 0;
            SetupPacket();
            if (TDMAReady) SendPacket();
        }
        if (PHYReceive((char *)&RxPacket,&Strength) == sizeof(PacketType)) {
//check for received RF packet and validate size
            LastReceived = ReceivePacket();
            if (ActiveNode[LastReceived - 1] == 0) ActiveNode[LastReceived - 1] =
1;
        }
        Sleep(); //put processor into idle mode and wait for interrupt
    }
}

void CSMALoop() {
    srand(ThisNode);
    ResetActiveNodes();
    while(1) { //infinite loop
        if (INTCONbits.TMR0IF) {
            INTCONbits.TMR0IF = 0;
            WriteTimer1(OneSecDelay);
            TerminalOut();
        }
        if (PIR1bits.TMR1IF) {
            PIR1bits.TMR1IF = 0;
            if (CSMAReady()) { // Wait one cycle (Total Nodes * Time Slot
Duration)
                SetupPacket();
                Strength = PHYGetRSSI();
                // if channel is free send the packet
                if (Strength < 10) {
                    SendPacket();
                } else {
                    CSMAWaitRandom();
                }
            }
        }
        if (PHYReceive((char *)&RxPacket,&Strength) == sizeof(PacketType)) {
//check for received RF packet and validate size
            LastReceived = ReceivePacket();
            if (ActiveNode[LastReceived - 1] == 0) ActiveNode[LastReceived - 1] =
1;
        }
        Sleep(); //put processor into idle mode and wait for interrupt
    }
}

void SetupPacket() {
    SvPacket[ThisNode - 1].GroupID = OurGroup; //set ID byte (e.g. group number)
    SvPacket[ThisNode - 1].NodeID  = ThisNode;
    SvPacket[ThisNode - 1].Temp    = getTemp();
    SvPacket[ThisNode - 1].Light   = getLight();
    SvPacket[ThisNode - 1].CRC     = 0; // Need to generate CRC against..
    SvPacket[ThisNode - 1].Strength= 0; //..a packet with known values first
```

```
    SvPacket[ThisNode - 1].CRC     = getCRC(SvPacket[ThisNode -1]);
    TxPacket = SvPacket[ThisNode - 1]; // save this nodes currentdata ready for
transmit
}

char TDMAReady() {
    // ready if just received new packet from previous node,...
    // or time out if you've waited for enough time for all nodes to send...
    // otherwise keep waiting...
    if ((LastReceived % NumNodes) == (ThisNode - 1)){
        WaitCounter = 0;
        return 1;
    } else if (WaitCounter >= NumNodes) {
        WaitCounter = 0;
        return 1;
    } else {
        WaitCounter = WaitCounter++;
        return 0;
    }
}

char CSMAReady() {
    // ready to try to send again if enough time has passed for all nodes to send
once...
    // otherwise keep waiting...
    if (WaitCounter >= NumNodes) {
        WaitCounter = 0;
        return 1;
    } else {
        WaitCounter = WaitCounter++;
        return 0;
    }
}

void CSMAWaitRandom() {
    WaitCounter = rand() % NumNodes;
}

void SendPacket() {
    PORTA = 0x05; //switch on LED 0 (must keep RA3 high)
    PHYTransmit((char *)&TxPacket,sizeof(PacketType)); //Transmit RF data packet
    PORTA = 0x04; //switch off LED 0 (must keep RA3 high)
}

void ResetActiveNodes() {
    char i;
    ActiveNode[ThisNode -1] = 1;
    for (i=0;i<NumNodes;i++) {
        if (i != ThisNode - 1) ActiveNode[i] = 0;
    }
}

unsigned char ReceivePacket() {
    unsigned int tCRC,gCRC;
    unsigned char receivedfrom = 0;
    PORTA = 0x06; //switch on LED 1 (must keep RA3 high)
    tCRC = RxPacket.CRC;  // store the transmitted CRC value
    RxPacket.CRC = 0; // Generate CRC vaule against packet with known CRC value
    gCRC = getCRC(RxPacket);  // generate a CRC value
```

```
    if (tCRC == gCRC) { // compare generated and stored CRC, only submit data if
same
        SvPacket[RxPacket.NodeID - 1] = RxPacket;
        SvPacket[RxPacket.NodeID - 1].Strength = Strength; // Store received
strenght value
        receivedfrom = RxPacket.NodeID;
    }
    PORTA = 0x04; //switch off LED 1 (must keep RA3 high)
    return receivedfrom;
}

void TerminalOut() {
    char Text[128];
    char Data[128];
    int i;
    char degreesymbol = 176;
    for (i=0;i<NumNodes;i++) {
        if (SvPacket[i].GroupID == OurGroup) { // Only output info from working
nodes
            sprintf(Text,"GroupID = %u NodeID = %u ",
                    SvPacket[i].GroupID,
                    SvPacket[i].NodeID);
            if (((OurProto != TDMA) && (OurProto != CSMA)) ||
                (OurProto == TDMA && ActiveNode[i] == 1)  ||
                (OurProto == CSMA && ActiveNode[i] == 1)){
                sprintf(Data,"Temp = %u%cC Light = %s RSSI = %u\r\n",
                    SvPacket[i].Temp,
                        degreesymbol,
                    lightstr[SvPacket[i].Light],
                    SvPacket[i].Strength); //Format ID, Data & RSSI as ASCII
string for terminal
            } else {
                sprintf(Data,"Node Inactive\r\n");
            }
            strcat(Text,Data);
            USARTOut(Text,strlen(Text)); //output string to USART
        }
    }
    sprintf(Text,"\n");
    USARTOut(Text,strlen(Text));
    // Latest Data reported so consider node inactive until sends again
    ResetActiveNodes();
}

// Get Temperature Sensor Resistance Value
unsigned char getTemp() {
    unsigned int inttemp = 0;
    float floattemp = 0;
    unsigned char temp   = 0;
    signed char offsets[NUMNODES] = {77,79,75,80}; // offset for room temp
    float slope = -0.55;
    OpenADC(ADC_FOSC_16 & ADC_RIGHT_JUST & ADC_16_TAD,
            ADC_CH6 & ADC_INT_ON & ADC_VREFPLUS_VDD & ADC_VREFMINUS_VSS,
            7);
    Delay10TCYx(5);
    ConvertADC();
    while(BusyADC());
    inttemp = ReadADC();
    // ADC returns 10 bit number, to get most significant bits, we shift right
```

```
    // twice to drop the 2 least significate bits and take the lower half by
casting
    temp = (char)(inttemp >> 2);
    // conversion - (y = mx+c) where m is 'slope' and c is 'offset'
    floattemp = (temp * slope) + offsets[ThisNode - 1];
    // convert to integer, with rounding
    temp = (int)floattemp;

    CloseADC();
    return temp;
}

// Get Light Sensor Resistance Value
unsigned char getLight() {
    unsigned int intlight = 0;
    unsigned char light   = 0;
    OpenADC(ADC_FOSC_16 & ADC_RIGHT_JUST & ADC_16_TAD,
            ADC_CH5 & ADC_INT_ON & ADC_VREFPLUS_VDD & ADC_VREFMINUS_VSS,
            7);
    Delay10TCYx(5);
    ConvertADC();
    while (BusyADC());
    intlight = ReadADC();
    // ADC returns 10 bit number, to get most significant bits, we shift right
    // twice to drop the 2 least significate bits and take the lower half by
casting
    light = (char)(intlight >> 2);
    // conversion - light report kept simple, only three values reported
    if (light < 16) {
        light = Bright;
    } else if (light > 64) {
        light = Shadow;
    } else {
        light = Normal;
    }
    CloseADC();
    return light;
}

unsigned int getCRC(PacketType packet) {
    // we use the XMODEM poly (0x8408) to generate the CRC
    unsigned short crc_tbl[16] = {
        0x0000, 0x1E3D, 0x1C39, 0x0204, 0x1831, 0x060C, 0x0408, 0x1A35,
        0x1021, 0x0E1C, 0x0C18, 0x1225, 0x0810, 0x162D, 0x1429, 0x0A14,
    };
    unsigned int i, ch;
    unsigned int acc = 0xFFFF;
    int *rawPacket = (int*)&packet;
    // Nibble-at-a-time processing. Reflected algorithm; lo-nibble first.
    for (i=0; i <= sizeof(&rawPacket); i++) {
        ch = rawPacket[i];
        acc = crc_tbl[(ch ^ acc) & 15] ^ (acc >> 4);
        acc = crc_tbl[((ch >> 4) ^ acc) & 15] ^ (acc >> 4);
    }
    return acc;
}
```