

DESIGN OF A WIRELESS SENSOR NETWORK USING LPC2103 MICROCONTROLLER

Dibal, P. Y.

(Department of Computer Engineering, University of Maiduguri, Maiduguri, Nigeria)

e-mail address: yoksa77@gmail.com

Abstract

This paper deals with the design of a Wireless Sensor Network (WSN) using the LPC2103 microcontroller. The concept of sensors and sensing in the context of sensors was introduced and the architecture of a wireless node was described. The design approach was presented and the design was described using a diagram. The implementation of the design was carried out using a firmware to run the WSN and achieve the desired functionality of the WSN. Results obtained from the tests were similar to the ones obtained in the simulation. It was concluded that based on the results obtained, the wireless node transmitted data packet in real-time to the PC hyper-terminal. The LEDs on the wireless node were switched ON and OFF based on the value of the data received via 14 character input. The project has been able to in practice implement and set up a basic wireless sensor network.

Keywords: Wireless sensor, network, microcontroller, simulation

1. Introduction

Sensing can be defined as a method through which information regarding processes, physical objects and occurrence of events are gathered. While on the other hand, a **Sensor** can be defined as an object that is employed in the performance of a sensing task (Waltenegu and Poellabauer, 2010). Sensors can be classified as either active sensors (having external power), or as passive sensors (detecting and deriving power from the environment).

A wireless sensor (Waltenegu and Poellabauer, 2010) apart from having the sensing component also has capabilities like communication, storage and on-board processing. Being equipped with all these facilities, a sensor node apart from being responsible for data collection also is responsible for analysis, correlation and fusion of its own sensor data and data from other sensor nodes as well.

According to (Waltenegu and Poellabauer, 2010), several steps are involved in the sensing process as shown Figure 1 below:

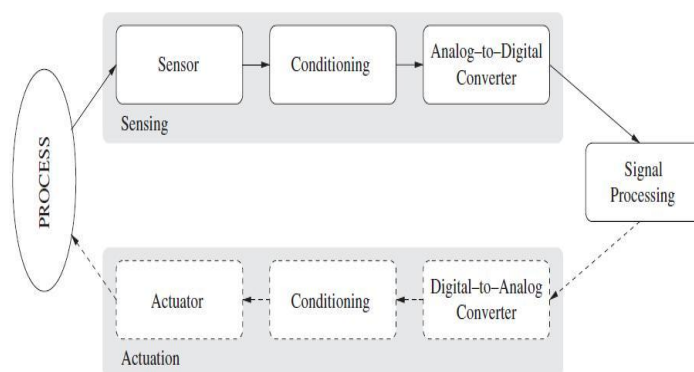


Figure 1: Sensing process

From the fore-going therefore, we can define a wireless sensor network as a network in which many sensors monitor large physical environments cooperatively by using wireless radios to communicate with each other and with a base station thus making it possible for the sensors to transmit their data to remote processing, analysis, visualization and storage systems.

The concept of a wireless sensor network according to (Pantazis and Vergados, 2007) is shown in Figure 2 below:

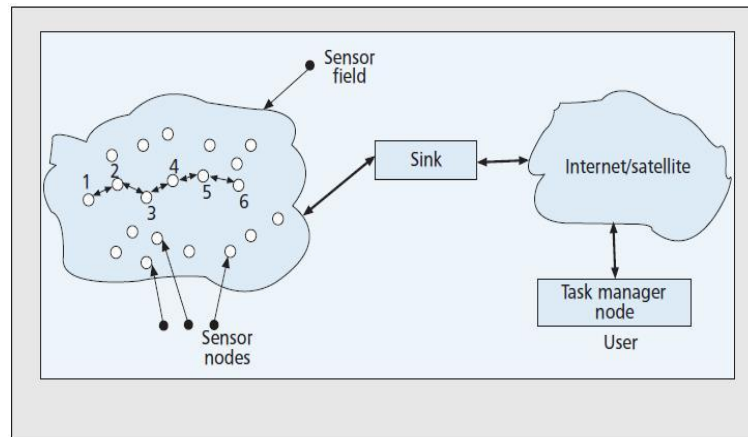


Figure 2: Wireless sensor network.

2. Materials and methods

2.1 Wireless sensor architecture

The architecture of a wireless sensor (Waltenegu and Poellabauer, 2010), is shown below in Figure 3:

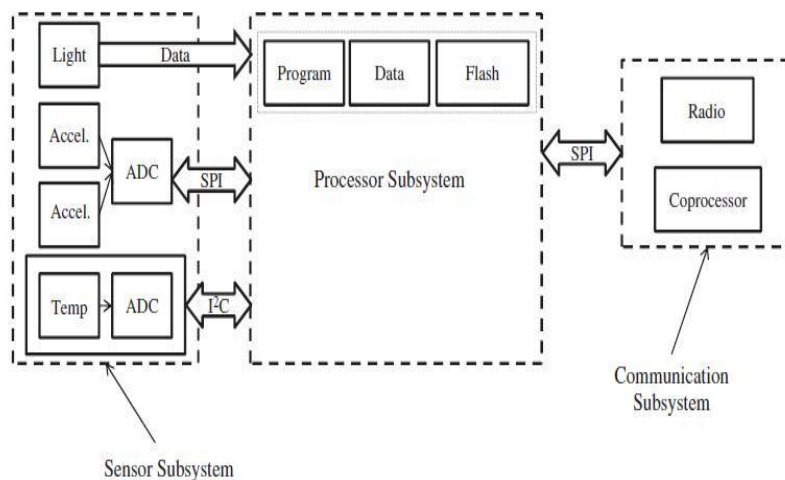


Figure 3: Wireless Sensor architecture.

The sensor subsystem (Waltenegu and Poellabauer, 2010), as shown in figure 3 combines different types of physical sensors, one or more analog-to-digital (ADC) converters and a multiplexing mechanism to share them. The output of a sensor being a continuous analog signal is converted by the ADC to a digital format and this involves:

- quantization of the analog signal in which the number of allowable discrete values is determined.

- determining the sampling frequency; in digital signal processing and communication engineering, this is often determined by the Nyquist rate.

The processor subsystem (Waltenegu and Poellabauer, 2010), as shown in figure 3 forms the core of the wireless sensor. The type of processor used is usually determined by the trade-off between flexibility and efficiency. Possible types of processors that can be used include: microcontrollers, digital signal processors, application specific ICs, field programmable gate arrays (FPGAs) etc.

The LPC2103 microcontroller (LPC2103 User Manual, 2009), is the microcontroller used for the processor subsystem of this design. The LPC2103 microcontroller is a microcontroller that is based on the 16-bit/32-bit ARM7TDMI-S CPU and has real time-time capability blended with 8kB, 16kB or 32kB of embedded high speed flash memory. The microcontroller allows 32-bit code execution to be achieved at maximum clock rate because it has 128-bit wide memory interface. This unique feature increases the performance of the interrupt service routine (ISR) and DSP algorithms.

The pin diagram of the LPC2103 microcontroller from (LPC2103 User Manual, 2009) is shown below in Figure 4:

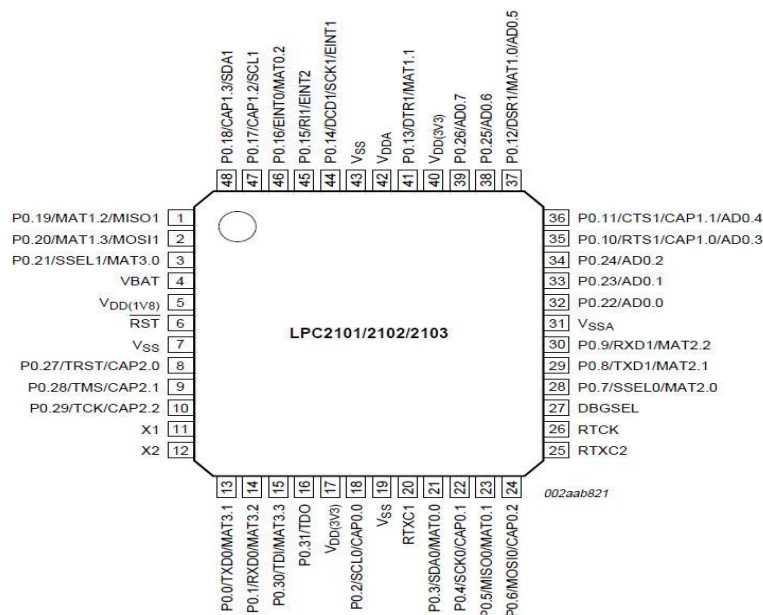


Figure 4: LPC2103 pin diagram

The LPC2103 microcontroller is usually housed on the MCB2103 Keil Development Board as shown in Figure 5 below:

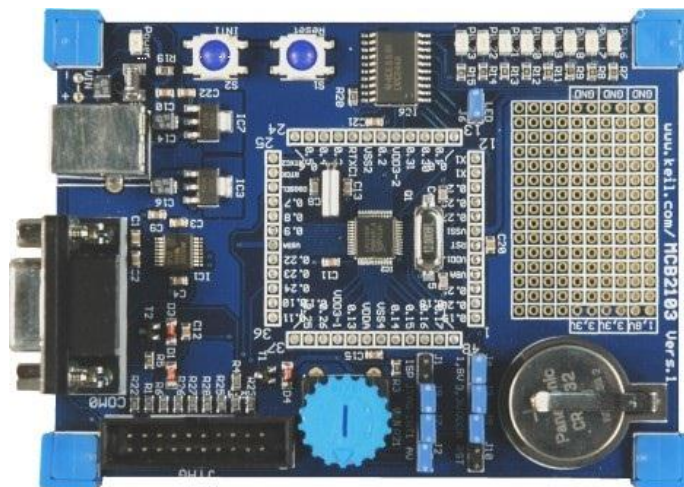


Figure 5: MCB2103 evaluation board

In the communication subsystem, as shown in figure 3, the most commonly used buses are the Data bus, Serial-Parallel Interface (SPI) and Inter-Integrated Circuit (I²C).

In the data bus, the Universal Asynchronous Receiver Transceiver (UART) is used for the transmission of information. In order for the PC or user to communicate with the wireless sensor network, a physical communication path must be established. This is achieved through the use of the Universal Asynchronous Receiver Transmitter (UART).

UART is defined as a component (Elmenreich and Martin, 2002), which is used extensively in serial communication. It has two basic parts: a transmitter (which is parallel-to-serial converter) and a receiver (which is serial-to-parallel converter). To achieve accurate connectivity, the parallel side of the UART is connected to the bus of a computer. Upon writing a byte to the UART's transmit data register, the UART then begins to transmit the byte on the serial line.

When a byte (Fang and Chen, 2011) is written to the UART's transmit data register for asynchronous transmission, a "start bit" is usually added at the beginning of each word that is scheduled for transmission. The purpose of the start bit is to alert the receiver at the other end that a word of data is about to be transmitted, therefore ensure the clock in the receiver to be in synchronization with that of the transmitter. To achieve an error margin of not more than 10% during transmission of the remaining bits in the word in terms of frequency drift, the two clocks must be accurate enough. Individual data bits of the word are then transmitted after the start bit with the Least Significant Bit (LSB) being the first to be sent. An equal amount of time is allocated to all the bits during transmission.

A parity bit (Fang and Chen, 2011) generated by the transmitter for the purpose of error checking is usually sent after the entire data word has been sent, after which a stop bit is sent by the transmitter. At the receiver end, when all the data word bits have been received, the

receiver checks for the parity bit and stop bit. If there is a failure of the stop bit to be where it is supposed to be, the entire word is considered by the UART as garbled and a framing error is reported to the host processor where the data was read. A typical UART frame format according to (Fang and Chen, 2011), is shown below in Figure 6:

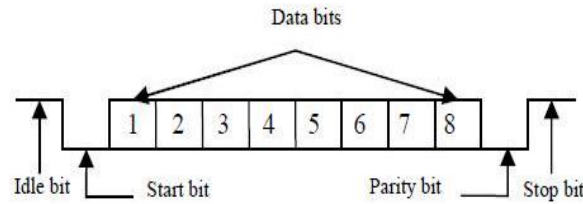


Figure 6: UART frame format

To address the problem of arithmetic error in baud rate setting (Elmenreich and Martin, 2002), the baud rate (BR) is set by choosing a value UBRS (UART Baud Rate Setting) as:

$$BR = \frac{f_{clk}}{C1 * (UBRS + C2)} \quad (1)$$

where C1 and C2 are UART implementation dependant. Solving for UBRS in equation (1), we obtain:

$$UBRS_{ideal} = \frac{f_{clk}}{C1 * BR} - C2 \quad (2)$$

However, equation (2) yields a rational number for UBRS, but most UARTs only accept UBRS values as integers. Therefore UBRS will be rounded to an integer value in approximation of the ideal value as expressed in equation (3) below:

$$UBRS_{real} = \frac{f_{clk}}{C1 * BR} - C2 + 0.5 \quad (3)$$

2.2 Design approach

The wireless sensor network will send and receive data from sensor nodes wirelessly. The data is then transmitted to the PC wirelessly also, through a Wireless Sensor Interface (WSI). The design approach to achieve this (WSS Manual, 2010), is shown below in Figure 7:

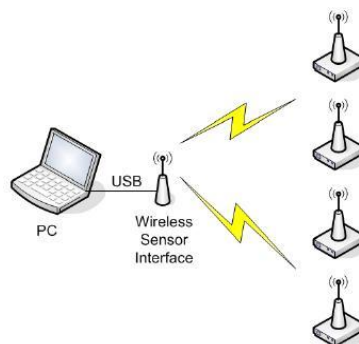


Figure 7: Sensors connecting to PC through wireless interface.

The wireless sensor interface transmits data from the wireless sensor network to the PC. When a sensor wants to transmit data to the PC, it does so through a Wireless Transceiver Module (WTM), which is connected to the UART interface of the LPC2103 microcontroller. The data from the WTM is transmitted wirelessly to the WSI, which is connected to the PC through a USB port. The data received by the PC is then displayed in a terminal interface. The flow of data between the WSI and WTM is bidirectional.

Both the WTM and the WSI shown in Figures 8a and 8b, operate in the 433MHz frequency band, with a supply voltage of 5V, and the Tx & Rx status LEDs. In both of them, the PIC16F876A microcontroller is at the heart of their operation.



Figure 8a: Wireless Transceiver Module

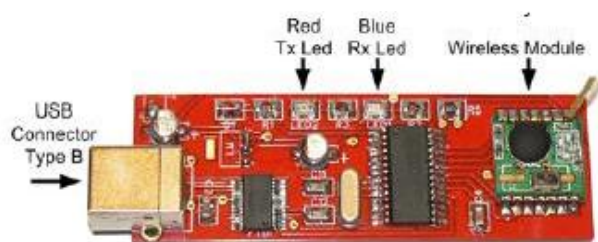


Figure 8b: Wireless Sensor Interface

2.3 Implementation of the design

To achieve proper coordination of the sensor field (Figure 2), there will be a master sensor node that will receive data from all the other sensor nodes (slave nodes), and then transmit this data to the WSI, which is connected to the PC for display in a hyper-terminal, so that the engineer monitoring the sensor field will know exactly what is going on.

The implementation of the design is carried out through the following steps, using Keil C and the Microvision (Keil, 2004) IDE:

2.3.1 UART initialization

In order for the sensors to communicate properly in the wireless network, both the master sensor node, the slave nodes, and the hyper-terminal on the PC, must all be initialized to the same baud rate, in this case 57,600bps.

Looking at Figure 4, we can see that the UART1 Tx & Rx are located at pins P0.8 and P0.9 respectively. From (LPC2103 User Manual, 2009), we initialize two registers i.e. PINSEL0 (Pin function Selection register) and the UART1 register as shown below in code listing 1 (Figure 9):


```

UILCR |= 0x80; // Set DLAB with 8 bits, no parity and 1 stop bit
UIFDR = 0xC1; // UART1 fractional division register
UIDLL = 0x09; // UART1 divisor latch LSB
UIDLM = 0x00; // UART1 divisor latch MSB
    
```

Figure 9: Code listing 1 - UART

These initialization values give a baud rate of 57,870bps, as shown below in Figure 10, which is within the acceptable margin for error.

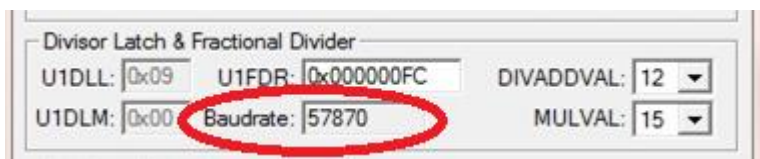


Figure 10: Initialized UART

Next, we initialize the VIC (Vectored Interrupt Controller) register, to handle the interrupt generated by the UART, as shown in code listing 2 (Figure 11).

```

VICVectAddr0 = (unsigned long)sio_irq; //assign address of IRQ
VICIntSelect = 0x00000000; //interrupt assigned to IRQ
    
```

Figure 11: Code listing 2 - VIC register

2.3.2 IRQ handler

The IRQ handler is designed to handle three possible interrupts that will be generated by the UART. These are: Receive Line Status (RLS) interrupt, Character Time-Out Indicator (CTI) interrupt and Receive Data Available (RDA) interrupt. Of particular importance is the RDA interrupt which is configured to be triggered once 14 characters are received. This interrupt invokes the *Receive_Packet* function which is responsible for receiving the 14 character via U1RBR. When the correct SLAVE_ID is entered, a *Send_Packet* function is invoked by the RDA. The *Send_Packet* function transmits a data to the terminal. The RDA interrupt then switches OFF the LEDs on the board if the first character of the *Receive_Data* packet is 'A'. But, if the first character of the *Receive_Data* is 'B', the RDA interrupt then switches the LEDs ON (treating a case of one sensor responding to a command from the PC). Figure 12 shows the Code listing 3 displaying the RDA handler.

```
// RDA interrupt - a complete packet has been received
else if ((IIRValue&0x07) == 0x02)
{
    // Extract data and reply if slave ID matches
    if((Recieve_Packet(&Recieve_Data[0])) == SLAVE_ID)
    {
        Send_Packet(SLAVE_ID, &Transmit_Data[0]);

        if(Recieve_Data[0]=='A')
        {
            IOCLR0 = 0x00FF0000;
        }
        else if(Recieve_Data[0]=='B')
        {
            IOSET0 = 0x00FF0000;
        }
    }
}
```

Figure 12: Code listing 3 display of RDA handler

The complete code listing for each of the wireless sensor nodes is shown in code listing 4 (Figure 13) and code listing 5 (Figure 14), respectively.

```

#include <LPC21xx.H>           /* LPC21xx definitions */
void sio_irq (void) __irq;    //Declare IRQ

void com_baudrate()
{
    U1LCR /= 0x80;           // Set DLAB
    U1FDR = 0xFC;          // Set fractional divider register
    U1DLL = 0x09;          // Set divisor latch LSB
    U1DLM = 0x00;          // Set divisor latch MSB
    U1LCR &= ~0x80;        // Clear DLAB
}

void init_uart (void)
{
    PINSEL0 = 0x00050000; // Enable Rx/D1 and Tx/D1
    U1LCR = 0x03;         // 8 bits, no Parity, 1 Stop bit
    U1IER = 0;           // Disable UART1 Interrupts
    VICVectAddr0 = (unsigned long)sio_irq; //Assign IRQ
    VICIntSelect = 0x00000000; //interrupt request assigned to IRQ
    VICVectCntl0 = 0x20 | 7; // UART1 Interrupt
    VICIntEnable = 1 << 7; // Enable UART1 Interrupt
    com_baudrate(); // setup for 57600 baud (12MHz)
    U1IER = 0x07;        // Enable UART1 RX and THRE Interrupts
    U1FCR = 0xC1; // Enable and reset TX and RX FIFO. trigger level 3
}

int main (void)
{
    init_uart (); //initialize the UART

    U1THR='H'; // Send test character
    U1THR='E'; // Send test character
    U1THR='L'; // Send test character
    U1THR='L'; // Send test character
    U1THR='O'; // Send test character
    U1THR=' '; // Send test character

    IOSET0 = 0x00FF0000;

    while (1) // Loop forever
    {

    }
}

```

Figure 13: Code listing 4 - Complete code listing to initialize

```

#include <LPC21xx.H>      /* LPC21xx definitions */
#include "uart.h"         /* uart definitions */

typedef unsigned char uint8_t;

// Slave ID
#define SLAVE_ID 'Z'
#define PACKET_DATA_SIZE (12U)
uint8_t Transmit_Data[PACKET_DATA_SIZE] = "FFFFFFFFFFFF";
uint8_t Recieve_Data[PACKET_DATA_SIZE];
volatile char UARTstatus;

void Send_Packet(uint8_t Slave_ID, uint8_t *Data)
{
    uint8_t Checksum=0x00, i;

    // Check the slave ID
    if(Slave_ID > 253)
    {
        return;
    }

    // TX slave ID
    U1THR = (Slave_ID);
    U1THR = ' ';

    // TX data
    for(i=0; i<=PACKET_DATA_SIZE; i++)
    {
        U1THR = (Data[i]);
        Checksum += Data[i];
    }

    // TX checksum
    U1THR=' ';
    U1THR=Checksum;
    U1THR=' ';
}

uint8_t Recieve_Packet(uint8_t *Data)
{
    uint8_t i, cChecksum=0x00, tChecksum;
    uint8_t Slave_ID=0xFF;

    // Get the slave ID
    Slave_ID=U1RBR;

    for (i=0; i<PACKET_DATA_SIZE; i++)
    {
        Data[i] = U1RBR;
        cChecksum += Data[i];
    }

    // Extract the transmitted checksum
    tChecksum +=U1RBR; //=cChecksum;

    return(Slave_ID);
}

```

```

void sio_irq(void) __irq
{
    uint8_t IIRValue, LSRValue, Dummy;
    IIRValue = ((uint8_t)U1IIR >> 0x01);
    LSRValue = (uint8_t)U1LSR;
    IODIRO = 0x00FF0000; //P0.16..23 defined as Outputs

    // Possible error ocurred during transmission or reception
    if ((IIRValue&0x07) == 0x03)
    {
        if (LSRValue & (LSR_OE | LSR_PE | LSR_FE |
            LSR_RXFE | LSR_BI))
        { //there are errors, read LSR to clear the interrupt
            UARTstatus = LSRValue;
            Dummy += U1RBR; //dummy read;clear interrupt
        }
    }

    // RDA interrupt - a complete packet has been recieved
    else if ((IIRValue&0x07) == 0x02)
    {
        // Extract data and reply if slave ID matches
        if ((Recieve_Packet(&Recieve_Data[0])) == SLAVE_ID)
        {
            Send_Packet(SLAVE_ID, &Transmit_Data[0]);

            if(Recieve_Data[0]=='A')
            {
                IOCLR0 = 0x00FF0000;
            }
            else if(Recieve_Data[0]=='B')
            {
                IOSET0 = 0x00FF0000;
            }
        }
    }
    VICVectAddr = 0xFF; // Acknowledge Interrupt
}

```

Figure 14: Code listing 5 - Code for wireless sensor node

3. Results and discussion

The simulation results show the output obtained from the Keil simulator after running the program. The output presented in Figure15 shows the performance of different functions that make up the total node program.

Module/Function	Calls	Time(Sec)	Time(%)
Blinky		3.080 s	100%
irqHANDLER		399.469 ms	13%
Send_Packet	1	4.867 μ s	0%
Recieve_Packet	1	4.317 μ s	0%
sio_irq	420484	399.460 ms	13%
wireless		2.645 s	86%
com_baudrate	1	0.517 μ s	0%
init_uart	1	0.800 μ s	0%
main	1	2.645 s	86%
Startup		35.041 ms	1%

Figure 15: Performance analyser for node program

It can be seen from Figure 15, that the IRQ handler (sio_irq) has the highest number of calls because it has been designed to continually run as long as 14 character input has not been fully received. The value for the number of calls is determined by how quickly the 14 characters arrive. The *Receive_Packet* function received 14 characters via U1RBR and if the characters are in the right order, it then invokes the *Send_Packet* function which transmits the data.

Figure 16 shows the output obtained at the hyper-terminal as a response from the program after getting the expected input.

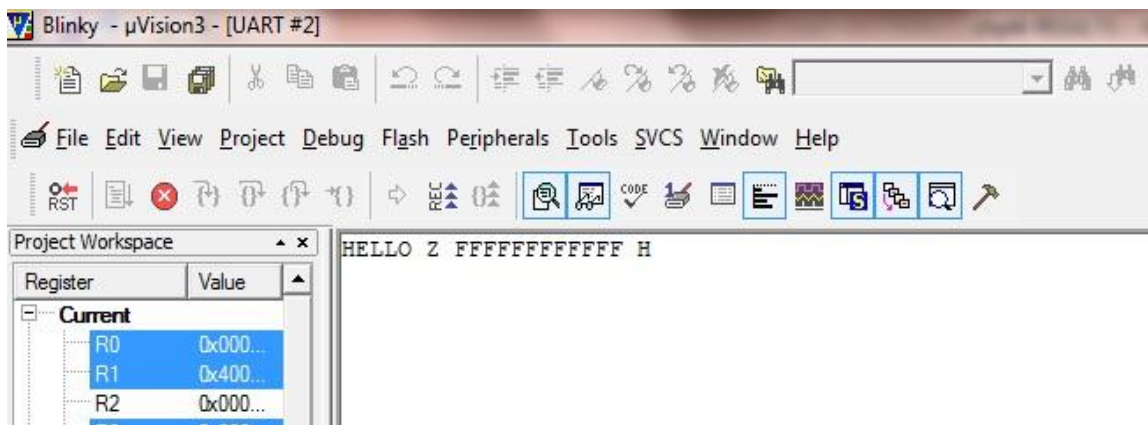


Figure 16: UART output at the hyper-terminal

When the first character of the Receive_Data packet is 'A', the RDA interrupt turns OFF the LEDs on the board. The test of that line code in the simulator confirms that the LEDs are switched OFF. The simulator result is shown in Figure 17 and highlighted in red.

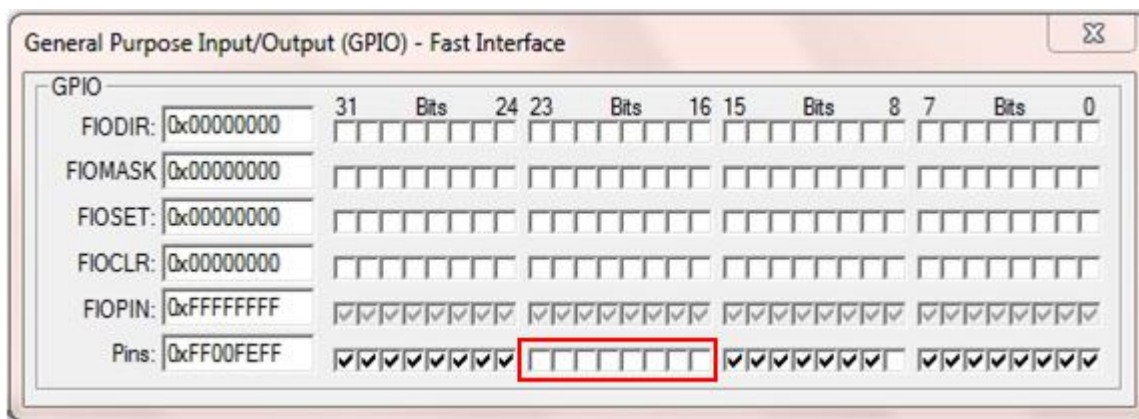


Figure 17: LEDs switched OFF

When the first character of the Receive_Data packet is 'B', the LEDs are switched ON. Testing that line of code on the simulator confirms the LEDs are actually switched ON (highlighted in red) as shown in Figure 18 below:

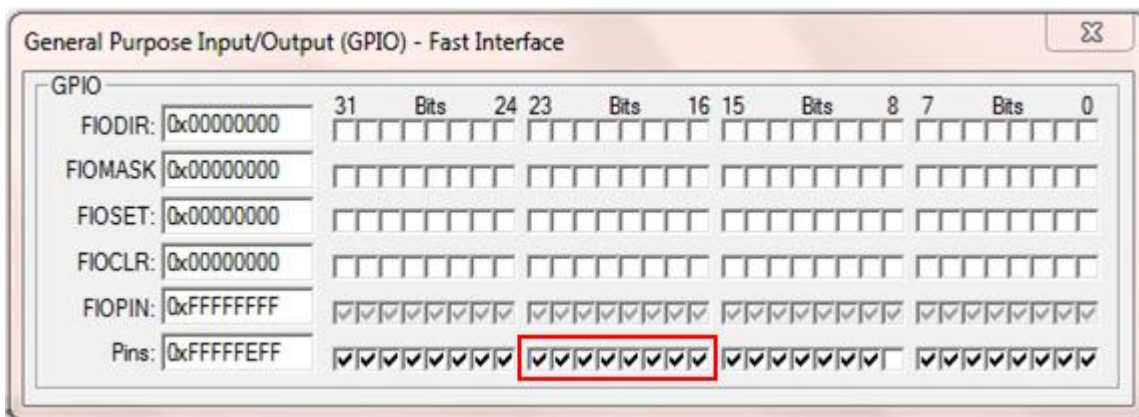


Figure 18: LEDs switched ON

Using flash magic, the source code was burnt to the microcontroller and the design was tested with one of the sensor nodes. When 14 character input was made through the PC, the microcontroller responded in real-time as expected and transmitted the predefined character to the hyper-terminal of the PC as shown in Figure 19.

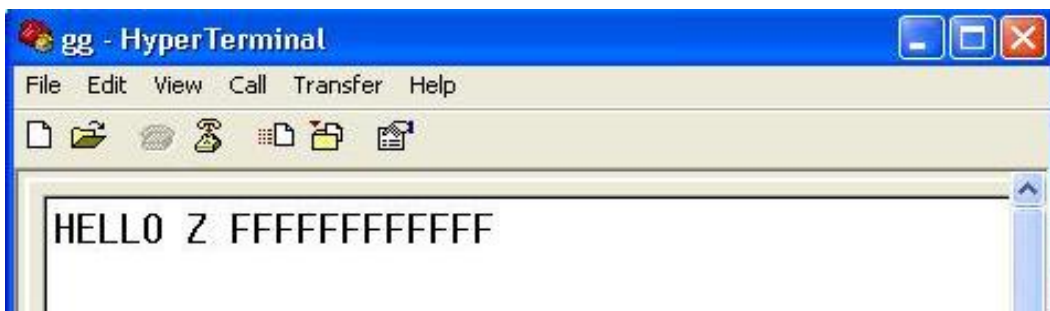


Figure 19: Transmitted data packet from microcontroller to hyper-terminal

During the first entry of the 14 characters, the *Receive_Data* had an 'A' as its first character and the RDA interrupt then switched OFF the LEDs on the MCB2103 board exactly as it happened in the simulator. This is shown in Figure 20.

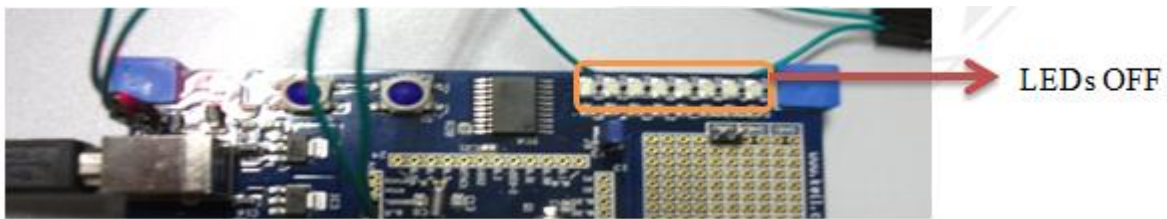


Figure 20: LEDs switched OFF when *Receive_Data* had 'A' as its first character

During the second entry of 14 characters, the *Receive_Data* had a 'B' as its first character and the RDA interrupt turned ON the LEDs on the MCB2103 board exactly as it did on the simulator. This is shown in Figure 21 below.

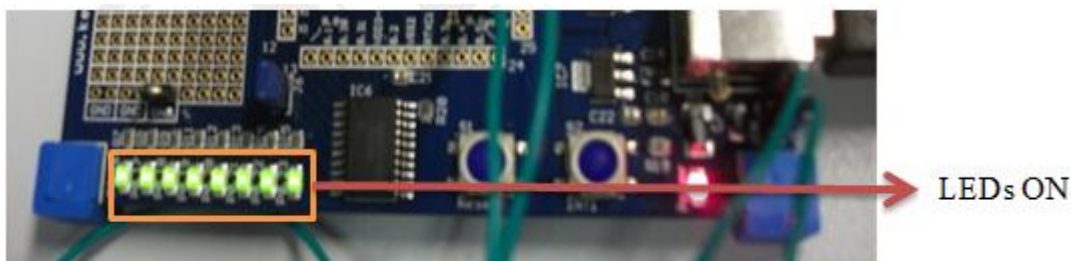


Figure 21: LEDs switched ON when *Receive_Data* has 'B' as its first character

4. Conclusions

The project set out with the aim of developing a wireless sensor network by writing a Keil-C code (firmware) for transmitting a packet and then control hardware components (in this case LEDs) on the network node. This code was tested in a simulator and the results were as expected i.e. a packet was transmitted and the LEDs in the simulator blinked as instructed by the firmware code.

The firmware was then downloaded to the LPC2103 microcontroller and then the performance of the wireless sensor network was tested in real-time. The results obtained from the test were like the results obtained in the simulation i.e. the wireless node transmitted data packet in real-time to the PC hyper-terminal and also the LEDs on the wireless node were switched ON and OFF based on the value of the data received via 14 character input.

References

Elmenreich, W. and Martin, D. 2002. Time-triggered communication with UARTs. 4th IEEE International Workshop on Factory Communication Systems, pp. 97-104.

Fang, Y. and Chen, X. 2011. Design and simulation of UART, serial communication module based on VHDL. 3rd International Workshop on Intelligent Systems and Applications, pp. 1-4.

Keil, J.M. 2004. *µVision3 IDE for Microcontrollers*. Available at: <http://www.evalkits.com/files/uv3.pdf> (Accessed: 29th April 2013).

NXP 2009. *LPC2101/02/03 user manual*. Eindhoven: NXP.

Pantazis, N. and Vergados, D. 2007. A survey on power control issues in wireless sensor networks. IEEE Communications Surveys and Tutorials, 9: 86-107.

Waltenegu, D. and Poellabauer, C. 2010. *Fundamentals of Wireless Sensor Networks*. West Sussex: John Wiley & Sons Ltd.

Wireless Sensor System 2010. *Wireless Sensor Interface*. Available at: http://77.162.55.232/usbscope/manual_wss/manual_wss.pdf (Accessed: 20th April 2013).