Contents

Chapter 1

Introduction

- **Introduction**
- **Applications Of WSN**

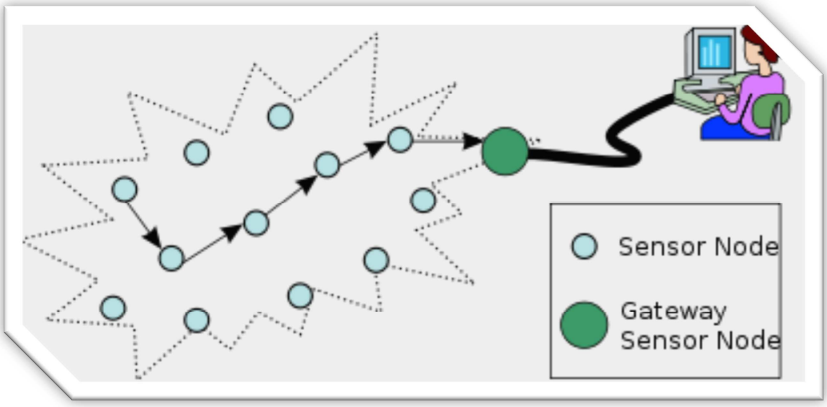
Introduction

A **wireless sensor network** (WSN) consists of spatially distributed autonomous sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants. The development of wireless sensor networks was motivated by military applications such as battlefield surveillance. They are now used in many industrial and civilian application areas, including industrial process monitoring and control, machine health monitoring, environment and habitat monitoring, healthcare applications, home automation, and traffic control.

In addition to one or more sensors, each node in a sensor network is typically equipped with a radio transceiver or other wireless communications device, a small microcontroller, and an energy source, usually a battery. A sensor node might vary in size from that of a shoebox down to the size of a grain of dust, although functioning "motes" of genuine microscopic dimensions have yet to be created. The cost of sensor nodes is similarly variable, ranging from hundreds of dollars to a few pennies, depending on the size of the sensor network and the complexity required of individual sensor nodes. Size and cost constraints on sensor nodes result in corresponding constraints on resources such as energy, memory, computational speed and bandwidth.

A sensor network normally constitutes a wireless ad-hoc network, meaning that each sensor supports a multi-hop routing algorithm (several nodes may forward data packets to the base station).

In computer science and telecommunications, wireless sensor networks are an active research area with numerous workshops and conferences arranged each year.



Applications Of WSN

The applications for WSNs are varied, typically involving some kind of monitoring, tracking, or controlling. Specific applications include habitat monitoring, object tracking, nuclear reactor control, fire detection, and traffic monitoring. In a typical application, a WSN is scattered in a region where it is meant to collect data through its sensor nodes.

Area monitoring

Area monitoring is a common application of WSNs. In area monitoring, the WSN is deployed over a region where some phenomenon is to be monitored. For example, a large quantity of sensor nodes could be deployed over a battlefield to detect enemy intrusion instead of using landmines. When the sensors detect the event being monitored (heat, pressure, sound, light, electro-magnetic field, vibration, etc.), the event needs to be reported to one of the base stations, which can take appropriate action (e.g., send a message on the internet or to a satellite). Depending on the exact application, different objective functions will require different data-propagation strategies, depending on things such as need for *real-time* response, *redundancy* of the data (which can be tackled via *data aggregation* and *information fusion* techniques), need for *security*, etc.

Fleet monitoring (outdoor and indoor location)

It is possible to put a mote with a GPS module on board of each vehicle of a fleet. The mote reports its coordinates so that the location is tracked with real time information. The motes can be equipped with temperature sensors to control any disruption of the cold chain. That helps to ensure the safety in the food and pharmaceutical industries and also some chemical shipments. Using the GSM cells helps to get the position in scenarios where there is not GPS coverage, like inside buildings, garages and tunnels. This alternative method consists in taking the information sent by the Mobile Phones Cells and look for their location in a previously saved Data Base.

Water Tower Level Monitoring

Water towers are used to add water and create water pressure to small communities or neighborhoods during peak use times to ensure water pressure is available to all users. Maintaining the water levels in these towers is important and requires constant monitoring and control. A wireless sensor network that includes submersible pressure sensors and float switches monitors the water levels in the tower and wirelessly transmits this data back to a control location. When tower water levels fall, pumps to move more water from the reservoir to the tower are turned on.

Vehicle Detection

Wireless sensor networks can use a range of sensors to detect the presence of vehicles ranging from motorcycles to train cars.

Chapter 2

Review

- Atmel AVR
- Sensors
- Wireless Sensors Network
- Wireless
- RS-232
- LCD
- Bascom AVR Compiler
- Visual Basic .NET

Atmel AVR

The AVR is a modified Harvard architecture 8-bit RISC single chip microcontroller (µC) which was developed by Atmel in 1996. The AVR was one of the first microcontroller families to use on-chip flash memory for program storage, as opposed to One-Time Programmable ROM, EPROM, or EEPROM used by other microcontrollers at the time.

Brief history

The AVR architecture was conceived by two students at the Norwegian Institute of Technology (NTH) Alf-Egil Bogen and Vegard Wollan.[1][2]

The original AVR MCU was developed at a local ASIC house in Trondheim, Norway, where the two founders of Atmel Norway were working as students. It was known as a µRISC (Micro RISC) [*citation needed*]. When the technology was sold to Atmel, the internal architecture was further developed by Alf and Vegard at Atmel Norway, a subsidiary of Atmel founded by the two architects. The designers worked closely with compiler writers at IAR Systems to ensure that the instruction set provided for more efficient compilation of high-level languages.[3] Atmel says that the name AVR is not an acronym and does not stand for anything in particular. The creators of the AVR give no definitive answer as to what the term "AVR" stands for.[2]

Note that the use of "AVR" in this article generally refers to the 8-bit RISC line of Atmel AVR Microcontrollers.

Among the first of the AVR line was the AT90S8515, which in a 40-pin DIP package has the same pinout as an 8051 microcontroller, including the external multiplexed address and data bus. The polarity of the RESET line was opposite (8051's having an active-high RESET, while the AVR has an active-low RESET), but other than that, the pinout was identical.

Basic families

AVRs are generally classified into five broad groups:

tinyAVR — the ATtiny series

- 0.5–8 kB program memory
- 6–32-pin package
- Limited peripheral set

megaAVR — the ATmega series

- 4–256 kB program memory
- 28–100-pin package
- Extended instruction set (Multiply instructions and instructions for handling larger program memories)
- Extensive peripheral set

XMEGA — the ATxmega series

- 16–384 kB program memory
- 44–64–100-pin package (A4, A3, A1)
- Extended performance features, such as DMA, "Event System", and cryptography support.
- Extensive peripheral set with DACs

Application specific AVR

- megaAVRs with special features not found on the other members of the AVR family, such as LCD controller, USB controller, advanced PWM, CAN etc.

Atmel At94k

FPSLIC (Field Programmable System Level Integrated Circuit), an AVR core on-die with an FPGA. The FPSLIC uses SRAM for the AVR program code, unlike all other AVRs. Partly due to the relative speed difference between SRAM and flash, the AVR core in the FPSLIC can run at up to 50 MHz.

Device architecture

Flash, EEPROM, and SRAM are all integrated onto a single chip, removing the need for external memory in most applications. Some devices have a parallel external bus option to allow adding additional data memory or memory-mapped devices. Almost all devices (except the smallest TinyAVR chips) have serial interfaces, which can be used to connect larger serial EEPROMs or flash chips.

Program memory

Program instructions are stored in non-volatile flash memory. Although they are 8-bit MCUs, each instruction takes one or two 16-bit words.

The size of the program memory is usually indicated in the naming of the device itself (e.g., the ATmega64x line has 64 kB of flash, however the ATmega32x has only 32 kB).

There is no provision for off-chip program memory; all code executed by the AVR core must reside in the on-chip flash. However, this limitation does not apply to the AT94 FPSLIC AVR/FPGA chips.

Internal data memory

The data address space consists of the register file, I/O registers, and SRAM.

Program execution

Atmel's AVRs have a two stage, single level pipeline design. This means the next machine instruction is fetched as the current one is executing. Most instructions take just one or two clock cycles, making AVRs relatively fast among the eight-bit microcontrollers.

The AVR family of processors were designed with the efficient execution of compiled C code in mind and has several built-in pointers for the task.

Internal registers

The AVRs have 32 single-byte registers and are classified as 8-bit RISC devices.

In most variants of the AVR architecture, the working registers are mapped in as the first 32 memory addresses (0000_{16}-$001F_{16}$) followed by the 64 I/O registers (0020_{16}-$005F_{16}$).

Actual SRAM starts after these register sections (address 0060_{16}). (Note that the I/O register space may be larger on some more extensive devices, in which case the memory mapped I/O registers will occupy a portion of the SRAM address space.)

Even though there are separate addressing schemes and optimized opcodes for register file and I/O register access, all can still be addressed and manipulated as if they were in SRAM.

In the XMEGA variant, the working register file is not mapped into the data address space; as such, it is not possible to treat any of the XMEGA's working registers as though they were SRAM. Instead, the I/O registers are mapped into the data address space starting at the very beginning of the address space. Additionally, the amount of data address space dedicated to I/O registers has grown substantially to 4096 bytes (0000_{16}-$0FFF_{16}$). As with previous generations, however, the fast I/O manipulation instructions can only reach the first 64 I/O register locations (the first 32 locations for bitwise instructions). Following the I/O registers, the XMEGA series sets aside a 4096 byte range of the data address space which can be used optionally for mapping the internal EEPROM to the data address space (1000_{16}-$1FFF_{16}$). The actual SRAM is located after these ranges, starting at 2000_{16}.

EEPROM

Almost all AVR microcontrollers have internal EEPROM for semi-permanent data storage. Like flash memory, EEPROM can maintain its contents when electrical power is removed.

In most variants of the AVR architecture, this internal EEPROM memory is not mapped into the MCU's addressable memory space. It can only be accessed the same way an external peripheral device is, using special pointer registers and read/write instructions which makes EEPROM access much slower than other internal RAM.

Sensors

A **sensor** is a device that measures a physical quantity and converts it into a signal which can be read by an observer or by an instrument. For example, a mercury-in-glass thermometer converts the measured temperature into expansion and contraction of a liquid which can be read on a calibrated glass tube. A thermocouple converts temperature to an output voltage which can be read by a voltmeter. For accuracy, most sensors are calibrated against known standards.

Use

Sensors are used in everyday objects such as touch-sensitive elevator buttons (tactile sensor) and lamps which dim or brighten by touching the base. There are also innumerable applications for sensors of which most people are never aware. Applications include cars, machines, aerospace, medicine, manufacturing and robotics.

A sensor is a device which receives and responds to a signal or stimulus. Here, the term "stimulus" means a property or a quantity that needs to be converted into electrical form. Hence, sensor can be defined as a device which receives a signal and converts it into electrical form which can be further used for electronic devices. A sensor differs from a transducer in the way that a transducer converts one form of energy into other form whereas a sensor converts the received signal into electrical form only.

Sensors in Nature

All living organisms contain biological sensors with functions similar to those of the mechanical devices described. Most of these are specialized cells that are sensitive to:

- Light, motion, temperature, magnetic fields, gravity, humidity, vibration, pressure, electrical fields, sound, and other physical aspects of the external environment
- Physical aspects of the internal environment, such as stretch, motion of the organism, and position of appendages (proprioception)
- Environmental molecules, including toxins, nutrients, and pheromones
- Estimation of biomolecules interaction and some kinetics parameters
- Internal metabolic milieu, such as glucose level, oxygen level, or osmolality
- Internal signal molecules, such as hormones, neurotransmitters, and cytokines
- Differences between proteins of the organism itself and of the environment or alien creatures

Wireless sensor network

A **wireless sensor network** (WSN) consists of spatially distributed autonomous sensors to cooperatively monitor physical or environmental conditions, such as temperature, sound, vibration, pressure, motion or pollutants. The development of wireless sensor networks was motivated by military applications such as battlefield surveillance. They are now used in many industrial and civilian application areas, including industrial process monitoring and control, machine health monitoring, environment and habitat monitoring, healthcare applications, home automation, and traffic control.

In addition to one or more sensors, each node in a sensor network is typically equipped with a radio transceiver or other wireless communications device, a small microcontroller, and an energy source, usually a battery. A sensor node might vary in size from that of a shoebox down to the size of a grain of dust, although functioning "motes" of genuine microscopic dimensions have yet to be created. The cost of sensor nodes is similarly variable, ranging from hundreds of dollars to a few pennies, depending on the size of the sensor network and the complexity required of individual sensor nodes. Size and cost constraints on sensor nodes result in corresponding constraints on resources such as energy, memory, computational speed and bandwidth.

A sensor network normally constitutes a wireless ad-hoc network, meaning that each sensor supports a multi-hop routing algorithm (several nodes may forward data packets to the base station).

In computer science and telecommunications, wireless sensor networks are an active research area with numerous workshops and conferences arranged each year.

Applications

The applications for WSNs are varied, typically involving some kind of monitoring, tracking, or controlling. Specific applications include habitat monitoring, object tracking, nuclear reactor control, fire detection, and traffic monitoring. In a typical application, a WSN is scattered in a region where it is meant to collect data through its sensor nodes.

Area monitoring

Area monitoring is a common application of WSNs. In area monitoring, the WSN is deployed over a region where some phenomenon is to be monitored. For example, a large quantity of sensor nodes could be deployed over a battlefield to detect enemy intrusion instead of using landmines. When the sensors detect the event being monitored (heat, pressure, sound, light, electro-magnetic field, vibration, etc.), the event needs to be reported to one of the base stations, which can take appropriate action (e.g., send a message on the internet or to a satellite). Depending on the exact application, different objective functions will require different data-propagation strategies, depending on things such as need for *real-time* response, *redundancy* of the data (which can be tackled via *data aggregation* and *information fusion* techniques), need for *security*, etc.

Environmental monitoring

A number of WSNs have been deployed for environmental monitoring. Many of these have been short lived, often due to the prototype nature of the projects. Examples of longer-lived deployments are monitoring the state of permafrost in the Swiss Alps: The PermaSense Project, PermaSense Online Data Viewer and glacier monitoring.

Machine Health Monitoring or Condition based maintenance

Wireless sensor networks have been developed for machinery condition-based maintenance (CBM) as they offer significant cost savings and enable new functionalities. In wired systems, the installation of enough sensors is often limited by the cost of wiring, which runs between $10–$1000 per foot.Previously

inaccessible locations, rotating machinery, hazardous or restricted areas, and mobile assets can now be reached with wireless sensors. Often, companies use manual techniques to calibrate, measure, and maintain equipment. This labor-intensive method not only increases the cost of maintenance but also makes the system prone to human errors. Especially in US Navy shipboard systems, reduced manning levels make it imperative to install automated maintenance monitoring systems. Wireless sensor networks play an important role in providing this capability

Industrial Monitoring

Water/Wastewater Monitoring

There are many opportunities for using wireless sensor networks within the water/wastewater industries. Facilities not wired for power or data transmission can be monitored using industrial wireless I/O devices and sensors powered using solar panels or battery packs. As part of the American Recovery and Reinvestment Act (ARRA), funding is available for some water and wastewater projects in most states.

Landfill Ground Well Level Monitoring and Pump Counter

Wireless sensor networks can be used to measure and monitor the water levels within all ground wells in the landfill site and monitor leachate accumulation and removal. A wireless device and submersible pressure transmitter monitors the leachate level. The sensor information is wirelessly transmitted to a central data logging system to store the level data, perform calculations, or notify personnel when a service vehicle is needed at a specific well.

It is typical for leachate removal pumps to be installed with a totalizing counter mounted at the top of the well to monitor the pump cycles and to calculate the total volume of leachate removed from the well. For most current installations, this counter is read manually. Instead of manually collecting the pump count data, wireless devices can send data from the pumps back to a central control location to save time and eliminate errors. The control system uses this count information to determine when the pump is in operation, to calculate leachate extraction volume, and to schedule maintenance on the pump.

Flare Stack Monitoring

Landfill managers need to accurately monitor methane gas production, removal, venting, and burning. Knowledge of both methane flow and temperature at the flare stack can define when methane is released into the environment instead of combusted. To accurately determine methane production levels and flow, a pressure transducer can detect both pressure and vacuum present within the methane production system.

Thermocouples connected to wireless I/O devices create the wireless sensor network that detects the heat of an active flame, verifying that methane is burning. Logically, if the meter is indicating a methane flow and the temperature at the flare stack is high, then the methane is burning correctly. If the meter indicates methane flow and the temperature is low, methane is releasing into the environment.

Water Tower Level Monitoring

Water towers are used to add water and create water pressure to small communities or neighborhoods during peak use times to ensure water pressure is available to all users. Maintaining the water levels in these towers is important and requires constant monitoring and control. A wireless sensor network that includes submersible pressure sensors and float switches monitors the water levels in the tower and wirelessly transmits this data back to a control location. When tower water levels fall, pumps to move more water from the reservoir to the tower are turned on.

Vehicle Detection

Wireless sensor networks can use a range of sensors to detect the presence of vehicles ranging from motorcycles to train cars.

Fleet monitoring (outdoor and indoor location)

It is possible to put a mote with a GPS module on board of each vehicle of a fleet. The mote reports its coordinates so that the location is tracked with real time information. The motes can be equipped with temperature sensors to control any disruption of the cold chain. That helps to ensure the safety in the food and pharmaceutical industries and also some chemical shipments. Using the GSM cells

helps to get the position in scenarios where there is not GPS coverage, like inside buildings, garages and tunnels. This alternative method consists in taking the information sent by the Mobile Phones Cells and look for their location in a previously saved Data Base.

Agriculture

Using wireless sensor networks within the agricultural industry is increasingly common. Gravity fed water systems can be monitored using pressure transmitters to monitor water tank levels, pumps can be controlled using wireless I/O devices, and water use can be measured and wirelessly transmitted back to a central control center for billing. Irrigation automation enables more efficient water use and reduces waste.

Windrow Composting

Composting is the aerobic decomposition of biodegradable organic matter to produce compost, a nutrient-rich mulch of organic soil produced using food, wood, manure, and/or other organic material. One of the primary methods of composting involves using windrows.

To ensure efficient and effective composting, the temperatures of the windrows must be measured and logged constantly. With accurate temperature measurements, facility managers can determine the optimum time to turn the windrows for quicker compost production. Manually collecting data is time consuming, cannot be done continually, and may expose the person collecting the data to harmful pathogens. Automatically collecting the data and wirelessly transmitting the data back to a centralized location allows composting temperatures to be continually recorded and logged, improving efficiency, reducing the time needed to complete a composting cycle, and minimizing human exposure and potential risk.

An industrial wireless I/O device mounted on a stake with two thermocouples, each at different depths, can automatically monitor the temperature at two depths within

a compost windrow or stack. Temperature sensor readings are wirelessly transmitted back to the gateway or host system for data collection, analysis, and logging. Because the temperatures are measured and recorded continuously, the composting rows can be turned as soon as the temperature reaches the ideal point. Continuously monitoring the temperature may also provide an early warning to potential fire hazards by notifying personnel when temperatures exceed recommended ranges.

Greenhouse Monitoring

Wireless sensor networks are also used to control the temperature and humidity levels inside commercial greenhouses. When the temperature and humidity drops below specific levels, the greenhouse manager must be notified via e-mail or cell phone text message, or host systems can trigger misting systems, open vents, turn on fans, or control a wide variety of system responses. Because some wireless sensor networks are easy to install, they are also easy to move as the needs of the application change.

Standards and specifications

Several standards are currently either ratified or under development for wireless sensor networks. In addition to the standards, there are also several non-standard, proprietary mechanisms and specifications.

6lowpan, ISA100, WirelessHART, and ZigBee are all based on the same underlying radio standard: IEEE 802.15.4 - 2006.

6LoWPAN

6LoWPAN is a working group within the IETF that has produced a standards track specification for the transmission of IPv6 packets over IEEE 802.15.4.

EnOcean

EnOcean is a system for wireless communication in the building automation world. It is not standardized with any of the generally approved standardization bodies.

EnviroNet

EnviroNet is a wireless sensor network generally geared toward industrial control and environmental automation applications. EnviroNet has non-standardized and standardized versions that comply with ISA100.

IEC 62591

The International Electrotechnical Commission (IEC) approved the WirelessHART specification as a full international standard (IEC 62591Ed. 1.0) in April 2010.

IEEE 1451

Also relevant to sensor networks is the emerging IEEE 1451 which attempts to create standards for the smart sensor market. The main point of smart sensors is to move the processing intelligence closer to the sensing device.

ISA100

ISA100 is a new standard under development that makes use of 6lowpan and provides additional agreements for industrial control applications. ISA100 is scheduled for completion in 2009.

WirelessHART

The WirelessHART standard is an extension of the HART Protocol and is specifically designed for Industrial applications like Process Monitoring and Control. WirelessHART was added to the overall HART protocol suite as part of the HART 7 Specification, which was approved by the HART Communication Foundation in June 2007.

ZigBee

ZigBee networking specification for transmission of packets over IEEE 802.15.4 is intended for uses such as embedded sensing, medical data collection, consumer devices like television remote controls, and home automation. Zigbee is promoted by a large consortium of industry players. ZigBee sets extra communication features such as authentication, encryption, association and application services in the upper layer.

Hardware

The main challenge is to produce *low cost* and *tiny* sensor nodes. With respect to these objectives, current sensor nodes are mainly prototypes. Miniaturization and low cost are understood to follow from recent and future progress in the fields of MEMS and NEMS. Some of the existing sensor nodes are given below. Some of the nodes are still in research stage.

Also inherent to sensor network adoption is the availability of a very low power method for acquiring sensor data wirelessly.

An overview of commonly used sensor network platforms, components, technology and related topics is available in the SNM - Sensor Network Museum [tm].

Software

Energy is the scarcest resource of WSN nodes, and it determines the lifetime of WSNs. WSNs are meant to be deployed in large numbers in various environments, including remote and hostile regions, with ad-hoc communications as key. For this reason, algorithms and protocols need to address the following issues:

- Lifetime maximization
- Robustness and fault tolerance
- Self-configuration

Security

- Mobility (when sensor nodes or base stations are moving)
- Middleware: the design of middle-level primitives between the software and the hardware

Operating systems

Operating systems for wireless sensor network nodes are typically less complex than general-purpose operating systems both because of the special requirements of sensor network applications and because of the resource constraints in sensor network hardware platforms. For example, sensor network applications are usually not interactive in the same way as applications for PCs. Because of this, the operating system does not need to include support for user interfaces. Furthermore, the resource constraints in terms of memory and memory mapping hardware support make mechanisms such as virtual memory either unnecessary or impossible to implement.

Wireless sensor network hardware is not different from traditional embedded systems and it is therefore possible to use embedded operating systems such as eCos or uC/OS for sensor networks. However, such operating systems are often designed with real-time properties. Unlike traditional embedded operating systems, however, operating systems specifically targeting sensor networks often do not have real-time support.

TinyOS[15] is perhaps the first operating system specifically designed for wireless sensor networks. Unlike most other operating systems, TinyOS is based on an event-driven programming model instead of multithreading. TinyOS programs are composed into *event handlers* and *tasks* with run to completion-semantics. When an external event occurs, such as an incoming data packet or a sensor reading, TinyOS calls the appropriate event handler to handle the event. Event handlers can post tasks that are scheduled by the TinyOS kernel some time later. Both the TinyOS system and programs written for TinyOS are written in a special programming language called nesC which is an extension to the C programming language. NesC is designed to detect race conditions between tasks and event handlers.

There are also operating systems that allow programming in C. Examples of such operating systems include Contiki, MANTIS, BTnut, SOS and Nano-RK. Contiki is designed to support loading modules over the network and supports run-time loading of standard ELF files. The Contiki kernel is event-driven, like TinyOS, but the

system supports multithreading on a per-application basis. Furthermore, Contiki includes protothreads that provide a thread-like programming abstraction but with a very small memory overhead. Unlike the event-driven Contiki kernel, the MANTIS and Nano-RK kernels are based on preemptive multithreading. With preemptive multithreading, applications do not need to explicitly yield the microprocessor to other processes. Instead, the kernel divides the time between the active processes and decides which process that currently can be run which makes application programming easier. Nano-RK is a real-time resource kernel that allows fine grained control of the way tasks get access to CPU time, networking and sensors. Like TinyOS and Contiki, SOS is an event-driven operating system. The prime feature of SOS is its support for loadable modules. A complete system is built from smaller modules, possibly at run-time. To support the inherent dynamism in its module interface, SOS also focuses on support for dynamic memory management. BTnut is based on cooperative multi-threading and plain C code, and is packaged with a developer kit and tutorial.

LiteOS is a newly developed OS for wireless sensor networks, which provides UNIX like abstraction and support for C programming language.

ERIKA Enterprise is one of the newcomers as operating systems for sensor networks. Being an open-source real-time kernel, ERIKA Enterprise provides an operating system API similar to the OSEK/VDX API used in automotive, together with the uWireless wireless software stack providing a 802.15.4 with Guaranteed Time Slot (GTS) support, which is very important when there is need for real-time traffic guarantees on wireless sensor networks.

Middleware

There is considerable research effort currently invested in the design of middleware for WSNs. In general approaches can be classified into distributed database, mobile agents, and event-based.

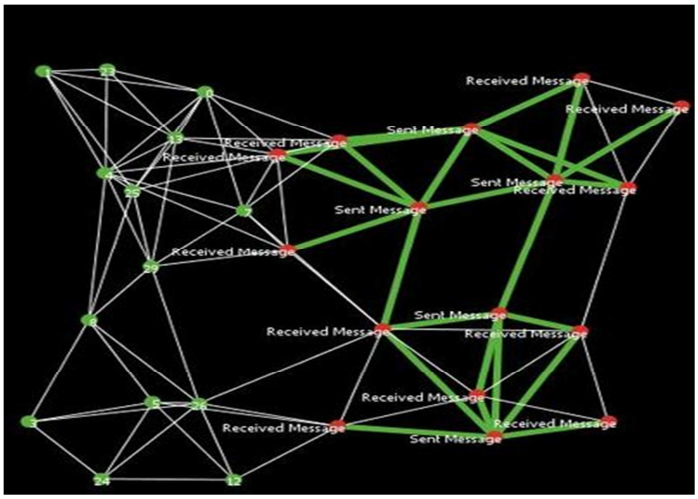
Algorithms

WSNs are composed of a large number of sensor nodes, therefore, an algorithm for a WSN is implicitly a *distributed algorithm*. In WSNs the scarcest resource is energy, and one of the most energy-expensive operations are data transmission and idle listening. For this reason, algorithmic research in WSN mostly focuses on the study and design of *energy aware* algorithms for saving energy by reducing the amount of data being transmitted - using techniques like data aggregation -, changing the transmission power of the sensor nodes or turning nodes off while preserving connectivity and coverage - applying Topology control algorithms -.

Another characteristic to take into account is that due to the constrained radio transmission range and the polynomial growth in the energy-cost of radio transmission with respect to the transmission distance, it is very unlikely that every node will reach the base station, so data transmission is usually multi-hop (from node to node, towards the base stations).

The *algorithmic* approach to modelling, simulating and analysing WSN differentiates itself from the *protocol* approach by the fact that the idealised mathematical models used are more abstract, more general, and easier to analyze. However, they are sometimes less realistic than the models used for protocol design, since an algorithmic approach often neglects timing issues, protocol overhead, the routing initiation phase and sometimes distributed implemenentation of the algorithms.

Simulators



Flooding Technique in Wireless Sensor Network by *Qasim Siddique*

There are network simulator platforms specifically designed to model and simulate Wireless Sensor Networks, like TOSSIM, which is a part of TinyOS and COOJA which is a part of Contiki. Traditional network simulators like ns-2 have also been used. A platform independent component based simulator with wireless sensor network framework, **J-Sim** can also be used. In addition, there is a simulator focused on the evaluation of topology control protocols in WSNs called Atarraya. An extensive list of simulation tools for Wireless Sensor Networks can be found at the CRUISE WSN Simulation Tool Knowledgebase.

Based on the OMNeT++ network simulator architecture, Mobility Framework and Castalia can be used for simulation of wireless sensor networks.

Based on Matlab, the Prowler (Probabilistic Wireless Network Simulator) toolbox is available. JProwler is a version of Prowler written in Java.

QualNet Network Simulator can be used to simulate Wireless Sensor Network

Developed using C# under Visual Studio 2008, **WSNSim** is available for download from WSNPedia. While WSNSim comes equipped with the behaviour of the classic LEACH and HEED clustering protocols in WSN, it is mainly a simulation framework that can be adopted to try other clustering and/or routing protocols in WSN, on a

common testbench. It also introduces the t-SNIPER algorithm that utilizes a socio-economic model of trust based routing scheme.

Netlogo can be used to simulate Wireless Sensor Network

Data visualization

The data gathered from wireless sensor networks is usually saved in the form of numerical data in a central base station. Additionally, the Open Geospatial Consortium (OGC) is specifying standards for interoperability interfaces and metadata encodings that enable real time integration of heterogeneous sensor webs into the Internet, allowing any individual to monitor or control Wireless Sensor Networks through a Web Browser. There are several techniques to retrieve data from the nodes, some of the protocols rely on flooding mechanisms, other map the data to nodes by applying the concept of DHT.

Wireless

Wireless communication is the transfer of information over a distance without the use of enhanced electrical conductors or "wires". The distances involved may be short (a few meters as in television remote control) or long (thousands or millions of kilometers for radio communications). When the context is clear, the term is often shortened to "wireless". Wireless communication is generally considered to be a branch of telecommunications.

It encompasses various types of fixed, mobile, and portable two-way radios, cellular telephones, personal digital assistants (PDAs), and wireless networking. Other examples of *wireless technology* include GPS units, garage door openers and or garage doors, wireless computer mice, keyboards and headsets, satellite television and cordless telephones.

Introduction

Handheld wireless radios such as this Maritime VHF radio transceiver use electromagnetic waves to implement a form of wireless communications technology.

Wireless operations permits services, such as long range communications, that are impossible or impractical to implement with the use of wires. The term is commonly used in the telecommunications industry to refer to telecommunications systems (e.g. radio transmitters and receivers, remote controls, computer networks, network terminals, etc.) which use some form of energy (e.g. radio frequency (RF), infrared light, laser light, visible light, acoustic energy, etc.) to transfer information without the use of wires. Information is transferred in this manner over both short and long distances.

Wireless communication

The term "wireless" has become a generic and all-encompassing word used to describe communications in which electromagnetic waves or RF (rather than some form of wire) carry a signal over part or the entire communication path. Common examples of wireless equipment in use today include:

- Professional LMR (Land Mobile Radio) and SMR (Specialized Mobile Radio) typically used by business, industrial and Public Safety entities.
- Consumer Two Way Radio including FRS (Family Radio Service), GMRS (General Mobile Radio Service) and Citizens band ("CB") radios.

- The Amateur Radio Service (Ham radio).
- Consumer and professional Marine VHF radios.
- Cellular telephones and pagers: provide connectivity for portable and mobile applications, both personal and business.
- Global Positioning System (GPS): allows drivers of cars and trucks, captains of boats and ships, and pilots of aircraft to ascertain their location anywhere on earth.
- Cordless computer peripherals: the cordless mouse is a common example; keyboards and printers can also be linked to a computer via wireless.
- Cordless telephone sets: these are limited-range devices, not to be confused with cell phones.
- Satellite television: allows viewers in almost any location to select from hundreds of channels.
- Wireless gaming: new gaming consoles allow players to interact and play in the same game regardless of whether they are playing on different consoles. Players can chat, send text messages as well as record sound and send it to their friends. Controllers also use wireless technology. They do not have any cords but they can send the information from what is being pressed on the controller to the main console which then processes this information and makes it happen in the game. All of these steps are completed in milliseconds.

Wireless networking (i.e. the various types of unlicensed 2.4 GHz WiFi devices) is used to meet many needs. Perhaps the most common use is to connect laptop users who travel from location to location. Another common use is for mobile networks that connect via satellite. A wireless transmission method is a logical choice to network a LAN segment that must frequently change locations. The following situations justify the use of wireless technology:

- To span a distance beyond the capabilities of typical cabling,
- To provide a backup communications link in case of normal network failure,
- To link portable or temporary workstations,
- To overcome situations where normal cabling is difficult or financially impractical, or
- To remotely connect mobile users or networks.

Wireless communication can be via:

- radio frequency communication,
- microwave communication, for example long-range line-of-sight via highly directional antennas, or short-range communication, or
- infrared (IR) short-range communication, for example from remote controls or via Infrared Data Association (IrDA).

Applications may involve point-to-point communication, point-to-multipoint communication, broadcasting, cellular networks and other wireless networks.

The term "wireless" should not be confused with the term "cordless", which is generally used to refer to powered electrical or electronic devices that are able to operate from a portable power source (e.g. a battery pack) without any cable or cord to limit the mobility of the cordless device through a connection to the mains power supply. Some cordless devices, such as cordless telephones, are also wireless in the sense that information is transferred from the cordless telephone to the telephone's base unit via some type of wireless communications link. This has caused some disparity in the usage of the term "cordless", for example in Digital Enhanced Cordless Telecommunications.

In the last fifty years, wireless communications industry experienced drastic changes driven by many technology innovations.

History

Photophone

The world's first, wireless telephone conversation occurred in 1880, when Alexander Graham Bell and Charles Sumner Tainter invented and patented the photophone, a telephone that conducted audio conversations wirelessly over modulated light beams (which are narrow projections of electromagnetic waves). In that distant era when utilities did not yet exist to provide electricity, and lasers had not even been conceived of in science fiction, there were no practical applications for their invention, which was highly limited by the availability of both sunlight and good weather. Similar to free space optical communication, the photophone also required a clear line of sight between its transmitter and its receiver. It would be several decades before the photophone's principles found their first practical applications in military communications and later in fiber-optic communications.

Radio

The term "wireless" came into public use to refer to a radio receiver or transceiver (a dual purpose receiver and transmitter device), establishing its usage in the field of wireless telegraphy early on; now the term is used to describe modern wireless connections such as in cellular networks and wireless broadband Internet. It is also used in a general sense to refer to any type of operation that is implemented without the use of wires, such as "wireless remote control" or "wireless energy transfer", regardless of the specific technology (e.g. radio, infrared, ultrasonic) that is used to accomplish the operation. While Guglielmo Marconi and Karl Ferdinand Braun were awarded the 1909 Nobel Prize for Physics for their contribution to wireless telegraphy.

Early wireless work

David E. Hughes, eight years before Hertz's experiments, transmitted radio signals over a few hundred yards by means of a clockwork keyed transmitter. As this was before Maxwell's work was understood, Hughes' contemporaries dismissed his achievement as mere "Induction". In 1885, T. A. Edison used a vibrator magnet for induction transmission. In 1888, Edison deployed a system of signaling on the Lehigh Valley Railroad. In 1891, Edison obtained the wireless patent for this method using inductance (U.S. Patent 465,971).

In the *history of wireless technology*, the demonstration of the theory of electromagnetic waves by Heinrich Hertz in 1888 was important. The theory of electromagnetic waves was predicted from the research of James Clerk Maxwell and Michael Faraday. Hertz demonstrated that electromagnetic waves could be transmitted and caused to travel through space at straight lines and that they were able to be received by an experimental apparatus. The experiments were not followed up by Hertz. Jagadish Chandra Bose around this time developed an early wireless detection device and help increase the knowledge of millimeter length electromagnetic waves. Practical applications of wireless radio communication and radio remote control technology were implemented by later inventors, such as Nikola Tesla.

The electromagnetic spectrum

Light, colors, AM and FM radio, and electronic devices make use of the electromagnetic spectrum. In the US, the frequencies that are available for use for communication are treated as a public resource and are regulated by the Federal Communications Commission. This determines which frequency ranges can be used for what purpose and by whom. In the absence of such control or alternative arrangements such as a privatized electromagnetic spectrum, chaos might result if, for example, airlines didn't have specific frequencies to work under and an amateur radio operator was interfering with the pilot's ability to land an airplane. Wireless communication spans the spectrum from 9 kHz to 300 GHz. (Also see Spectrum management)

RS-232

In telecommunications, **RS-232** (Recommended Standard 232) is a standard for serial binary single-ended data and control signals connecting between a *DTE* (Data Terminal Equipment) and a *DCE* (Data Circuit-terminating Equipment). It is commonly used in computer serial ports. The standard defines the electrical characteristics and timing of signals, the meaning of signals, and the physical size and pinout of connectors.

Scope of the standard

The Electronics Industries Association (EIA) standard RS-232-C as of 1969 defines:

- Electrical signal characteristics such as voltage levels, signaling rate, timing and slew-rate of signals, voltage withstand level, short-circuit behavior, and maximum load capacitance.
- Interface mechanical characteristics, pluggable connectors and pin identification.
- Functions of each circuit in the interface connector.
- Standard subsets of interface circuits for selected telecom applications.

The standard does not define such elements as

- character encoding (for example, ASCII, Baudot code or EBCDIC)
- the framing of characters in the data stream (bits per character, start/stop bits, parity)
- protocols for error detection or algorithms for data compression
- bit rates for transmission, although the standard says it is intended for bit rates lower than 20,000 bits per second. Many modern devices support speeds of 115,200 bit/s and above

Details of character format and transmission bit rate are controlled by the serial port hardware, often a single integrated circuit called a UART that converts data from parallel to asynchronous start-stop serial form. Details of voltage levels, slew rate, and short-circuit behavior are typically controlled by a line-driver that converts from the UART's logic levels to RS-232 compatible signal levels, and a receiver that converts from RS-232 compatible signal levels to the UART's logic levels.

History

RS-232 was first introduced in 1962. The original DTEs were electromechanical teletypewriters and the original DCEs were (usually) modems. When electronic terminals (smart and dumb) began to be used, they were often designed to be interchangeable with teletypes, and so supported RS-232. The C revision of the standard was issued in 1969 in part to accommodate the electrical characteristics of these devices.

Since application to devices such as computers, printers, test instruments, and so on was not considered by the standard, designers implementing an RS-232 compatible interface on their equipment often interpreted the requirements idiosyncratically. Common problems were non-standard pin assignment of circuits on connectors, and incorrect or missing control signals. The lack of adherence to the standards produced a thriving industry of breakout boxes, patch boxes, test equipment, books, and other aids for the connection of disparate equipment. A common deviation from the standard was to drive the signals at a reduced voltage: the standard requires the transmitter to use +12V and −12V, but requires the receiver to distinguish voltages as low as +3V and -3V. Some manufacturers therefore built transmitters that supplied +5V and -5V and labeled them as "RS-232 compatible."

Later personal computers (and other devices) started to make use of the standard so that they could connect to existing equipment. For many years, an RS-232-compatible port was a standard feature for serial communications, such as modem connections, on many computers. It remained in widespread use into the late 1990s. In personal computer peripherals it has largely been supplanted by other interface standards, such as USB. RS-232 is still used to connect older designs of peripherals, industrial equipment (such as PLCs), console ports and special purpose equipment such as a cash drawer for a cash register.

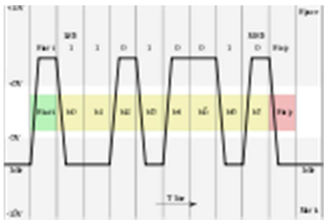
The standard has been renamed several times during its history as the sponsoring organization changed its name, and has been variously known as EIA RS-232, EIA 232, and most recently as TIA 232. The standard continued to be revised and updated by the Electronic Industries Alliance and since 1988 by the Telecommunications Industry Association (TIA). Revision C was issued in a document dated August 1969. Revision D was issued in 1986. The current revision is *TIA-232-F Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange*, issued in 1997. Changes since Revision C have been in timing and details intended to improve harmonization with the CCITT standard V.24, but equipment built to the current standard will interoperate with older versions.

Related ITU-T standards include **V.24** (circuit identification) and **V.28** (signal voltage and timing characteristics).

Standard details

In RS-232, user data is sent as a time-series of bits. Both synchronous and asynchronous transmissions are supported by the standard. In addition to the data circuits, the standard defines a number of control circuits used to manage the connection between the DTE and DCE. Each data or control circuit only operates in one direction, that is, signaling from a DTE to the attached DCE or the reverse. Since transmit data and receive data are separate circuits, the interface can operate in a full duplex manner, supporting concurrent data flow in both directions. The standard does not define character framing within the data stream, or character encoding.

Voltage levels



Diagrammatic oscilloscope trace of voltage levels for an uppercase ASCII "K" character (0x4b) with 1 start bit, 8 data bits, 1 stop bit

The RS-232 standard defines the voltage levels that correspond to logical one and logical zero levels for the data transmission and the control signal lines. Valid signals are plus or minus 3 to 15 volts - the range near zero volts is not a valid RS-232 level. The standard specifies a maximum open-circuit voltage of 25 volts: signal levels of ±5 V, ±10 V, ±12 V, and ±15 V are all commonly seen depending on the power supplies available within a device. RS-232 drivers and receivers must be able to withstand indefinite short circuit to ground or to any voltage level up to ±25 volts. The slew rate, or how fast the signal changes between levels, is also controlled.

For data transmission lines (TxD, RxD and their secondary channel equivalents) logic one is defined as a negative voltage, the signal condition is called marking, and has the functional significance. Logic zero is positive and the signal condition is termed spacing. Control signals are logically inverted with respect to what one would see on the data transmission lines. When one of these signals is active, the voltage on the line will be between +3 to +15 volts. The inactive state for these signals would be the opposite voltage condition, between -3 and -15 volts. Examples of control lines would include request to send (RTS), clear to send (CTS), data terminal ready (DTR), and data set ready (DSR).

Because the voltage levels are higher than logic levels typically used by integrated circuits, special intervening driver circuits are required to translate logic levels. These also protect the device's internal circuitry from short circuits or transients that may appear on the RS-232 interface, and provide sufficient current to comply with the slew rate requirements for data transmission.

Because both ends of the RS-232 circuit depend on the ground pin being zero volts, problems will occur when connecting machinery and computers where the voltage between the ground pin on one end, and the ground pin on the other is not zero.

Connectors

RS-232 devices may be classified as Data Terminal Equipment (DTE) or Data Communication Equipment (DCE); this defines at each device which wires will be sending and receiving each signal. The standard recommended but did not make mandatory the D-subminiature 25 pin connector. In general and according to the standard, terminals and computers have male connectors with DTE pin functions, and modems have female connectors with DCE pin functions. Other devices may have any combination of connector gender and pin definitions. Many terminals were manufactured with female terminals but were sold with a cable with male connectors at each end; the terminal with its cable satisfied the recommendations in the standard.

Presence of a 25 pin D-sub connector does not necessarily indicate an RS-232-C compliant interface. For example, on the original IBM PC, a male D-sub was an RS-232-C DTE port (with a non-standard current loop interface on reserved pins), but the female D-sub connector was used for a parallel Centronics printer port. Some personal computers put non-standard voltages or signals on some pins of their serial ports.

The standard specifies 20 different signal connections. Since most devices use only a few signals, smaller connectors can often be used.

Pinouts

The following table lists commonly-used RS-232 signals and pin assignments.

Signal			Origin		DB-25 pin
Name	**Typical purpose**	**Abbreviation**	**DTE**	**DCE**	
Data Terminal Ready	OOB control signal: Tells DCE that DTE is ready to be connected.	DTR	•		20
Data Carrier Detect	OOB control signal: Tells DTE that DCE is connected to telephone line.	DCD		•	8
Data Set Ready	OOB control signal: Tells DTE that DCE is ready to receive commands or data.	DSR		•	6
Ring Indicator	OOB control signal: Tells DTE that DCE has detected a ring signal on the telephone line.	RI		•	22
Request To Send	OOB control signal: Tells DCE to prepare to accept data from DTE.	RTS	•		4
Clear To Send	OOB control signal: Acknowledges RTS and allows DTE to transmit.	CTS		•	5
Transmitted Data	Data signal: Carries data from DTE to DCE.	TxD	•		2
Received Data	Data signal: Carries data from DCE to DTE.	RxD		•	3
Common Ground		GND	common		7
Protective Ground		PG	common		1

The signals are named from the standpoint of the DTE. The ground signal is a common return for the other connections. The DB-25 connector includes a second "protective ground" on pin 1. Connecting this to pin 7 (signal reference ground) is a common practice but not essential.

Data can be sent over a secondary channel (when implemented by the DTE and DCE devices), which is equivalent to the primary channel. Pin assignments are described in following table:

Signal	Pin
Common Ground	7 (same as primary)
Secondary Transmitted Data (STD)	14
Secondary Received Data (SRD)	16
Secondary Request To Send (SRTS)	19
Secondary Clear To Send (SCTS)	13
Secondary Carrier Detect (SDCD)	12

3-wire and 5-wire RS-232

A minimal "3-wire" RS-232 connection consisting only of transmit data, receive data, and ground, is commonly used when the full facilities of RS-232 are not required. Even a two-wire connection (data and ground) can be used if the data flow is one way (for example, a digital postal scale that periodically sends a weight reading, or a GPS receiver that periodically sends position, if no configuration via RS-232 is necessary). When only hardware flow control is required in addition to two-way data, the RTS and CTS lines are added in a 5-wire version.

LCD

Overview

A **liquid crystal display** (**LCD**) is a thin, flat electronic visual display that uses the light modulating properties of liquid crystals (LCs). LCs do not emit light directly.

They are used in a wide range of applications including: computer monitors, television, instrument panels, aircraft cockpit displays, signage, etc. They are common in consumer devices such as video players, gaming devices, clocks, watches, calculators, and telephones. LCDs have displaced cathode ray tube(CRT) displays in most applications. They are usually more compact, lightweight, portable, and less expensive. They are available in a wider range of screen sizes than CRT and other flat panel displays.

LCDs are more energy efficient and offer safer disposal than CRTs. Its low electrical power consumption enables it to be used in battery-powered electronic equipment. It is an electronically-modulated optical device made up of any number of pixels filled with liquid crystals and arrayed in front of a light source (backlight) or reflector to produce images in colour or monochrome. The earliest discovery leading to the development of LCD technology, the discovery of liquid crystals, dates from 1888. By 2008, worldwide sales of televisions with LCD screens had surpassed the sale of CRT units.

Specifications

Important factors to consider when evaluating an LCD monitor:

- Resolution: The horizontal and vertical screen size expressed in pixels (e.g., 1024×768). Unlike CRT monitors, LCD monitors have a native-supported resolution for best display effect.
- Dot pitch: The distance between the centers of two adjacent pixels. The smaller the dot pitch size, the less granularity is present, resulting in a sharper image. Dot pitch may be the same both vertically and horizontally, or different (less common).

- Viewable size: The size of an LCD panel measured on the diagonal (more specifically known as active display area).
- Response time: The minimum time necessary to change a pixel's colour or brightness. Response time is also divided into rise and fall time. For LCD monitors, this is measured in btb (black to black) or gtg (gray to gray). These different types of measurements make comparison difficult.
- Input lag - a delay between the moment monitor receives the image over display link and the moment the image is displayed. Input lag is caused by internal digital processing such as image scaling, noise reduction and details enhancement, as well as advanced techniques like frame interpolation. Input lag can measure as high as 3-4 frames (in excess of 67 ms for a 60p/60i signal). Some monitors and TV sets feature a special "gaming mode" which disables most internal processing and sets the display to its native resolution.
- Refresh rate: The number of times per second in which the monitor draws the data it is being given. Since activated LCD pixels do not flash on/off between frames, LCD monitors exhibit no refresh-induced flicker, no matter how low the refresh rate. High-end LCD televisions now feature up to 240 Hz refresh rate, which allows advanced digital processing to insert additional interpolated frames to smooth up motion, especially with lower-framerate 24p material like the Blu-ray disc. However, such high refresh rates may not be supported by pixel response times, and additional processing can introduce considerable input lag.
- Viewing angle: (coll., more specifically known as viewing direction).

Passive-matrix and active-matrix addressed LCDs

A general purpose alphanumeric LCD, with two lines of 16 characters.

LCDs with a small number of segments, such as those used in digital watches and pocket calculators, have individual electrical contacts for each segment. An external dedicated circuit supplies an electric charge to control each segment. This display structure is unwieldy for more than a few display elements.

Small monochrome displays such as those found in personal organizers, or older laptop screens have a passive-matrix structure employing super-twisted nematic (STN) or double-layer STN (DSTN) technology—the latter of which addresses a colour-shifting problem with the former—and colour-STN (CSTN)—wherein colour is added by using an internal filter. Each row or column of the display has a single electrical circuit. The pixels are addressed one at a time by row and column addresses. This type of display is called *passive-matrix addressed* because the pixel must retain its state between refreshes without the benefit of a steady electrical charge. As the number of pixels (and, correspondingly, columns and rows) increases, this type of display becomes less feasible. Very slow response times and poor contrast are typical of passive-matrix addressed LCDs.

High-resolution colour displays such as modern LCD computer monitors and televisions use an active matrix structure. A matrix of thin-film transistors (TFTs) is added to the polarizing and colour filters. Each pixel has its own dedicated transistor,

allowing each column line to access one pixel. When a row line is activated, all of the column lines are connected to a row of pixels and the correct voltage is driven onto all of the column lines. The row line is then deactivated and the next row line is activated. All of the row lines are activated in sequence during a refresh operation. Active-matrix addressed displays look "brighter" and "sharper" than passive-matrix addressed displays of the same size, and generally have quicker response times, producing much better images.

Blue Phase mode

Blue phase LCDs do not require a liquid crystal top layer. Blue phase LCDs are relatively new to the market, and very expensive because of the low volume of production. They provide a higher refresh rate than normal LCDs, but normal LCDs are still cheaper to make and actually provide better colours and a sharper image.

Military use of LCD monitors

LCD monitors have been adopted by the military instead of CRT displays due to being smaller, lighter and more efficient. Using night vision imaging systems with an LCD monitor is needed to have the monitor be compliant with MIL-L-3009 (formerly MIL-L-85762A). These LCD monitors go through extensive certification so that they pass the standards for the military. These include MIL-STD-901D - High Shock (Sea Vessels), MIL-STD-167B - Vibration (Sea Vessels), MIL-STD-810F – Field Environmental Conditions (Ground Vehicles and Systems), MIL-STD-461E/F – EMI/RFI (Electromagnetic Interference/Radio Frequency Interference), MIL-STD-740B – Airborne/Structureborne Noise, and TEMPEST - Telecommunications Electronics Material Protected from Emanating Spurious Transmissions.

Bascom-AVR Basic Compiler

BASCOM-AVR is the original **Windows BASIC COMPILER** for the **AVR** family**.** It is designed to run on **W95/W98/NT/W2000/XP and Vista**

Key Benefits

- Structured BASIC with labels.
- Structured programming with IF-THEN-ELSE-END IF, DO-LOOP, WHILE-WEND, SELECT- CASE.
- Fast machine code instead of interpreted code.
- Variables and labels can be as long as 32 characters.
- Bit, Byte, Integer, Word, Long, Single , DOUBLE and String variables.
- Support for the DOUBLE. Not found in any AVR compiler, BASCOM gives you the advantage to crunch huge numbers with the DOUBLE(8 byte Floating Point)
- Large set of Trig Floating point functions.
- Date & Time calculation functions.
- Compiled programs work with all AVR microprocessors that have internal memory.
- Statements are highly compatible with Microsoft's VB/QB.
- Special commands for LCD-displays , I2C chips and 1WIRE chips, PC keyboad, matrix keyboad, RC5 reception, software UART, SPI , graphical LCD, send IR RC5, RC6 or Sony code.
- TCP/IP with W3100A chip.
- Local variables, user functions, library support.
- Integrated terminal emulator with download option..
- Integrated simulator for testing.
- Integrated ISP programmer (application note AVR910.ASM).
- Integrated STK200 programmer and STK300 programmer. Also supported is the low cost Sample Electronics programmer. Can be built in 10 minutes! Many other programmers supported via the Universal Interface.
- Editor with statement highlighting.
- Context sensitive help.
- Perfectly matches the following boards :
  - MAVRIC and the MAVRIC-II from BDMICRO.
  - AVR robot controller (ARC 1.1) from L. Barello
  - Active Mega8535 Micro Board from Active Robots
- DEMO version compiles 4KB of code. Well suited for the AT90S2313.
- English an German Books available

Supported Statements

Decision and structures

IF, THEN, ELSE, ELSEIF, END IF, DO, LOOP, WHILE, WEND, UNTIL, EXIT DO, EXIT WHILE, FOR, NEXT, TO, STEP, EXIT FOR, ON .. GOTO/GOSUB, SELECT, CASE.

Input and output

PRINT, INPUT, INKEY, PRINT, INPUTHEX, LCD, UPPERLINE, LOWERLINE,DISPLAY ON/OFF, CURSOR ON/OFF/BLINK/NOBLINK, HOME, LOCATE, SHIFTLCD LEFT/RIGHT, SHIFTCURSOR LEFT/RIGHT, CLS, DEFLCDCHAR, WAITKEY, INPUTBIN, PRINTBIN, OPEN, CLOSE, DEBOUNCE, SHIFTIN, SHIFTOUT, GETATKBD, SPC, SERIN, SEROUT

Numeric functions

AND, OR, XOR, INC, DEC, MOD, NOT, ABS, BCD, LOG, EXP, SQR, SIN,COS,TAN,ATN, ATN2, ASIN, ACOS, FIX, ROUND, MOD, SGN, POWER, RAD2DEG, DEG2RAD, LOG10, TANH, SINH, COSH.

I2C

I2CSTART, I2CSTOP, I2CWBYTE, I2CRBYTE, I2CSEND and I2CRECEIVE.

1WIRE

1WWRITE, 1WREAD, 1WRESET, 1WIRECOUNT, 1WSEARCHFIRST, 1WSEARCHNEXT.

SPI

SPIINIT, SPIIN, SPIOUT, SPIMOVE.

Interrupt programming

ON INT0/INT1/TIMER0/TIMER1/SERIAL, RETURN, ENABLE, DISABLE, COUNTERx, CAPTUREx, INTERRUPTS, CONFIG, START, LOAD.

Bit manipulation

SET, RESET, ROTATE, SHIFT, BITWAIT, TOGGLE.

Variables

DIM, BIT , BYTE , INTEGER , WORD, LONG, SINGLE, STRING , DEFBIT, DEFBYTE, DEFINT, DEFWORD.

Miscellaneous

REM, ' , SWAP, END, STOP, CONST, DELAY, WAIT, WAITMS, GOTO, GOSUB, POWERDOWN, IDLE, DECLARE, CALL, SUB, END SUB, MAKEDEC, MAKEBCD, INP,OUT, ALIAS, DIM , ERASE, DATA, READ, RESTORE, INCR, DECR, PEEK, POKE, CPEEK, FUNCTION, READMAGCARD, BIN2GREY, GREY2BIN, CRC8, CRC16, CHECKSUM.

Compiler directives

$INCLUDE, $BAUD and $CRYSTAL, $SERIALINPUT, $SERIALOUTPUT, $RAMSIZE, $RAMSTART, $DEFAULT XRAM, $ASM-$END ASM, $LCD, $EXTERNAL, $LIB.

String manipulation

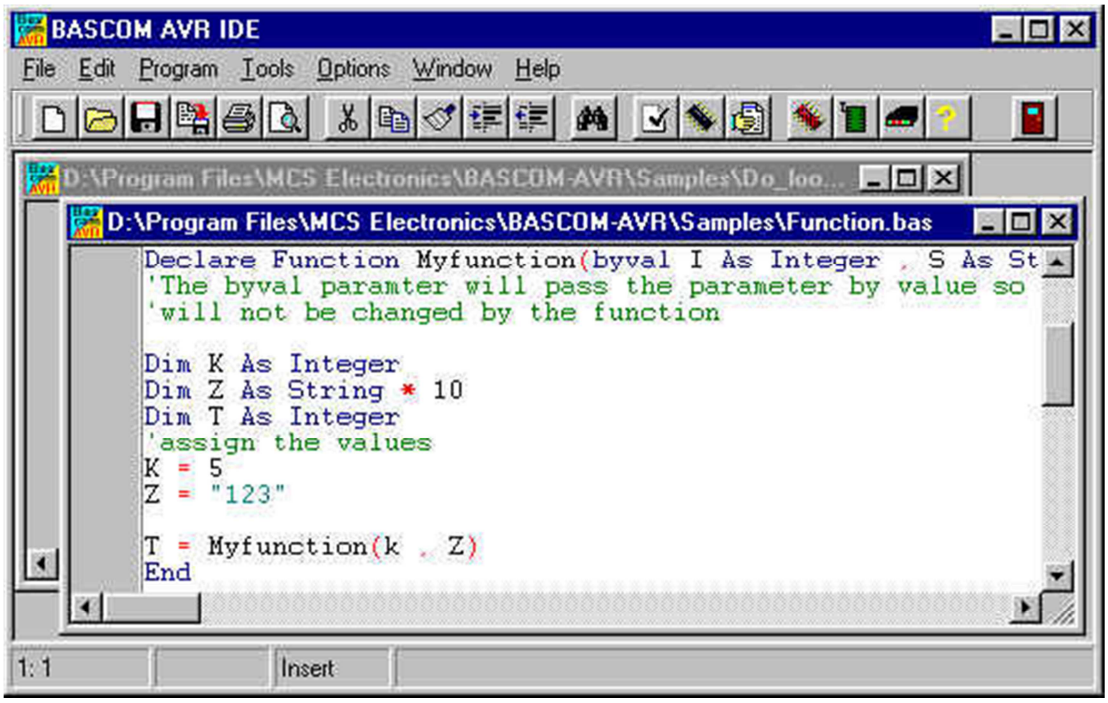
STRING, SPACE, LEFT, RIGHT, MID, VAL, HEXVAL, LEN, STR, HEX, LTRIM, RTRIM, TRIM, LCASE, UCASE, FORMAT, FUSING, INSTR.

And many other functions, statements and directives

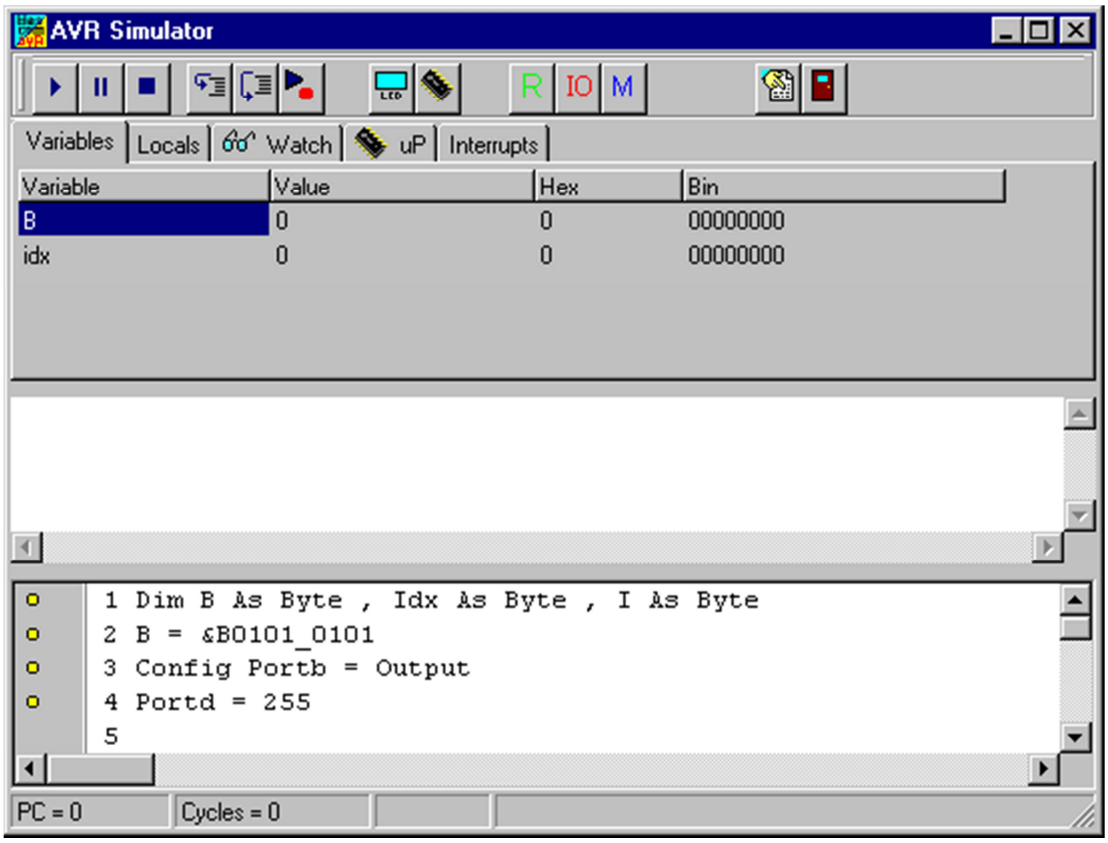
Usage

- **Write the program in BASIC**
- **Compile it to fast machine binary code**
- **Test the result with the integrated simulator(with additional hardware you can simulate the hardware too).**
- **Program the chip with one of the integrated programmers.**
  (hardware must be purchased separately)

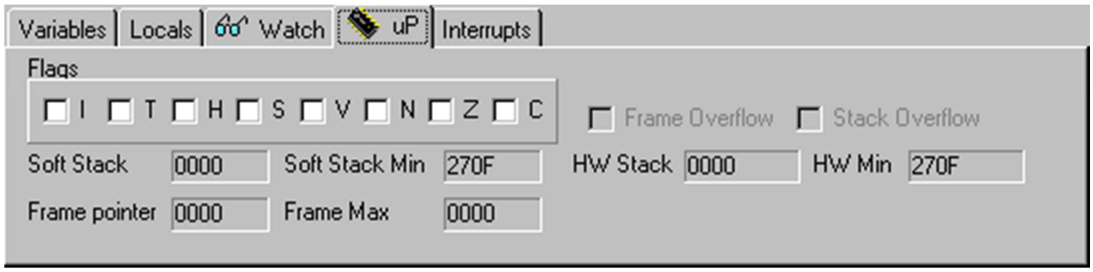
The program can be written in a comfortable MDI color coded editor.
Besides the normal editing features, the editor supports Undo, Redo, Bookmarks and block indention.



The simulator let you test your program before writing it to the uP.
You can watch variables, step through the program one line at the time or run to a specific line,
or you can alter variables.
To watch a variables value you can also point the mouse cursor over it.

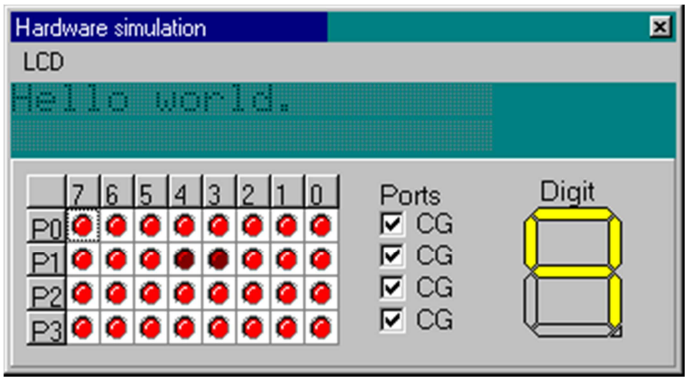


uP TAB of simulator



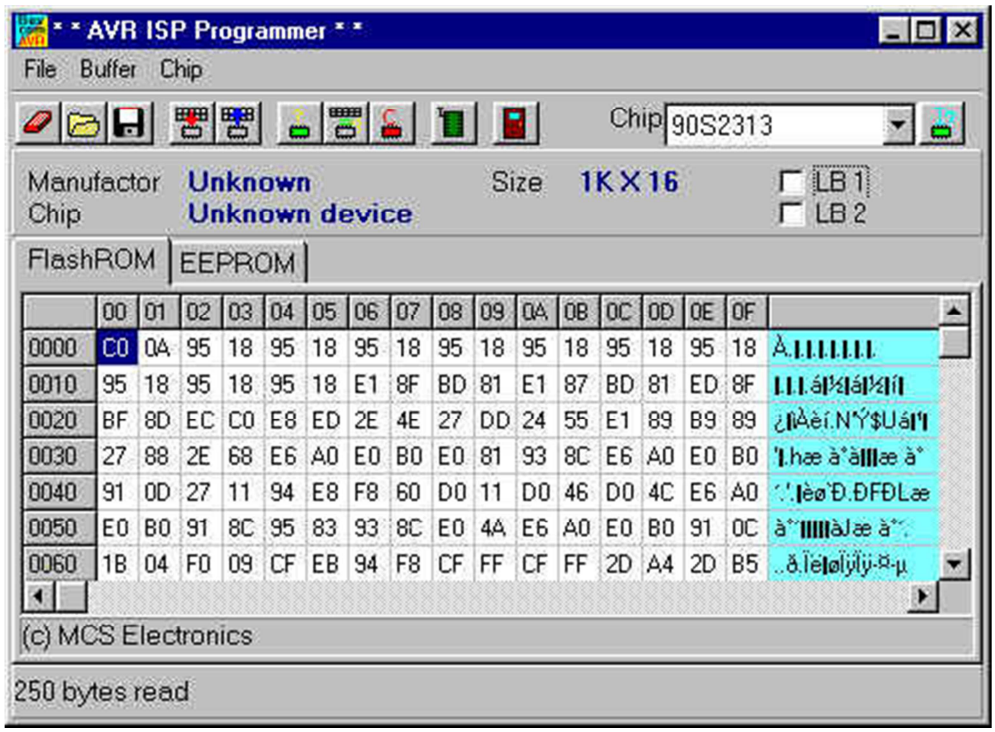
A powerful feature is the hardware emulator, to emulate the LCD display, and the ports.

The LCD emulator also emulates custom build LCD characters!



It is simple to control various graphical LCD displays.

You can even simulate the hardware ports with the special basmon monitor program!
When you are done with the simulator it is time to program the chip using one of the supported programmer drivers.



Visual Basic .NET

Visual Basic .NET (**VB.NET**) is an object-oriented computer programming language that can be viewed as an evolution of Microsoft's Visual Basic (VB) which is generally implemented on the Microsoft .NET Framework. Microsoft currently supplies Visual Basic Express Edition free of charge.

Visual Basic 2008 (VB 9.0)

Visual Basic 9.0 was released together with the Microsoft .NET Framework 3.5 on November 19, 2007.

For this release, Microsoft added many features, including:

- A true conditional operator, "If(boolean, value, value)", to replace the "IIf" function.
- Anonymous types
- Support for LINQ
- Lambda expressions
- XML Literals
- Type Inference
- Extension methods

Visual Basic 2010 (VB 10.0)

In 2007, Microsoft planned to use the Dynamic Language Runtime (DLR) for the upcoming Visual Basic 10 (2010), formerly known as VBx. However, Microsoft shifted to a co-evolution strategy between Visual Basic and sister language C# to bring both languages into closer parity with one another. Visual Basic's innate ability to interact dynamically with CLR and COM objects has been enhanced to work with Dynamic languages built on the DLR such as IronPython and IronRuby. The Visual Basic compiler was improved to infer line continuation in a set of common contexts in this version, lifting in many cases the requirement for the " _" line continuation character. Also in version 10.0 existing support of inline Functions was complemented with support for inline Subs as well as multi-line versions of both Sub and Function lambdas.

For a full list of language features added to Visual Basic 10.0 see the "What's New in Visual Basic 2010" document published by Microsoft.

.NET Framework 4 and Visual Basic 2010 were released together.

Visual Studio 2010 is available now, and was released on April 12 2010.

Also see Release Candidate.

Microsoft currently offers Visual Studio 2010 RC free-of-charge.

Relation to older versions of Visual Basic (VB6 and previous)

Whether Visual Basic .NET should be considered as just another version of Visual Basic or a completely different language is a topic of debate. This is not obvious, as once the methods that have been moved around and that can be automatically converted are accounted for, the basic syntax of the language has not seen many "breaking" changes, just additions to support new features like structured exception handling and short-circuited expressions. Two important data type changes occurred with the move to VB.NET. Compared to VB6, the Integer data type has been doubled in length from 16 bits to 32 bits, and the Long data type has been doubled in length from 32 bits to 64 bits. This is true for all versions of VB.NET. A 16-bit integer in all versions of VB.NET is now known as a Short. Similarly, the Windows Forms GUI editor is very similar in style and function to the Visual Basic form editor.

The version numbers used for the new Visual Basic (7, 7.1, 8, 9, ...) clearly imply that it is viewed by Microsoft as still essentially the same product as the old Visual Basic. The things that *have* changed significantly are the semantics—from those of an object-based programming language running on a deterministic, reference-counted engine based on COM to a fully object-oriented language backed by the .NET Framework, which consists of a combination of the Common Language Runtime (a virtual machine using generational garbage collection and a just-in-time compilation engine) and a far larger class library. The increased breadth of the latter is also a problem that VB developers have to deal with when coming to the language, although this is somewhat addressed by the *My* feature in Visual Studio 2005.

The changes have altered many underlying assumptions about the "right" thing to do with respect to performance and maintainability. Some functions and libraries no longer exist; others are available, but not as efficient as the "native" .NET alternatives. Even if they compile, most converted VB6 applications will require some level of refactoring to take full advantage of the new language.

Comparative samples

The following simple example demonstrates similarity in syntax between VB and VB.NET. Both examples pop up a message box saying "Hello, World" with an OK button.

Classic VB example:

```
Private Sub Command1_Click()

   MsgBox "Hello, World"

End Sub
```

A VB.NET example, MsgBox or the MessageBox class can be used:

```
Public Class Form1

   Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

      MsgBox("Hello, World")

   End Sub

End Class
```

```
Public Class Form1

   Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click

      MessageBox.Show("Hello, World")

   End Sub

End Class
```

- Both Visual Basic 6 and Visual Basic .NET will automatically generate the Sub and End Sub statements when the corresponding button is clicked in design

view. Visual Basic .NET will also generate the necessary Class and End Class statements. The developer need only add the statement to display the "Hello, World" message box.

- Note that all procedure calls must be made with parentheses in VB.NET, whereas in VB6 there were different conventions for functions (parentheses required) and subs (no parentheses allowed, unless called using the keyword Call).
- Also note that the names Command1 and Button1 are not obligatory. However, these are default names for a command button in VB6 and VB.NET respectively.
- In VB.NET, the Handles keyword is used to make the sub Button1_Click a handler for the Click event of the object Button1. In VB6, event handler subs must have a specific name consisting of the object's name ("Command1"), an underscore ("_"), and the event's name ("Click", hence "Command1_Click").
- There is a function called MsgBox in the Microsoft.VisualBasic namespace which can be used similarly to the corresponding function in VB6. There is a controversy about which function to use as a best practice (not only restricted to showing message boxes but also regarding other features of the Microsoft.VisualBasic namespace). Some programmers prefer to do things "the .NET way", since the Framework classes have more features and are less language-specific. Others argue that using language-specific features makes code more readable (for example, using int (C#) or Integer (VB.NET) instead of System.Int32).
- In VB 2008, the inclusion of ByVal sender as Object, ByVal e as EventArgs has become optional.

The following example demonstrates a difference between VB6 and VB.NET. Both examples close the active window.

Classic VB Example:

```
Sub cmdClose_Click()
    Unload Me
End Sub
```

A VB.NET example:

```
Sub btnClose_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
btnClose.Click
    Me.Close()
End Sub
```

Note the 'cmd' prefix being replaced with the 'btn' prefix, conforming to the new convention previously mentioned.

Visual Basic 6 did not provide common operator shortcuts. The following are equivalent:

VB6 Example:

```
Sub Timer1_Timer()
    Me.Height = Me.Height - 1
End Sub
```

VB.NET example:

```
Sub Timer1_Tick(ByVal sender As Object, ByVal e As EventArgs) Handles Timer1.Tick
    Me.Height -= 1
End Sub
```

Cross-platform and open-source development

The creation of open-source tools for VB.NET development have been slow compared to C#, although the Mono development platform provides an implementation of VB.NET-specific libraries and a VB.NET 8.0 compatible compiler written in VB.NET, as well as standard framework libraries such as Windows Forms GUI library.

Serial Communication with VB.Net

One of the thing I really miss while using VB.Net is the lack of serial communication support.Yes, you can use VB6's MsComm32.ocx but if you, like me, have installed VB.Net on a separate hard disk you may experience some problems installing and using it in design mode.
Don't forget moreover that if you use that control, you must register it on user machine when you deploy your application, loosing the 'XCopy installation mode' provided by VB.Net.
With this class you can easily use serial communication using native VB.Net and API features.
Before describing you how to use the class, let me point out that I've just written it as a demonstration of VB.Net interface to serial communication, you'd probably need to customize it for your special 'Rs232' needs.

Initializing and Opening the Com port

Create an instance of CRs232 then set COM parameters before invoking the Open method
Here's an example:

```
Dim moRS232 as New Rs232()
With moRs232
        .Port = 1                                   '// Uses COM1
        .BaudRate = 2400                             ' // 2400 baud rate
        .DataBit = 8                                ‘// 8 data
bits
        .StopBit = Rs232.DataStopBit.StopBit_1       '// 1 Stop bit
        .Parity = Rs232.DataParity.Parity_None        '// No Parity
        .Timeout = 500                              '// 500 ms of timeout admitted to get
all required bytes
End With
```

```
'// Initializes and Open
moRS232.Open ()
```
You can, optionally control the state of DTR/RTS lines after the Port is open
```
 '// Set state of RTS / DTS
moRS232.Dtr = True
moRS232.Rts = True
```

Transmitting data to COM Port

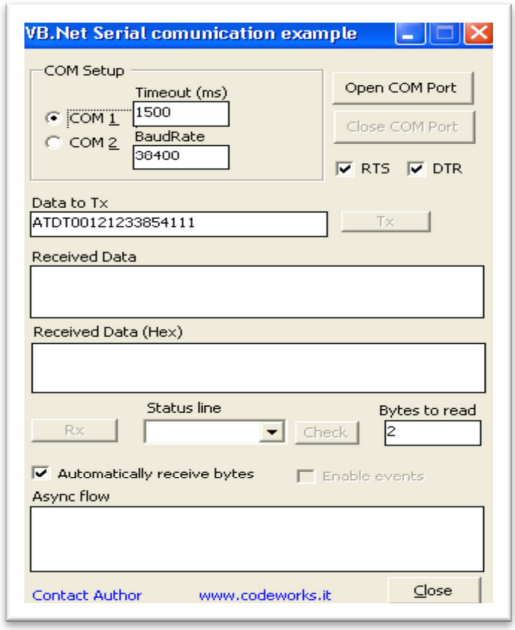
The class has 2 buffers one for Tx and one for Rx, to transmit data just set the TxData property with the informations you wish to send then invoke the Tx method.
example:
```
moRS232.Write(txtTx.Text)
```

Receiving data from COM Port

Just invoke the Rx method passing it the number of bytes you want to read from COM port, then read the Rxdata property.
example:
```
moRS232.Read(10)                '// Gets 10 bytes from serial communication buffer
Dim sRead as String=moRs232.InputStreamString
```
Please note that thread is blocked for the period of time set by Timeout property and that in case the class cannot read all required bytes from COM buffer a Timeout exception is raised.
If the number of bytes to read is omitted, the class assumes 512 bytes have to be read from buffer.

The class is very simple and surely misses some error control, but as prefaced I just wanted to give you an example of what you can do with VB.Net without resorting to ocx'es or other 3rdy parties controls

More details



The zip file that you can download here includes also a small Windows Form example that can help you understand how to use my class.

How to use the sample

- Verify that COM1 or COM2 are not in use (sample uses COM1 or COM2 but class can handle more than 9 ports...)
- Select default timeout (in milliseconds) and Baudrate (other parameters like parity... are fixed inside code)
- Press **Open COM Port**
- Type the string you wish to send to device (e.g ATDT12345) and press **Tx**
- Verify on target device that bytes are sent.
- If you want to read some bytes from COM port, type the number of char you wish to read into **Bytes to read** textbox and press **Rx**
- Received data will be displayed inside Received data textbox

If not enough character are received inside allocated timeout, you will have an error and you will se what has

been read from serial port.
If you want to test the status of Carrier Detect (CD) or RTS and other lines just seelct it from Status line combobox and press Check

RTS and DTR checkboxes will allow you to change the status of RTS and DTR lines respectively.
Automatically receive bytes allows you to read incoming bytes after you press Tx button simulating a client-server connection.

Enabling events
This is the most requested features i got from you and finally, here it is!
Check **Enable events** to start intercepting events coming from serial port (CD,RTS,Ring...) and to get characters as soon as they get into COM port.

Events are signaled by a CommEvent event, and you can use Mask parameter to detect what event has occurred, if you set a number of character into Bytes to read texbox, as soon as those character arrives into serial port you will get a CommEvent that you can use to get received characters.
Here's an extract from sample code:

```
If (mask And Rs232.EventMasks.RxChar) > 0 Then
 Dim s as String= source.InputStreamString) '// Here you have a string with received characters
End If
```

Please note that in order to get data asyncronously you have to set **EnableEvents** property to true (see code on sample Form)

Chapter 3

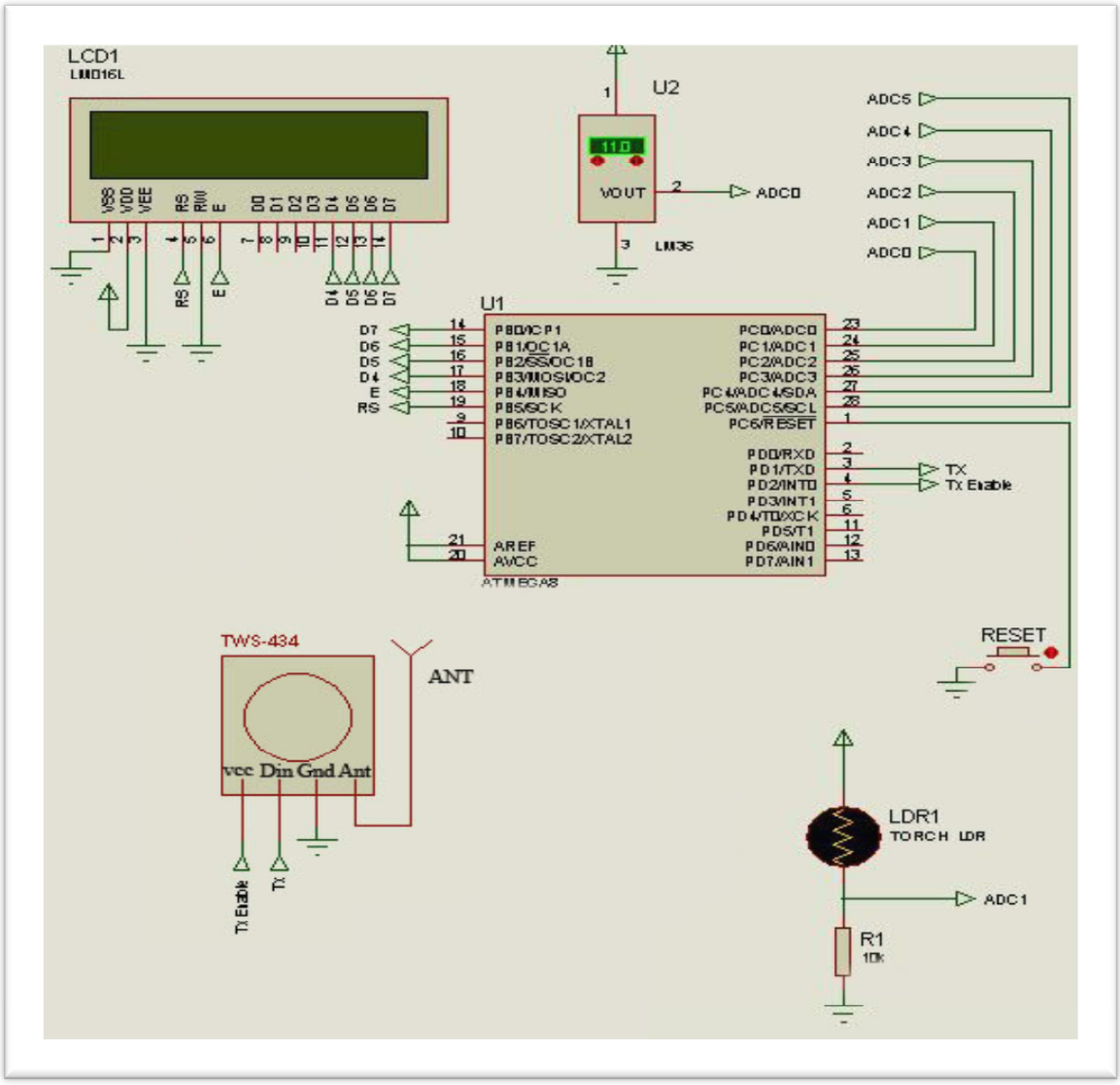
Proposed System

- **Sensor Node**
- **RF Receiver Board**
- **Applications of WSN**

Sensor Node

Node Circuit Diagram

In this circuit we are using “Atmel ATMega8” MCU that’s get Analog values from Sensors via ADC Ports then convert them to digital values to display them on LCD , after that it enable RF Transmitter Module using “Tx Enable” Pin then send sensors values in addition of node ID to RF Transmitter (TWS-434) throw MUC’s UART port using specific RF Packet Format.



Sensor Node Circuit Diagram

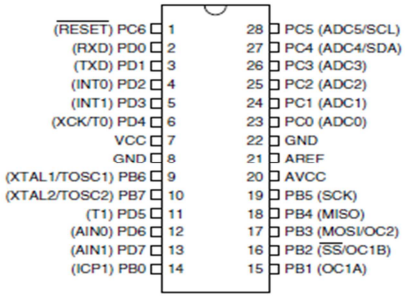
Node Components

- **Atmel AVR ATMEGA8 Microcontroller**

The purpose of using AVR ATmega8 Microcontroller that atmega8 meeting our requirements in additional of small size , low power usage and low cost .

ATmega8 Short Features :

- High-performance, Low-power 8-bit MCU
- 8K Bytes of programmable Flash memory
- 512 Bytes EEPROM
- 1K Byte Internal SRAM
- 6-channel ADC
- 23 Programmable I/O Lines
- 2.7 - 5.5V Operating Voltages
- 0 - 8 MHz Internal Crystal

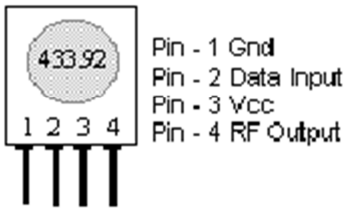


- **TWS-434 RF Transmitter**

TWS-434 is Small Size RF Transmitter Module that uses 434 MHz Signal, Support Serial Communication with low Power Usage and Low Cost.

TWS-434 Short Features :

- 433.92 MHz Frequency
- AM Modulation
- 2 - 12v Operating Voltages
- 2.4 Bps Data Rate
- ~30-70m Range

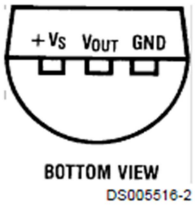


- **LM35 Temperature Sensor**

The LM35 series are precision integrated-circuit temperature sensors, whose output voltage is linearly proportional to the Celsius (Centigrade) temperature.

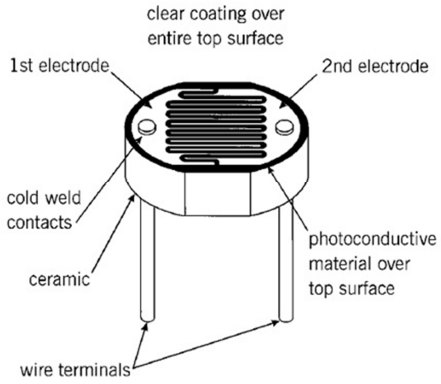
LM35 Short Features :

- Linear + 10.0 mV/°C scale factor
- 4 – 30v Operating Voltage
- Rated for full −55° to +150°C range
- Suitable for remote applications



- **LDR Light Sensor**

Light Dependent Resistors are very useful especially in light/dark sensor circuits. Normally the resistance of an LDR is very high, sometimes as high as 1000 000 ohms



- **LMB162A LCD**

LMB162A is 16x2 Characters LCD , in our sensor node it can be used to display local sensors values directly on node , or it can be removed and use data pins as I/O ports

LMB162A Short Features :

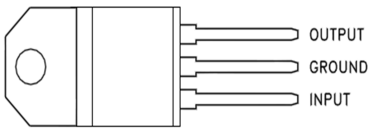
- 4 – 5.3v Operating Voltage
- 16x2 Characters
- LED Backlight

- **L7805CV Voltage Regulator**

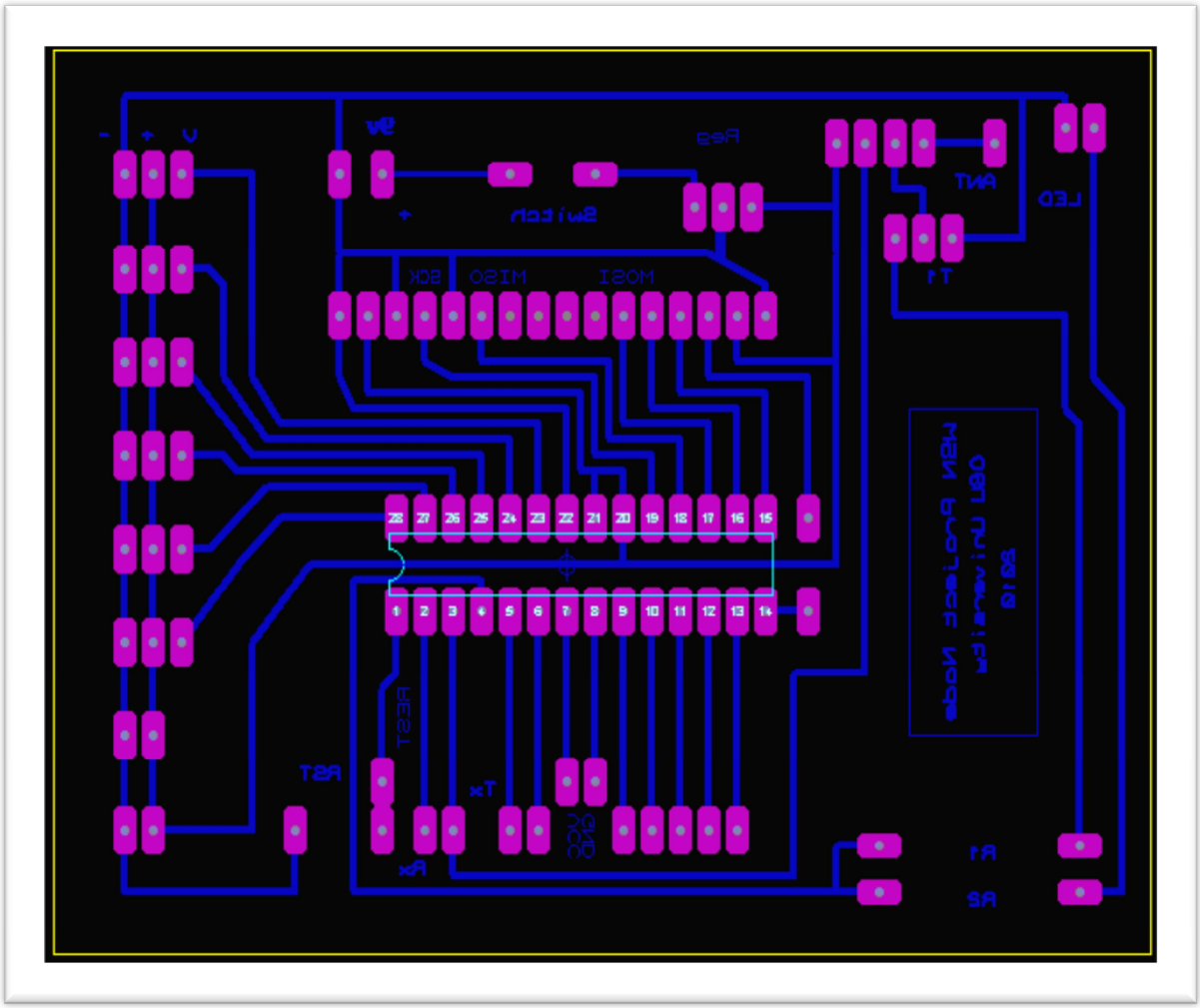
L7805 is a voltage Regulator Used to convert 9v Battery to 5v Output Voltage that used in All Node Components.

L7805CV Short Features :

- 5 – 18v Input Voltage
- 5v and 1.5A Output
- Low Voltage Drop

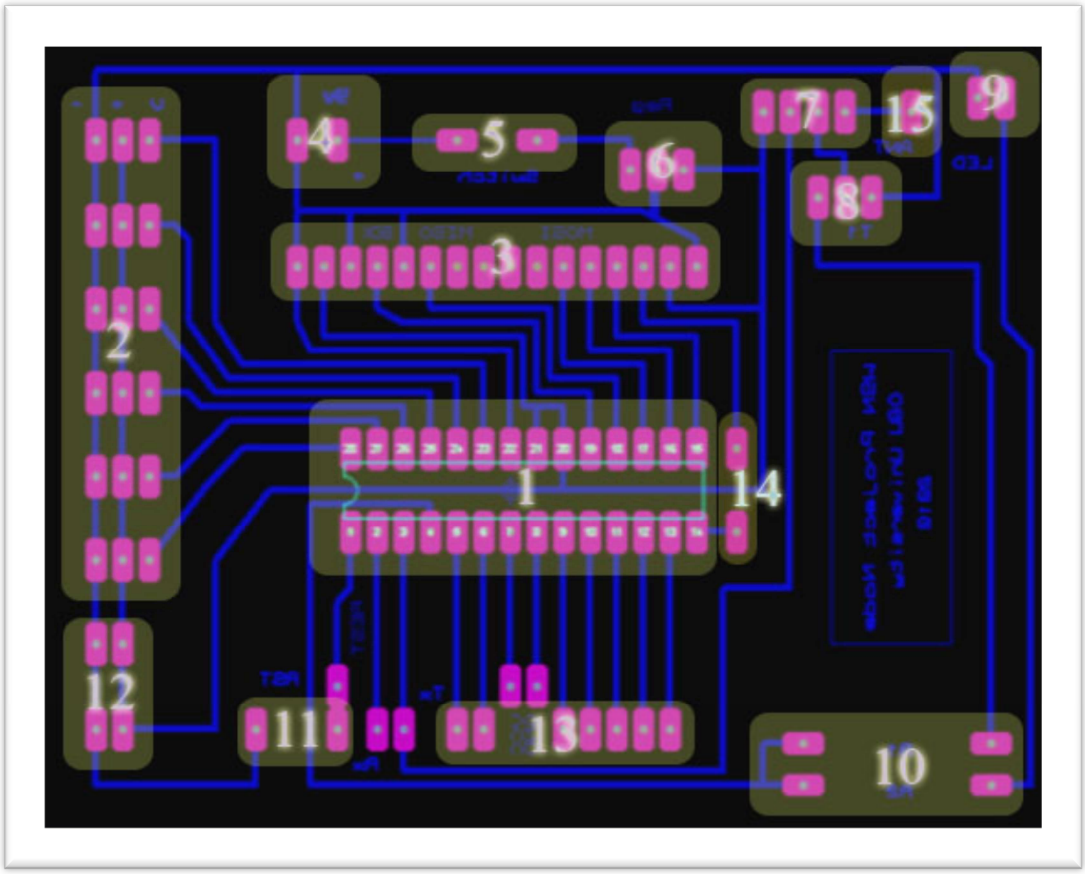


Node Layout



Sensor Node Layout

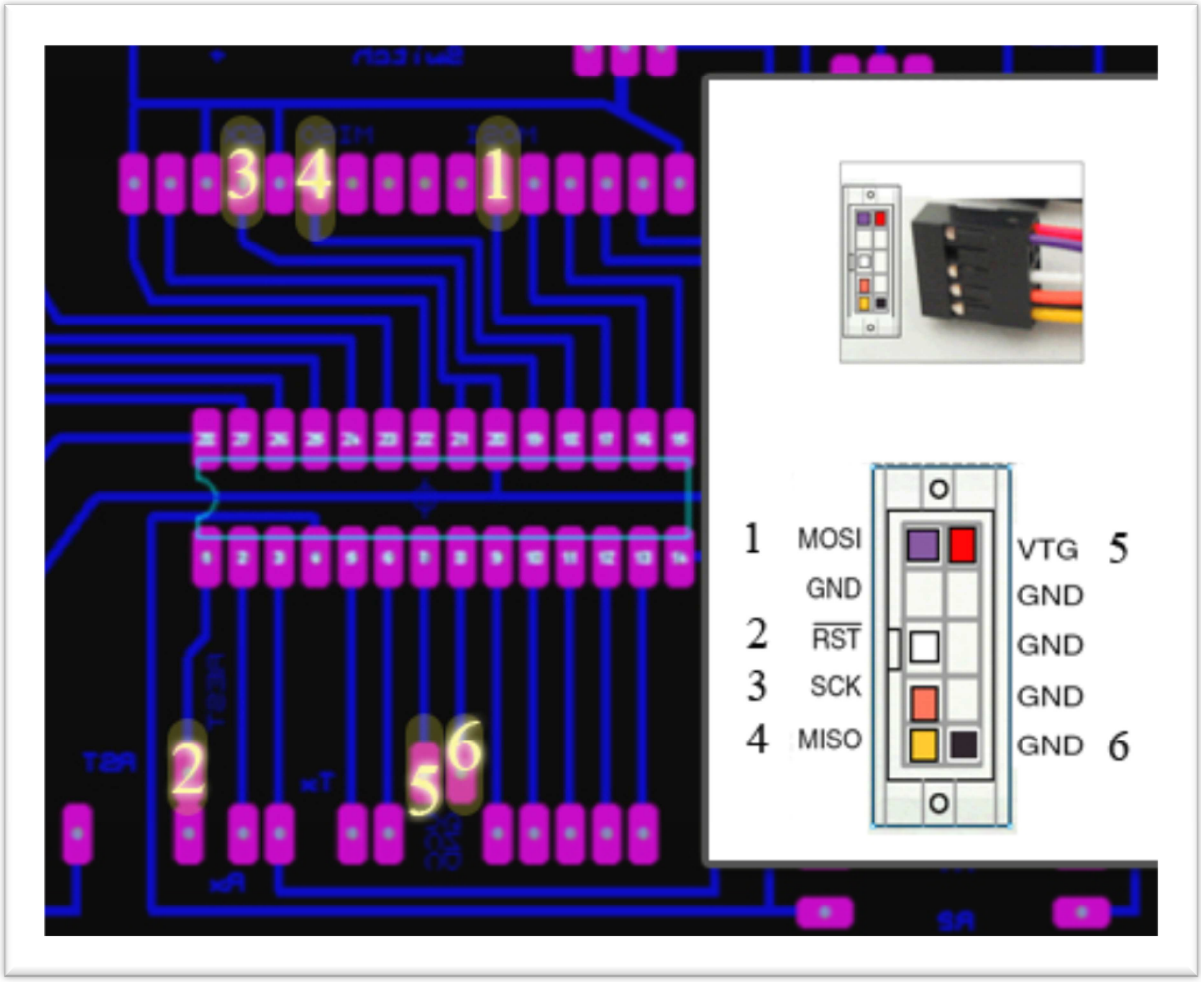
Layout Description



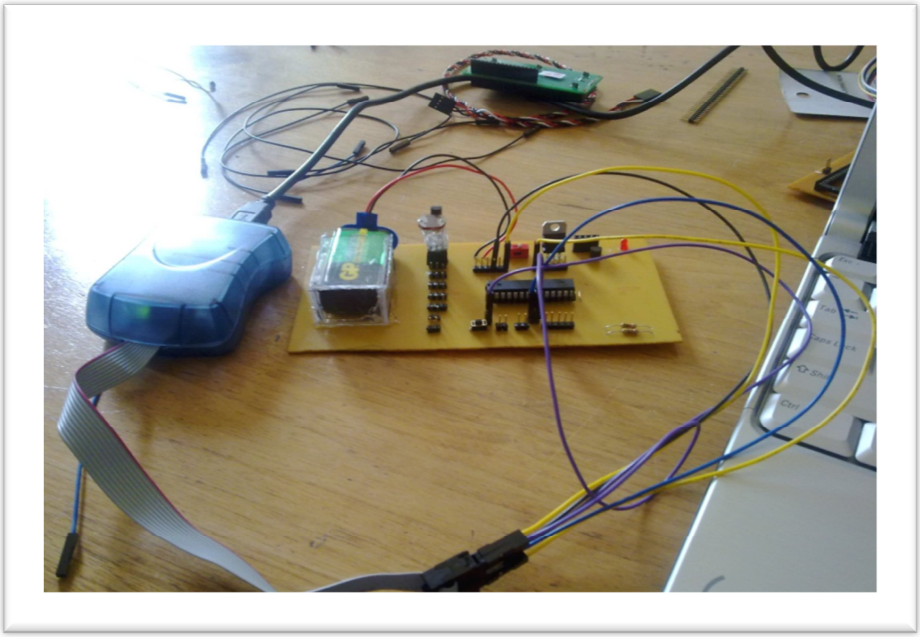
1. Atmega8 MCU Socket
2. Sensors Sockets
3. LCD Socket
4. 9v Battery Socket
5. ON / OFF Switch
6. 5v Regulator
7. RF Transmission Module Socket
8. Transmission Enable Transistor
9. Transmission LED
10. Resistors for Voltage Regulating
11. Reset Button
12. Additional 5v Output Ports
13. Additional Digital I/O Ports
14. Wire
15. Antenna

Node Programming Mode

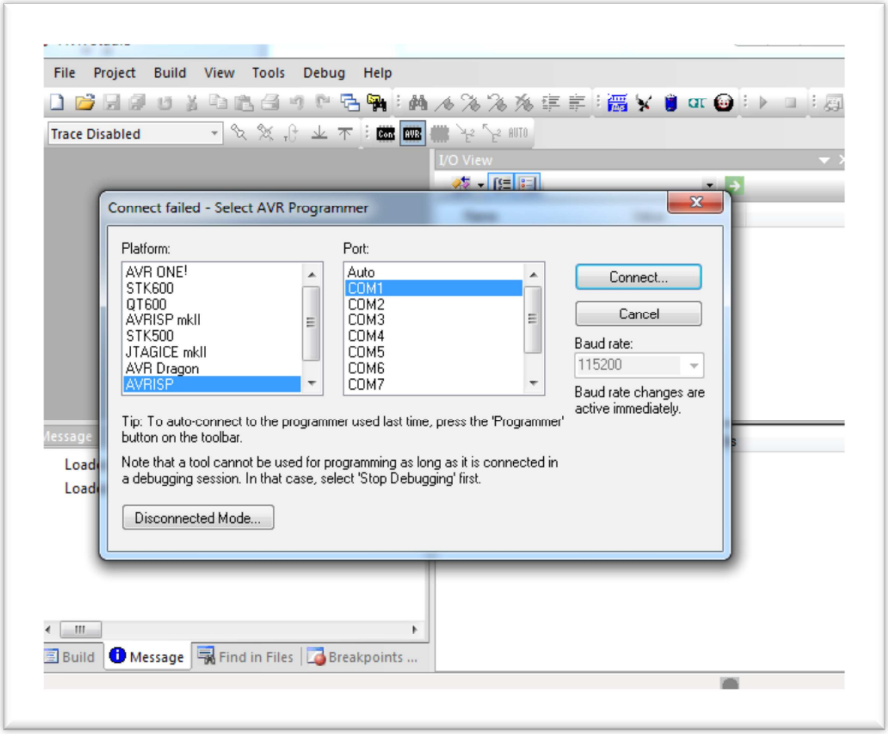
Node can be connected to AVR Programmer Using ISP Socket , ISP Socket can be connected to any ISP AVR Programmer , and we can use any programming software to write hex file to MCU *(Ex: AVR Studio)* , the figure below show how to connect ISP Pins to Node



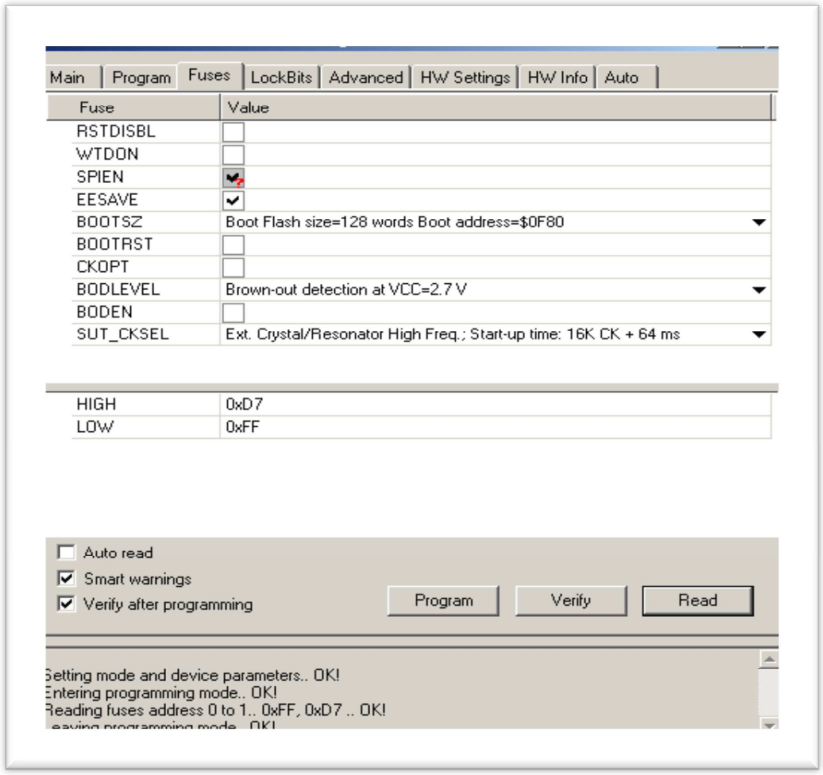
ISP Programmer Pins Connection



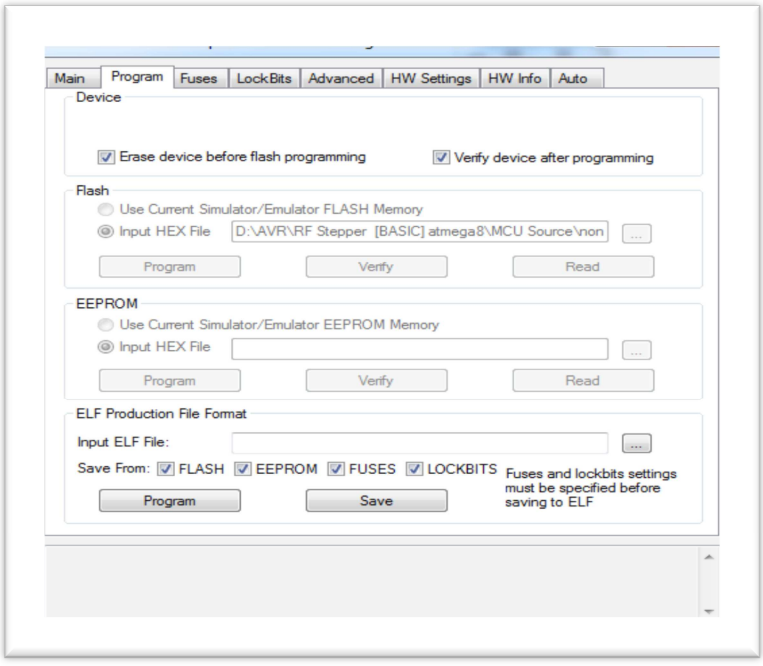
Sensor Node Connected to AVR ISP Programmer



AVR Studio Software – Programmer Type and Port Selection

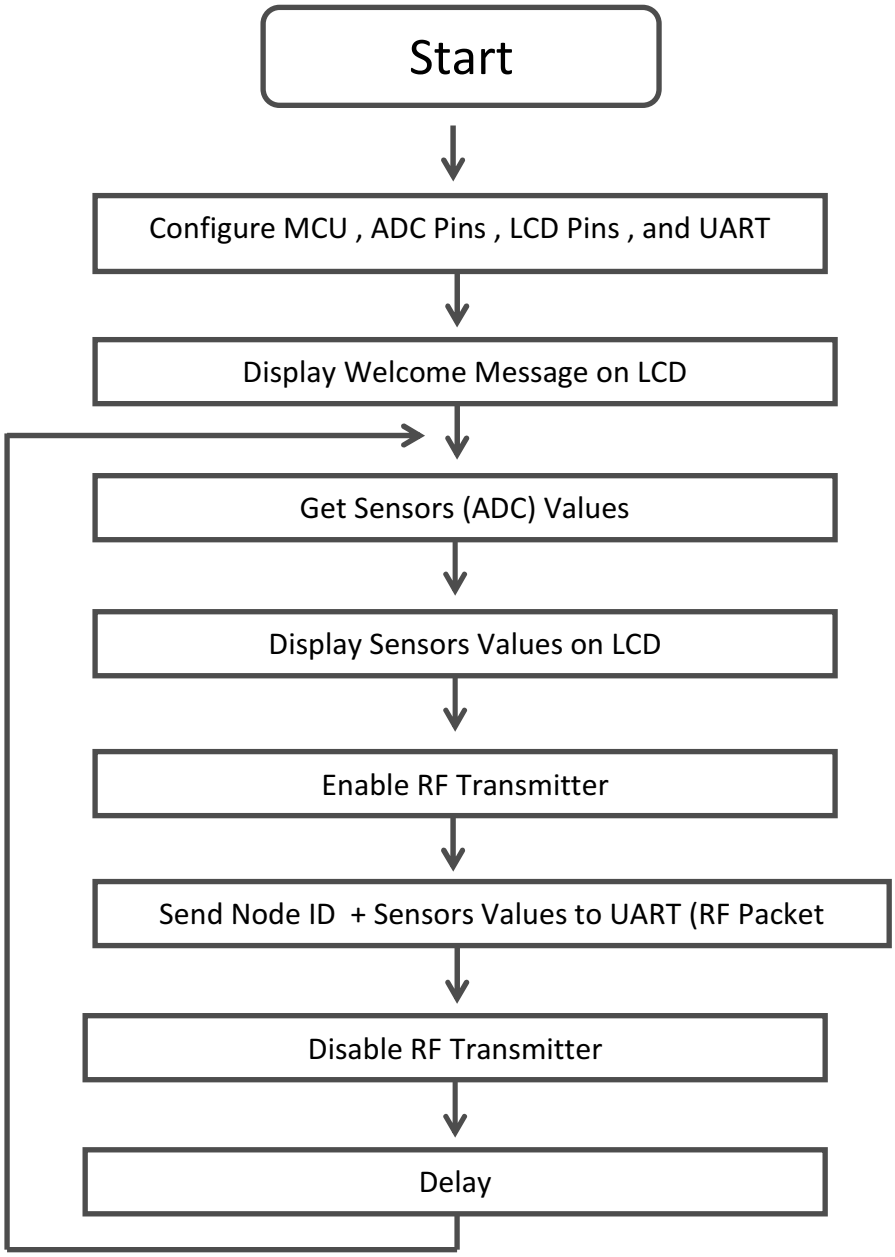


AVR Crystal Clock Settings



Write Hex File to AVR Flash Memory

Node MCU Program Flowchart



Node MCU Program Code

```
$regfile = "m8def.dat"
$crystal = 1000000

'------------------------------

Dim I As Integer
Dim Node_id As String * 1
Dim Sensor1 As Word
Dim Sensor2 As Word
Dim Ubfr As String * 50
Dim Chk As Integer

'------------------------------

Node_id = "1"

'-----------------

Call Sensors_init()
Call Lcd_init()
Call Uart_init()
Call Lcd_welcome()

Do

Sensor1 = Get_tempreture(0)
Sensor2 = Get_light(1)

 Call Lcd_clear()
Lcd "Node ID: " ; Node_id
  Lowerline
 Lcd "T: " ; Str(sensor1) ; " C  L: " ; Str(sensor2)
Ubfr = Node_id + "%" + Str(sensor1) + "-" + Str(sensor2)
Call Enable_tx()
Call Uart_send()
Waitms 20
Call Uart_send()
Call Disable_tx()

  Wait 1

Loop
```

Node MCU Program Code Description

$regfile = "m8def.dat"
$crystal = 1000000

Here we define the MCU type and Crystal Frequency

Dim I As Integer
Dim Node_id As String * 1
Dim Sensor1 As Word
Dim Sensor2 As Word
Dim Ubfr As String * 50
Dim Chk As Integer

Define variables to save data from functions

Node_id = "1"

Define the node ID , we have to use different ID for each Sensors Node.

Call Sensors_init()

Configure sensors ADC ports

Call Lcd_init()
Configure LCD Ports and Settings

Call Uart_init()

Configure Serial Communication Port and Settings

Call Lcd_welcome()

Display Welcome Message on LCD

Do
Start Loop

Sensor1 = Get_temperature(0)
Get Temperature sensor value that connected to sensor socket number "0" and save it into variable Sensor1

Sensor2 = Get_light(1)

Get Light Sensor value that connected to sensor socket number "1" and save it into variable Sensor1

Call Lcd_clear()

Clear LCD

Lcd "Node ID: " ; Node_id ;

Display Node ID on LCD

Lowerline

Go to the second LCD Line

Lcd "T: " ; Str(sensor1) ; " C L: " ; Str(sensor2)

Display Temperature and Light Sensors Values on LCD

Ubfr = Node_id + "%" + Str(sensor1) + "-" + Str(sensor2)

Load Node ID + Sensors Values to Variable “Ubfr” to Send it later using UART

Call Enable_tx()
Enable RF Transmission , we have to enable transmission before sending then disable it to avoid Signal Noises

Call Uart_send()

Send “Ubfr” variable Contents to RF Module Using UART

Waitms 20
Call Uart_send()

Wait 20ms and Send it Again

Call Disable_tx()

Disable RF Transmission.

Wait 1
Loop

Wait 1 Sec. Then Loop Again.

Node MCU Program Compiler

We used function to make a compiler for Sensor Node , those functions make node programming more easier and simpler , so the programmer can use those functions directly without having a knowledge in MCU pinout.

Node MCU Program Compiler Functions

Sub Sensors_init()
Config Adc = Single , Prescaler = Auto
End Sub

Function “sensors_init” used to initialize sensors pins.

Function Sensor_value(pin As Integer)
Sensor_value = Getadc(pin)
End Function

Function “Sensor_Value” can be used to get any sensor value directly using sensor pin value from 0 to 5.

Function Get_light(pin As Integer)
Get_light = Getadc(pin) / 2
End Function

Function “Get_light” can be used to get light sensor value directly using sensor pin value from 0 to 5 , this function divide ADC value by 2 to minimize the scale.

Sub Lcd_init()
Config Lcdmode = Port
Config Lcdpin = Pin , Db7 = Portb.0 , Db6 = Portb.1 , Db5 = Portb.2 , Db4 = Portb.3 , E = Portb.4 , Rs = Portb.5
Config Lcd = 16 * 2
End Sub

Function “Lcd_init” used to initialize LCD pins and configure LCD Size settings.

```
Function Get_temperature(pin As Integer)
Dim Tmpr As Single
Dim Tmpr_word As Word
 Tmpr_word = Getadc(pin)
Tmpr = Tmpr_word * 5
Tmpr = Tmpr / 1023
Tmpr = Tmpr * 100
Tmpr = Round(tmpr)
Get_temperature = Tmpr
End Function
```

Function “get_temperature” used to get Temperature sensor value in Celsius using sensor pin value , this function convert ADC value into Celsius Value.

```
Sub Lcd_clear()
Cls
End Sub
```

Function “Lcd_clear” used to clear LCD so we can display new contents again

```
Sub Lcd_welcome()
 Cls
 Locate 1 , 3
Lcd "WSN Project"
Locate 2 , 5
Lcd "Welcome"
Wait 1
End Sub
```

Function “Lcd_Welcome” used to display Welcome Message on LCD for 1 Sec , This function is using locate command to define start letter for every line that align the text in center.

```
Sub Uart_init()
Open "comd.1:1200,8,n,1" For Output As #1
Config Portd.2 = Output
End Sub
```

Function “Uart_init” used to initialize UART Port Settings and Transmitter Enable Pin that used in RF Transmission.

```
Sub Enable_tx()
Set Portd.2
End Sub
```

Function "Enable_tx" used to enable RF transmitter

```
Sub Disable_tx()
Reset Portd.2
End Sub
```

Function "Disable_tx" used to Disable RF transmitter

```
Sub Uart_send()
Dim Xx As Integer
Dim Chksb As String * 2
Xx = Check_sum(ubfr)
Chksb = Chr(xx)
Print #1 , "UU" ; ":" ; Ubfr ; ";" ; Chksb ; "UU"
End Sub
```

Function "Uart_send" used to send saved data in variable "Ubfr" to UART Port in RF Packet Format :

2 NULL BYTES	START BYTE	DATA	END BYTE	Checksum Byte	2 NULL Byte

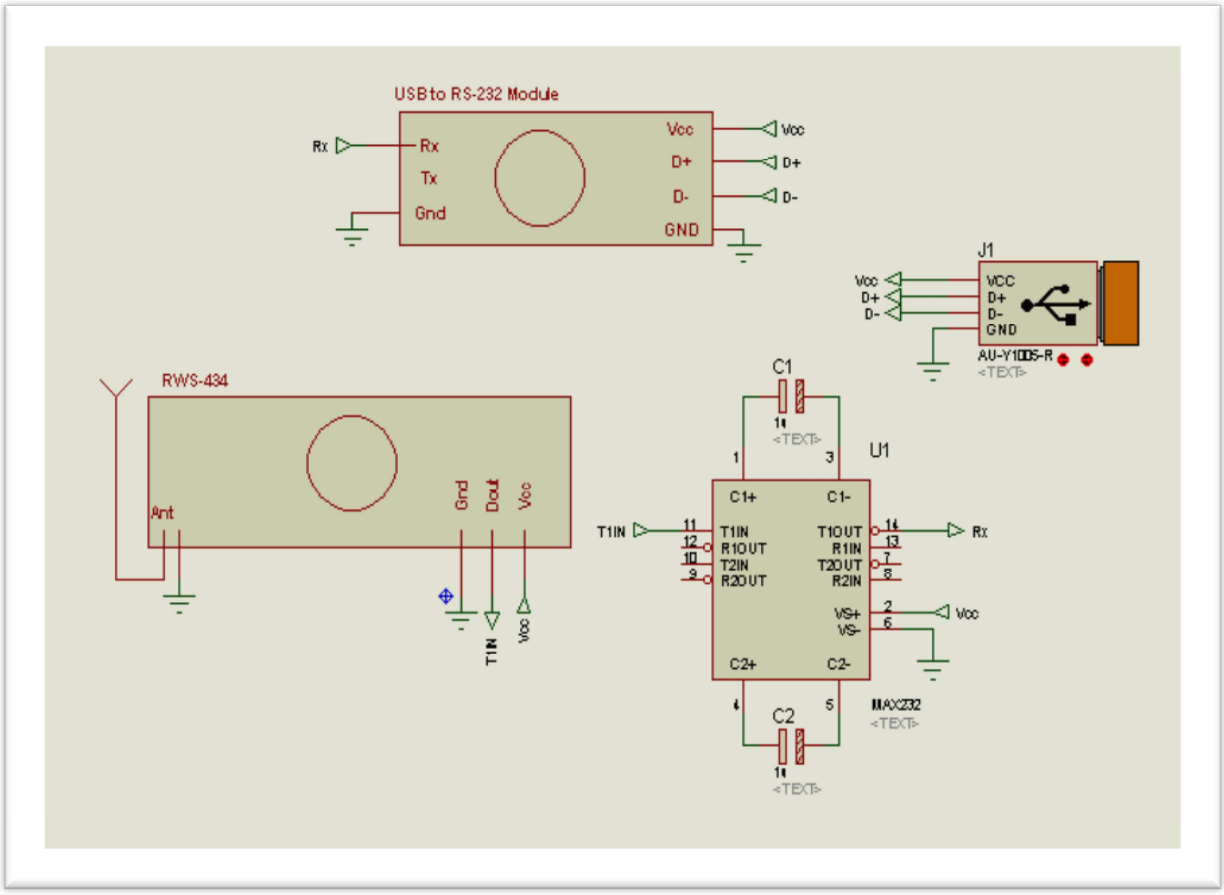
```
Function Check_sum(s$ As String * 50)
Dim Chkx As Integer
Dim M As Byte
Dim Bz As String * 10
Chkx = 0
For M = 1 To Len(s$)
Bz = Mid(s$ , M , 1)
 Chkx = Chkx + Asc(bz)
If Chkx > 100 Then Chkx = Chkx Mod 100
Next
Check_sum = Chkx
 End Function
```

Function "Check_sum" used to generate checksum byte to be used in RF Package.

RF Receiver Board

Receiver Diagram

in this Circuit we are using (USB to RS-233) Module to receive serial data from RF Module (RWS-434) into USB Port in Computer , we also using “MAX232” to convert voltage levels.



Receiver Board Diagram

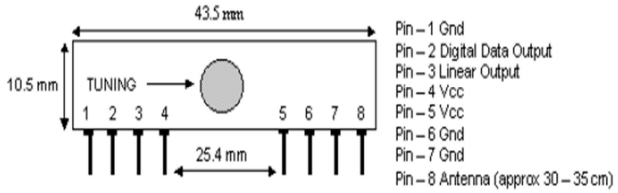
Node Components

- **RF Receiver Module (RWS-434)**

RWS-434 is Small Size RF Receiver Module that uses 434 MHz Signal, Support Serial Communication with low Power Usage and Low Cost.

RWS-434 Short Features :

- 433.92 MHz Frequency
- AM Modulation
- 4.5 - 5v Operating Voltages
- 2.4 Bps Data Rate
- ~30-70m Range



- **USB to RS232 Module**

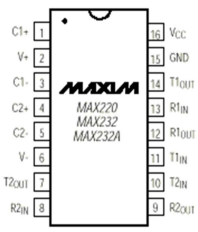
This Module used to Connect the Receiver Board into USB port in computer Directly without needing to RS232 (COM) port be available in computer , This Module Convert Serial Data from that's Received from RF Receiver Module into USB packet Format , we removed the USB and RS232 Ports From Module and Connect it Directly to Receiver Board.

- **Max232 RS232 Driver**

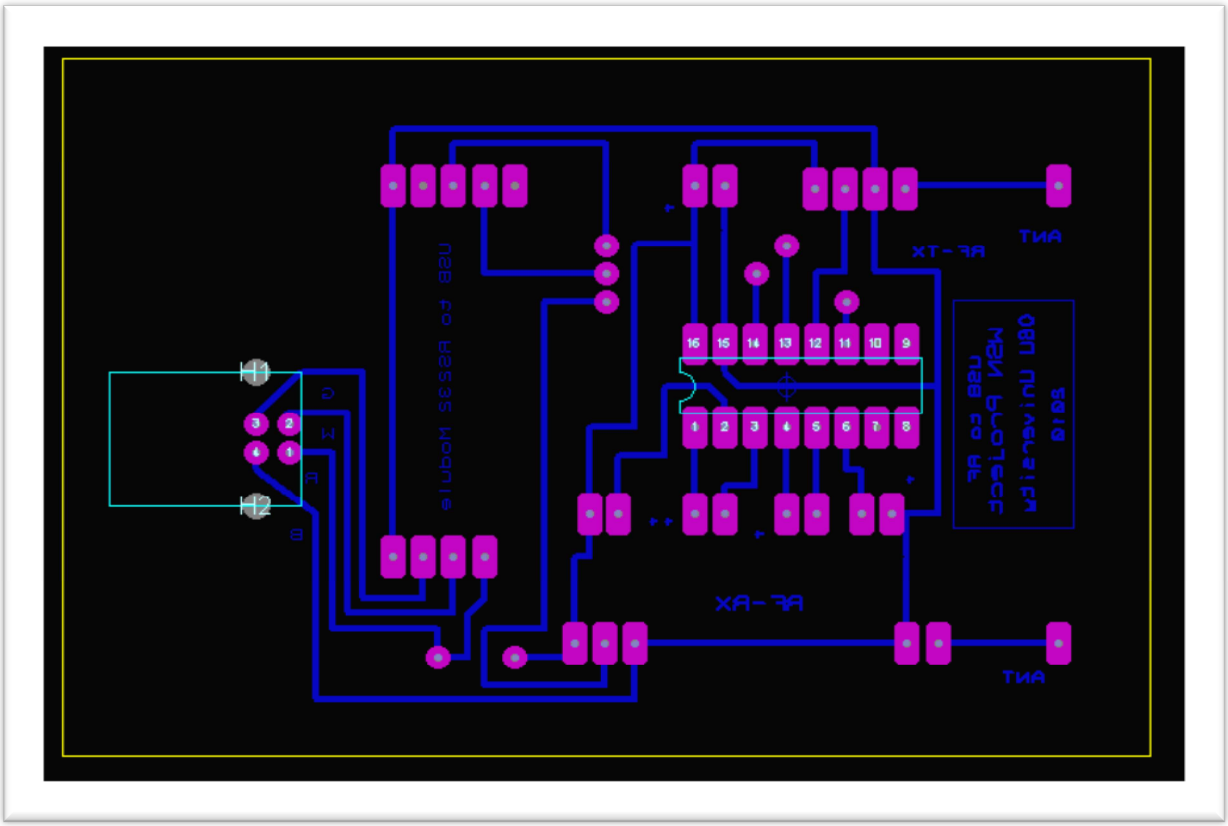
Because of RF Receiver Module uses data voltage levels 0,5v and "USB to RS232" Accept only RS232 Voltage Levels -12v , +12 v we are using max232 to convert those levels between RF Module And "USB to RS232" Module

Max232 Short Features :

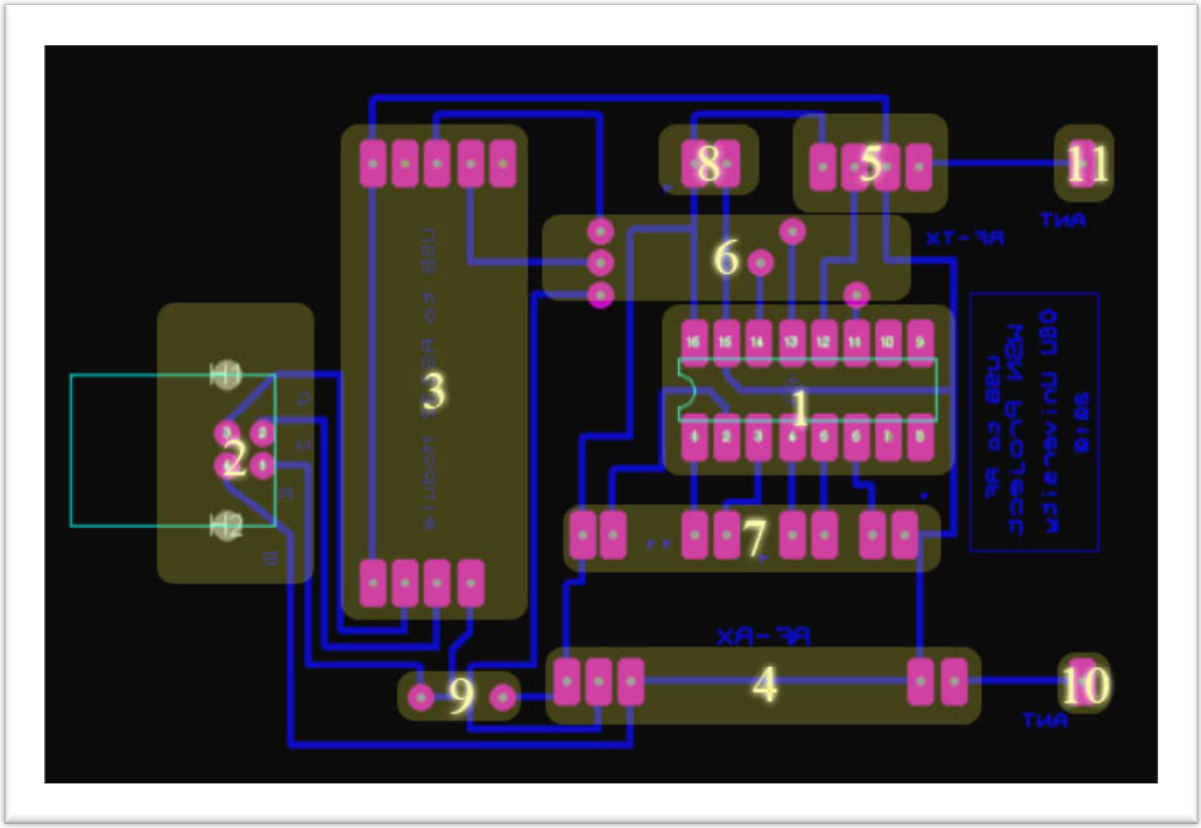
- 5v Operating Voltage
- 2 RS232 Drivers
- 120 Kbps Data Rate



Receiver Layout



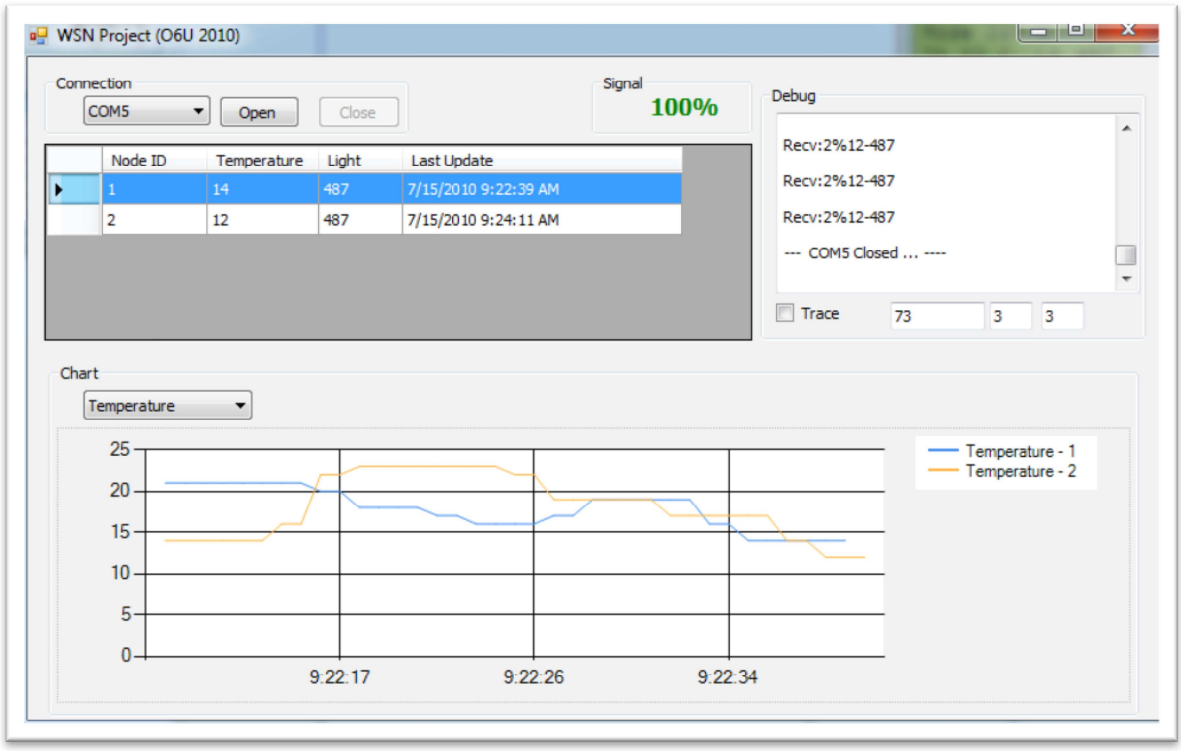
Receiver Layout Description



1. Max232 Socket
2. USB Port Socket
3. USB to RS232 Socket
4. RF Receiver Module Socket
5. RF Transmitter Module Socket (Optional)
6. Wires
7. Capacitors
8. Capacitor
9. Wire
10. Rx Antenna
11. Tx Antenna

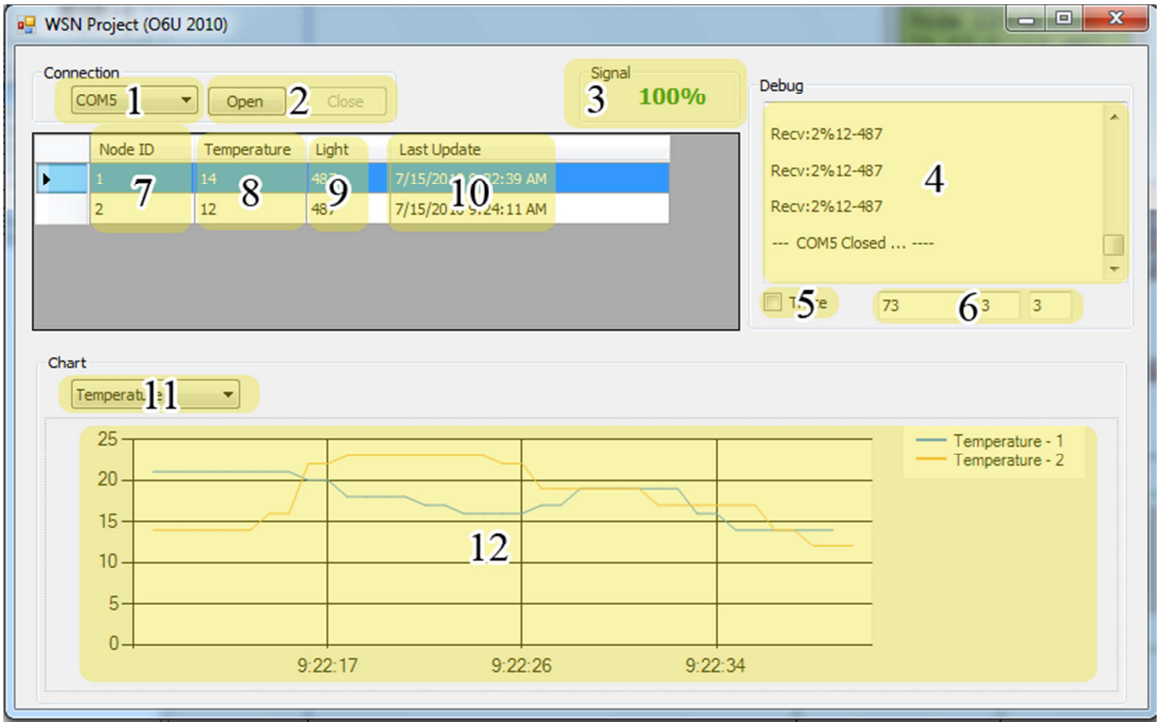
Receiver High Level Software

High Level Receiver Board Software Designed and Programmed Using Visual Basic .NET Language , This Program Receives the data from Receiver Board via USB Port and Display Each Node Sensors Data in Grid List in Additional of Signal Quality and Data Chart.



Receiver Board Software Main GUI Window.

Receiver High Level Software Description



1. Combo Box to Select which COM port will be used to receive data , when connecting the Receiver Board into USB Port it will Install a virtual COM Port Directly , that can be detected from From Device Manager.
2. Close and Open Buttons to Start Or Stop Receiving Data.
3. Signal Quality
4. Debug Box Display Successfully Received data command
5. Trace Mode Enable You to See All Received data into Debug Box even if it's wrong data.
6. Data Counters
7. First Columns in Grid Box Show Active Nodes IDs.
8. Second Column Display Temperature Sensors values in each Node.
9. Third Column Display Light Sensors Values in each Node.
10. Fourth Column Display the Time of Last Data Received for each Node.
11. Combo Box to Select which Sensors to show in Chart.
12. Data Chart for Selected Sensor that show Nodes Values Change Chart.

Receiver Software Source Code

```
Public Class Form1

    Dim snd_string As String
Dim last_snd_string As String

Dim data_arr(4) As String

    Dim data_add(3) As String

    Dim snd_c As Integer

    Dim s As String

    Dim s_buf As String

    Dim chksumnow As Boolean

    Dim chksumbyte_saved As Boolean

    Dim chksumbyte As String

    Dim sbuf_max As Integer

    Dim chart_name As String

Private Sub Form1_Leave(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Leave

        com1.Close()

        End

    End Sub

Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

        Dim a() As String

        Dim x As String

        a = IO.Ports.SerialPort.GetPortNames()

        For Each x In a

            ComboBox1.Items.Add(x)

        Next
```

```
        ComboBox1.Sorted = True
        ComboBox1.SelectedIndex = 0
        sensors_arr.SelectedIndex = 0
    End Sub

Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button2.Click
        Try
            com1.PortName = ComboBox1.SelectedItem
            com1.Open()

            TextBox1.AppendText(vbNewLine & "---  " & ComboBox1.SelectedItem & "
Opened , Waiting Nodes Signal  ----" & vbNewLine & vbNewLine)

            Button2.Enabled = False
            Button3.Enabled = True
        Catch
            MessageBox.Show(Err.Description)
        End Try
    End Sub
Private Sub com1_DataReceived(ByVal sender As Object, ByVal e As
System.IO.Ports.SerialDataReceivedEventArgs) Handles com1.DataReceived
        TextBox1.Invoke(New myDelegate(AddressOf DataReceived), New Object() {})
End Sub

Public Delegate Sub myDelegate()
    Function get_row_by_text(ByVal txt As String) As Integer
        Dim row_id As Integer
        row_id = -1
```

```
        For i = 0 To grid.RowCount - 1
            If grid.Rows(i).Cells(0).Value = txt Then
                row_id = i
                Exit For
            End If
        Next
        Return row_id
    End Function
    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button3.Click
        com1.Close()
        TextBox1.AppendText(vbNewLine & "---  " & ComboBox1.SelectedItem & "
Closed ... ----" & vbNewLine)
        Button2.Enabled = True
Button3.Enabled = False
    End Sub
Private Sub sensors_arr_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles sensors_arr.SelectedIndexChanged
        Chart1.Series.Clear()
    End Sub
Function check_sum(ByVal s As String)
        Dim x As Integer
        x = 0
        For i = 1 To Len(s)
            x = x + Asc(Mid(s, i, 1))
            If (x > 100) Then x = x Mod 100
Next
    Return x
    End Function
```

```
Public Sub DataReceived()
        On Error Resume Next
        sbuf_max = 20
        If Len(s_buf) > sbuf_max Then s_buf = ""
total_bytes.Text = Val(total_bytes.Text) + 1
Dim row_id, strp, endp As Integer
        s = com1.ReadLine
        If s <> "" Then
           If trace.Checked = True Then
              TextBox1.AppendText(s)
           End If
strp = InStr(s, ":") + 1
           endp = InStr(s, ";")
If strp > 0 And endp > 0 And (endp - strp) > 0 And Len(s) >= (endp + 1) Then
              s_buf = Mid(s, strp, endp - strp)
              chksumbyte = 0
              chksumbyte = Mid(s, endp + 1, 1)
  If check_sum(s_buf) = Asc(chksumbyte) Then
TextBox1.AppendText(vbNewLine & "Recv:" & s_buf & vbNewLine)
If IsNumeric(Mid(s_buf, 1, 1)) Then
'----- update grid -------'
data_add = Split(s_buf, "%")
row_id = get_row_by_text(data_add(0))
If row_id = -1 Then
grid.RowCount = grid.RowCount + 1
grid.Rows(grid.RowCount - 1).Cells(0).Value = data_add(0)
row_id = grid.RowCount - 1
```

```
End If
data_arr = Split(data_add(1), "-")
grid.Rows(row_id).Cells(1).Value = data_arr(0)
grid.Rows(row_id).Cells(2).Value = data_arr(1)
grid.Rows(row_id).Cells(3).Value = Date.Now()
'------- end grid update ----------'
'------- chart update -------------'
chart_name = sensors_arr.SelectedItem & " - " & grid.Rows(row_id).Cells(0).Value

If Chart1.Series.IndexOf(chart_name) = -1 Then
Chart1.Series.Add(chart_name)

Chart1.Series(chart_name).ChartType =
DataVisualization.Charting.SeriesChartType.Line
End If
Chart1.Series(chart_name).Points.AddXY(TimeOfDay.Hour & ":" &
TimeOfDay.Minute & ":" & TimeOfDay.Second,
grid.Rows(row_id).Cells(sensors_arr.SelectedIndex + 1).Value)
'--------------------------------'

                rf_num.Text = Val(rf_num.Text) + 1
                    rf_success.Text = Val(rf_success.Text) + 1

                End If
            Else
                TextBox1.AppendText(vbNewLine & "WRNG:" & s_buf & "(" &
check_sum(s_buf) & " | " & Asc(chksumbyte) & ")" & vbNewLine)
                rf_num.Text = Val(rf_num.Text) + 1
            End If
```

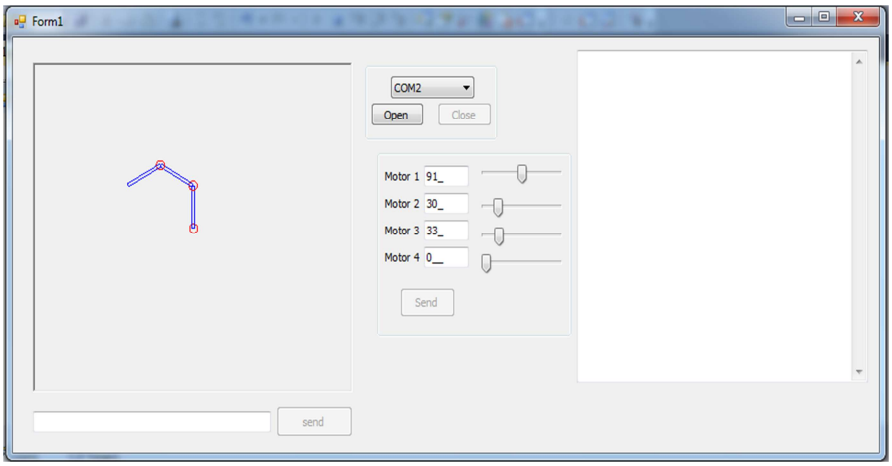
```
If Val(rf_num.Text) >= 10 Then
                rf_signal.Text = Int((Val(rf_success.Text) / 10) * 100)
                If Val(rf_signal.Text) < 25 Then
                   rf_signal.ForeColor = Color.Red
                ElseIf Val(rf_signal.Text) < 70 Then
                   rf_signal.ForeColor = Color.DarkGoldenrod
                Else
                   rf_signal.ForeColor = Color.Green
                End If
                rf_signal.Text &= "%"
                rf_num.Text = "0"
                rf_success.Text = "0"
            End If
        End If
    End If
  End Sub

End Class
```

Applications of Wireless Sensor Network

Robot Arm Control Using Sensor Node

We also used sensor node to control Robot Arm using digital output pins to control Arm Motors and use Variable resistors connected to ADC Ports to return Motors Positions (Source Code and Circuit Diagram Attached in CD)

Robot Arm

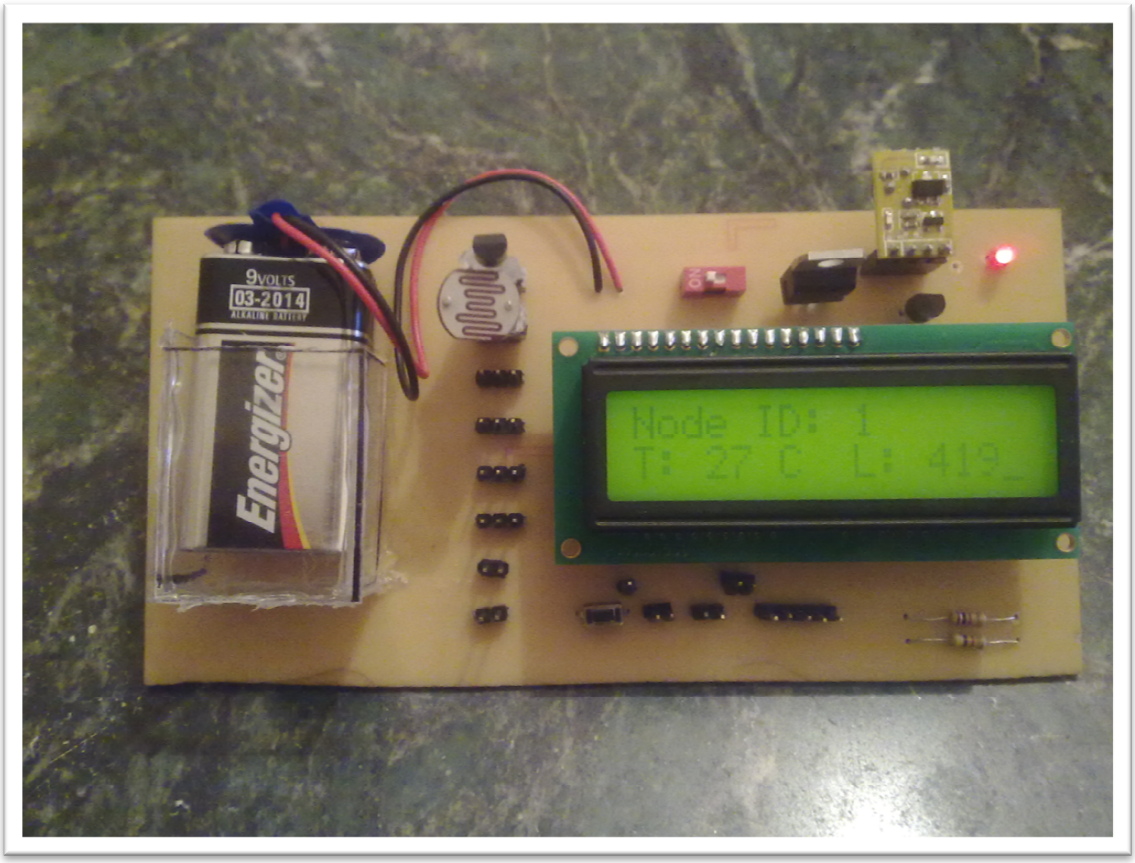


Robot Arm Control Software

Chapter 4

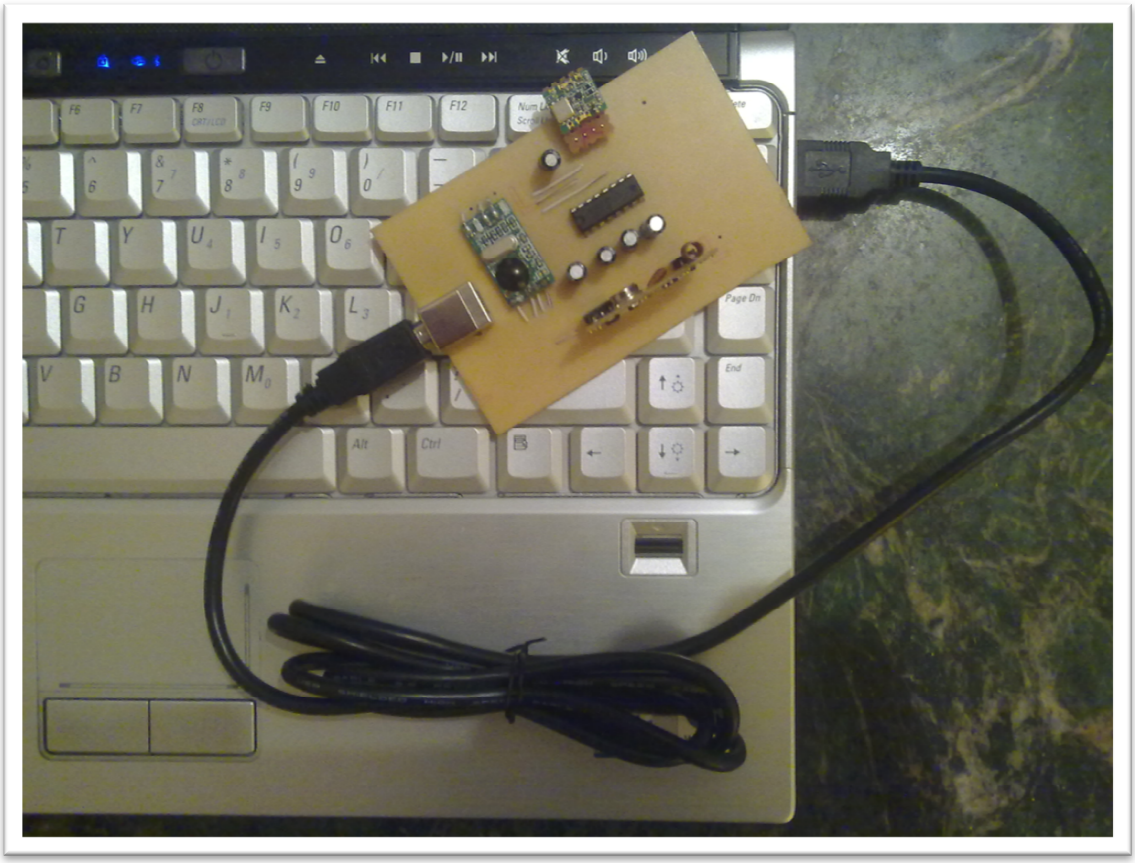
- **Testing And Results**

Sensor Node



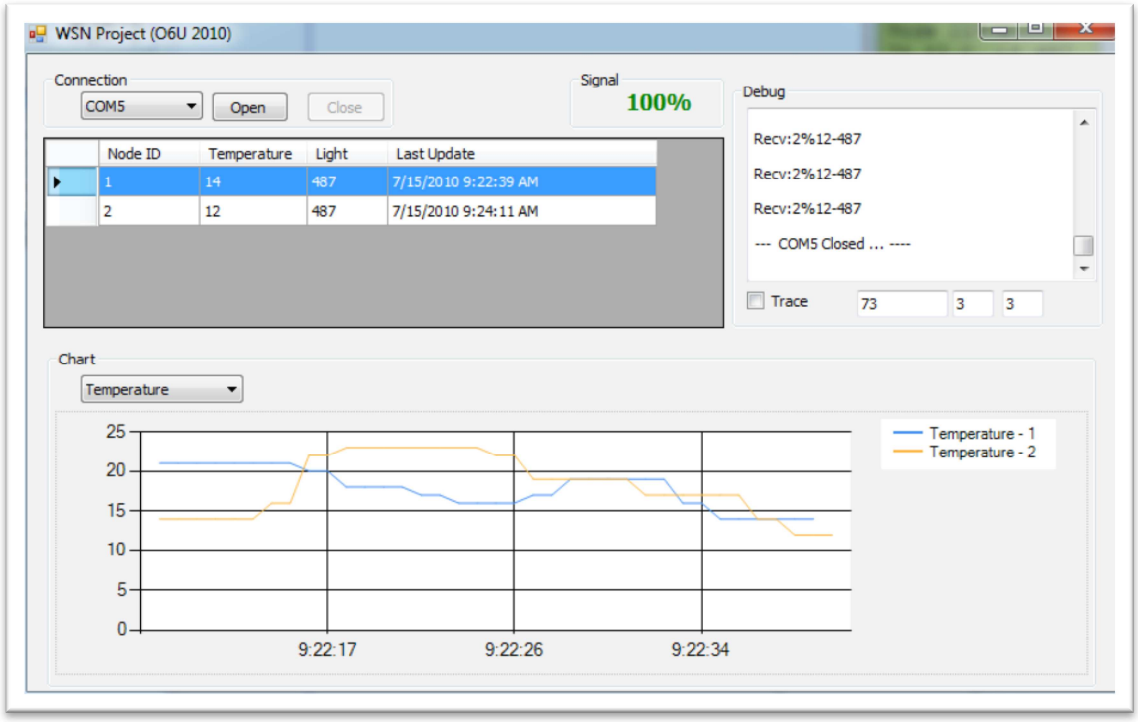
Sensor Node Display Node ID at the 1st LCD line and Temperature and Light Sensors Values at the 2nd Line

Receiver Board



Receiver Board Connected to Computer via USB Port

Receiver Board Computer Software



Receive Board Computer Software Displays Sensors Node IDs and Sensors Values

Chapter 5

- **Data Sheets**