Journal of Information Engineering and Applications
ISSN 2224-5782 (print) ISSN 2225-0506 (online)
Vol.6, No.1, 2016

A Hypothetical Database-Driven Web-Based Meteorological Weather Station with Dynamic Datalogger System

Vincent A. Akpan[1*] Reginald O. A. Osakwe[2] Sylvester A. Ekong[3]
1.Department of Physics Electronics, The Federal University of Technology, P.M.B. 704 Akure, Ondo State, Nigeria
2.Department of Physics, The Federal University of Petroleum Resources, P.M.B. 1221 Effurun, Delta State, Nigeria
3.Department of Physics & Energy Studies, Achievers University, P.M.B. 1030, Owo, Ondo State, Nigeria

Abstract
This paper proposes the development and implementation strategies of a hypothetical database-driven web-based meteorological (meteo) weather station with a dynamic datalogger system to provide up-to-the-minute real-time ground-based weather information online to any interested client. The meteo weather station will provide weather data/information for eight parameters namely: relative humidity; solar radiance; wind speed; wind direction; barometric pressure; temperature; nefobasimeter for monitoring and measuring cloud height, thickness and number of layers detection; and rainfall. The hypothetical design present techniques that can be used to capture and log meteo data in a dynamic relational database management system (DRDBMS) and implements a TCP/IP network server. The meteo weather information will be collected from sensors incorporated into measuring instruments and transmitted via 1–Wire network and stored in the TINI's non–volatile static random access memory (NV-SRAM). The TINI processes and uploads the information over a TCP/IP network via a switch, router, common gateway interface (CGI), very small aperture terminal (VSAT) via an Internet service provider (ISP) to the Internet for any interested user in the world. More so, although the acquired data will be made available as they are being logged to both the Internet and the database (DRDBMS) but the data will be automatically deleted every 24 hours at 00:00G MT from the TINI's NV-SRAM to free the memory for the next day data at the same 00:00G MT. As a result of this, the DRDBMS which is an object for data/information storage using the MySQL stores the meteo data/information which can be retrieved by any interested client on request. However, the request will be made possible via the use of web pages, where the each meteo data/information will be displayed and accessed using special user login codes (username and password) upon subscription.
Keywords: DBMS, Dynamic datalogger, embedded systems, html, hypothetical weather station, JavaScript, meteorology, MySQL, PHP, TCP/IP protocol, Tiny INternet interface (TINI), WAMP server, web development.

1. Introduction

Weather affects all aspects of human endeavors and thousands of lives are lost each year through weather-related disaster involving flashflood, thunder tornadoes, severe snow and ice storms and even Tsunami (Batlan, 1985; Dotto; 1988; Lamb, 1982, 1988). Adaptation or replacement of erratic tradition weather prediction systems to a minute-to-minute modern web-based meteorological data acquisition system has become a matter of grave concern to the world as well as to the scientific community (Asheville, 2004; McBridge, 1973; TMS, 2004; WMO, 2004; Flavio, 2001; Oniarah, 2004). When two Englishmen meet, their first talk is about the weather (Anealle, 2003; Moran and Morgan, 1986). Furthermore, in recent years, though, the weather has become more than a conversation starter. Why has weather become a matter of grave concern to people all over the world? This is because weather which was always unpredictable anyway, seems to be increasingly erratic (Anealle, 2003; Felix, 2000; Guzzi, 1990; Moran and Morgan, 1986).

Weather affects all aspects of human endeavours and thousands of lives and properties are lost each year through weather-related disasters involving flash floods, lightening strikes, tornadoes, and severe snow and ice storms (Anealle, 2003; Felix, 2000; Guzzi, 1990; Moran and Morgan, 1986). Weather is a very complex phenomenon (Brush, 1984; Ludlum, 1993). The science of the atmosphere, especially of weather is called meteorology. Meteorology is the most international of scientific activities (Brush, 1984; McBridge, 1973; Miller and Thompson, 1979; Moran, 1991). Its realm is the atmosphere of the entire globe, and its practice involves the daily cooperation of every action on earth. The progress of meteorological knowledge over the past three centuries has been dependent mainly on the state of contemporary technology. Thus the growth of technology as a science has followed, namely: 1). The development of instruments for observing the weather elements, such as temperature, humidity and air pressure; 2). Improved methods of communication for gathering weather data and transmitting forecasts; 3). The use of balloons, air crafts, rockets and satellites for probing the upper atmosphere and relaying data to the earth; and 4). The utilization of high-speed computers for processing and analyzing the vast quantities of data required (Ashville, 2004; Oniarah, 2004; TMS, 2004; WMO, 2004).

The application of scientific principles and procedures to the prediction of future states of the

atmosphere is called weather forecasting. Modern weather forecasting is done with exact, quantitative methods, much of the processing of weather data is done by high-speed digital computers. Unfortunately, most observations are made in the more developed, middle-latitude countries. There are few or no observations collected in less developed countries and across the oceans, which cover most of the globe. Commercial ships and aircraft provide valuable observations from the major shipping lanes and aircraft routes, while upper air information is obtained from selected continental, island, and a few ship locations. Satellite-based remote sensors offer the exciting possibility of eventually measuring atmospheric structure with sufficient capacity to obviate the need for an expensive people-based system.

Station coverage is heavily biased toward the major industrial nations; and principal ship routes. In such regions, station separation may be as small as 50kilometers (30 miles), with 100 kilometers (60 miles) more typical. In the more remote corners of the globe, the station separation can easily reach 1,000 kilometers (6000 miles). A complete surface observation must include temperature, dew point, pressure, and change in pressure over the last three hours, present weather, state of the sky, cloud type, and height and amount of precipitation if it occurred. Numerous other types of information are added as required. In addition weather ships, buoys, and commercial ships report water temperature and information on the state of the sea including the period and height of waves (Anthes, 1978; Budyko and Golitsyn, 1988; Guzzi, 1990; Herman and Goldberg, 1985; Lutgens and Edward, 1989).

The local National Weather Service forecast offices have the responsibility of preparing a local and regional forecast for the next 48 hours based upon the guidance package Miller and Thompson, 1979; Oniarah, 2004; TMS, 2004).. The public forecast prepared by the local National Weather Service (NWS) office is necessarily general. The forecast is communicated to the various media outlets for public dissemination. Hazardous weather warnings or watches are communicated to the responsible local officials, police, and civil-defense agencies. Likewise, marine and agricultural forecasts, including freeze and frost warnings, are routinely prepared. Many companies and individuals prepare private weather forecasts for specific users. A public utility, for example, needs much more specific information on temperature and wind than can be provided by the Local National Weather Service office if it is to provide economical and efficient gas and electric service. The same can be said of suppliers of home heating oil with snow and ice amount forecasts.

The general public is most familiar with private meteorologists through local TV and radio broadcasts. However, both public and private weather forecasters essentially have access to the same weather information from government. Why then do the forecasts on occasion disagree within the same area? National Weather Service forecasters in advanced developed countries rarely have such a luxury, as several hours often elapse before their forecast reach the public. The National Weather Service communicates to the public through middlemen represented by local media and newspapers. Broadcasters frequently read obsolete forecasts or edit the forecast for broadcast convenience, which often alters or destroys the content of the forecast. Local newspapers, with rare exceptions, decline to put the time on their printed forecast. The question is “why can’t individual access these forecast through the World Wide Web (Internet)?

There are technological basis for meteorological date acquisition techniques which includes the development of local traditional meteorological instruments, High–Speed communication and data processing as well as the conquest of the Air technique. There are also some recent development on forecasting techniques such as empirical forecasting, trends towards scientific forecasting and numerical forecasting by computer and many more.

2. Statement of the Problem

Data collection and analysis techniques are at the top in meteorological scale of preference and such techniques includes instrumentation; radiosonde, remote sensing, frequency of data sampling, data communication, preparation of weather forecasts, horizontal and vertical movement, equation’s of weather, dissemination of public weather forecasts, local forecasts, private forecasts , weather forecasting of the future, and some possible hints for amateur weather forecasters (Anthes, 1978; Budyko and Golitsyn, 1988; Lutgens and Edward, 1989).

The main objective of this hypothetical study is to develop a technique that will be used for the design and construction of an inexpensive ground-based embedded web-based meteorological weather station with dynamic datalogger system. The sub-objectives include: *1)*. Conversion of the physical form of atmospheric weather conditions into electrical signal for the purpose of measurement, calibration and display; *2)*. Development of signal conditioning circuits for configuring the atmospheric electrical signals obtained in the first step *1)* just mentioned above; *3)*. Initialization, configuration and Networking of the TINI microcontrollers for data conversion and routing; *4)*. Configuration of the host computer as a network sever for secured Internet communication and relational database management manipulation; *5)*. Database design and development for the meteo data management; *6)*. Database-driven web-site design for the embedded web-based meteo data logger System; and *7)*. The integration of the above sub-objective for the design of the embedded web-based meteo data logger system.

The scope of this hypothetical study will be limited to seven weather parameters for a particular location. The data logging interval shall be five (5) minutes to a maximum of 24 hours. The meteo data should be made available on the internal over 24 hours period and after which it should be deleted and stored permanently in the host database (archive) management system. The data logger should start fresh data logging at 00.00 GMT. The significance of this study if successfully implemented, is that it will provide accurate up-to-date meteorological information about the best community or state via the Internet and can be expanded to encompass the host nation.

3. The Development of the Hypothetical Web-Based Embedded Meteo Data Logger System

3.1 The Block Diagram Description of the Meteo Data Logger System

This section presents the techniques that can be used to capture and log meteorological (meteo) data and implement a TCP/IP network server, making this data available to both the administrator and remote clients. Ultimately, the web server will accept connections over both Ethernet and the public switched telephone network (PSTN) using point-to–point protocol (PPP) to manage dial-up connections. Support for dial–up networking is primarily what makes the data logger truly removed. This will allow access to any client computer in the world with internet access without requiring the presence of Ethernet network at the data collection sight. The block diagram of the web-based embedded meteorological datalogger is shown in Fig. 1.

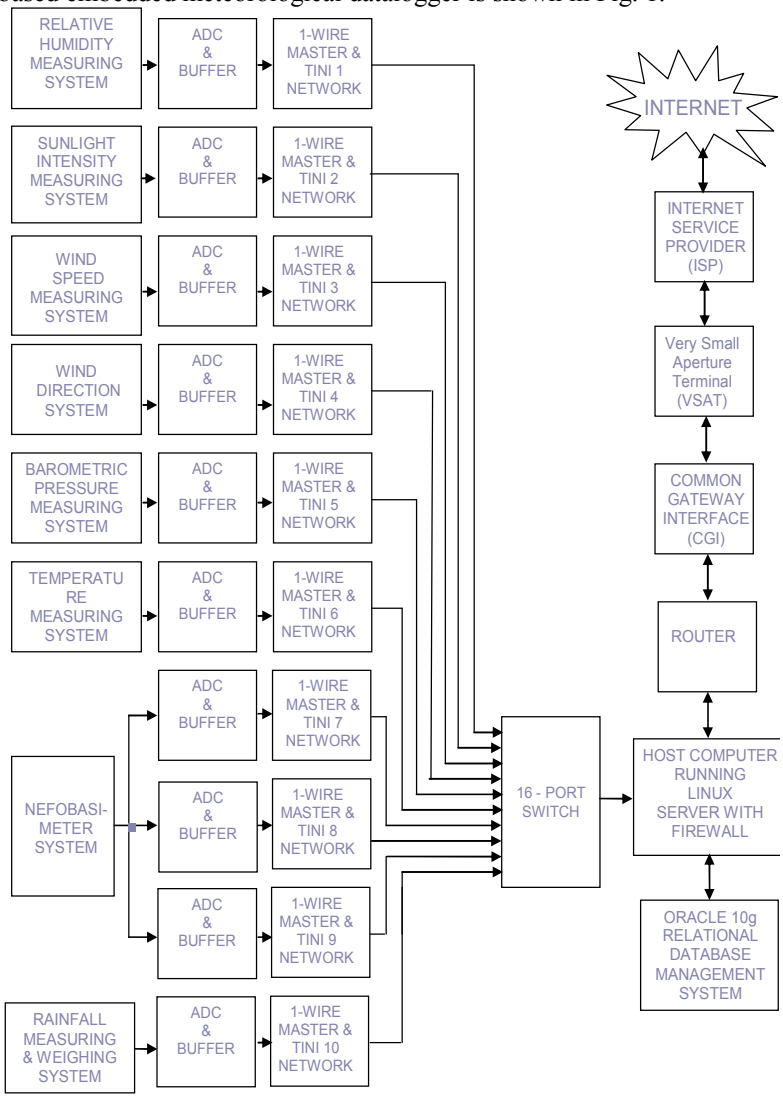


Fig. 1: Simplified block diagram of the proposed hypothetical ground-based embedded web-based meteorological (meteo) weather data logger system.

The main point here is that meteo information will be collected from sensors and other electromechanical devices (or possibly multiple devices), processed and uploaded to any interested client over a TCP/IP network via a switch (or a hub), router, common gate interface (CGI), Internet service provider (ISP) and

Internet. Although the acquired data on the TINI microcontroller will be made available as they are being logged to any client, these data will automatically be deleted every 24 hours to free the TINI's 512 kilobyte nonvolatile static random access memory (SRAM) for the next day set of data starting at 00.00 GMT. Before these data are deleted, they will be transferred to a database system. Therefore, a data based system will be developed for this meteo data storage that can be accessed by any interested client upon request from the meteo station management team.

3.2 Measuring Humidity on the 1-Wire Net

Humidity is an important factor in many manufacturing operations and also affects personal comfort. With the proper sensing element, it can be measured over the 1-wire net. The sensing element specified here develops a linear voltage version relative humidity (RH) output that is ratiometric to supply voltage. That is, when the supply voltage varies, the sensor output follows in direct proportion. This requires measuring both the voltage across the sensor element and its output voltage. In addition, calculating true RH requires knowing the temperature at the sensing element. Because it contains all the necessary functions for calculating, the DS2438 with its two analog-to-digital converters (ADCs) and a temperature sensor makes an ideal choice for constructing a humidity sensor. In Fig. 2, the analog output of the HIH–3610 humidity sensing element is converted to digital by the main ADC input of a DS2438. The bus master first has integrated circuit 1 (IC1), the DS2438, report the supply voltage level on its V_{DD} PM, which is also the supply-voltage for 1-wire master (WM1), the sensing element. Next the master has IC1 read the output voltage of WM1 and reports local temperature from its on-clip sensor. Finally, the master calculates true RH from the three parameters, supplied by IC1. Thus, calculating true RH with a ratiometric sensor requires that the supply voltage, sensor temperature, and output voltage all be measured. Conveniently all three sensor parameters can be monitored with a single, new 1-wire IC. The DS2438 smart Battery monitor, designed to work in battery packs to perform multiple functions such as tagging and fuel gauging, contains a voltage and temperature A/D converter needed for RH calculations.

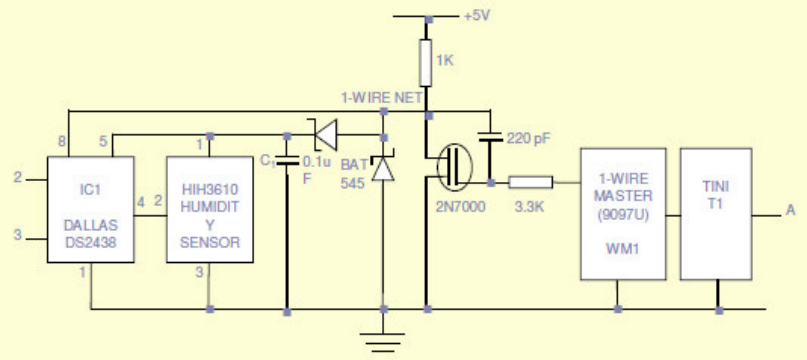


Fig. 2: The block diagram of the proposed humidity measuring system.

3.3 Measuring Solar Radiance on the 1-Wire Net

The amount of sun light and its duration are additional parameters easily measured with 1-wire sensors. The amount is a measure of air and sky conditions, while duration is related to the equinoxes and optical implementations tend to be complex, the electronics can be easily created using a DS2438. Fig. 3 illustrates a solar–radiance sensor built using a sense resistor connected in series with a photodiode. Light striking the photodiode generates photocurrents that develop a voltage across the sense resistor that is read by the ADC. Optical filters can be added to control both the wavelength and optical band-pass to which the sensor responds.

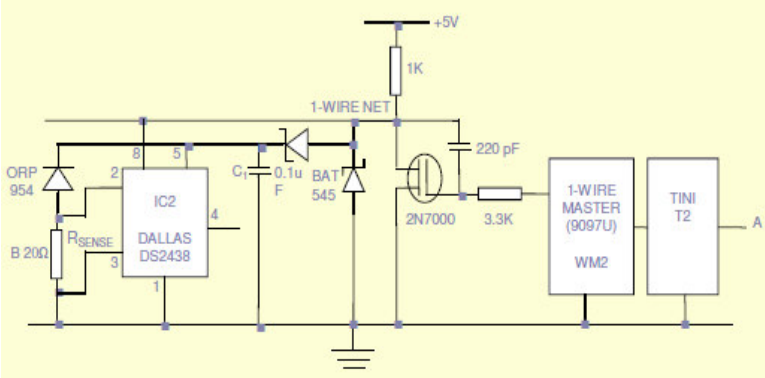


Fig. 3: The block diagram of the proposed solar radiance measuring system.

3.4 Wind Speed Measurement on the 1-Wire Net

The weather station handles wind speed measurements in a similar manner as that of the wind direction discussed in the next sub-section (Awtrey, 1997; Awtrey, 1998). The sensor consists of two magnets mounted on a second rotor attached to the wind cup's axle as shown in Fig. 4(a). The magnet operates a reed switch connected to the DS2423 counter chip that provides the sensors serial number. One magnet is mounted in each of the two outermost holes of the rotor, providing two counts per resolution, and thus improving response of low wind speeds. The two-magnet arrangement also provides rotational balance to the rotor, an important consideration given that the rotor can reach 2400 rpm in 160 km/h (100 mph) winds. The instrumentation diagram of the proposed wing speed measuring system is shown in Fig. 4(b).

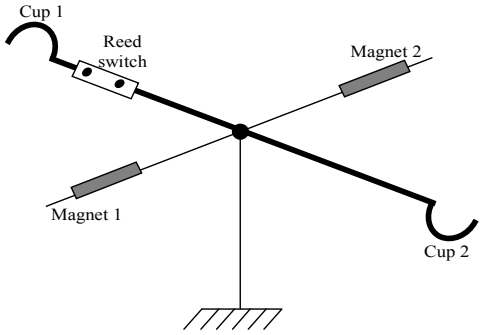


Fig. 4(a): The setup for the wind speed measuring instrument.

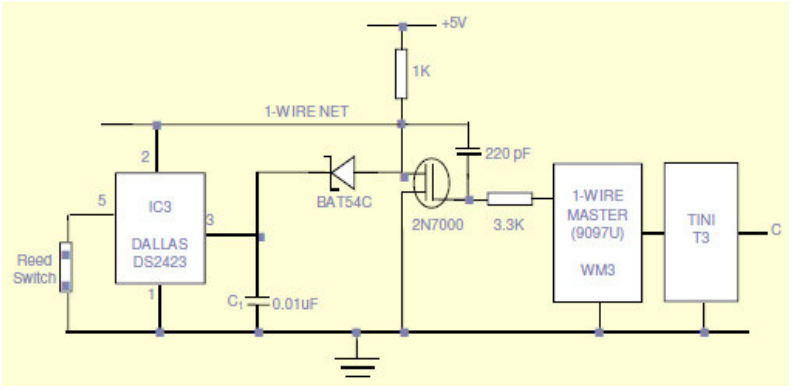


Fig. 4(b): The block diagram of the proposed wind speed measuring system.

The counter chip keeps tracks of the total number of wind cup revolutions and transmits the data to the bus master at demand. The chip contains two 32-bit counters and can be powered for 10 years with a small lithium battery. Power from the counter chip comes from diode CR_1 and capacitor C_1 which is the half wave rectifier that steals power from the data line. The counter can be reset to zero order then the parasitic power is lost.

The bus master calculates wind speed by taking the difference between two values stored in the counter,

one generated before and the other generated after a clocked interval. The calculation also takes into account other factors such as the relationship of revolutions per minute to kilometers per hour.

3.5 Measuring Wind Directions on the 1-Wire Net.

The original 1–Wire weather station used DS2401s to label each of the eight magnetic read switches in its wind direction sensor, as shown in Fig. 5. A single DS2450 quad ADC can perform the same function with five resistors. As the wind rotates the wind vane, a magnet mounted on a tracking rotor opens and closes one (or two) of the reed switches. When a reed switch closes, it changes the voltages seen at the input pins of IC4, the DS2450. for example, if the magnet is in a position to close S_1 (North), the voltage seen on pin 7 changes from V_{CC} to $V_{CC}/2$, or approximately from 5 V to 2.5 V. Since all 16 wind vane positions produce unique 4 bit signals from the ADC, it is only necessary to indicate month, or specify which direction the wind vane is currently pointing to initialize the sensor. Because two reed switches are closed when the magnet is midway between them, just eight reed switches indicate 16 compass points. If the circuit is biased with +5 V as shown in Fig 5. Referring to the schematic and position 2 in Table 1, which list the voltage seen at the ADC inputs for all 16 cardinal points (wind directions). Observed that when S_1 and S_2 are closed, 3.3 V is applied to ADC inputs B and C.

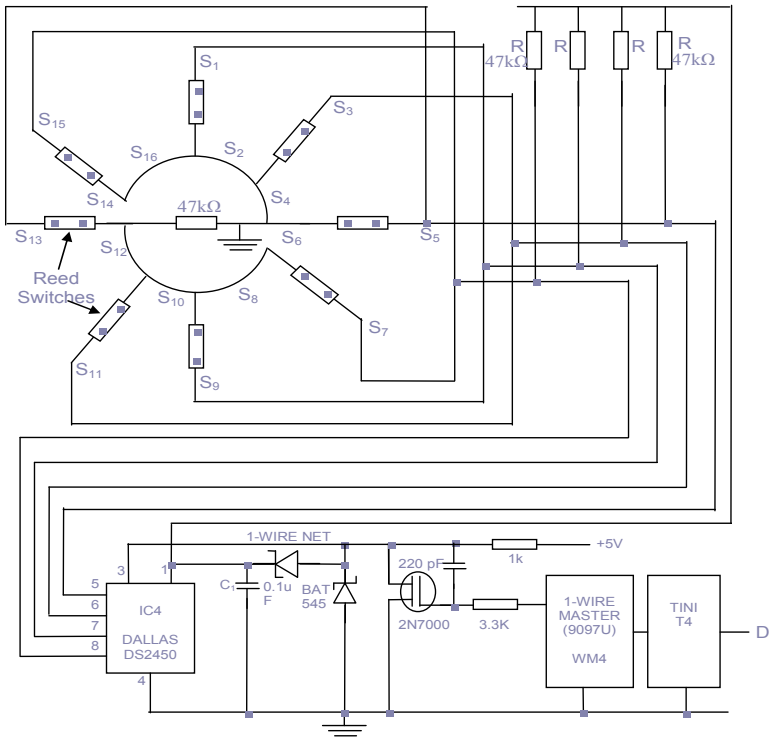


Fig. 5: The block diagram of the proposed wind direction measuring system.

Table 1: Wind vane position versus the voltage at the four DS2450 ADC inputs.

Cardinal points	Voltage Input at D (V)	Voltage Input at C (V)	Voltage Input at B (V)	Voltage Input at A (V)	Wind Directions
1	5	2.5	5	5	N
2	5	3.3	3.3	5	NNW
3	5	5	2.5	5	NW
4	5	5	3.3	3.3	WWN
5	5	5	5	2.5	W
6	0	5	5	2.5	WWS
7	0	5	5	5	SW
8	0	0	5	5	SSW
9	5	0	5	5	S
10	5	0	0	5	SSE
11	5	5	0	5	SE
12	5	5	0	0	EES
13	5	5	5	0	E
14	2.5	5	5	0	EEN
15	2.5	5	5	5	NE
16	3.3	3.3	5	5	NNE

This occurs because the parallel combination of pull up resistors R_2 and R_3 act as a single resistor half their value connected in series with R_1 to form a voltage divider with 0.66 V_{CC} across R_1. Note that this condition occurs twice more at switch positions 4 and 16 generating 3.3 V at those directions.

3.6 Measuring Barometric Pressure on the 1-Wire Net

Barometric pressure is another important meteorological parameter that can be measured over a 1–Wire net using the DS2438. By selecting a ratiometric pressure sensor that contains comprehensive on-chip signal conditioning circuitry, the circuit is very straight forward as illustrated in Fig. 6. Note that both the output voltage representing atmospheric pressure and the supply voltage across the element must be known to accurately calculate barometric pressure. Because the MPXA4115 pressure sensor can require as mush as 10 mA at 5 V, an external power source is needed. Note that external power should also be connected to the DS2438's power pin. This allows the DS2438 to measure the supply voltage applied to the pressure sensing element. Flexible tubing can be routed to sample the outside air pressure and avoid unwanted pressure changes (noise) caused by doors and windows opening and closing or elevators moving inside the building.

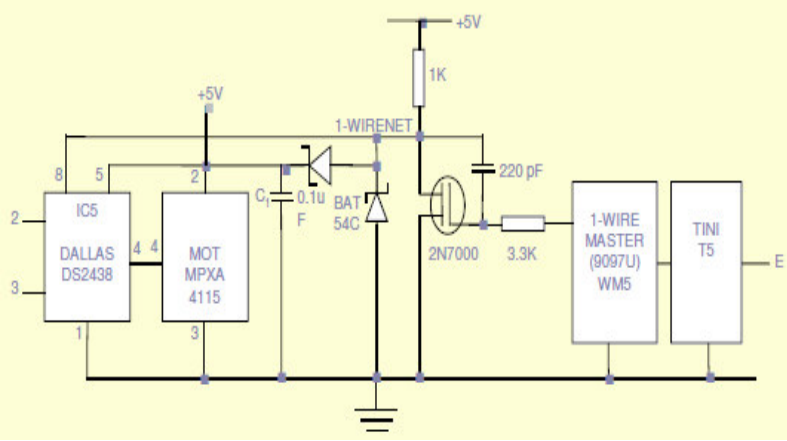


Fig. 6: The block diagram of the proposed barometric pressure measuring system.

3.7 Temperature Measurement on 1–Wire Net

One can measure extreme temperatures using conventional thermocouples (TC) that are directly digitalized using a DS2760 multi-function with the 1–Wire chip as shown in Fig. 7. The twisted pair cable of the 1–Wire net covers the distance between the TC and the bus master, effectively replacing the extensive TC extension cable normally used. Because of its unique ID address, multiple smart TCs can be arbitrarily placed where needed along the net, greatly minimizing the positioning and cost of an installation. With an LSB of 15.625 µV, the chip can directly digitize the millivolt (mV) level output produced between the hot and cold junctions of the typical TC as its on–chip temperature sensor continuously monitors the temperature at the cold junction of the TC. It contains user–accessible memory for storage of sensor–specific data such as TC type, location and the data it is

placed onto service.

This information minimizes the probability of error due to the mislabeling of sensors. Thus, a DS2760 can be used with any TC type because the bus master's calculations are based on the stored data and the temperature of the cold junction, as reported by the on–chip temperature sensor. Adding R_1 allows V_{DD} to be measured, which is useful in troubleshooting to verify that the voltage availability on the 1–Wire net is within acceptable limits as in circuit.

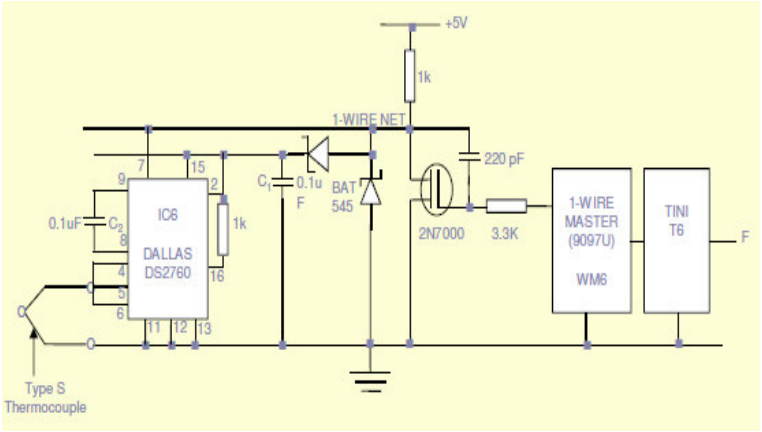


Fig. 7: The block diagram of the proposed temperature measuring system.

3.8 Height of Clouds Measurement on the 1–Wire Net

This nefobasimeter sensor system evaluates the height of one or more cloud layers existing over the airport. Usually, the nefobasimeter cloud measuring system use modern laser technologies for cloud height, thickness and possible number of layers as illustrated in Fig. 8. The transmitter and the receiver use the same lens to the object to obtain better result in the rank from 0 to 300ft. heating, thermostat and automatic cleaners for the control of temperature of the optics and electronic systems, including automatic conditioning of window are also incorporated to the equipment.

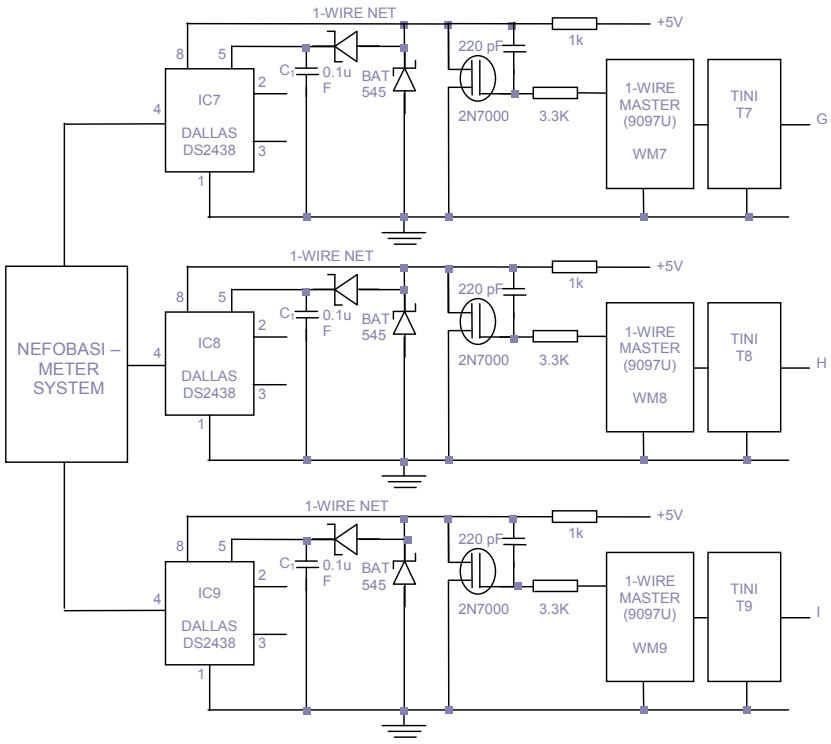


Fig. 8: The block diagram of the proposed nefobasimeter for monitoring and measuring system for cloud height, thickness and number of layers detection.

It measures from 0 to 25,000 ft with a minimum precision of +/– 20 ft or approximately 2%. For bases smaller or equal to 3000 m (10,000 ft) the resolution will be of 30 m (100 ft), and for bases greater than 3,000 m the resolution will be of 300 m (1,000 ft). The cycle of measurement has to be programmable between 15 and 120 seconds, having an automatic system such that in case of cloudless sky optimizes the emission of the laser. They detect simultaneously up to three cloud layers, having two types of ports: a data port with data of height of clouds and status, and another port including all possibilities of maintenance. Like transmission meters, nefobasimeter having an automatic compensation of the contamination of the lenses (dirt). The control unit of the sensor allows the presentation of the three layers of simultaneous clouds, or two cloud layers and vertical visibility, if the lowest layer is not enough defined, including the thickness of the same ones.

3.9 Rain Measurement on the 1–Wire Net

Many events call for measurement of either a total or a count per unit time (rate). Examples include wind speed and rainfall or the number of times a wheel has rotated, from which the revolution per minute (rpm) and distance can be computed. A magnetically activated reed switch used as an input to a DS2423 counter allows such events to be easily measured. In stead of using the basic reed switch as in (Awtrey, 1997; Awtrey, 1998), a sophisticated rain gauge system based on strain gauges configurations similar to those described in (Babalola and Akpan, 2006; Akpan and Babalola, 2008; Akpan and Babalola, 2009) can be adapted which is proposed here as shown in Fig. 9.

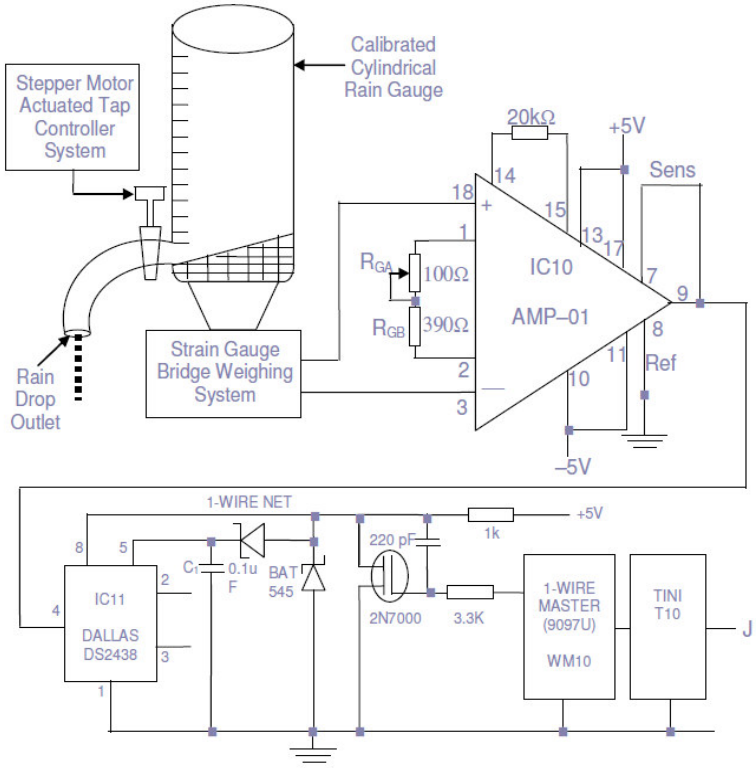


Fig. 9: The block diagram of the proposed rainfall measuring system.

The dual diode BAT54S served to protect the circuit from signals that go below ground and with C_1 provides a local source of power. While the DS2423 has an internal pull-up resistor to keep the input from floating its high value (≈ 22-MΩ) can make it susceptible to noise. To avoid generating spurious count during turn-on and minimize noise pickup, an external 1-MΩ pull-down resistor is substituted. Except for lithium backup, the counter circuit used in the 1-Wire rain gauge is shown in Fig. 9 build around the DS2438. Here, a small permanent magnet moves past the reed switch each time a tipping bucket fills and empties. This momentarily closes the reed switch that increments the counter, indicating 0.01 inch of rain has fallen. Note that a similar circuit was used in the 1-Wire weather station to measure wind speed. Conveniently, the DS2423 also contains 4096 bits of user-accessible SRAM that can be used for temporary storage, or where lithium backup is provided, for calibration, location, and routine inspection information.

4. TINI: Hardware, Software, 1–Wire Network and Network Connectivity

4.1 TINI Initialization and Network Configuration

Tiny internet Interface (TINI) is a platform developed by Dallas semiconductor (DSC-MIP, 2014) to provide system designers and software developers with a simple, flexible, and cost-effective means to design a wide variety of hardware devices that can connect directly to corporate and home networks (Akpan and Osakwe, 2015a; DSC-MIP, 2014). The platform used in this project is the TINI Board Model (TBM) 390 (DSC, 2013; DSC-MIP, 2014; Loomis, 2001). The TBM 390 is a combination of a small but powerful chip-set and a java programmable runtime environment. The chip-set provides processing, control, device-level communication and networking capabilities. TINI's networking capability extends the connectivity of any attached device by allowing interaction with remote systems and users through standard network applications.

The TINI hardware consists of a microcontroller, flash ROM (read-only-memory) and static RAM (random access memory). The flash memory stores TINI's runtime environment and it maintains the memory content even in the absence of system power; it is reprogrammable. The static RAM contains the system data area as well as the garbage collected heap from which all java objects are allocated. It also stores all file system data. Fig. 10 is an illustration of a full-featured TINI hardware implementation (Loomis, 2001).

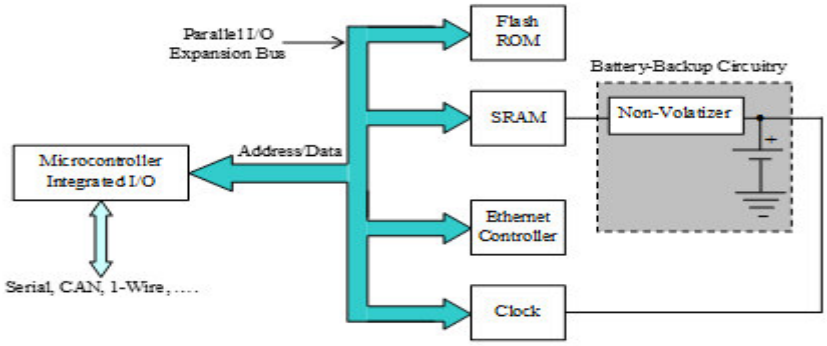


Fig. 10: A full-featured TINI hardware implementation.

The circuitry performs two functions. First, it keeps the clock running in the absence of main power (V_{cc}), ensuring that an accurate time can always be read from the clock. The lithium cell alone performs this task. Also the lithium cell in conjunction a small chip known as an SRAM non-volatizer, maintains the contents of the state RAM in the absence of main power. The TBM 390 is a compact (31.8 mm x 102.9 mm) 72 – pin SIMMM board. It is an Ethernet-ready hardware implementation and supports all the functionality shown in Fig. 11. It requires only a single +5 V power supply.

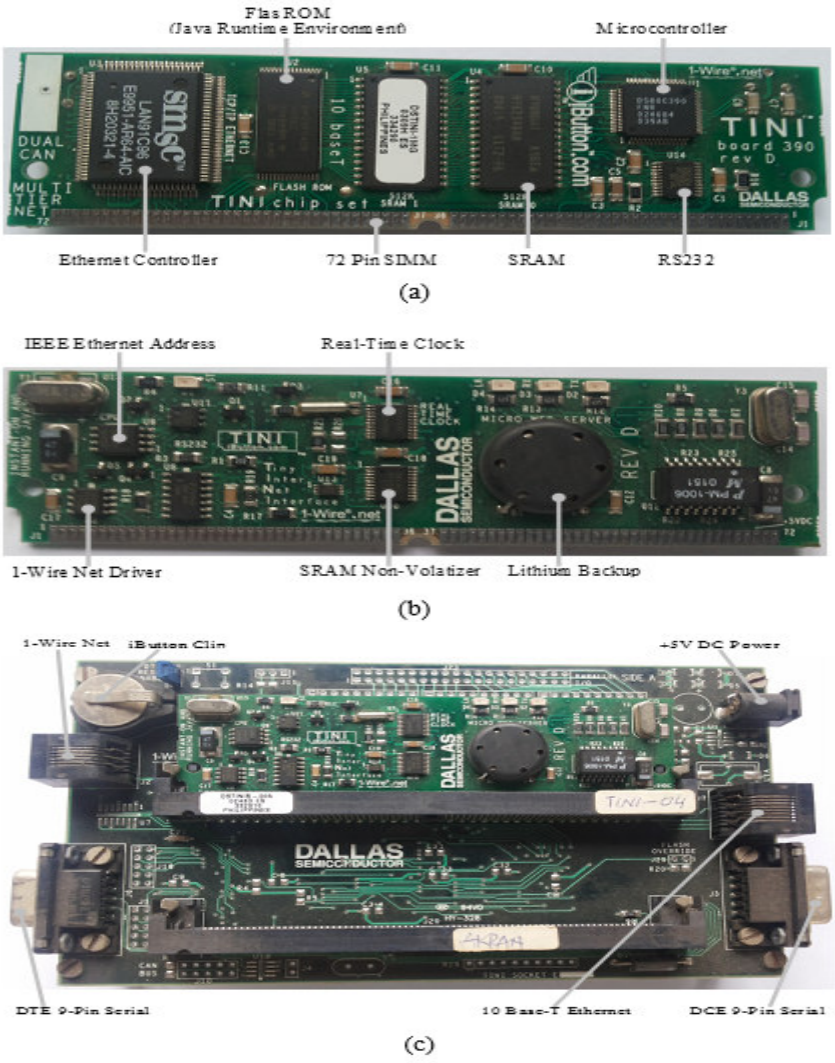


Fig. 11: (a) Top view of the TBM 390 TINI board model, (b) bottom view of the TBM 390 TINI board model and (c) The complete E10 socket with the TINI board model.

4.1.1 The TINI Board Model

The board model 390 (TBM 390) is a complete TINI hardware reference design that is currently available with either 512 kilobytes or 1 megabyte of non volatile, static RAM. It is also available as a 72 – pin SIMM module and the TINI employed in this project is a 512-kilobyte of NVSRAM. The top and bottom view of the TBM390 as well as the E10 socket with TINI board are shown in Fig. 11 (a), (b) and (c) respectively. The E10 socket board shown Fig. 11(c) is aimed at aiding the application development process. It provides the following physical connectors:

1). 72–Pin SIMM Connector: the SIMM connector accepts the TINI board shown in Fig. 11(a) and (b).

2). 9 – Pin Female DB9 Connector: this connector provides a limited DCE (Data Communication's Equipment) serial port using a straight-through serial cable. This port is typically used for loading the run time environment and bootstrap application. Hardware handshake lines are not supported.

3). 9 – Pin Male DB-9 Connector: this connector provides a DTE (Data Terminal Equipment) serial port for straight–through connections to DCE devices such as analog modems. Most TINI applications that control serial devices use the DTE port. In this case TINI is the DTE device, replacing all hardware handshake lines except DSR (Data Set Ready) and RI (Ring Indicate).

4). RJ45: The RJ45 connector accepts a standard 10 BAX-T Ethernet cable providing connectivity to an Ethernet network. Use a straight-through cable for connecting TINI directly to a PC (personal computer) or Workstation.

5). RJ11: the connector provides access to the 1–Wire network using standard telephone cable.

6). Power Jack: the E10 accepts a regulated +5 V DC power supply.

The E10 also provides integrated circuit and discrete components foot prints to support additional I/O option such as parallel, CAN and additional Serial Ports. A more comprehensive detail on the initialization and configuration of TINI and Java communications application programmer interface (API) for real-time embedded networked applications as applicable in this study can be found in (Akpan and Osakwe, 2015a).

4.1.2 The TINI Runtime Environment

Providing hardware essentials for developing embedded network devices is only half of the job. A large amount of software is also required to free application developers from having to worry about the details of creating layers of infrastructure to provide support for executing multiple tasks, network protocol stacks, and an application programming interface. A well-defined runtime environment that provides all of these features allows the developer to focus primarily on the details of the application. For this reason a runtime environment was developed from the beginning as an integral part of the overall platform. The software that comprises TINI's runtime environment can be divided into two categories: native code executed directly by the microcontroller and an API interpreted as bytecodes by the Java Virtual Machine. Application code is written in Java and utilizes the API to exploit the capabilities of the native runtime and the underlying hardware resources. It is also possible to write native libraries that can be loaded from within an application to meet defined real-time requirements. A graphical representation of the runtime environment is shown in Fig. 12. Java programs running on TINI are mostly defined in applications and not applets. They are stand-alone programs that begin execution from a "main" method with the following signature.

public static void main(String[] args)

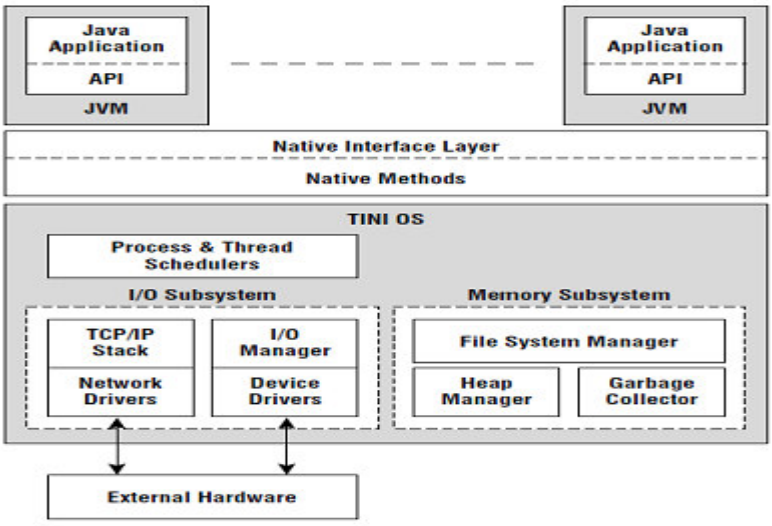


Fig. 12: The TINI runtime environment.

Also, unlike applets, they have no "*sandbox*" restrictions. On TINI, Java applications have full privileges and access to all system resources, even more so than on other platforms that support a Java runtime environment. This is particularly important for embedded applications because they are closely coupled with physical devices. Also, unlike other Java platforms, on TINI there is usually no system administrator to perform configuration and maintenance. This means that the application is responsible for configuring as well as controlling the entire system. For these reasons an application that controls an embedded system must have complete access to even low-level functionality provided by the OS (operating system).

4.2 TINI Hardware, Implementation and Network Connectivity

The configuration shown in Fig. 10 extends the reach of embedded devices to Ethernet networks. It also provides an accurate time reference for time-stamping purposes. Without the clock, commonly used Java methods such as *java.lang.System.currentTimeMillis* and *java.util.Date* methods that use *currentTimeMillis* return constant, and therefore useless. Peripheral devices such as the Ethernet controller and clock are included into the system's memory map. Another addition that is shown in Fig. 10 is the battery-back circuitry. The battery is a very small, single-cell lithium battery. Both the SRAM and clock used in TINI designs have very low stand-by power requirements, which mean that an appropriately chosen lithium cell will keep the clock running and the SRAM data persistent for over 10 years.

4.2.1 Protocol Conversion via TINI

TINI is designed to meet the functional requirements for commercial and industrial embedded network applications. However, because of its low-cost hardware and the availability of free software development tools, it is beginning to find a home in the educational and hobbyist arenas as well. TINI can be used for traditional

stand-alone embedded tasks such as monitoring and controlling a local device or system, but the majority of applications utilize TINI's networking capabilities. A few applications of the technology include the following: *1). Industrial controls*: TINI's integrated Controller Area Network (CAN) support is instrumental in implementing factory automation equipment, networked switches, and actuators; *2). Web-based equipment monitoring and control*: It can be used for communication with equipment to provide remote diagnostics and data collection for purposes such as monitoring device utilization; *3). Protocol Conversion*: TINI-based systems can be used to connect legacy devices to Ethernet networks. Depending on the I/O capabilities of the legacy system, this may be a job that can be done with a PC or workstation as well. However, TINI can do the job at a fraction of the cost and size; *4). Environmental monitors*: Using TINI's built-in support for 1-Wire networking, an application can query sensors and report the results to remote hosts.

The schematic of Fig. 13 shows a model in which TINI is employed as a protocol converter (or link) between a legacy embedded device and an Ethernet network (Akpan and Osakwe, 2015a, 2015b; Akpan et al., 2013; Loomis, 2001). The legacy device may communicate with the outside world using an RS232 serial port, Controller Area Network (CAN), or perhaps some type of parallel interface. The Java application running on TINI performs the task of communicating with the attached device in its native language (using a device-specific communication protocol) and presents the results to remote systems reachable via a TCP/IP network. The link provided by TINI is bidirectional, allowing a remote system to control as well as monitor a device.

Furthermore, Fig. 13 focuses on an embedded system that controls and provides network connectivity to a single device. However, TINI can also serve to interconnect various types of networks by bridging the gap between smaller, localized networks of inexpensive and lightweight devices and a "big world" TCP/IP network such as the Internet.

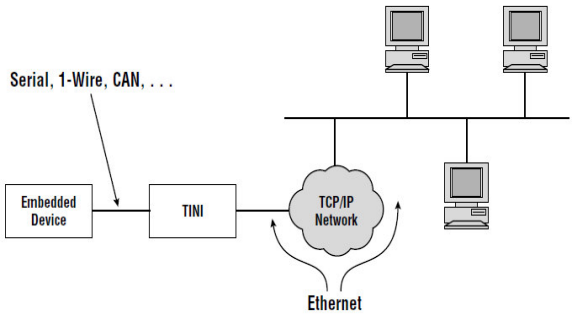


Fig. 13: Protocol conversion via the TINI.

In general, TINI applications interface to other equipment and networks as opposed to humans. Due to the embedded control and I/O-centric nature of most embedded network applications, there is no built-in hardware or API support for a human interface. TINI-based systems often provide a remote display by implementing a network server, such as an HTTP server, allowing the user to interact with the system using a network client such as a Web browser. Local display and data entry can be obtained by interfacing to a PDA (personal digital assistant) over a wireless link such as infrared (IR) or a hard-wired serial link. TINI systems requiring dedicated human interfacing capability can be implemented using liquid crystal displays (LCDs) and keypads.

4.2.2 1-Wire Chips and iButton: Device and Network Configuration

1-Wire chips provide network connectivity to otherwise mute entities. A 1-Wire network is a collection of one or more uniquely addressable devices that share a single conductor for communication and power. The single conductor is often referred to as a bus. The 1-Wire devices attached to the bus are always slaves. This implies the existence of a master that initiates all communication with the devices. The extremely simple hardware configuration of a 1-Wire network is shown in Fig. 14.

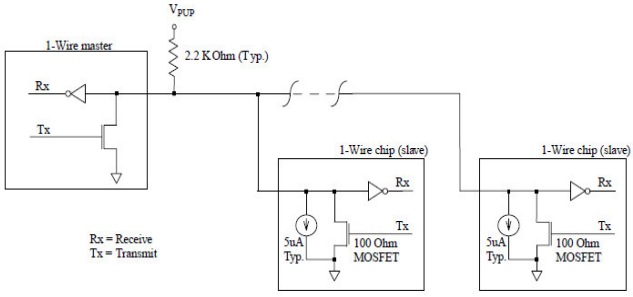


Fig. 14: 1-Wire network hardware configuration.

Although; an in-depth treatment of 1-Wire networking is beyond the scope of this work; a thorough treatment of 1-Wire networking can be found in (MIP, 2008; DSC, 2013; DSC-MIP, 2014; MAX, 2014a, 2014b). This work presents only brief description of the 1-Wire API and examines adapters and containers, the classes that represent them, and how they are used to monitor and control devices on a 1-Wire network.

A 1-Wire Network is a complex arrangement of devices, wire and connections. The radius of a 1-wire network is the distance from the master end to the furthest slave device in meters. The weight of the network is the total amount of connected 1-wire in the network in meters. In general, the weight of the network limits the rise time on the cable, while the radius establishes the timing of the slowest signal reflections. As a rule, no 1-wire network may ever have radius greater than 75 m. At this distance, the port will fail due to the time delay of the cable. In practice, however other factors usually limit the radius to smaller values than this 75 m (DSC, 2013).

4.2.2.1 1-Wire Network Topologies

At least, three major topologies exist for networking 1-wire devices, namely (MIP, 2008):

1). Linear topology: The 1-wire bus is a single pair, starting at the master and extending farthest. Slate devices other slaves devices are attached to the pair along its length without significant (> 3 m) branches or "stubs";

2). Stubbed Topology: The 1-wire bus is a single main line starting at the master and extends main line starting at the master and extending to the farthest slave device. Other slave devices are attached to the main line through branches stubs 3m or more in length; and

3). Star Topology: The 1-Wire bus is spelt at or near the master end and extends multiple branches of varying lengths, presumably with slave device along or at the ends of the branches. When different topologies are intermixed it becomes much more difficult to determine the effective limitations of the network.

To allow networks to grow in complexity, without growing in weights and radius methods have been devised wherein the network is divided into section that are electronically switched ON one at a time as in switched networks described in MIP (2008). Using 1-Wire switching devices like the DS2009, the network may physically resemble one topology but may electrically resemble another. However, the linear topology scheme is proposed for implementation in this work (MIP, 2008).

4.2.2.2 Limitations of 1-Wire Networking

Several factors determine the maximum radius and weight of a network (MIP, 2008; DSC, 2013; DSC-MIP, 2014; MAX, 2014a, 2014b). Some of these factors can be controlled and some cannot. The master-end interface has a great deal of influence on the allowable size of a 1-Wire network. The interface must provide sufficient drive current to overcome the weight of the cable of the slave devices. It must also perform the 1-Wire waveform with timing that are within specification and are outsized for the change and discharge times of the network and if must provide importance match to the network so that signals are reflected back to the line to interface with other network devices.

When the network is small, very single master-end interfaced are acceptable. Capacitance is low, related energies arrive too soon to pose a problem and cable losses are at a minimum, but when line lengths becomes longer and more and more devices are connected complex forces come into play and the master-end interface must be able to handle them. Network radius is limited by the timing of wave form reflection and the time delay produced by the cable as well as by resistance of the cable and degradation of signal levels at about 75m, the delay in getting a response from slave at the far end of the cable back to the master is simply beyond the limits of the protocol (DSC, 2013). Network weight is limited by the ability of the cable to be changed and discharged quickly enough to satisfy the 1-Wire Protocol.

4.2.2.3 Master-End Interface Device

A number of master-end interfaces for personal computers have been developed over the years and a wide variety of methods have been employed to interface 1-wire networks to microcontrollers, but there has been little consistency in that each master was designed with a different intended use and was not always reliable when pressed into alternative service (MAX, 2014b). The master-end hardware in most common use today includes (MAX, 2014b): DS9097 PC serial port adaptor, DS9097U PC serial port adaptor (DS2480.B-based), DS1410E PC parallel port adaptor, DS9097U-E25 serial port adaptor w/EPROM programming, Microcontroller with slew-rate limited FET and 1-kΩ resistor, microcontroller with advanced bus interface, and Microcontroller with DS2480B bus interface.

4.2.3 Transmission Control/Internet Protocol (TCP/IP) Networking

TINI's main objective is to provide a powerful a platform for developing embedded applications that connect non–networked devices to the network. TINI's broad networking is its most compelling feature, and Java suitability for writing networked applications is one of the primary reasons TINI provides a Java runtime environment. This project does not cover a general treatment of TCP/IP networking or writing network applications in Java rather it focuses primarily on programming for TINI's networking environment. Six application layer protocols are supported in the TINI API as shown in Fig. 15.

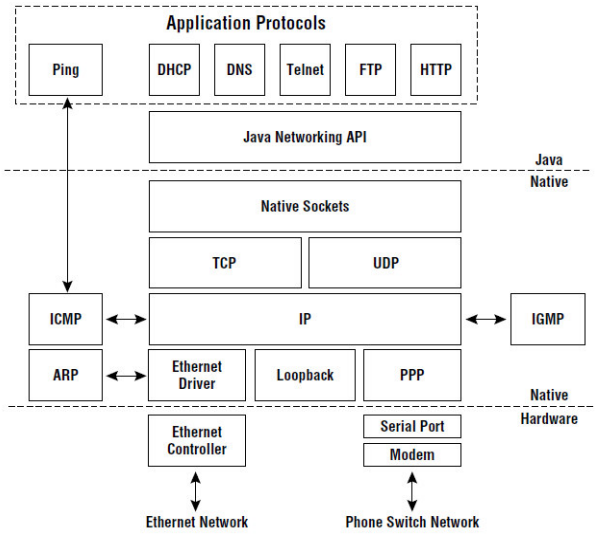


Fig. 15: Network protocol stack.

All of the application layer protocols except *ping* are implemented using the socket classes in the java.net packages as the network transport mechanism. Ping is not really a protocol but it is and application wrapper over a subject of ICMP. The ping class directly, invokes native methods that are exposed in the network stack;s ICMP module. Support for most of the application layer protocols is provided by the sub-packages of *com.dalsemi.tininet* which includes *com.dalsemi.tninet.http*, *com.dalsemi,tininet.icmp*, *com.dalsemi.tininet.dhcp*, *com.dalsemi.tininet.dus*. The FTP and Telnet protocols are implemented the *com.dalsemi.shell.server.ftp* and *com.dalsemi.shell.server.telnet* respectively. Both FTP and Telnet are implemented as servers and are typically used by system shells such as slush support for using FTP as a client is of course available using the URL (uniform resource locator) classes in the java.net package. The protocols in Fig. 15 are those for which the TINI networking API provides supports. Other networking protocols can be written in Java and can run on TINI with little or no changes to the codes.

4.3. TINI's Input/Output (I/O) Ports, Memory Map and Parallel Bus

4.3.1 TINI Integrated Input/Output (I/O) Ports

All TINI's hardware peripheral devices described are all interfaced to the microcontroller's address and data busses. However, a broad range of devices that are interesting to network-enable with TINI don't have support to interface to a full parallel bus. Often these devices have some form of serial interface. This usually results in a lower communication bandwidth. But a serial interface also reduces the required pin count, simplifies communication, and often lowers cost when compared with devices that have parallel bus-type interfaces. Serial interrupts also have the advantage of adding no load to either of the microcontroller's busses. Support for the following low-level serial communication protocols has been integrated onto the microcontroller.

1). Serial communication: Synchronous serial protocols using 1-wire interface and asynchronous serial communication, based on the RS232-C standard, are supported. TINI's controller provides two integrated UART (Universal Asynchronous Receiver Transmitter) circuits to facilitate serial communication. Asynchronous serial ports are extremely common in legacy devices.

2). Controller Area Network (CAN): Originally developed at Bosch-Siemens, CAN is now described in two ISO standards. It provides a reliable serial communications bus that is commonly used in automotive and industrial control applications. TINI's microcontroller provides two integrated CAN controllers. The application programming interface for communicating with CAN devices supported by TINI can be found in DSC (2014).

3). 1-Wire net: Developed by Dallas Semiconductor, the 1-Wire net is a network of small sensors, actuators, and memory elements that all share the same conductor for both communication and power.

4). TTL I/O: These general purpose, bidirectional microcontroller port pins may be used for various control tasks and are not necessarily tied to any type of serial communication device.

Utilizing the microcontroller's integrated I/O capabilities instead of the memory-mapped I/O, reduces both total device count and the cost of communicating with an external device because it burdens the CPU less than communicating with devices interfaced to the microcontroller's busses. For example, the microcontroller's CPU core runs at full speed, executing the runtime environment, while the UART is simultaneously sending and receiving serial characters. Communicating with bus interfaced peripherals, on the other hand, requires the CPU to stop its current operation and execute instructions to read data from or write data to the device.

4.3.2 TINI Memory Map

A memory map specifies where memory and other peripheral devices are decoded in the microcontroller's address space. The memory map used by TINI, shown in Fig. 16, consists of the following three distinct segments: code, data, and peripheral. The segment sizes shown in Fig. 16 are maximums and are all multiples of 1 megabyte. If, for example, only 512 kilobytes of flash ROM exists in the code segment, the starting address of the data segment remains 0×100000. In other words, the starting addresses of the different segments are always as shown in Fig. 16. But the ending address may be less than those indicated, depending on how much of the space is actually occupied by the memory chips. The minimum memory requirement for the code and data segments is 512 kilobytes each.

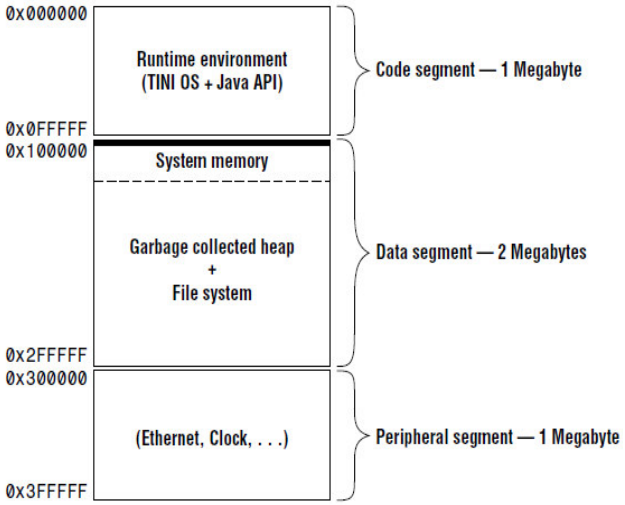


Fig. 16: TINI's memory map.

The code and data segments are occupied by memory chips, and the peripheral segment is occupied by other types of hardware components such as the Ethernet controller and real-time clock (Akpan and Osakwe, 2015a, 2015b; Akpan et al, 2013). Other peripheral devices that support a parallel bus interface compatible with the microcontroller's bus can also be mapped into the peripheral segment. As a caution, adding hardware in this fashion also adds capacitive loading to either or both the data and address busses (depending on the device). The system designer must be aware of this loading to ensure reliable system operation.

The Ethernet controller and real-time clock occupy these address ranges are for: Ethernet controller is [0x300000 - 0x0x307FFF] and real-time clock is 0x310000. It is, however, advisable to avoid these ranges for interfacing any device other than an Ethernet controller or real-time clock (DSC, 2014). The rest of this address range is available for adding other peripheral devices.

However, there is also a separate 4-megabyte peripheral area, known as peripheral chip enable (PCE) space, that can be used to interface large (up to four 1-megabyte) external memory chips or other hardware devices directly to the microcontroller's address and data busses. However most hardware is mapped in the peripheral segment, shown in Fig. 16, because it can be accessed more efficiently by the controller. The microcontroller uses four pins to control the PCE space. If no devices are mapped into this space, the microcontroller pins can be dedicated for use as general purpose port pins. Unlike the Ethernet controller and the real-time clock address region, these peripheral areas are available for use either for interfacing hardware directly to the controller's address and data busses or general purpose TTL I/O, but not both.

4.3.3 TINI's Parallel Bus and Pin-outs

TINI's parallel bus is used, at a minimum, for interfacing with external memory chips for code and data storage. Peripheral devices such as an Ethernet controller and real-time clock are also accessed via the parallel bus. The block diagram shown in Fig. 17 presents a fairly generic configuration for interfacing external devices to the bus. As their names suggest, the address bus specifies the target address of the read or write operations, while the data bus transfers the binary data to and from the device. The combination of certain control signals and possibly certain address lines can be used in conjunction with decoding logic to act as an enable signal for the peripheral. The purpose of the enable signal is to ensure that only bus operations intended for the device are actually seen by the device. Note that some devices, including many memory chips, can be interfaced directly to the bus without using any decode logic.

An important point to be made here is that Fig. 17 shows the address and data bus signals connected directly to the external device. This is often appropriate. However, depending on the total number of devices on the parallel bus, either or both the address and data signals may require external buffering to ensure reliable

system operation. Buffers are chips that provide isolation from the capacitive loading of bus interfaced peripherals. Whether or not buffering is required, and how to buffer the bus if it is required, are design specific issues.

The signals that comprise the microcontroller's parallel bus can be grouped into the following categories, namely: *1)*. Data (a bidirectional data bus), 2*)*. Address (a unidirectional address bus, driven by the controller), and *3)*. Control (provides signals for distinguishing between read and write operations as well as device or chip selection).

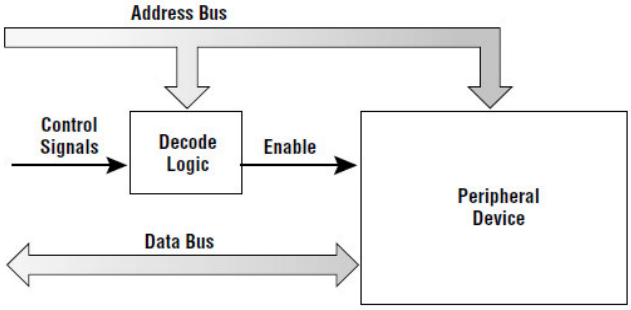
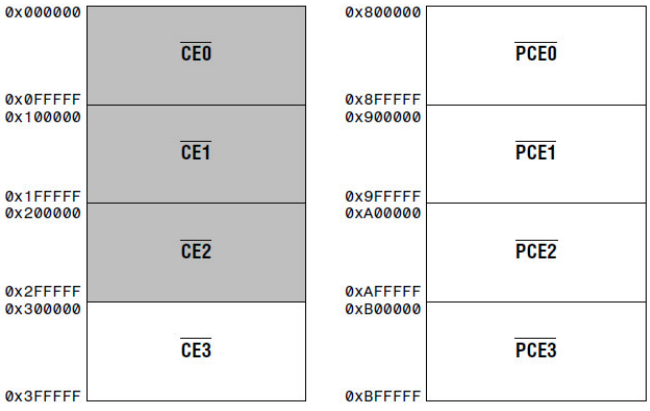


Fig. 17: Interfacing external peripherals to the TINI's microcontroller bus.



(a) (b)

Fig. 18: DataPort mapping of (a) CE and (b) PCE address ranges on TINI's microcontroller bus. The shaded portions are not accessible using data port.

Table 2: TINI's bus control signals

S/N	Designator	Full Signal Name	Description
1	*D0 – D7*	Data Bus	8-bit wide bidirectional data bus
2	*A0 – A19*	Address Bus	20-bit wide address bus
3	*CE0 – CE3*	Chip Enables	Chip enable lines are used to select memory or attached peripherals. Code fetches must occur from memory chip enabled by one of these signals.
4	*PCE0 – PCE3*	Peripheral Chip Enables	Peripheral chip enable lines are commonly used to enable memories for purposes of data storage only. No native code can be fetched from memory chips enabled by these signals.
5	$\overline{PSEN}$	Program Store Enabled	Strobe line used to control code fetches (reads) from external memory devices enabled by CE lines. It can also be used for data fetches.
6	$\overline{RD}$	Read Strobe	Read strobe line used for data fetches from memory and other peripheral devices enabled by PCE lines.
7	$\overline{WR}$	Write Strobe	Strobe line used for data writes to memory and other peripheral devices.
8	*DRST*	Device Reset	This is not a formal signal defined in the parallel bus. On TINI, it is used to reset external devices.

All data, address, and control signals are listed, along with brief descriptions, in Table 2. However, a complete description of the microcontroller including all the signals described in Table 2 can be found in (Akpan and Osakwe, 2015b; DSC-MIP, 2014; Loomis, 2001). The data bus (D0–D7) is an 8-bit bidirectional bus. All data transfer occurs on this bus, including code fetches from flash ROM, data fetches from static RAM, and read and write operations to bus interfaced peripherals. External devices are addressed using the 20-bit address bus (A0–A19) along with one of eight predecoded "chip select" signals. The 20-bit address bus provides a 1-

megabyte address range. However, this range is extended by the eight chip select lines that each decodes a separate megabyte of address space. The chip selects come in two flavors: chip enables (CEs) and peripheral chip enables (PCEs). There are four CE signals (CE0–CE3) and four PCE signals (PCE0–PCE3).

The memory map, shown in Fig. 18, is split into two separate 4-megabyte ranges. The CE space contains all memory chips used as program and data storage for the runtime environment. It also contains a 1-megabyte peripheral area for addressing high-speed devices that support a parallel bus interface. A more detailed memory map of the CE space is contained in Fig. 16. The primary difference between the CE and PCE signals is that the PCE signals can only be used for data reads and writes. In other words, the microcontroller cannot fetch *native* code from memory devices that are enabled using the PCE signals. This is why the flash ROM and static RAM used by the runtime environment are accessed using the CE signals.

The CE addresses correspond to true physical addresses in the microcontroller's memory map. The starting address of memory enabled by PCE signals is somewhat arbitrary because it's a virtual address mapping. Real PCE addresses actually overlap CE addresses. This requires the microcontroller to change memory maps when transitioning from accessing devices mapped into CE space to accessing devices mapped into PCE space. Applications accessing devices in PCE space don't need to worry about the details of this address map swapping because they are managed automatically by the parallel I/O driver. However, the system designer should be aware that there is overhead associated with swapping between CE and PCE memory maps. Data transfer rates on block move operations are about three times faster when only CE mapped devices are involved.

TINI's runtime environment does not reserve any of the PCE space for peripheral devices. This implies that all four PCE signals, and the four megabytes of address space they control, are wide open for system designers. However, many high-speed peripheral devices are mapped into the CE3 address space because it can be accessed more efficiently by the microcontroller. If no devices are mapped into PCE space, the four PCE pins can be used as general purpose port pins. The system designer is free to use the peripheral area either for interfacing hardware directly to the microcontroller's parallel bus or as general purpose TTL I/O but not both.

5. Overview of the Database-Driven Website Development

5.1 Overview of the Proposed Design Workflow

Databases are a fantastic way to store most types of relational data and information. A database allows one to store rational information in a logical format making it easy to manage and retrieve. It is possible to take almost any existing database and integrate it into website (DBS, 2007). A database driven website is a website that uses a database to gather, display, or manipulate information. A database driven website is a website that has most of its web pages content in a database (Quackit, 2012). According to Yank (Yank, 2002, 2012), database driven website programming can also be called or characterized as server side programming. The reason it is so called because the action or magic that allows the web pages to connect to the database is actually taking place in the server. The web pages that are created when a database website design solution is used are actually called dynamic web pages.

According to DB Net Solution (DBS, 2007), people have encountered databases such as Microsoft Access or software based on databases. Database allows storage of relational information in a logical format, thus making it easy to manage and retrieve such information. It is also possible to take almost any existing database and integrate it into a website. Quackit (2012) stated that some websites have a combination of static and dynamic content. The reason for this is that smaller website will be static and vice versa. There is little need to configure a database just to store a handful of web pages; it is much easier and cheaper to keep such pages as files on the server.

Today, websites are often developed using dynamic pages driven by databases. The web pages that are created when a database websites design solution is used are actually called dynamic web pages. A website with dynamic content usually has a CMS (content management system) to assist the content providers in updating the website. According to Quackit (2012), a CMS is usually provided in the form of an administration area where content providers need to log in before they can add contents. The information contained in the databases is usually kept up to date by using a website content management system. The content management system can be designed and tailored to specific requirement. Once logged in, they can create, update and delete articles. They may be able to upload files such as word documents, PDF files, e.t.c. They may be able to upload images too. All this content can be stored in the database. Some may be stored on the file system too. An example, are documents and images that can be stored in the database, they are sometimes stored on the file system as performance and database size are key reasons.

5.2 Reasons for Using Database Driven Web Pages

According to Rogers (Rogers, 2012), the following are the reasons for using database driven web pages: *1)*. Storing and distribution of large amounts of information (e.g. Library/information centers, news articles, e.t.c.); *2)*. The need to add, update, maintain and display historical data (e.g. transactional data entry systems, surveys,

polls, customer information); *3)*. To manage membership-based or organization. Storing information about members, such as their bios and profiles, creating membership directories and implement the appropriate security requirements in members only content areas; *4)*. For the maintenance of large volume of static web pages; *5)*. For centralization of data; *6)*. To share the same content across several different pages. Storing that information a database ensures that only the database table is updated; and *7)*. For more effective website maintenance and content management.

5.3 Advantages of a Database Driven Website

In the following we highlight some advantages of the proposed database driven website:
1). Time: Managing a website can become time consume g and costly, especially if a large number of services is to be offered on the website. It is simpler if each of this service occupies a single record in a database. Once a service is changed the database can simply be uploaded unto the website, and the web pages are updated in one simple action.
2). Skills: No HTML or website design knowledge is required. Using database completing simple forms website can easily be updated. Large complex websites can be maintained by user with very little computer experience.
3). Cost: Using database driven website is cost effective. Content management tools give freedom to create dynamic web page content, rather than being dependent upon a third party, a web designer.
4). Interactivity: Database driven website present visitors with information which relates specifically to individual needs. It also provides interactive elements to ensure that users have reasons to return to the website on an ongoing basis.
5). Ability to evolve: Website has the ability to evolve without incurring extra costs. Website can be as large as possible. Just by adding more records to the database extra web pages will automatically be created.
6). Databases add a new dynamic angle to the website giving additional leverage and accessibility to content providing a richer and more interactive user experience and at the same time reducing the support and maintenance overhead associated with maintaining static websites.
7). Updating: Incorporating a database into websites can reduce the hassle and complexity of maintaining and updating the website. It is highly searchable and organized making content far more accessible to the end user. Traditional software based systems can be migrated.

5.4 An Overview of a Database Driven Web-based Design

Web applications are by nature distributed applications. In particular, this means that one part of the application is executed on the web server while another runs on the client computer within a web browser window as illustrated in Fig. 19.

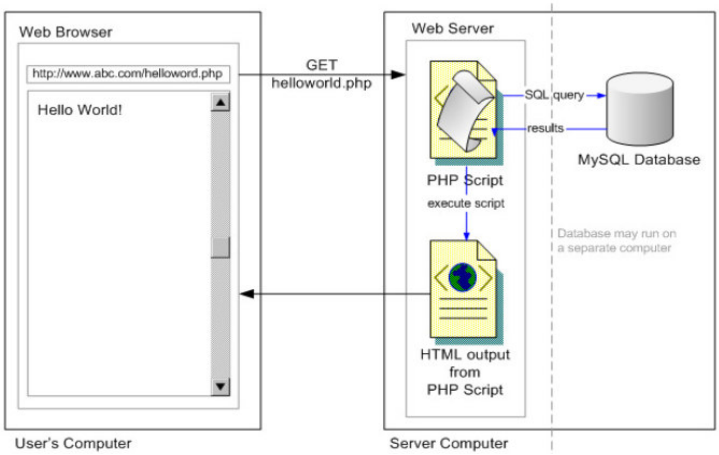


Fig. 19: A web browser requests a document from the web server. The web server parses and executes a PHP script which dynamically assembles the HTML code that is then sent back to the browser which renders the HTML code into a webpage.

When the user types in a web address into the URL (i.e. uniform resource locator) and hits the enter key (or clicks on a link or chooses a bookmark), the web browser sends this address into the so-called *HTTP* (HyperText Transfer Protocol) request to the web server. If the web server receives a request for a static *HTML* file (typically indicated by a file name ending in *.html* or *.htm*) it merely locates the corresponding file on its hard-drive and sends the requested information back to the computer that had requested it. If the web server receives a request for dynamic files (often indicated by *.php*, *.asp*, *.cfm*, *.jsp* etc.), it does a little bit manipulation before sending the response back to the web browser.

5.4.1 The Main Task of the Web Browser

The web browser is responsible for displaying the layout described by the HTML file it has received (this process is called "page rendering"). Once the web page is rendered on the screen the user can interact with it: it involves clicking on the links, fill in forms etc. Since HTML alone provides only a crude level of interactivity (namely links, form elements, buttons and image maps), JavaScript was developed. JavaScript is a programming language which gives the web designer a number of options for creating sophisticated user interfaces (e.g. input validation, pop-up windows, cascading menus using cascaded style sheet, *CSS*). JavaScript code is often contained within an *HTML* file enclosed by the *<script>* tag. JavaScript is always executed by the web browser.
The web server is fully ignorant about JavaScript (although, there is something like server-side JavaScript but this subject is outside the scope of this study and we refer interested readers to (Bean, 2010; Junion-Metz and Stephens, 1998; Lloyd, 2006). However, the web server just serves up any HTML, JavaScript or pictures that have been requested without analyzing or executing any code. For clarity and to avoid confusion; beyond its name, JavaScript has very little in common with Java, which is a programming language that is often used for programs (such as java servlets, java server pages, e.t.c.) running on the web server as well as for highly-interactive (but sometimes slow and crash-prone) java applets running on the client computer. Therefore, when creating a mental map of web technologies, it is better to place JavaScript and Java in opposite corners.

5.4.2 The Main Task of the Web Server

So far, we have only discussed how a web server handles requests for static files (HTML pages, images, etc.). However, if we want to construct our web pages from data contained within a database, the web server performs additional tasks. If the web server receives a request for a dynamic file (indicated by a file ending .php, .asp, .cfm, .jsp, e.t.c.); the web server will locate the file on its hard-drive and interpret (the so-called parse) and execute its contents. PHP (a recursive acronym for "*P*HP: *H*ypertext *P*reprocessor") is a programming language for creating dynamic web pages and is always executed on the web server (as opposed to the web browser on the client computer). Typically PHP commands output certain HTML code depending on user input or data read from a database etc. This resulting HTML code is then sent back to the web browser for rendering.

5.5 Setting up a Database for the Database Driven Web-Based Applications

According to Jochen (2012), databases store data persistently. A database is a piece of software which often runs on the same physical computer as the web server. Data can be collected via the web, web applications are by nature distributed applications. In particular, one part of the application is executed on the web server while another runs on the client computer within a web browser window. Before building a dynamic web site, the tools needed are:

1). PHP: This is a server-side scripting language. It is a plug-in web server that enables it to do more than just exact copies of the files that the browsers ask for. With PHP, web server will be able to run little programs call PHP scripts. It can do little tasks like retrieving up to the minute information from a database and use it to generate a web page on the fly before sending it to the browser that requested it. For PHP scripts to retrieve information from a database, a database must exist. It processes the page request and fetches the data from the MySQL database, then spits it out dynamically as the nicely-formatted HTML page that the browser expects (Yank, 2002, 2009).

2). MySQL: This is a relational database management system or RDBMS. It is a software program that is able to organize and manage many pieces of information efficiently while keeping track of how all of those pieces of information are related to each other. It also makes that information really easy to access with server-side scripting languages like PHP.

3). WampServer: There WAMP stands for Windows, Apache, MYSQL and PHP is an all-in-one program that included built-in copies of recent versions of the Apache web server, PHP and MySQL.

4). Opera Browser: This is used to interpret the codes written in HTML and other scripting languages, the browser is a used to read and give visual interpretation to the html codes.

The whole idea of a database driven website is to allow the content of the site to reside in a database, and for the content to be dynamically pulled from the database to create web pages for viewing purposes with a regular web browser. The whole idea of what happens when a page on database-driven website is visited is illustrated in Fig. 20. Thus, at one end of the system it is possible to have a visitor to the website that uses a web browser to load, say for example *http://www.abc.com/helloworld.php*, and expect to view a standard HTML web page. At the other end, it is possible to have the content of the website, which sits in one or more tables in MySQL database that only understands how to respond to SQL queries (commands).

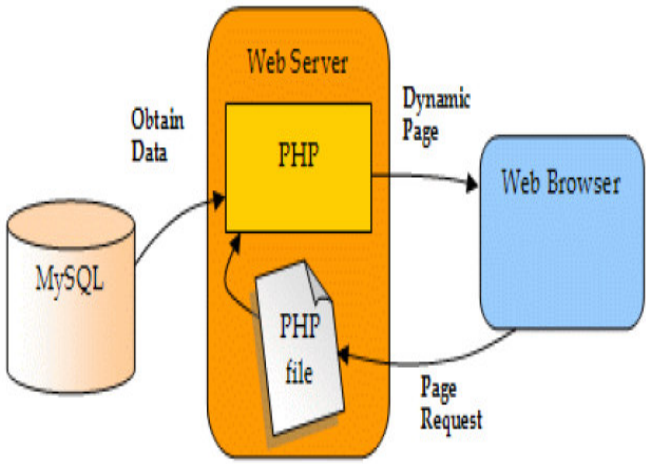


Fig. 20: Schematics of how a web browser receives information from a database (MySQL database) via PHP sitting on a web server.

As shown in Fig. 20, the PHP scripting language is the go-between that speaks both languages. It processes the page request and fetches the data from MySQL database, then outputs out dynamic information as nicely-formatted HTML page that the browser expects. With PHP, presentation aspects of the site can be written (the fancy graphics and page layouts) as templates in the HTML. Where the content belongs in those templates, PHP code is used to connect to the MySQL database and uses SQL queries. According to Yank (2002, 2009), the following happens when a database-driven website is visited:

1). The visitor's web browser requests the web page using a standard URL.

2). The web server software (Apache, IIS, or whatever) recognizes that the requested file is a PHP script, and so the server interprets the file using its PHP plug-in, before responding to the page request.

3). Certain PHP commands connect to the MySQL

4). The MySQL database responds by sending the requested content to the PHP script.

5). The PHP script stores the content into one or more PHP variables, and then uses the now familiar echo function to output the content as part of the web page.

6). The PHP plug-in finishes up by handing a copy of the HTML it has created to the Web server.

7). The Web server sends the HTML to the Web browser as it would a plain HTML file, except that instead of coming directly from an HTML file, the page is the output provided by the PHP plug-in.

5.6 Installing, Setting-Up and Configuring the Software Required for the Development of the Proposed Database-Driven Web-Based Meteorological Weather Station

In the current workflow, five major software are employed for the development of the database driven web-based NYSC posting system, namely: *1)* Apache web server for hosting the application, *2)* Adobe Dreamweaver for writing HTML for the web-based application, *3)* PHP for writing scripts required for the application, *4)* MySQL as the database server for writing SQL, *5)* Google Chrome browser for reading and displaying information developed using the first four tools. These software were installed on a Dual Core Pentium CPU computer having 320 GB of hard disk with 2.00 GB RAM running Microsoft® Windows® 7 operating system.

5.6.1 Setting-Up and Configuring the PHP

1). Open the php.ini file with a favorite text editor. If there is no particular preference, just double-click the file to open it in Notepad. It's a large file with a lot of confusing options, but look for the line that begins with *doc_root* (Notepad's *Edit* > *Find*… feature will help). Out of the box, this line looks like:

doc_root =

At the end of this line, add the path to the web server's document root directory. For the Apache server, this is the htdocs folder in the main Apache web server directory. If Apache is installed in the default location, the path should be "C:\Program Files\Apache. Software Foundation\Apache2.2\htdocs". If Apache is installed elsewhere, find the *htdocs* folder and type its path:

doc_root="C:\wamp\bin\apache\Apache2.2.11\htdocs"

2). Just a little further down in the file, look for the line that begins with *extension_dir*, and set it so that it points to the ext subfolder in the PHP folder:

extension_dir = "C:\PHP\ext"

3). Scrolling down further in the file, and one will find a bunch of lines beginning with *;extension=*. These are optional extensions to PHP, disabled by default. We want to enable the MySQL extension so that PHP can communicate with MySQL. To do this, we remove the semicolon from the start of the *php_mysqli.dll* line becomes:

extension=php_mysqli.dll

Note: Please note that we are referring to "*php_mysqli*" and not "*php_mysql*". Just above the line for php_mysqli.dll there is a line *for php_mysql.dll*. The *i* in *php_mysqli* stands for *improved*. Here, we want to enable this new improved *MySQL* extension. The one without the *i* is obsolete, and some of its features are incompatible with current versions of *MySQL*.

4). Scrolling down even further down in the file, and looking for a line that starts with *;session.save_path*. Once again, we remove the semicolon to enable this line, and set it to the current *Windows Temp* folder to have the following:

session.save_path = "C:\Windows\Temp"

Save the changes made and we close the text editor.

5). The above tasks takes care of setting-up and configuring of the PHP.

5.6.2 Setting-Up and Configuring the Apache Web Server to PHP as a Plug-in

1). Run Notepad as Administrator. This is necessary because the Apache configuration file, by default, can only be edited by an administrator. To do this, find the Notepad icon in the Start Menu (under All Programs > Accessories) and right-click on it. Click the Run as administrator menu item.

2). Choose File > Open in Notepad. Browse to the *conf* subfolder in the Apache installation folder (*C:\wamp\bin\apache\Apache2.2.11\conf*), and select the *httpd.conf* file located there. In order to make this file visible for selection, it may be necessary to select All Files from the file type drop-down menu at the bottom of the Open window.

3). Look for the existing line in this file that begins with *DirectoryIndex*, shown here:

<IfModule dir_module>
DirectoryIndex index.html
</IfModule>

This line tells Apache which filenames to use when it looks for the default page for a given directory. Add index.php to the end of this line:

<IfModule dir_module>
DirectoryIndex index.html index.php
</IfModule>

4). All of the remaining options in this long and intimidating configuration file should have been set up correctly by the Apache install program. Next we need to add the following lines to the very end of the file:

LoadModule php5_module "C:/PHP/php5apache2_2.dll"
AddType application/x-httpd-php .php
PHPIniDir "C:/PHP"

Please be sure that the *LoadModule* and *PHPIniDir* lines point to the PHP installation directory, and note the use of forward slashes (/) instead of backslashes (\) in the paths as well as in the path names.

5). Save all changes and close Notepad.

6). Restart Apache using the Apache Service Monitor system tray icon. If all is well, Apache will start up again without complaint.

7). Double-click the Apache Service Monitor icon to open the Apache Service Monitor window. If PHP is installed correctly, the status bar of this window should indicate the version of PHP that have been installed, as shown in Fig. 21.

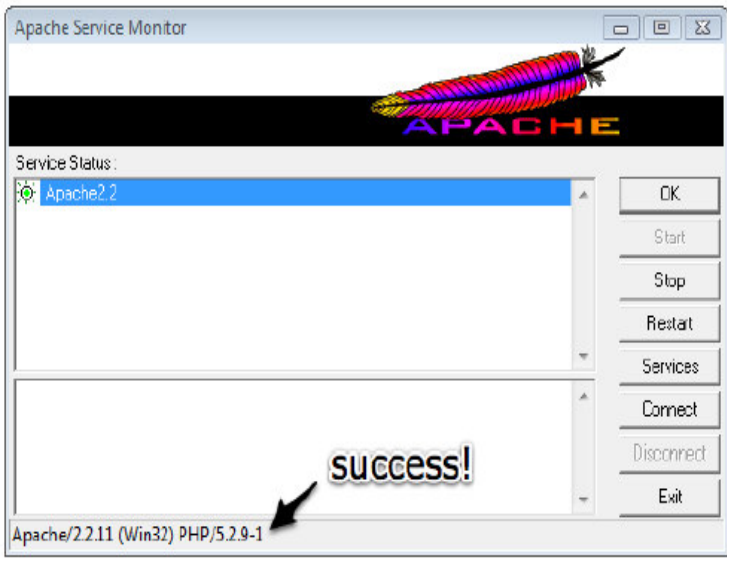


Fig. 21: The PHP version number indicates that Apache is configured to support PHP script development.

8). Click OK to close the Apache Service Monitor window.

5.6.3 Setting-Up and Configuring MySQL Database Server

1). *Server Type*: Assume that one is setting up MySQL for development purposes on a desktop computer, choose Developer Machine.

2). *Database Usage*: Unless we know that we will need support for transactions (as such support is usually superfluous for most PHP applications), then we choose Non-Transactional Database Only.

3). *Connection Limit*: Select Decision Support (DSS)/OLAP to optimize MySQL for a relatively modest number of connections.

4). *Networking Options*: Uncheck the Enable Strict Mode option to ensure MySQL's compatibility with older PHP code that might be needed for use in that case.

5). *Default Character Set*: Select Best Support For Multilingualism to tell MySQL to assume that we want to use UTF-8 encoded text, which supports the full range of characters that are in use on the Web today.

6). *Windows Options*: Allow MySQL to be installed as a Windows Service that's launched automatically; also select Include Bin Directory in Windows PATH to make it easier to run MySQL's administration tools from the command prompt.

7). *Security Options*: Uncheck the Modify Security Settings option. It's best to learn how to set the root password mentioned at this juncture without the assistance of the wizard, so we will show how to do this in the section called "Post-Installation Set-up Tasks".

Once the wizard has been completed, the system should now be fully equipped with for running *MySQL* server! To verify that the *MySQL* server is running properly, we use the *Ctrl+Alt+Del* tabs and choose the option to open the *Task Manager*. Then click the *Show Processes* from all users button unless it has already been selected. If all is well, the server program (*mysqld.exe*) should be listed on the *Processes* tab. It will also start up automatically whenever the system is restarted.

To add the MySQL command prompt programs that come with the Wamp Server to the Windows system path, we perform the following tasks:

1). Open the Windows Control Panel. Locate and double-click the System icon.

2). In our Windows 7, we click the "*Advanced System*" settings link in the sidebar.

3). Click the Environment Variables… button.

4). In the list labeled User variables for *user*, look for a variable named *PATH*.

(i) If it exists, select it and click the Edit… button.

(ii) If there's no variable, click the New… button and fill in the Variable name by typing *PATH*.

5). Add the path to Wamp Server's MySQL bin directory as the Variable value:

(i) If the Variable value is empty, just type in the path. In our case, the path is:
"C:/wamp/bin/mysql/mysql5.1.36/bin"

(ii) If there is already text in the Variable value field, add a semicolon (;) to the end of the value, we type the path name thereafter. If the Variable value field does not exist, it must be created.

6). Click the OK button in each of the open windows to apply and effect the changes.

5.7 Implementation Steps: The Research Methodology for Design of the Database-Driven Web-Based Hypothetical Weather Station

Step 1: Statement of the Problem

In order to reduce data and information redundancy, the very first step involves creating a layout of the overall design for the meteorological weather station based on objectives highlighted in the "Statement of the Problem" discussed in Section 2 which are re-summarized here again as follows: *1)*. Conversion of the physical form of atmospheric weather conditions into electrical signal for the purpose of measurement, calibration and display; *2)*. Development of signal conditioning circuits for configuring the atmospheric electrical signals obtained in the first step *1)* just mentioned above; *3)*. Initialization, configuration and Networking of the TINI microcontrollers for data conversion and routing; *4)*. Configuration of the host computer as a network sever for secured Internet communication and relational database management manipulation; *5)*. Database design and development for the meteo data management; *6)*. Database-driven web-site design for the embedded web-based meteo data logger System; and *7)*. The integration of the above sub-objective for the design of the embedded web-based meteo data logger system.

The scope of this hypothetical study will be limited to seven weather parameters for a particular location. The data logging interval shall be five (5) minutes to a maximum of 24 hours. The meteo data should be made available on the internal over 24 hours period and after which it should be deleted and stored permanently in the host database (archive) management system. The data logger should start fresh data logging at 00.00 GMT. The significant of this study if successfully implemented which will provide accurate up-to-date meteorological information about the best community or state via the Internet and can be expanded to encompass the host nation.

Step 2: Creation of the Database

The seconds step is to design the database. In the current workflow, MySQL software is used to create a relational and dynamic database system named *meteo_weather_station* while the main database file is *meteo_weather_station .sql.* Here, the database is created from the MySQL console using the following syntax:

mysql> CREATE DATABASE meteo_weather_station;

All subsequent development within the database is done from the MySQL graphical user interface (GUI). Using the MySQL GUI different character types, lengths, as well as other properties are specified for the different fields.

Step 3: Writing the PHP Script Files

The third step is to systematically write codes to implement the NYSC posting system. The term "systematically" used here implies that the HTML codes are written while PHP functions are inserted accordingly which will link the HTML code pages to the database in a cascaded format using the cascaded style sheet (CSS) tool properties. In the current workflow, Adobe Dreamweaver™ is used.

Step 4: Deploying and Viewing the Design in a Google Chrome Web Browser

The fourth step is to view the codes in a browser. When the URL (uniform resource locator) is set to the domain name, the webpage comes up from where the user can enter in the required information. The browser is actually used to interpret the codes written in HTML and other scripting languages. In the current workflow, Google Chrome Version 23.0.1271.64m 2012 has been used as the browser since it supports JavaScript and HTML version 5.

Step 5: Viewing the Database Information and Status

The last but optional step for verification of the database information may be performed. Here, the database is re-visited in order to view the information that has been collected from the user. At this point as well as at any other point, the information collected can be edited, modified or deleted.

6. Conclusion and Future Work

The fairly complete but comprehensive hypothetical design of a database-driven web-based meteorological weather station with dynamic datalogger capabilities over a TCP/IP network with emphasis on its instrumentation, control, electronic measurements, programming and information technology has been proposed and presented in this paper. The proposed system and its description is a new dimension which can be used for other weather and meteorological measurements for the purpose of estimation, forecasting, modeling and prediction.

Practical aspects on the initialization and configuration of the TINI and the associated Java communications API for real-time embedded networked applications with the complete Java source codes can be found at Akpan and Osakwe (2015a, 2015b) and Akpan et al., (2013), and can be readily adapted for the proposed design. Furthermore, a demonstration of an efficient networking of the TINI for real-time weather datalogging and deployment over Ethernet and serial communication link with the complete java source codes can also be found from Akpan and co-workers (Akpan et. al, 2013) and can be readily adapted for the proposed design. The database-driven website design approach proposed here has been adapted for the development and implementation of a dynamic database-driven web-based automatic NYSC posting system (Akpan et al, 2015)

and can as well be readily adapted and deployed for the database-driven website for the proposed meteo system design and development.

The techniques proposed and presented in this study can be adopted in refineries, oil companies, manufacturing industries, pipe line installation companies for supervisory control and data acquisition systems (SCADA), robotics and artificial intelligence control systems, as well as process and complex system control (Akpan and Osakwe, 2014b). The database design techniques can also be extended to libraries, banks, institutions and commercial sectors for relational database management system design. The meteorology data logger system can also be installed in remote locations where there are no Ethernet or telephone link using the recently introduced industrial wireless communication RADIOLINX developed by Pro-soft Technology. The TINI microcontroller presented in this proposed study can be used in conjunction with RADIOLINX for high voltage transmission and distribution lines, pipelines, railway tracks, water and wastewater monitoring and control amongst other applications.

References

Akpan, V. A. and Babalola, M. T. (2009). "The development of a Model automatic electromechanical vending machine". *Journal of Science & Technology Research*, vol. 8, no. 3, pp. 121 – 131.

Akpan, V. A. and Osakwe, R. O. A. (2015a). "Initialization and configuration of TINI & Java communications API for real-time embedded networked applications", *Journal of Advanced Research in Networking and Communication Engineering (JoARNCE),* vol. 2, no. 2, pp. 1 – 26 Available [Online]: http://technology.adrpublications.com/index.php/JoARNCE/article/view/225/222.

Akpan, V. A., and Osakwe, R. O. A. (2015b). "Real-time embedded system for online temperature and pressure measurements for automated fluid flow monitoring and control", *Journal of Advanced Research in Instrumentation and Control Engineering (JoARICE)*, vol. 2, no. 1, pp. 29 – 70. Available [Online]: http://technology.adrpublications.com/index.php/JoARICE/article/view/163/178

Akpan, V. A., Osakwe, R. A. O. and Amaku, A. (2013): Efficient Networking of TINI for Real-Time Weather Data Logging & Deployment over Ethernet and Serial Communication Links, vol. 3, no. 11, pp. 973 – 999. [Online] Available: http://iet-journals.org/archive/2013/november_vol_3_no_11/819767137735515.pdf

Akpan, V.A. and Babalola, M. T. (2008). "An experimental configuration of strain gauges for weighing systems design", *International Journal of Pure and Applied Science*, vol. 1, no. 2, pp. 1 – 7.

Anealle, K. M. (2003). "What's Happening to the weather?", *Awake, Jehovah's Witnesses*, *August ed.,* Edo State, Nigeria.

Anthes, R. A, (1978). "The Atmosphere", 2nd edition, Merrill, U.S.A.

Asheville (2004). "World data center for Meteorology", Asheville. Available [Online]: http:/www.noaa.gov/oa/wmo/wdcamet.html.

Awtrey, D. (1997). "Transmitting data and power over a One-Wire bus", *Sensors*, vol. 14, no. 2, pp. 48 – 51.

Awtrey, D. (1998). "The 1-Wire weather station", *Sensors*, vol. 15, no. 6, pp. 34 – 40.

Babalola, M. T. and Akpan, V. A. (2006). "Experimental determination of the Poisson's ratio (μ) and the gauge factor (k) values of a practical strain gauge (120 – SRA Foil Type)", *Ife Journal of Science*, vol. 8, no. 1, pp. 47 – 58.

Batlan, L. J. (1985). "Weather in your Life", W.H. Freeman, U.S.A.

Bean, J. (2010). "SOA and Web Services Interface Design: Principles, Techniques, and Standards", Morgan Kaufmann Publishers, U.S.A.

Brush, S. G. (1984). "*The History of Meteorology and Geophysics*" Edited by Martin Cochins and Robert Mutthauf, Drosoft Technology, Garland.

Budyko, M. I. and Golitsyn, G. S. (1988). "*Global Climate Catastrophies*", Springer-Verlag, The Netherlands.

DBS (2007). DB NetSolution, "Database Driven Websites", Available [Online], Downloaded 23rd Oct, 2012: http://www.dbnetsolutions.co.uk/Articles/DatabaseDrivenWebsites.aspx.

Dotto, L. (1988). "*Thinking the Unthinkable: Civilization and Rapid Climate Change*", Humanities Press.

DSC (2013). Dallas Semiconductor Corporation, "1-Wire Networking", Available [Online]: http://www.ibutton.com/ibuttons/standard.pdf.

DSC (2014). The DS80C390 Datasheet, Available [Online]: http://www.dalsemi.com/datasheets/pdfs/80c390.pdf

DSC-MIP (2014). Dallas Semiconductors inc. (Maxim Integrated Products), "Tiny Internet Interface (TINI) and IBUTTONS", Available [Online]: http://www.ibutton.com.

Felix O. I. (2000). "The Nigeria Experience in Satellite Imagery, Reception and Interpretation", *Nigerian Meteorological Services*, Control Forecast, Ojo, Nigeria.

Flavio, N. (2001). "Meteorological Data Acquisition System", *ESTI Technical Documentation*, ESTI. Available [Online]: http://216.239.39.104/2+meteorological+data+acquisition+systems.

Guzzi, R. (1990). "*Meteorology and Environmental Sciences*", World Scientific Publication.

Herman J. R. and Goldberg, R. A. (1985). "*Sun, Weather and Climate*", Dover.
Jochen, R. (23rd Oct., 2012): "Creating Database-driven Websites with PHP & MySQL", Available [Online]: http://www.hosting.vt.edu/tutorials/phpmysql/.
Junion-Metz, G. and Stephens, B. (1998). "Creating a Power Web Site: HTML, TABLES, IMAGEMAPS, FRAMES, AND FORMS",Neal-Schuman Publishers, New York, U.S.A.
Lamb, H. H, (1988). "*Weather, Climate and Human Affairs*", Rout Ledge.
Lamb, H. H., (1982). "*Climate, History and the Modern World*", Rout Ledge.
Lloyd, I. (2006). "Build Your Own Web Site The Right Way Using HTML & CSS", Sitepoint, Collingwood, U.S.A.
Loomis, D. (2001). "*The TINI Specification and Developer's Guide*", First Edition, Addison-Wesley, Boston.
Ludlum, D. (1993). "*The American Weather Book*", Houghton.
Lutgens, F. K. and Edward I. T. (1989). "*The Atmosphere*", 4th ed., Prentice–Hall, U.S.A.
MAX (2014a). Maxim Integrated Products, "Advanced 1-Wire network driver, AN 244 Reference schematic", 2014. Available [Online]: http://www.maximintegrated.com/an244.
MAX (2014b). Maxim Integrated Products, "DS9097U-DS9097U-S09 Universal 1-Wire Com Port Adapter", 2014. Available [Online]: http://www.maximintegrated.com/datasheet/index.mvp/id/2983.
McBridge, L. W. (1973). "*Experiments in Meteorology: Investigation for the Amateur Scientist*", Doubleday.
Miller, A. and Thompson J.C. (1979). "*Elements of Meteorology*", 3rd ed., Merrill.
MIP (2008). Maxim Integrated Products, "Guidelines for reliable long line 1-Wire® networks, Tutorial 148", Sept. 22, 2008. Available [Online]: www.maximintegrated.com/app-notes/index.mvp/id/148.
Moran, J. M. (1991). "*Meteorology*", 3rd ed., Macmillan, U.S.A.
Moran, J. M., and Morgan M. D. (1986). "*Meteorology: The Atmosphere and the Science of Weather*", Burgess.
Oniarah, A. (2004). "CLIPs focal point, Climate/Investigation Service Division, Directorate of Weather Forecast Services", *Nigerian Meteorological Agency*, Lagos Nigeria. Available [Online]: http://www.wmo.ch/web/wcp/clips2001/html/FP_reports_ 2003/Nigeria%202003.doc.
Quackit (23rd Oct., 2012): "Database Driven Website", Available [Online]: http://www.quackit.com/database/tutorial/database_driven_website.cfm.
Rogers, K. (2012). "Top 10 Reasons for Using Database Driven Web Pages", 24th Oct., 2012. Available [Online]: http://www.websmithgroup.com/blog/web-application-development/top-1.
TMS (2004). "Weather observations and Radio Soundings at the Meteorological Station". Available [Online]: http://home.online.no /~Vteigen/gass.html.
WMO (2004). "Basic Facts About WMO", *World Meteorological Organization (WMO)*. Available [Online]: http://www.wmo.ch/web-en/wmofact.html.
Yank, K. (2009). "Build Your Own Database Driven Website Using PHP & MySQL", 4th ed., Sitepoint, Collingwood, U.S.A.
Yank, K. (24th Jan., 2002). "Build Your Own Database Driven Website Using PHP & MySQL: Part 4", Available [Online]: http://www.databasejournal.com/features/mysql/article.php/1402281/Bui

About The Authors

Vincent A. Akpan received his Ph.D. degree in Electrical & Computer Engineering from the Aristotle University of Thessaloniki (AUTH), Thessaloniki, Greece in 2011.
Since 2004, he has been with The Federal University of Technology, Akure (FUTA), Nigeria where he is currently Lecturer I with FUTA, Akure, Nigeria. He has been an Visiting as well as an Adjunct Lecturer to a number of Universities in Nigeria with the Departments of Computer Science, Electrical and Electronics Engineering, Information & Communication Technology (ICT) and Mechatronics Engineering, His integrated research interests include: intelligent adaptive control, computational intelligence, instrumentation, real-time embedded systems, signal processing & machine vision, nonlinear system identification, as well as robotics & automation. He is the co-author of a book and has authored and/or co-authored more than 55 articles in refereed journals and conference proceedings. He is a regular reviewer in more than 20 local and international scientific/academic journals and several IEEE sponsored conferences. He is the managing editor of the Nigerian Journal of Pure and Applied Physics (NJPAP) as well as an editorial board member for the American Journal of Intelligent Systems (AJIS), American Journal of Embedded Systems and Applications (AJESA) and Automation, Control and Intelligent Systems (ACIS), USA.
Dr. Akpan is a member of the IEEE, USA, The IET, UK, a fellow of The Institute of Policy Management Development (IPMD), Nigeria and a member of the Nigerian Institute for Biomedical Engineering (MNIBE). He was one among the two Nigerian recipients of the 2005/2006 Greek State Scholarship (IKY) for a Ph.D. programme tenable in Greece.

Reginald A. O. Osakwe received his Ph.D in electronics from the University of Ibadan, Nigeria in 1995.
Since 2007 he has been with the Department of Physics, The Federal University of Petroleum Resources, Effurun, Delta State, Nigeria where he is currently a Senior Lecturer. During the periods 1998 – 2009 he has been a visiting lecturer to the Shell Intensive Training Programme (SITP), Warri, Delta State under the Shell Petroleum Development Company (SPDC) of Nigeria. His research interests are on microprocessor architecture, embedded computer systems, chaotic dynamics, and image and signal processing systems.
Dr. Osakwe is a member of the Nigeria Institute of Physics (NIP). He was a recipient of The Federal Government of Nigeria Scholarship to study in the U.S.A. leading to the award of an M.Sc.

Sylvester A. Ekong had his B. Sc. Degree in Physics (With Electronics) from the Delta State University (DELSU), Abraka, Nigeria in 2005; and a Master of Technology (M. Tech.) degree in Condensed Matter Physics from The Federal University of Technology, Akure (FUTA), Nigeria in 2011.
Between 2006 and 2007 he was a Physics and Further Mathematics tutor at Government Model Secondary School, Ajikamai, Shendam, Plateau State, Nigeria; Between 2007 and 2009 he was a tutor of Physics and Further Mathematics and also the practical Physics coordinator at St. John's College, Kachia, Kaduna State, Nigeria; He was, between 2009 and 2011, a tutor and practical Physics coordinator with Idris Primier College, Akure, Ondo State; and also between 2011 and 2012 he was a Physics and Further Mathematics tutor with Model secondary school, Alagbaka, Akure, Ondo State, Nigeria. After obtaining his M. Tech. degree, in November 2011 he joined as an Assistant lecturer in 2012, the Department of Physics and Energy Studies, Achievers University, Owo, Nigeria. He has co-authored over 15 articles in refereed journal and conference proceedings. His research interests are on — digital signal processing, soft-computing, Theoretical & Computational condensed matter and chaotic systems.
Mr. Ekong is a member of The African Review of Physics (ARP) and The National Association of Theoretical and Mathematical Physics (NAMP). He is currently a Ph.D. student in the Department of Physics, University of Benin, Benin City, Nigeria.