

**DESIGN AND CONSTRUCTION OF A  
DIGITAL TEMPERATURE DATA  
LOGGER**

**SISAN BOYO IYINBOH**

**2004/18755EE**

DEPARTMENT OF ELECTRICAL AND COMPUTER  
ENGINEERING, FEDERAL UNIVERSITY OF TECHNOLOGY,  
MINNA.

**DECEMBER 2009**

**DESIGN AND CONSTRUCTION OF A  
DIGITAL TEMPERATURE DATA  
LOGGER**

**SISAN BOYO IYINBOH**

**2004/18755EE**

**A THESIS PRESENTED AND SUBMITTED IN PARTIAL  
FULFILMENT FOR THE REQUIREMENT OF FIRST DEGREE  
IN ELECTRICAL AND COMPUTER ENGINEERING, FEDERAL  
UNIVERSITY OF TECHNOLOGY, MINNA.**

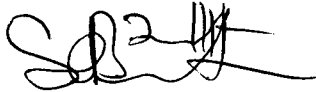
**DECEMBER 2009**

## **DEDICATION**

This project is dedicated to God Almighty, my father Mr. Solomon Boyo Iyinboh, my mother Mrs. Victoria Iyinboh, my brothers and sisters, my uncles Mr. Clement and Benjamin Okpaghoro and also Mr. Misan Tenumah, my grandmother Mrs. Alice Tenumah , for the moral, spiritual and financial support given to me.

## ATTESTATION / DECLARATION

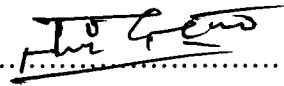
I Sisan Boyo Iyinboh, declare that this work was done by me and has never been presented elsewhere for the award of a degree. I also hereby relinquish the copyright to the Federal University of Technology, Minna



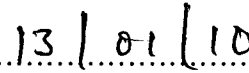
Sisan Boyo Iyinboh



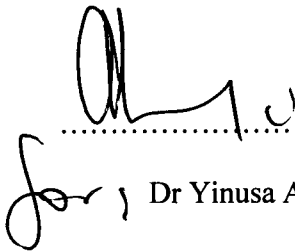
Date



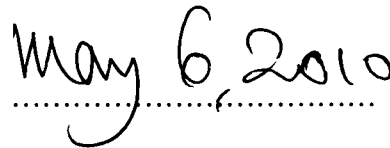
Eng'r J. G. Kolo  
(Supervisor)



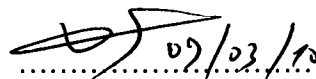
Date



Dr Yinusa A. Adediran  
(H.O.D)



Date



External Examiner

Date

## **ACKNOWLEDGEMENT**

I give God all the glory for His wisdom and His grace and supernatural direction upon me.

Also, I appreciate my entire family, uncles and grandmother for their endless support towards ensuring that this goal is achieved.

More so, I appreciate my friend and brother Sheyi Mackson Ejejigbe for availing me the use of his computer throughout the period of my work. Also to my beloved friend Godiya Lafiya L adama, thank you for your wonderful support may God richly bless you.

Finally, my special thanks goes to my supervisors Eng'r J.G. Kolo and Mallam Umaru for their tremendous assistance, contributions and suggestions towards ensuring that this project is successful. To my friends and well-wishers may the grace of God be with you all.

## **ABSTRACT**

The quest of establishing an effective mechanism for temperature measurement and monitoring, brought about the design of this device ( Digital Temperature Data Logger).

Digital temperature data logger is an electronic device that measures and records temperature data over time in relation to a location. It employs temperature sensor in converting physical temperature to electrical quantity which is then digitized by an analog-to-digital converter for easy analysis by a microcontroller. The microcontroller analyses the digitized temperature data and stores it to the memory of the device along with the time and date of acquisition stamped to it. Viewing and analysis of the collected data is achieved by interfacing the device with a personal computer via the PC-resident software developed for the device.

## LIST OF FIGURES

Fig. 1.1 Block diagram of the digital temperature data logger.

Fig. 3.1 System power supply.

Fig. 3.2 Battery charger system.

Fig. 3.3 Analysis of the voltage setting resistances.

Fig. 3.4 Sensor circuit.

Fig. 3.5 ADC- microcontroller interface.

Fig. 3.6 DS1307/24C32- microcontroller interface.

Fig. 3.7 LEDs status indicator.

Fig. 3.8 Pin assignment of DS1307.

Fig. 3.9 Pin assignment of 24C32.

Fig.3.11 Logic level translator.

## **LIST OF TABLES**

Table 3.1 Data record implemented on the 24C32 device.

Table 3.2 Pin description of DS1307.

Table 3.3 Function table of 24C32.

Table 4.1 Digital logger user interface.

Table 4.2 Temperature display.



## TABLE OF CONTENT

CONTENT	PAGES
COVER PAGE .....	i
DEDICATION .....	ii
ATTESTATION/DECLARATION .....	iii
ACKNOWLEDGEMENT .....	iv
ABSTRACT .....	v
LIST OF FIGURES .....	vi
LIST OF TABLES .....	vii

### CHAPTER ONE

1.1 INTRODUCTION .....	1-2
1.2 AIMS AND OBJECTIVES OF THE PROJECT .....	2-3
1.3 METHODOLOGY .....	3-4
1.4 APPLICATION OF TEMPERATURE DATA LOGGER .....	4-5
1.5 SCOPE OF WORK .....	5

### CHAPTER TWO

2.1 HISTORICAL BACKGROUND .....	6
2.1.1 HISTORY OF TEMPERATURE AND ITS MEASUREMENT .....	6-8
2.1.2 DATA LOGGER .....	8-9
2.2 THEORITICAL BACKGROUND .....	9
2.2.1 TEMPERATURE SENSORS .....	9-10
2.2.2 ANALOG-TO- DIGITAL CONVERSION .....	10
2.2.3 MICROCONTROLLER .....	11
2.3 PREVIOUS WORKS AND MODIFICATION .....	12

### CHAPTER THREE

3.1 AN OVER VIEW OF THE DESIGN .....	13
3.2 POWER SUPPLY .....	13-15
3.3 BATTERY CHARGING SYSTEM .....	15-17
3.4 LM35 TEMPERATURE SENSOR .....	17-18

3.5 ADC0804 ANALOG TO DIGITAL CONVERTER	.....	18-20
3.6 SYSTEM CONTROL	.....	20-24
3.7 DS1307 REAL TIME CLOCK CHIP ( RTCC)	.....	24-25
3.8 24C32( EEPROM)	.....	25-26
3.9 LOGIC LEVEL TRANSLATOR	.....	26-27
3.10 PC-RESIDENT TERMINAL SOFTWARE	.....	27-28
3.11 APPARATUS/DEVICES USED IN CONSTRUCTION	.....	28

**CHAPTER FOUR**

4.1 TEST	.....	29-30
4.2 RESULT AND DISCUSSION OF RESULT	.....	30-31

**CHAPTER FIVE**

5.1 CONCLUSION	.....	33
5.2 LIMITATIONS	.....	33
5.3 PROBLEMS ENCOUNTERED	.....	33
5.4 POSSIBLE IMPROVEMENT ON THE PROJECT	.....	33
REFERENCE	.....	34
APPENDICES	.....	35

# CHAPTER ONE

## 1.1 INTRODUCTION

Temperature has effect on virtually every aspect of life, it impacts the physical, chemical, and biological world in numerous ways. The engineering field is not an exception as the efficiency and function ability of generating plants, machines, metals, solid state devices and other mechanical, and agricultural processes etc are dependent on it in one way or the other.

Furthermore, the rating of an electronic or electrical device depends on the capability of the device to dissipate heat. As miniaturization continues, Engineers are more concerned about heat dissipation and change in properties of the device and its material make up with respect to temperature[1]. Hence a TEMPERATURE DATA LOGGER is required for the purpose of monitoring temperature at specific intervals, to foster means of activating a temperature control mechanism to keep the various processes equipments etc that are temperature dependent within their optimal temperature for maximum efficiency.

Data logging is in its simplest term, the procurement of information in order to learn more about a process or system. It forms the basis of an understanding of diverse range of systems. The logging and saving of information provides for increased knowledge and sometimes improved management of how and why different processes work. To begin this process, a DATA LOGGER is required[2] .

A data logger is an electronic device that records data over time or in relation to location either with a built-in instrument or sensor or via external instruments and sensors. Increasingly, but not entirely, they are based on a digital processor (or computer). They generally are small, battery powered, portable, and equipped with a microprocessor,

internal memory for data storage, and sensors. Some data loggers interface with a personal computer and utilize software to activate the data logger and view and analyze the collected data, while others have a local interface device (keypad, LCD) and can be used as a stand-alone device[3] .

One of the primary benefits of using data loggers is the ability to automatically collect data on a 24-hour basis. Upon activation, data loggers are typically deployed and left unattended to measure and record information for the duration of the monitoring period. This allows for a comprehensive, accurate picture of the environmental conditions being monitored, such as TEMPERATURE and relative humidity[3]. Furthermore, given the extended recording times of data loggers, they typically feature a time- and date-stamping mechanism to ensure that each recorded data value is associated with a date and time of acquisition.

## **1.2 AIMS AND OBJECTIVES**

1. To fabricate a device that can archive temperature measurements for future use.
2. To establish an effective mechanism for temperature monitoring, so as to foster means of activating a temperature control system.
3. To produce a device that will provide for increased knowledge of the temperature of an environment, equipment and processes for improved management and to enhance the function ability of the processes.
4. To provide a portable, accurate and cost effective device for temperature data acquisition.
5. To reduce the frequency of visit to weather stations to record temperature data.

6. To produce a device that will enhance the efficiency of temperature data collection.

### **1.3 METHODOLOGY**

The temperature data logger consists of a sensor, analog-to-digital converter, microcontroller, internal memory and system interface.

The temperature data logger works with the sensor to convert the temperature which is a physical quantity into electronic signals such as voltage or current. These electronic signals are then converted or digitized into binary data. The binary data is then easily analyzed by the microcontroller and stored on internal memory of the temperature data logger. The stored data is then downloaded to a computer through the system interface. The block diagram of the digital temperature data logger is shown in fig 1.0 below.

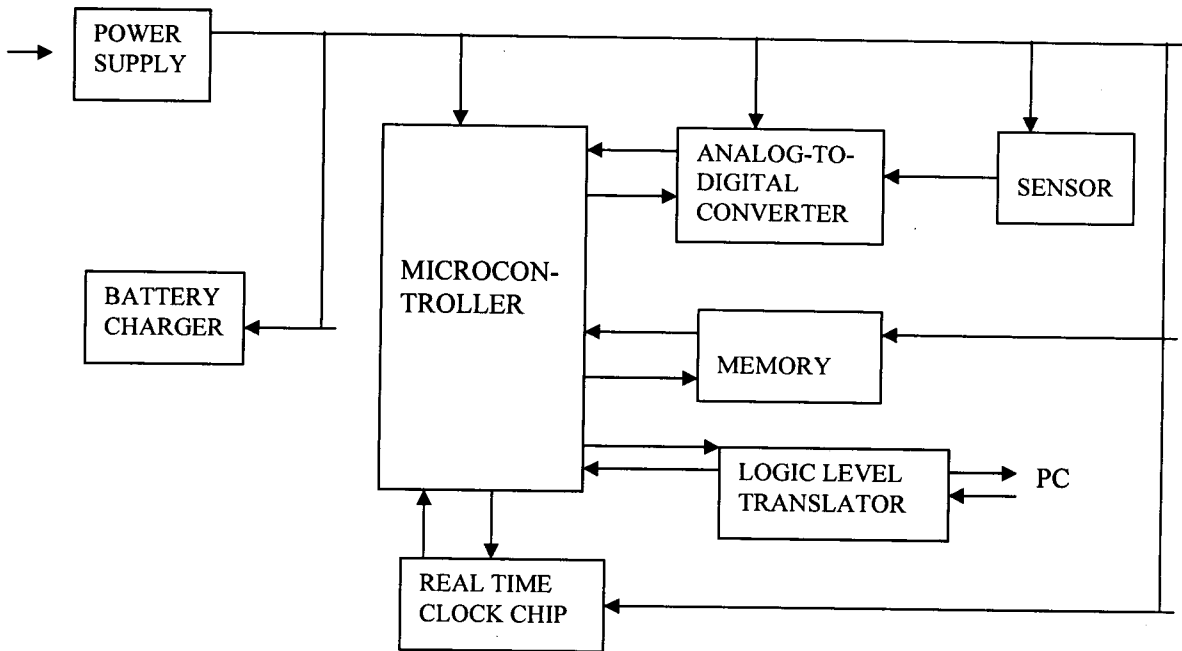


Fig 1.1 Block Diagram Of Digital Temperature Data Logger.

## 1.4 APPLICATION OF TEMPERATURE DATA LOGGER

The ability to take sensor measurement and store the processed data is a characteristic of data logger. However, a data logging application rarely requires only data acquisition and storage. Inevitably, the ability of the user to analyze and present the data to determine results and decisions based on the logged data is required. Temperature data logger finds its application in the following:

1. Weather stations.
2. Shipment companies, for monitoring temperature during shipment of produce, meat, dairy, vaccines foods pharmaceuticals or adhesives etc to ascertain if they should be accepted or not.
3. Food processing and Storage industries, to monitor the temperature of temperature-sensitive commodities to ensure that they are being stored or processed at their optimal temperature for freshness or efficacy.

4. Pharmaceutical and Life Science Industries.
5. Medical Laboratories, to monitor temperature when carrying out analysis and test.
6. Poultry Farms.
7. Hospitals, to monitor temperature of incubators where premature babies are kept.
8. Performance testing of temperature resistant paints.
9. Performance testing of air conditioning equipments.
10. Recording of temperature for instrumentation and machinery.
11. Manufacturing of oven, for testing ovens to ensure proper temperature gradient throughout the chambers.

## **1.5 SCOPE OF WORK**

The processes employed in the actualization of this project from the conception stage to the conclusive stage are thus highlighted:

Chapter one discusses what the project is all about, including the aims and objectives, narrates the methodology applied in achieving the project as well as the applications of the project.

Chapter two discusses the historical background, theoretical background, modification done on the project with respect to previous works etc.

Chapter three contains details of the design and implementation of the project with each module carefully drawn and explained.

Chapter four contains the tests, results and discussion of the results etc.

Chapter five gives the summary ( i.e. conclusions ) of the entire project work etc.

## **CHAPTER TWO**

### **LITERATURE REVIEW**

#### **2.1 HISTORICAL BACKGROUND**

##### **2.1.1 HISTORY OF TEMPERATURE AND ITS MEASUREMENT**

Intuitively, people have known about temperature for a long time: fire is hot and snow is cold. Greater knowledge was gained as man attempted to work with metals through the bronze and iron ages. Some of the technological processes required a degree of control over temperature, but to control temperature you need to be able to measure what you are controlling [4].

Until about 260 years ago temperature measurement was very subjective. For hot metals the colour of the glow was a good indicator. For intermediate temperatures, the impact on various materials could be determined. In other words a number of fixed points could be defined, but there was no scale or any way to measure the temperature between these points. It is, however possible that there is a gap in the recorded history of technology in this regard as it is difficult to believe that the Egyptians, Assyrians, Greeks, Romans or Chinese did not measure temperatures in some way. Galileo invented the first documented thermometer in about 1592. This type of thermometer is sensitive, but is affected by changes in atmospheric pressure [4].

By the early 18th century, as many as 35 different temperature scales had been devised. In 1714, Daniel Gabriel Fahrenheit invented both the mercury and the alcohol thermometer. Although the mercury thermometer is not as sensitive as the air thermometer. Mercury freezes at  $-39^{\circ}$  Celsius, so it cannot be used to measure temperature below this point. Alcohol, on the other hand, freezes at  $-113^{\circ}$  Celsius, allowing much lower temperatures to be measured. At the time, thermometers were



calibrated between the freezing point of salted water and the human body temperature. Also, Anders Celsius chose to use one hundred degrees as the freezing point and zero degrees as the boiling point of water. Sensibly the scale was later reversed and the Centigrade scale was born [4] .

The early 1800's were very productive in the area of temperature measurement and understanding. William Thomson (later Lord Kelvin) postulated the existence of an absolute zero. In 1821 T J Seebeck discovered that a current could be produced by unequally heating two junctions of two dissimilar metals, the thermocouple effect. Also, Sir Humphrey Davy discovered that all metals have a positive temperature coefficient of resistance and that platinum could be used as an excellent temperature detector (RTD). These two discoveries marked the beginning of serious electrical sensors[4] .

Gradually the scientific community learnt how to measure temperature with greater precision. For example it was realized by Thomas Stevenson that air temperature measurement needed to occur in a space shielded from the sun's radiation and rain. For this purpose he developed what is now known as the Stevenson Screen [4].

The late 19th century saw the introduction of bimetallic temperature sensor. These thermometers contain no liquid but operate on the principle of unequal expansion between two metals. Although not as accurate as liquid in glass thermometers, Bimetallic Thermometer are more hardy, easy to read and have a wider span, making them ideal for many industrial applications[4] .

The 20th century has seen the discovery of semiconductor devices, such as: the thermistor, the integrated circuit sensor, a range of non-contact sensors and also fibre-optic temperature sensors. Also, Lord Kelvin was finally rewarded for his early work in temperature measurement. The increments of the Kelvin scale were changed from degrees to Kelvin. Now we no longer say "one-hundred degrees Kelvin;" we instead say "one-

hundred Kelvin". The "Centigrade" scale was changed to the "Celsius" scale, in honour of Anders Celsius. The 20th century also saw the refinement of the temperature scale. Temperatures can now be measured to within about 0.001°C over a wide range, although it is not a simple task. The most recent change occurred with the updating of the International Temperature Scale in 1990 to the International Temperature Scale of 1990 (ITS-90). This document also covers the recent history of temperature standards [4].

### **2.1.2 DATA LOGGER**

The terms data logging and data acquisition are often used interchangeably. However, in a historical context they are quite different. A data logger is a data acquisition system, but a data acquisition system is not necessarily a data logger. Data loggers typically have slower sample rates. A maximum sample rate of 1 Hz may be considered to be very fast for a data logger, yet very slow for a typical data acquisition system. Also, Data loggers are implicitly stand-alone devices, while typical data acquisition system must remain tethered to a computer to acquire data [3].

Data acquisition and logging system is a practice that has been in existence for a long time even right from the prehistoric era. This reflected in the invention of merchets, the oldest known astronomical tool by the Egyptians around 600BC. A pair of merchets was used to establish a north-south line (or meridian) by aligning them the pole stand. They could then be used to mark off night time hours by determining when certain other stars crossed the meridian[5].

Over the years there has been an evolution in data logging and the type of loggers that are used. In the past, the equipment was bulky and mechanical, using huge paper chart recorders. Now, sophisticated computers and microprocessors retrieve the information in far more detail than could have been processed previously.

Loggers are used in everyday life unknowingly by you the public. The next time that you are in a supermarket and hand over your credit card or store card, a data logging device may track your spending movements by the store. It can assess which items you have bought, how many times a month you buy them and even, how many times you use the store. In today's society nearly all information from the weather to our shopping habits ends up in a data logger. The information is archived and saved for use at a later date[2] .

## **2.2 THEORETICAL BACKGROUND**

### **2.2.1 TEMPERATURE SENSORS**

A sensor is a device that measures a physical quantity and converts it into a signal which can be read by an observer or by an instrument. For example, A thermocouple converts temperature to an output voltage which can be read by a voltmeter. For accuracy, all sensors need to be calibrated against known standards[6]. Sensors are used in everyday objects such as touch-sensitive elevator buttons and lamps which dim or brighten by touching the base. There are also innumerable applications for sensors of which most people are never aware. Applications include cars, machines, aerospace, medicine, manufacturing and robotics etc.

A sensor's sensitivity indicates how much the sensor's output changes when the measured quantity changes. Sensors that measure very small changes must have very high sensitivities. Sensors need to be designed to have a small effect on what is measured, making the sensor smaller often improves this and may introduce other advantages. Technological progress allows more and more sensors to be manufactured on a microscopic scale as micro sensors using MEMS technology. In most cases, a micro

sensor reaches a significantly higher speed and sensitivity compared with macroscopic approaches[6] .

Temperature sensor or transducer is a device that senses temperature variation in an environment to give useful electrical signal[7]. Its properties changes with change in temperature. Some temperature sensors in use today are thermocouples, thermistors, resistance temperature detector (RTD) and sensor integrated circuits.

Descriptively, a thermocouple consists of two different conductors coupled together at their ends. As it senses temperature, the thermoelectric voltage developed between the two junctions is proportional to the temperature. But a thermistor is a device whose resistance value changes with its temperature [8]. It offers greater accuracy and stability than thermocouple [9], but its non-uniform resistance temperature characteristics can be disadvantageous in some application where it is required to obtain a more linear variation [10].

However, the integrated circuit temperature sensor (LM35) a precision semiconductor giving an output of 10mV per degree centigrade. Unlike devices with outputs proportional to the absolute temperature (in degree Kevin), there is no longer offset voltage which in most application will have to be removed. It does not require any external calibration.

### **2.2.2 ANALOG-TO-DIGITAL CONVERSION**

Analog-to-digital conversion is the complementary process of converting a continuous range of analog signals into digital codes. Such conversion process are necessary to interface real-world systems, which typically monitor continuously varying analog signals, with digital systems that process, store, interpret and manipulate the analog values[11].

### 2.2.3 MICROCONTROLLER

A microcontroller (also microcontroller unit, MCU or  $\mu\text{C}$ ) is a small computer on a single integrated circuit consisting of a relatively simple CPU combined with support functions such as a crystal oscillator, timers, serial and analog I/O etc. Program memory in the form of NOR flash or OTP ROM is also often included on chip, as well as a, typically small, read/write memory [12].

Considering the AT89S51 microcontroller, this is a low power, high performance CMOS-8bit microcomputer with 4Kbytes of flash programmable and erasable read only memory(PEROM). The device is manufactured by Atmel's high density non-volatile memory technology and is compatible with the industry standard MCS-51TM instruction set. The on-chip flash allows the programmed memory to be reprogrammed in system or by a convectional non-volatile memory programmer, by combining a versatile 8-bit central processing unit (CPU) with flash on a monolithic chip. The 89S51 is a powerful microcomputer which provides a high flexible and cost effective solution to many embedded control [13].

The 89S51 posses the following standard features: 4Kbytes of flash, 128bytes of RAM, 32 I/O lines, three 16-bit timers, five vector two-level interrupt architecture, a full duplex serial port on-chip oscillator and circuitry. Also, it is designed with static logic for operation down to zero frequency and supports two software selectable power saving mode. The idle mode stops the CPU while allowing the RAM, timer, serial port and interrupt system to continue functioning. The power down mode saves the RAM contents but freezes the oscillator, disabling all other chip function until the next hardware reset [13].

## **2.3 PREVIOUS WORKS AND MODIFICATION**

In previous designs related to this topic, specifically by Oitolaiye David A (design and construction of temperature logging and control device) in year 2007 and Sule Ezekiel Andrew (design and construction of a microcontroller-based temperature data acquisition and logging system) in year 2008, both of this department. In their designs, the logged data are logged and stored directly inside the memory of a computer, which is to say the devices does not have internal memory of its own. Also, there was no provision for battery charger for charging the backup battery.

However, this design (design and construction of a digital temperature data logger), the device has its own internal memory for storage of the logged data, which can later be interfaced with the computer and then saved to the computer's memory. Also, the design made provision for battery charger for the sake of charging the backup battery.

## CHAPTER THREE

### DESIGN AND CONSTRUCTION OF TEMPERATURE DATA

#### LOGGER

#### 3.1 AN OVERVIEW OF THE DESIGN

The digital temperature data logger was designed around the following subsystems:

1. Power Supply
2. An LM35 Temperature Sensor
3. Analog- to Digital Converter
4. 8- bit Microcontoller
5. DS1307 Real Time Clock Chip
6. 4KB EEPROM( 24C32)
7. Logic Level Translator
8. PC- Resident Terminal Software

#### 3.2 POWER SUPPLY

A dual-source power supply was used:

1. A mains-derived supply.
2. A battery source.

The mains-derived supply was obtained from a 15V2A step down transformer and a bridge rectifier. The power to the battery charging subsystem was directly derived from this source.

The 15VAC voltage was converted into a DC voltage of amplitude given by the relation:

$$V_{DC} = V_{Rms} \sqrt{2} - 1.4 \dots \dots \dots (3.1)$$

The system power supply is shown in fig 3.1 below:

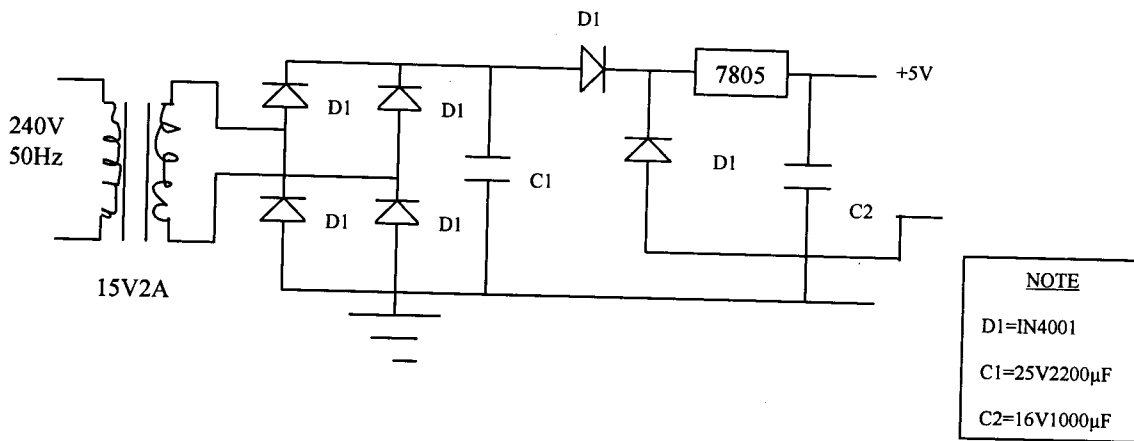


Fig 3.1 System power supply

### 3.3 BATTERY CHARGING SYSTEM

The battery charger was designed around a 3-terminal adjustable voltage regulator -LM317. To match the charging characteristics of the lead- acid battery, a constant-current, constant- voltage charging scheme was implemented.

In this charging algorithm, the battery is charged at a fixed current until the terminal voltage attains a preset maximum after which charging is discontinued, and the battery held at a float voltage. The charging current was set by a resistance calculated from the expression:

$$R = V/I = 1.25/I \dots\dots\dots (3.6)$$

1.25 = Internal reference voltage on the LM317.

I = output current = charging current.

The charger subsystem was designed to handle a series connection of two 6V, 4.5AH cell.

The rule of thumb concerning battery charging postulates that the maximum charging current should be less than or equal to  $Q/5A$ , and the minimum be  $Q/20A$ .

For a 4.5AH battery, this translates into:

$$I_{max} = 4.5/5 = 0.9A \dots\dots\dots (3.7)$$



$$I_{min} = 4.5/20 = 0.225A \dots\dots\dots(3.8)$$

The battery was rapid charged at the maximum specified current of 1.2A to reduce the charging period. The battery terminal voltage was fixed at 13.8V by a zener diode. The circuit diagram of the battery charger is shown in fig 3.2 below:

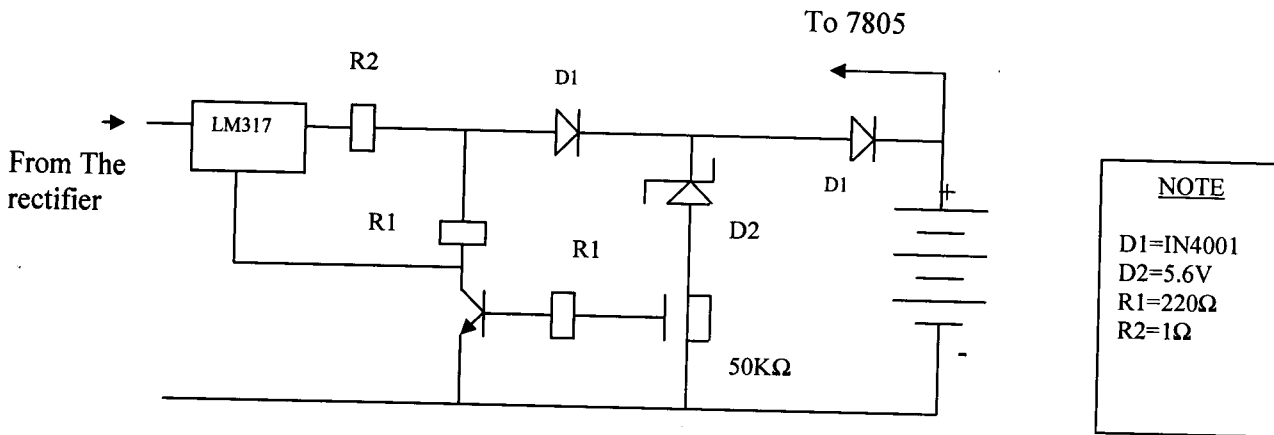


Fig 3.2 Battery Charger System

The charger also incorporated a shutdown feature to power down the LM317 regulator when the maximum terminal voltage is attained. The shutdown function was realized using a C9014 transistor as shown in fig 3.1.

The regulator shutdown occur when the 13.8V terminal voltage is attained, the 50KΩ was adjusted to provide the required output DC voltage across the charger terminals with the battery disconnected.

The terminal voltage was set by the relative resistances on either side of the 50KΩ resistance as shown by the following equations:

At  $V_{batt(max)}$ , the transistor has base voltage of about 0.7V between the bases – emitter junction. The base voltage is determined by the resistances on either sides of the 50KΩ potentiometer as shown in fig 3.3 below:

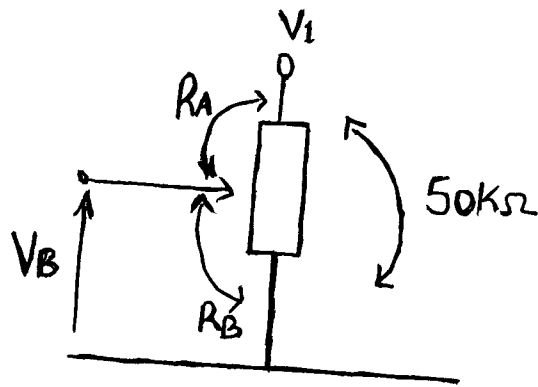


Fig 3.3 Analysis of The Voltage setting resistances.

$$0.7 = (V_1 * R_B) / (R_A + R_B) \dots\dots\dots(3.9)$$

$$0.7 = (V_1 * R_B) / 50000 \dots\dots\dots(3.1.1)$$

$$V_1 R_B = 35000 \dots\dots\dots(3.1.2)$$

$$V_1 = \text{voltage at the upper terminal of the potentiometer} = V_{\text{batt(max)}} - V_2 \dots\dots(3.1.3)$$

$$= V_{\text{batt(max)}} - 5.6V \dots\dots\dots(3.1.4)$$

Merging the equations (1) and (2),

$$[ V_{\text{batt(max)}} - 5.6V ] R_B = V_{BE} * 50000 \dots\dots\dots(3.1.5)$$

$$V_{\text{batt(max)}} - 5.6V = 50000V_{BE} / R_B \dots\dots\dots(3.1.6)$$

$$V_{\text{batt(max)}} = [(50000V_{BE} / R_B) + 5.6]V \dots\dots\dots(3.1.7)$$

Thus, the maximum battery terminal voltage is directly influenced by the transistor's

$$V_{BE}(= 0.7V), \text{ and } R_B = 50000 - R_A \dots\dots\dots(3.1.8)$$

### 3.4 LM35 TEMPERATURE SENSOR

Since the temperature to be measured is a non-electrical quantity, a transducer was required to convert it into an electrical quantity. For better accuracy and sensitivity, an integrated circuit temperature sensor ( LM35 ) was used. The LM35 has an operational

range of 0°C – 100°C, with an output voltage related to the ambient temperature by the expression:

$$V_{out} = [ T^{\circ}C * 0.01 ] V \dots\dots\dots(3.1.9)$$

The output voltage changes by 10mV for a degree change in temperature, fig 3.4 shows the sensor circuit. The LM35 has the following specifications as shown in the appendix 1

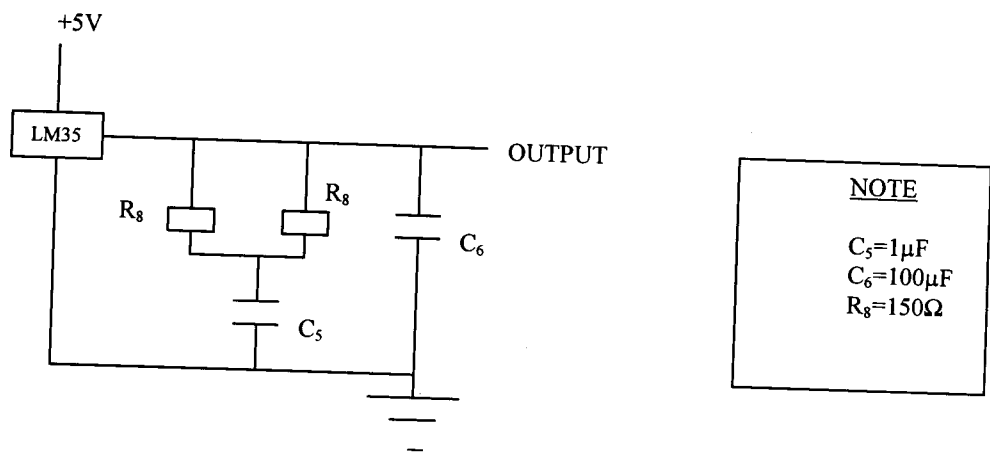


Fig 3.4 Sensor Circuit

The sensor was interfaced directly with an 8 - bit Analog - to Digital Converter ( ADC ) that translated the analog output to a digital value that can processed by the microcontroller.

### 3.5 ADC0804 ANALOG – TO –DIGITAL CONVERTER

For conversion of the analog temperature reading to its digital equivalent, an analog – to – digital converter was required. An 8 – bit device was used, the ADC0804 was used. It operates on a supply voltage range of 4.5V to 6.5V and in this case, a voltage of 5V was chosen, since about the same voltage of 5V is required by the microcontroller,

ADC0804, LM35 etc. The top view and pin description of the ADC0804 are shown in appendix 2 respectively. The device was run off a clock source given by the expression:

$$f_{\text{clock}} = 1 / 1.1RC \dots\dots\dots (3.2.1)$$

$$R = 10K\Omega \dots\dots\dots (3.2.2)$$

$$C = 150\text{Pf} \dots\dots\dots (3.2.3)$$

$$f_{\text{clock}} = [ 1 / (1.1 * 10^4 * 1.5 * 10^2 * 10^{-12}) ] \text{ Hz} \dots\dots\dots (3.2.4)$$

It was interfaced with the microcontroller over P1 as shown in fig 3.5. The device was setup for a 1-bit change at the output for a 10mV input change by making  $V_{\text{ref}} = 1.28\text{V}$ . The span voltage was thus 2.56V.

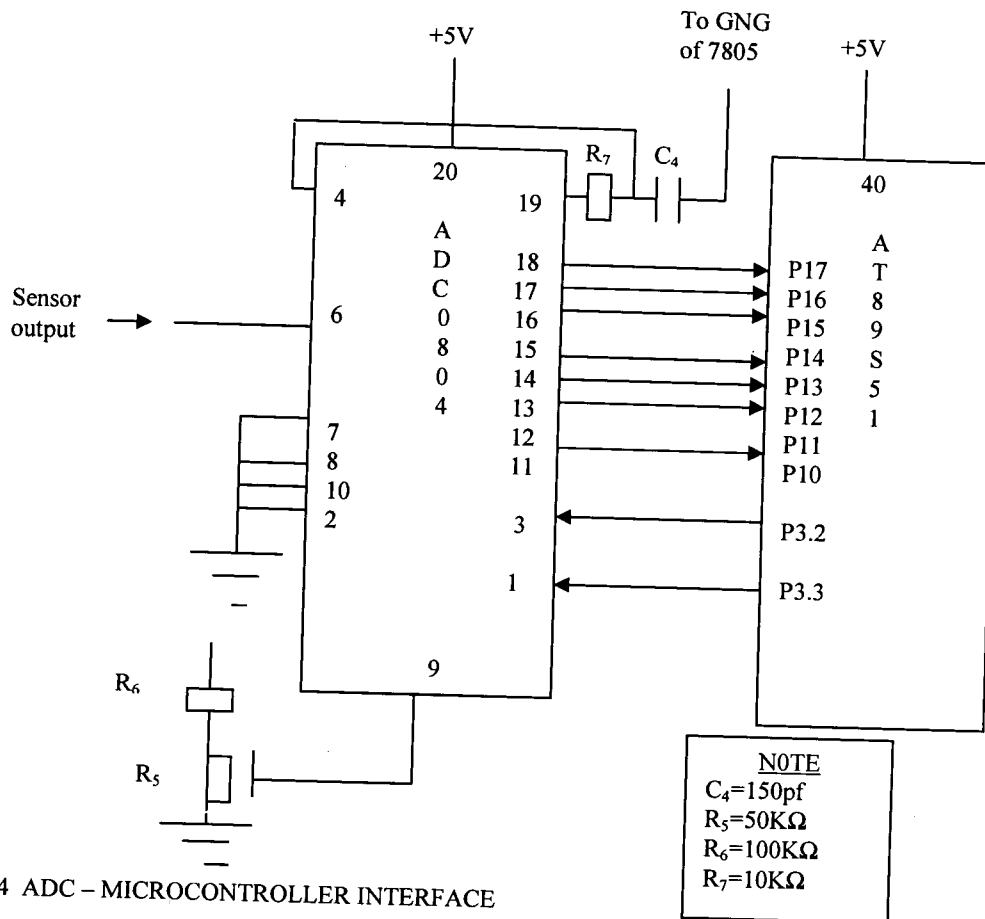


FIG 3.4 ADC – MICROCONTROLLER INTERFACE

Conversion was initiated by strobing WR (3) low, with CS (1) asserted, then high conversion is perfected in about 100 $\mu$ s after which the data can be read. The converted data is processed and stored in the 24C32 EEPROM device attached to the I<sup>2</sup>C bus on P2.0 and P21.

### 3.6 SYSTEM CONTROLLER

An 8 – bit microcontroller was embedded in the system realization. A low power device AT89S51 microcontroller was used, its pin configuration and description are shown in appendix 3.

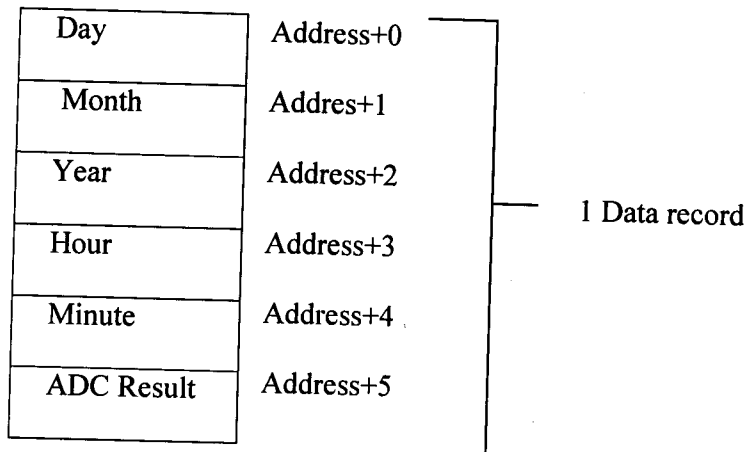
The device was configured for a serial port connection at 9600bps for data upload and download. A crystal frequency of 11.05912MHz was used. The microcontroller was interfaced with the ADC over port1(P1), the logic level translator over P3.0 and P3.1, the real time clock chip ( RTTC ) and 24C32 memory over P2.0 / P21, port 3(P3) pins 4, 5, 6 were attached to three LED indicators ( GREEN, ORANGE AND RED ).The system software was modulated for ease of maintainability and debugging.

At power-up, the software initializes system's variables and the serial port. During this phase, a check of the DS1307 Real Time Clock Chip is made. A signature message is read from six RAM locations on the Real Time Clock Chip IC, the signature byte matches " DS1307 ", it is assured that the timer has not been powered down since last read, otherwise the system performs an initialization of the Real Time Clock Chip by resetting the Time / Date information to 01 / 01 /09 00:00. The RTTC was also configured for 1Hz generation of pin1 by resetting the oscillator control to bits in register address 07h.

The 1Hz output was converted to the P3.3 ( INT1) input which generates a 1Hz periodic interrupt . These interrupts are converted in software to get the sampling

intervals. The software also handles temperature reading storage for every conversion. Four samples are taken each hour, i.e. a sample is taken every 15 minutes. The digitized reading is stored alongside the Time /Date information as a sequential record in the 24C32 device as shown in table 3.5 below:

Table 3.1 Data Record Implemented on The 24C32 Device.



Since each sampling occupies 6 bytes, and the 24C32 device has addressable locations only up to 4096, then maximum number of samples possible before a memory full condition is:

$$4096 / 6 = 682 \text{ mod } 4$$

A 7-day data storage mechanism was implemented in which, assuming the memory is empty (erased), at 4 samples per hour, 96 samples are taken per day. The 96 samples are stored as uniquely identifiable records in  $96 * 6 = 576$  memory locations. The number of days before the memory is exhausted is thus:

$$4096 / 576 = 7 \text{ R } 64$$

Two bytes were used for storing the R-byte pointer used to indicate the next memory location to be written into. The samples are stored this way until the memory is

exhausted. A memory full condition is indicated by the RED LED flashing at 1Hz. When data is read from the device, the memory is re-initialized.

To store data into the EEPROM and access the calendar chip, a software I<sup>2</sup>C simulation was implemented as the generic 8051 devices have two hardware I<sup>2</sup>C bus. The EEPROM and RTTC were both placed on the I<sup>2</sup>C bus made on P2.0 and P2.1 as shown in Fig 3.6:

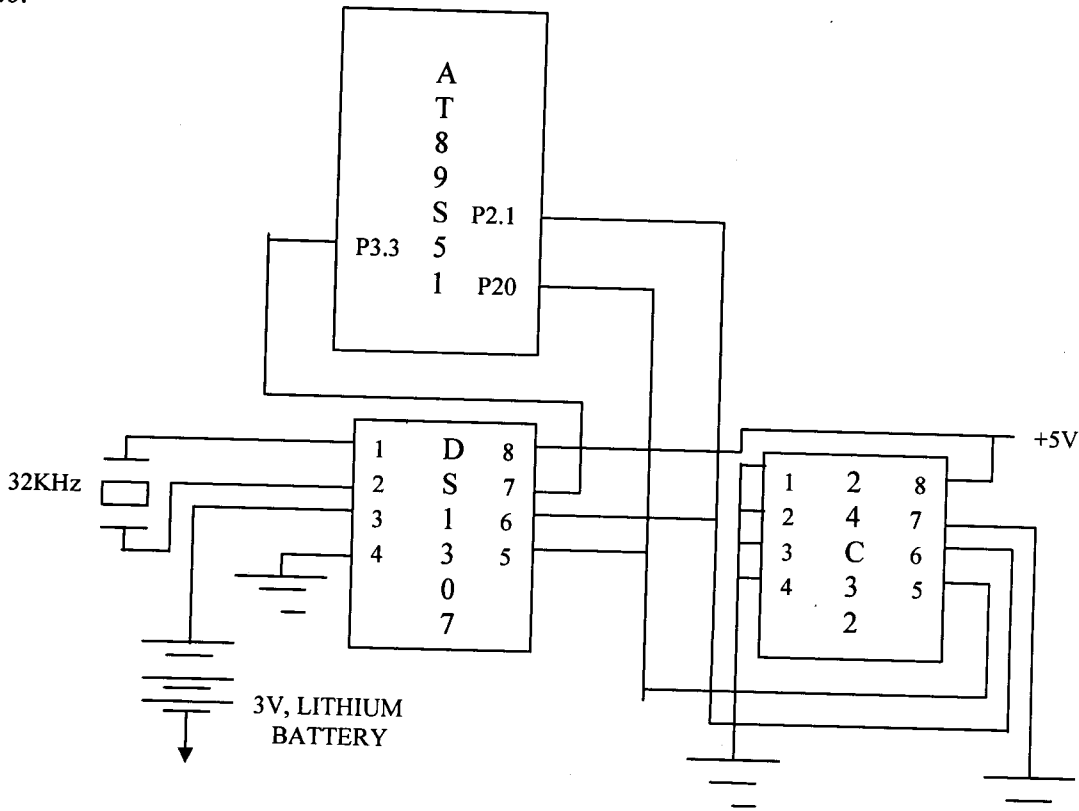


Fig 3.6 DS1307 / 24C32 – microcontroller interface.

Communication with the two devices was implemented serially over the bit-banged interface. P2.0 was designated SDA and P2.1 SCL. Bit-level manipulation was effected in converting the byte-wide data from the microcontroller to serial data needed in the implementation of Philips I<sup>2</sup>C bus specifications, and vice versa. The system software was also coded to effect data transfer to and from any connected PC over its

serial port. The processes executable via the visual basic routines resident on the PC include

1. Time / Date Set.
2. Memory Clear.
3. Memory Dump.

The high level communication interface was effected via a command-response handshaking protocol. Command sent from the terminal machine are executed, and the result of the command sent back to the High Level Language. Messages are then posted on the Graphic User Interface (GUI), notifying the user of the state of the LOGGER.

For easier debugging, three LEDs were provided on the unit. Green LED – system normal and operational, Orange LED – RTCC error / memory error and Red LED – memory full. Fig 3.7 shows the status indicator circuit.

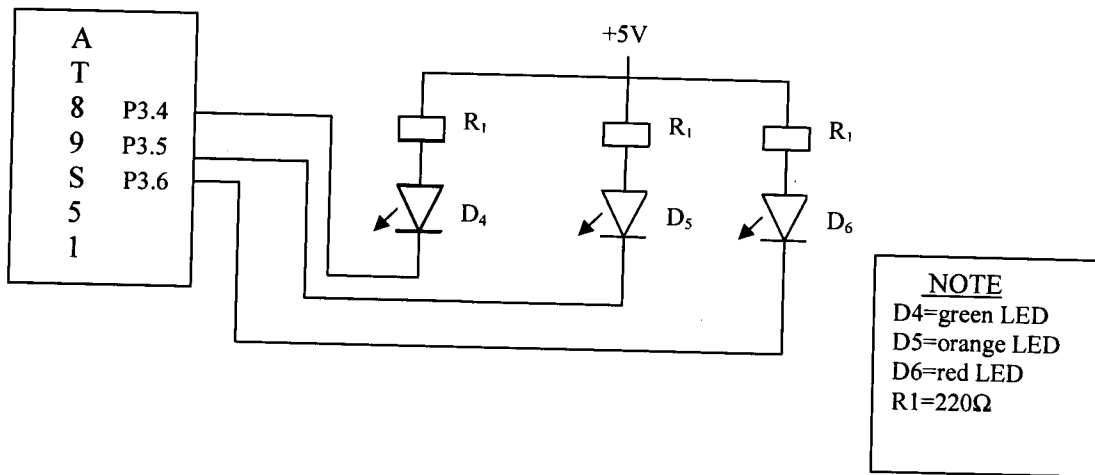


Fig 3.7 LED Status Indicators.

The LEDs were driven from the +5V supply via current limiting resistances. The values which were evaluated using:

$$R_S = (V_S - V_{LED}) / I_{LED} \dots \dots \dots (3.2.5)$$

$$V_S = \text{Supply Voltage} = 5V \dots \dots \dots (3.2.6)$$



$$V_{LED} = \text{LED Forward Voltage} = 1.7V \dots\dots\dots (3.2.7)$$

$I_{LED}$  = LED Forward current.

Typically, a minimum and maximum LED current of 5mA and 20mA are quoted, provided a maximum and minimum current – limiting resistances of

$$R_{max} = (5 - 1.7) / 0.05 = 3.3 / 0.05 = 660\Omega \dots\dots\dots (3.2.8)$$

$$R_{min} = (5 - 1.7) / 0.02 = 3.3 / 0.02 = 165\Omega \dots\dots\dots (3.2.9)$$

A 220Ω resistance was selected to yield a current slightly above 10mA. The Orange and Red LEDs were activated mutually exclusively, i.e. only one is turned on during an error condition.

### 3.7 DS1307 REAL TIME CLOCK CHIP (RTCC)

The DS1307 Serial Real-Time Clock is a low-power; full binary- coded decimal (BCD) clock/calendar plus 56 bytes of NV SRAM. Address and data are transferred serially via a 2-wire, bi-directional bus. The clock/calendar provides seconds, minutes, hours, day, date, month, and year information. The end of the month date is automatically adjusted for months with fewer than 31 days, including corrections for leap year. The clock operates in either the 24-hour or 12-hour format with AM/PM indicator. The DS1307 has a built-in power sense circuit that detects power failures and automatically switches to the battery supply[14].The pin assignment and function table is shown in fig 3.8 and table 3.2

Table 3.1 DS1307 function table

Name	Function
X <sub>1</sub> , X <sub>2</sub>	32.768KHz Crystal connection
V <sub>BAT</sub>	4.3V Battery Input
GND	Ground
VCC	Primary Power Supply
SDA	Serial Data
SCL	Serial Clock
SQW/OUT	Square wave/ Output Driver

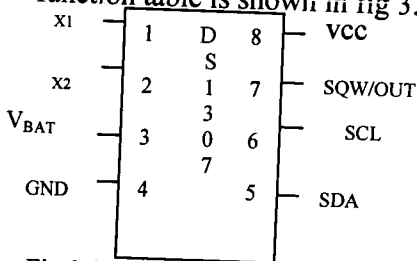


Fig 3.8 Pin Assignment of DS1307

The device was configured for operation at address 11010000. Every 15 minutes, the RTCC. Registers were read to extract the Time / Date information required for storage along with the ADC samples. The device has 56 RAM location, six of which were used for holding the signature bytes required by software to know whether the device been reset (battery power / main supply removed), or has been periodically initialized and functioning properly.

For a non-initialized device, the software default to the following settings:

Seconds: 00h

Minute : 00h

Hour : 00h

Date : 01h

Month : 01h

Year : 09h

RAM location 8 – 13: “DS1307”.

Once configured, the device generates a periodic 1Hz output on Pin7 while uploading the internal RTCC. registers as well.

### **3.8 24C32 (EEPROM).**

The Microchip Technology Inc. 24C32 is a 4K\*8 (32K-bit ) Serial Electrically Erasable PROM. This device has been developed for advanced, low power applications such as personal communications or data acquisition. The 24C32 features an input cache for fast write loads with a capacity of eight 8-byte pages, or 64bytes. It also features a 4K-bit block of ultra-high endurance memory for data that changes frequently. The 24C32 is capable of both random and sequential reads up to 32K boundary. Functional address lines allow up to eight 24C32 devices on the same bus, for up to 256K-bits address space.

Advanced CMOS technology makes this device ideal low-power non-volatile code and applications. The 24C32 is available in the standard 8-pin plastic DIP and 8-pin Surface mount SOIC package[15].The pin assignment and function table of the 24C32 shown in fig 3.9 and Table 3.3

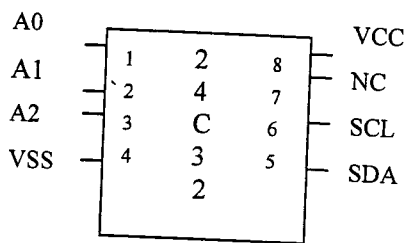


Fig 3.9 Pin Assignment of 24C32

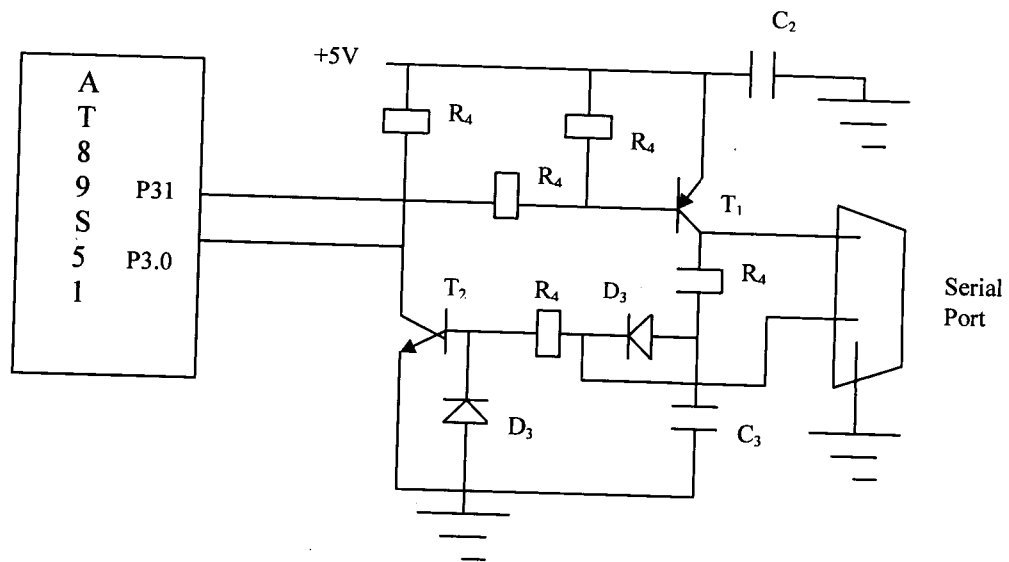
Table 3.3 function table of 24C32

Name	Function
A0, A1,A2	User Configurable Chip Select
VSS	Ground
SDA	Serial Address/ Data I/O
SCL	Serial Clock
VCC	+4.5V To 5.5V Power Supply
NC	No Connection

data applications A 24C32 device was provided for bulk storage, the device has 4096 addressable byte – wide memory locations. The device was configured at address 00h on the I<sup>2</sup>C bus by taking Pins1, 2 and 3 to ground. 682 different records sets can be stored on the part before the memory buffer indicator is activated.

### 3.9 LOGIC LEVEL TRANSLATOR

To effect communication over the serial port on the controller and the terminal host system, a logic level translator was required to convert the 0 – 5V signaling voltages to the ±3V—±12V bipolarity signaling voltages required on the motherboard. The logic level translator was effected using discrete components as shown below in fig 3.10



**NOTE**

C2=1000 $\mu$ F  
 C3=10 $\mu$ F  
 D3=IN4148  
 R4=3.3K $\Omega$   
 T1=25A1015  
 T2=C9014

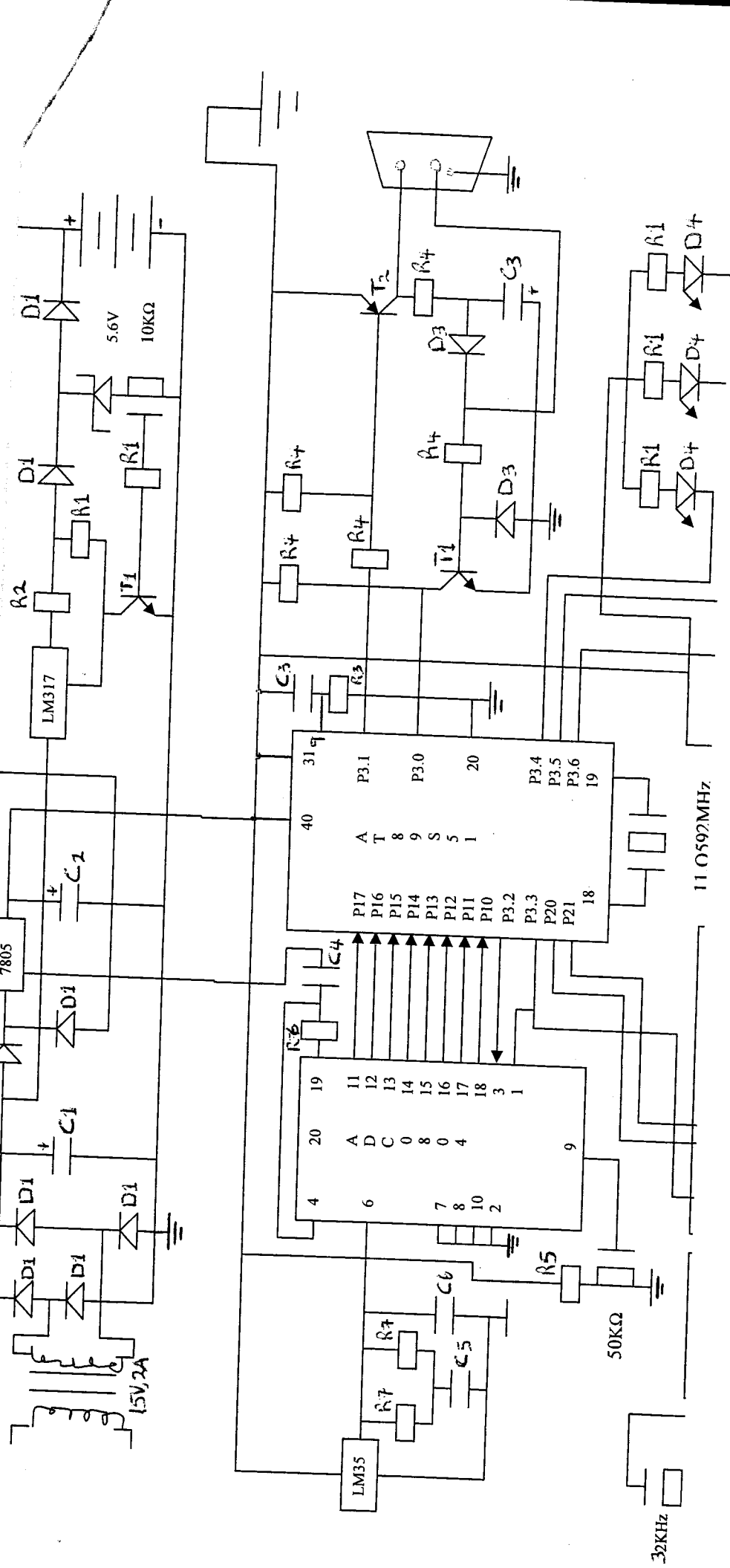
Fig 3.11 LOGIC LEVEL TRANSLATOR

The circuit was adopted from an Atmel application note.

### 3.10 PC-RESIDENT TERMINAL SOFTWARE.

For a useful utilization of the logged data, a high level language visual basic application was provided on the host system. The application enabled the following processes executed over the serial interface.

1. RTCC. Initialization (time set).
2. Memory initialization.
3. Data dumping from logger onboard memory.
4. Data storage to file.
5. Data display on screen.



The visual basic interface was configured with a 16KB buffer to hold the data inflow from the logger. Various visual messages were posted on screen reflecting the rate of the serial link. The dumped data can be saved to a file for later analysis, or displayed on a customized page on the system for visual analysis.

### **3.11 APPARATUS/ DEVICES USED IN CONSTRUCTION**

Some of the apparatus used in the construction of the device are:

1. Vero-Board
2. Soldering Iron
3. Soldering Lead
4. Lead Sucker
5. Multi-meter
6. Wire/ Jumpers
7. A Plastic Material.

## CHAPTER FOUR

### TEST, RESULTS AND DISCUSSION OF RESULT

#### 4.1 TEST

In carrying out the test of the digital temperature logger, the following materials were applied:

1. A personal computer on which the PC-resident terminal software of this device ( a high level visual basic application / program ) was provided.
2. Visual Basic Software (Visual Basic 6.0 )
3. Serial-to-serial cable ( with which the interfacing of the device to the computer was done).

At the outset, the device was connected to both battery and PHCN power supply, but the PHCN source of power was applied and then the device was positioned inside the room to log temperature of the room. Two hours later, the PHCN power supply was turned off and the device was run automatically by the back-up battery as source of it's power supply. This was done so as to test the automated power change designed for the device in the event of power outage. About an hour later, the device was then taken and interfaced with the computer so as to access the logged data.

To access the logged data, the device was connected to a personal computer via a serial-to-serial cable. But then the device was not recognized by the computer not until the PC-resident terminal software (the temperature data logger user interface) for the device was provided on the computer along with an installation of Visual Basic 6.0 (in which the PC-resident terminal software was

developed ).Having completed these processes, the device was then re-interfaced with the computer. This time the device was recognized and was accessed via the PC-resident software.

## 4.2 Result And Discussion Of Result

On opening, the user interface welcome screen displays on the computer screen as illustrated in the table 4.1 below.

Table 4.1 Digital Temperature User Interface.

<b>DIGITAL TEMPERATURE LOGGER USER INTERFACE</b>				
<b>TIME-STAMPED DIGITAL TEMPERATURE LOGGING SYSTEM</b>				
<b>TIME</b>				
HOUR	MINUTE	SECONDS	SAVE FILE	
<b>DATE</b>				
DAY	MONTH	YEAR	ABOUT	
SET TIME/DATE	CLEAR NVM	UPLOAD DATA	VIEW DATA	CLOSE

The buttons in table 4.1 are explained thus:

The time and date button allows the user to set the time , after setting, the user then clicks the “ set time/date” button to activate it. This is done again only when the CMOS battery is removed or replaced.



Whereas, the data view process is initiated by first clicking on the “upload data “ button which will prompt the microcontroller to dump data from the nonvolatile memory (24C32) to the computer . The uploaded data is then viewed by clicking on the view data button. On clicking the button, the temperature data is then displayed as shown in table 4.2 below.

Table 4.2 Temperature display.

DATE	TIME	TEMPERATURE (°C)	
21/11/09	06:55	28	<b>RELOAD</b>
21/11/09	07:10	27	<b>NEXT</b>
21/11/09	07:25	28	<b>PREVIOUS</b>
21/11/09	07:40	28	<b>CLOSE</b>
21/11/09	07:55	28	
21/11/09	08:10	28	
21/11/09	08:25	28	
21/11/09	08:40	29	
21/11/09	17:06	39	
21/11/09	17:31	36	
21/11/09	17:46	38	
21/11/09	18:03	38	

From the table 4.2 above, the “reload button” prompts the microcontroller to keep logging temperature data to memory, while the “next” and “previous” buttons allows the user to access next and previous data. The “close” button is used to close the page.

Furthermore, the uploaded temperature data can be stored to a file in the computer memory by clicking on the “save file” button. But the data will be displayed in hexadecimal values. After which the memory can then be cleared or erased by clicking on the “clear nvm” button for the logger to log in new sets of data.

## REFERENCES

1. Jerry C. Whitaker, The electronics handbook, Technical Press, Inc., Beaverton, Oregon.
2. What is-data-logging, 2009, @ <http://www.wisegeek.com>.
3. Data logger, 2009,@ <http://en.Wikipedia.org>.
4. Temperature history, 2009 @ <http://www.capgo.com>.
5. Temperature, 2009, @ <http://physics.nist.gov.com>
6. Sensor, 2009, @ <http://en.wikipedia.org>.
7. Paul Horowitz and Winfield Hill, The art of electronics, Cambridge University Press, pp988.
8. Thomas E. Newman, Electricity and electronics 1995, pp88ff.
9. Giorgio Rizzoni, Principle and application of electrical Engineering, revised 4<sup>th</sup> edition, New York Mc Graw Hills pp713-715.
10. Km Leatherman, Automatic controls for heating and air conditioning, principles and application, vol.15, Pergamon Press, pp9,15,23-25.
11. Jerry C. Whitaker, The electronic handbook, Technical Press, Inc., Beaverton, Oregon., pp723 .
12. Embedded systems dictionary by Jack Ganssle and Mike Barr, p173.
13. 89S51 microcontroller, 2009,@ <http://en.wikipedia.org>.
14. DS1307, 2009 @ <http://www.datasheetcatalog.com>.
15. 24C32, 2009, @ <http://www.datasheetcatalog.com>.

# APPENDIX 1

## Applications

The LM35 can be applied easily in the same way as other integrated-circuit temperature sensors. It can be glued or cemented to a surface and its temperature will be within about 0.01°C of the surface temperature.

This presumes that the ambient air temperature is almost the same as the surface temperature; if the air temperature were much higher or lower than the surface temperature, the actual temperature of the LM35 die would be at an intermediate temperature between the surface temperature and the air temperature. This is especially true for the TO-92 plastic package, where the copper leads are the principal thermal path to carry heat into the device, so its temperature might be closer to the air temperature than to the surface temperature.

To minimize this problem, be sure that the wiring to the LM35, as it leaves the device, is held at the same temperature as the surface of interest. The easiest way to do this is to cover up these wires with a bead of epoxy which will insure that the leads and wires are all at the same temperature as the surface, and that the LM35 die's temperature will not be affected by the air temperature.

The TO-46 metal package can also be soldered to a metal surface or pipe without damage. Of course, in that case the V- terminal of the circuit will be grounded to that metal. Alternatively, the LM35 can be mounted inside a sealed-end metal tube, and can then be dipped into a bath or screwed into a threaded hole in a tank. As with any IC, the LM35 and accompanying wiring and circuits must be kept insulated and dry, to avoid leakage and corrosion. This is especially true if the circuit may operate at cold temperatures where condensation can occur. Printed-circuit coatings and varnishes such as Humiseal and epoxy paints or dips are often used to insure that moisture cannot corrode the LM35 or its connections.

These devices are sometimes soldered to a small lightweight heat fin, to decrease the thermal time constant and speed up the response in slowly-moving air. On the other hand, a small thermal mass may be added to the sensor, to give the steadiest reading despite small deviations in the air temperature.

Temperature Rise of LM35 Due To Self-heating (Thermal Resistance)

	TO-46, no heat sink	TO-46, small heat fin*	TO-92, no heat sink	TO-92, small heat fin**	SO-8 no heat sink	SO-8 small heat fin**	TO-202 no heat sink	TO-202 *** small heat fin
Still air	400°C/W	100°C/W	180°C/W	140°C/W	220°C/W	110°C/W	85°C/W	60°C/W
Moving air	100°C/W	40°C/W	90°C/W	70°C/W	105°C/W	90°C/W	25°C/W	40°C/W
Still oil	100°C/W	40°C/W	90°C/W	70°C/W	105°C/W	90°C/W	25°C/W	40°C/W
Stirred oil	50°C/W	30°C/W	45°C/W	40°C/W				
(Clamped to metal, Infinite heat sink)	(24°C/W)			(55°C/W)			(23°C/W)	

\* Wakefield type 201, or 1" disc of 0.020" sheet brass, soldered to case, or similar.

\*\* TO-92 and SO-8 packages glued and leads soldered to 1" square of 1/16" printed circuit board with 2 oz. foil or similar.

## Typical Applications (Continued)

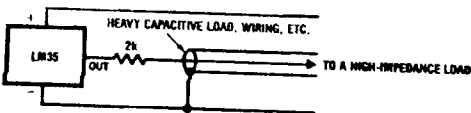


FIGURE 3. LM35 with Decoupling from Capacitive Load  
TL/H/5516-19

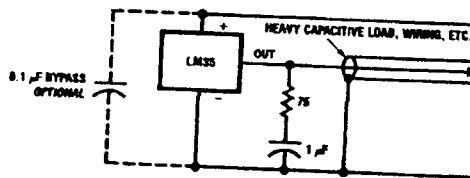


FIGURE 4. LM35 with R-C Damper  
TL/H/5516-20

### CAPACITIVE LOADS

Like most micropower circuits, the LM35 has a limited ability to drive heavy capacitive loads. The LM35 by itself is able to drive 50 pf without special precautions. If heavier loads are anticipated, it is easy to isolate or decouple the load with a resistor; see Figure 3. Or you can improve the tolerance of capacitance with a series R-C damper from output to ground; see Figure 4.

When the LM35 is applied with a 200Ω load resistor as shown in Figure 5, 6, or 8, it is relatively immune to wiring

capacitance because the capacitance forms a bypass from ground to input, not on the output. However, as with any linear circuit connected to wires in a hostile environment, its performance can be affected adversely by intense electromagnetic sources such as relays, radio transmitters, motors with arcing brushes, SCR transients, etc. as its wiring can act as a receiving antenna and its internal junctions can act as rectifiers. For best results in such cases, a bypass capacitor from  $V_{IN}$  to ground and a series R-C damper such as 75Ω in series with 0.2 or 1 μF from output to ground are often useful. These are shown in Figures 13, 14, and 16.

# APPENDIX 2



December 1994

ADC0801/ADC0802/ADC0803/ADC0804/ADC0805  
8-Bit  $\mu$ P Compatible A/D Converters

## ADC0801/ADC0802/ADC0803/ADC0804/ADC0805 8-Bit $\mu$ P Compatible A/D Converters

### General Description

The ADC0801, ADC0802, ADC0803, ADC0804 and ADC0805 are CMOS 8-bit successive approximation A/D converters that use a differential potentiometric ladder—similar to the 256R products. These converters are designed to allow operation with the NSC800 and INS8080A derivative control bus with TRI-STATE® output latches directly driving the data bus. These A/Ds appear like memory locations or I/O ports to the microprocessor and no interfacing logic is needed.

Differential analog voltage inputs allow increasing the common-mode rejection and offsetting the analog zero input voltage value. In addition, the voltage reference input can be adjusted to allow encoding any smaller analog voltage span to the full 8 bits of resolution.

- Differential analog voltage inputs
- Logic inputs and outputs meet both MOS and TTL voltage level specifications
- Works with 2.5V (LM336) voltage reference
- On-chip clock generator
- 0V to 5V analog input voltage range with single 5V supply
- No zero adjust required
- 0.3" standard width 20-pin DIP package
- 20-pin molded chip carrier or small outline package
- Operates ratiometrically or with 5 V<sub>DC</sub>, 2.5 V<sub>DC</sub>, or analog span adjusted voltage reference

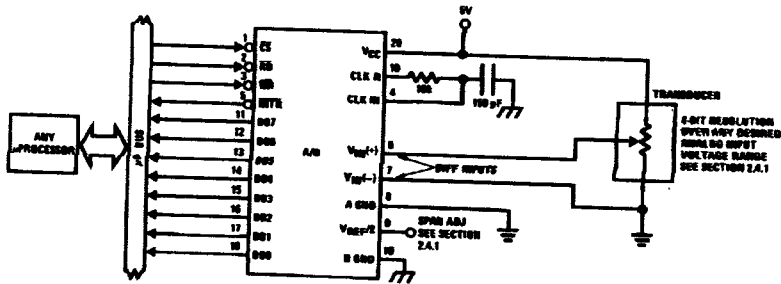
### Features

- Compatible with 8080  $\mu$ P derivatives—no interfacing logic needed - access time - 135 ns
- Easy interface to all microprocessors, or operates "stand alone"

### Key Specifications

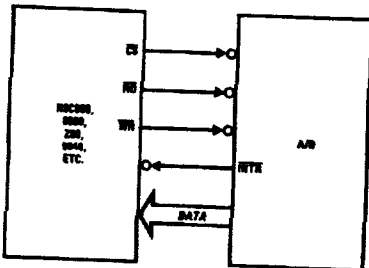
- Resolution 8 bits
- Total error  $\pm 1/4$  LSB,  $\pm 1/2$  LSB and  $\pm 1$  LSB
- Conversion time 100  $\mu$ s

### Typical Applications



TL/H/5671-1

8080 interface



TL/H/5671-21

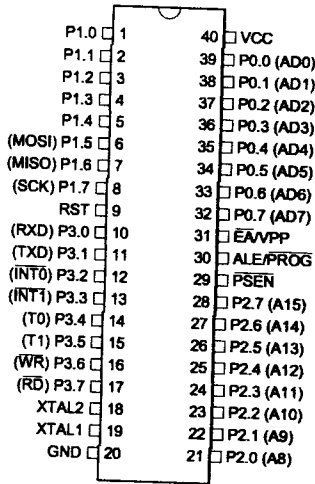
Error Specification (Includes Full-Scale, Zero Error, and Non-Linearity)

Part Number	Full-Scale Adjusted	V <sub>REF</sub> /2 = 2.500 VDC (No Adjustments)	V <sub>REF</sub> /2 = No Connection (No Adjustments)
ADC0801	$\pm 1/4$ LSB		
ADC0802		$\pm 1/2$ LSB	
ADC0803	$\pm 1/2$ LSB		
ADC0804		$\pm 1$ LSB	
ADC0805			$\pm 1$ LSB

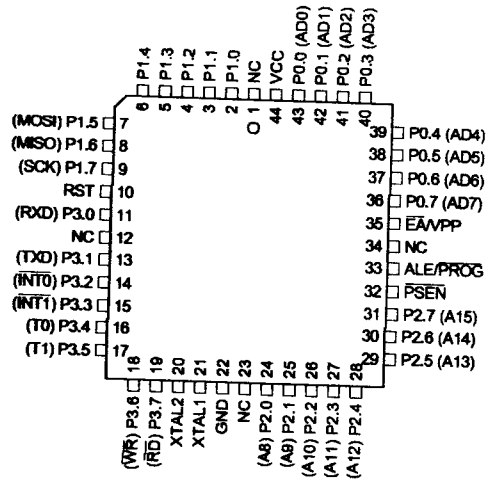
TRI-STATE® is a registered trademark of National Semiconductor Corp.  
Z-80® is a registered trademark of Zilog Corp.

**Pin Configurations**

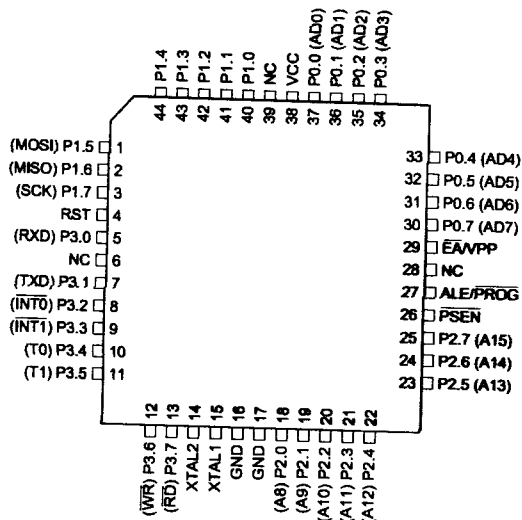
**PDIP**



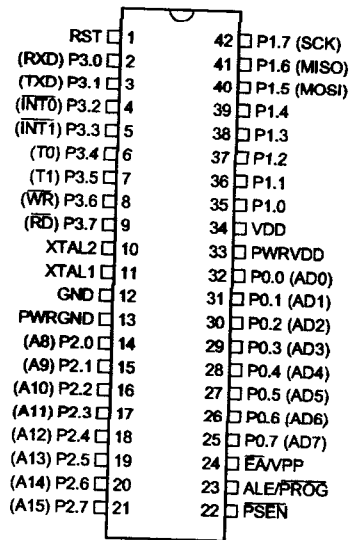
**PLCC**



**TQFP**



**PDIP**





## In Description

CC

Supply voltage (all packages except 42-PDIP).

ND

Ground (all packages except 42-PDIP; for 42-PDIP GND connects only the logic core and the embedded program memory).

DD

Supply voltage for the 42-PDIP which connects only the logic core and the embedded program memory.

WRVDD

Supply voltage for the 42-PDIP which connects only the I/O Pad Drivers. The application board **MUST** connect both VDD and PWRVDD to the board supply voltage.

WRGND

Ground for the 42-PDIP which connects only the I/O Pad Drivers. PWRGND and GND are weakly connected through the common silicon substrate, but not through any metal link. The application board **MUST** connect both GND and PWRGND to the board ground.

Port 0

Port 0 is an 8-bit open drain bi-directional I/O port. As an output port, each pin can sink eight TTL inputs. When 1s are written to port 0 pins, the pins can be used as high-impedance inputs.

Port 0 can also be configured to be the multiplexed low-order address/data bus during accesses to external program and data memory. In this mode, P0 has internal pull-ups.

Port 0 also receives the code bytes during Flash programming and outputs the code bytes during program verification. **External pull-ups are required during program verification.**

Port 1

Port 1 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 1 output buffers can sink/source four TTL inputs. When 1s are written to Port 1 pins, they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 1 pins that are externally being pulled low will source current ( $I_{IL}$ ) because of the internal pull-ups.

Port 1 also receives the low-order address bytes during Flash programming and verification.

Port Pin	Alternate Functions
P1.5	MOSI (used for In-System Programming)
P1.6	MISO (used for In-System Programming)
P1.7	SCK (used for In-System Programming)

Port 2

Port 2 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 2 output buffers can sink/source four TTL inputs. When 1s are written to Port 2 pins, they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 2 pins that are externally being pulled low will source current ( $I_{IL}$ ) because of the internal pull-ups.

Port 2 emits the high-order address byte during fetches from external program memory and during accesses to external data memory that use 16-bit addresses (MOVX @ DPTR). In this application, Port 2 uses strong internal pull-ups when emitting 1s. During accesses to external data memory that use 8-bit addresses (MOVX @ RI), Port 2 emits the contents of the P2 Special Function Register.

Port 2 also receives the high-order address bits and some control signals during Flash programming and verification.

# AT89S51

2487B-MICRO-12/03

Port 3 is an 8-bit bi-directional I/O port with internal pull-ups. The Port 3 output buffers can sink/source four TTL inputs. When 1s are written to Port 3 pins, they are pulled high by the internal pull-ups and can be used as inputs. As inputs, Port 3 pins that are externally being pulled low will source current ( $I_{IL}$ ) because of the pull-ups.

Port 3 receives some control signals for Flash programming and verification.

Port 3 also serves the functions of various special features of the AT89S51, as shown in the following table.

Port Pin	Alternate Functions
P3.0	RXD (serial input port)
P3.1	TXD (serial output port)
P3.2	$\overline{INT0}$ (external interrupt 0)
P3.3	$\overline{INT1}$ (external interrupt 1)
P3.4	T0 (timer 0 external input)
P3.5	T1 (timer 1 external input)
P3.6	$\overline{WR}$ (external data memory write strobe)
P3.7	$\overline{RD}$ (external data memory read strobe)

Reset input. A high on this pin for two machine cycles while the oscillator is running resets the device. This pin drives High for 98 oscillator periods after the Watchdog times out. The DISRTO bit in SFR AUXR (address 8EH) can be used to disable this feature. In the default state of bit DISRTO, the RESET HIGH out feature is enabled.

Address Latch Enable (ALE) is an output pulse for latching the low byte of the address during accesses to external memory. This pin is also the program pulse input ( $\overline{PROG}$ ) during Flash programming.

In normal operation, ALE is emitted at a constant rate of 1/6 the oscillator frequency and may be used for external timing or clocking purposes. Note, however, that one ALE pulse is skipped during each access to external data memory.

If desired, ALE operation can be disabled by setting bit 0 of SFR location 8EH. With the bit set, ALE is active only during a MOVX or MOVC instruction. Otherwise, the pin is weakly pulled high. Setting the ALE-disable bit has no effect if the microcontroller is in external execution mode.

Program Store Enable ( $\overline{PSEN}$ ) is the read strobe to external program memory.

When the AT89S51 is executing code from external program memory,  $\overline{PSEN}$  is activated twice each machine cycle, except that two  $\overline{PSEN}$  activations are skipped during each access to external data memory.

External Access Enable.  $\overline{EA}$  must be strapped to GND in order to enable the device to fetch code from external program memory locations starting at 0000H up to FFFFH. Note, however, that if lock bit 1 is programmed,  $\overline{EA}$  will be internally latched on reset.

$\overline{EA}$  should be strapped to  $V_{CC}$  for internal program executions.

This pin also receives the 12-volt programming enable voltage ( $V_{PP}$ ) during Flash programming.

Input to the inverting oscillator amplifier and input to the internal clock operating circuit.

Output from the inverting oscillator amplifier





## APPENDIX 4

```

INCLUDE 89c51.mc
;*****
****
adc_port EQU p1
adc_select BIT p3.2
adc_Write BIT p3.3
;*****
****
sda BIT p2.0
scl BIT p2.1
read_flag EQU 00000001b
write_flag EQU 00000000b
nvm_address EQU 10100000b
rtcc_address EQU 11010000B
seconds_Address EQU 00h
sig_byte_address EQU 8
sig_byte_offset EQU 8
;*****
****
adc_Value DATA 8
seconds DATA 9
minutes DATA 10
hours DATA 11
day_week DATA 12
day DATA 13
month DATA 14
year DATA 15
control DATA 16
;*****
****
data_2_write DATA 23
data_Read DATA 24
slave_Address DATA 25
address_lo DATA 26
address_hi DATA 27
count DATA 28
;*****
****
count1 DATA 29
count2 DATA 30
count3 DATA 31
temp1 DATA 33
temp2 DATA 34
ERROR DATA 35
nvm_interval DATA 36
interval_temp DATA 37
count1_reload DATA 38
count2_reload DATA 30
count3_reload DATA 40
R6_TEMP DATA 41
R7_TEMP DATA 42
new_count DATA 43
;*****
****

stack EQU 90
;*****
****
sec5 BIT 127 ; check
for the correct positions here!!!
rtcc BIT sec5-1
mem_full BIT sec5-2
timeout BIT sec5-3
error_led BIT sec5-4
new_mem BIT sec5-5
;*****
****
pointer_Address EQU 4094
interval_Address EQU 4092
NVM_SELECT EQU 4093
;*****
****
buffer DATA 50 ; 60 before
host_time_Date_lenght equ 7
;*****
****
led_Green bit p3.4
led_orange bit p3.5
led_red bit p3.6
;*****
****

;*****
****
org 0000h
LJMP start_up
;*****
****
org 0003h
RETI
;*****
****
org 000bh
LJMP tf0_isr
;*****
****
org 0013h
RETI
;*****
****
org 001bh
RETI
;*****
****
org 0023h
LJMP serial_isr
;*****
****

org 0030h
start_up: CLR ea
MOV sp,#stack
call sys_init

```

```

;*****
;*****
main:      call get_temp
          call store_temp
          SETB EA
          SJMP main
;*****
;*****
get_Temp:  JNB sec5,$
          CLR sec5
          CPL led_orange
          CLR EA
          CLR adc_write
          SETB adc_write
          MOV R2,#100
          DJNZ R2,$
          MOV adc_value,
adc_port
          RET
;*****
;*****
host_Error: MOV error,#8
          CALL GET_eRROR
          RET
;*****
;*****
sys_init:  call long_delay
          CLR adc_select
          SETB sda
          SETB scl
          CLR mem_full
          call clear_Error
          MOV tmod,#22h
          MOV scon,#50h
          MOV tcon,#0
          MOV th0,#16
          MOV tl0,#16
          MOV th1,#0fDh
          MOV tl1,#0fDh
          MOV count1,#240
          MOV count2,#16
          MOV
interval_temp,#150
set up for 15-min sampling
interval. change this!!!
          MOV new_count,#6
          SETB tr0
          SETB tr1
          SETB ti
          SETB ren
          CLR ri
          call init_timer
          MOV ie,#10010010b
          RET
;*****
;*****
store_temp: call
load_time_date ; CLEARED
;*****
;*****
error_store_Temp JC
load_pointer call ; CLEARED
error_store_temp JC
call write_temp
error_Store_temp JC
store_pointer call
error_store_temp JC
RET
error_store_temp: call get_Error
RET
;*****
;*****
write_temp: JB mem_full,
exit_Write_temp
MOV
data_2_write, day
call write_nvm
JC
error_write_temp call
inc_Address MOV
data_2_write, month
call write_nvm
JC
error_write_temp call
inc_Address MOV
data_2_write, year
call write_nvm
JC
error_write_temp call
inc_Address MOV
data_2_Write, hours
call write_nvm
JC
error_write_temp call
inc_address MOV
data_2_write,minutes
call write_nvm
JC
error_write_temp call
inc_Address MOV
data_2_write, adc_Value

```

```

                call write_nvm
                JC
error_write_temp RET
exit_write_temp: MOV error,#10
                ; return 10 if memory full
                SETB C
                RET
error_write_temp: MOV error,#0
                RET
;*****
;*****
load_time_date:  MOV R0,#seconds
                MOV
address_lo,#seconds_address
                MOV count,#8
load_time_loop:  call read_rtcc
                JC
error_load_date_time
                MOV @R0,
data_read
                inc r0
                inc address_lo
                djnz count,
load_time_loop
                CLR C
                RET
error_load_Date_time:  MOV
error,#1
                RET
;*****
;*****
load_pointer:    MOV
address_hi,#high(pointer_address)
                MOV address_lo,
#low(pointer_address)
                call read_nvm
                JC
ERROR_LOAD_POINTER
                MOV R7_TEMP,
data_read
                call
inc_Address
                call read_nvm
                JC
ERROR_LOAD_POINTER
                MOV r6_TEMP,
data_Read
                MOV address_hi,
r7_TEMP
                MOV address_lo,
r6_TEMP
                CALL
GET_ADDRESS
                CLR C
                RET

```

```

ERROR_LOAD_POINTER:  MOV
ERROR,#2
                RET
;*****
;*****
store_pointer:    MOV R7_TEMP,
address_hi
                MOV r6_TEMP,
address_lo
                MOV
address_hi,#high(pointer_address)
                MOV
address_lo,#low(pointer_address)
                MOV
data_2_Write,r7_TEMP
                call write_nvm
                JC
ERROR_STORE_POINTER
                call
inc_address
                MOV
data_2_Write, r6_TEMP
                call write_nvm
                JC
ERROR_STORE_POINTER
                CLR C
                RET
ERROR_STORE_POINTER:  MOV
ERROR,#3
                RET
;*****
;*****
inc_address:      MOV A,
address_lo
                add a,#1
                mov address_lo,
a
                clr a
                addc a,
address_hi
                mov address_hi,
a
                ret
;*****
;*****
tf0_isr:         DJNZ count1,
exit_isr
                MOV count1,#240
                DJNZ count2,
exit_isr
                MOV count2,#16
                cpl led_Green
                DJNZ
interval_temp, exit_isr

```

```

MOV
interval_Temp, #150 ;
sample every 15 minutes
DJNZ
new_count, exit_isr
MOV
new_count, #6 write_time_out
SETB sec5 RET
exit_isr: RETI write_abort: CALL
;*****
;***** write_time_out CALL
write_nvm: MOV ret
slave_address, #nvm_address
CLR rtcc ;*****
call write ;*****
ret ;*****
;***** read: CALL i2c_Start
;***** SLAVE_Address MOV A,
;***** #write_flag ORL A,
;***** CALL write_byte
;***** JC read_Abort
;***** JB rtcc,
;***** SKIP_READ1 MOV A,
;***** ADDRESS_HI CALL WRITE_BYTE
;***** SKIP_READ1: JC READ_ABORT
;***** address_LO MOV A,
;***** CALL write_byte
;***** JC read_abort
;***** CALL i2c_Start
;***** MOV A,
;***** SLAVE_Address ORL A,
;***** #read_flag ORL A,
;***** CALL write_byte
;***** JC read_Abort
;***** CALL read_byte
;***** MOV data_Read,
;***** A
;***** CALL NO_ack
;***** CLR C
;***** CALL I2C_STOP
;***** RET
;***** read_Abort: CALL i2c_Stop
;***** RET
;*****
;*****
;***** i2c_start: SETB SDA
;***** SETB SCL
;***** CALL dly_7us
;***** CLR SDA
;***** LCALL dly_5us
;***** CLR SCL
data_2_Write

```

```

                CALL dly_7us
                CLR C
                RET
;*****
;*****
i2c_stop:      CLR SDA
                CALL dly_5us
                SETB SCL
                CALL dly_7us
                SETB SDA
                RET
;*****
;*****
no_Ack:       SETB SDA
                NOP
                NOP
                SETB SCL
                NOP
                NOP
                NOP
                CLR SCL
                RET
;*****
;*****
write_byte:   MOV R7,#8
write_loop:   RLC A
                MOV SDA, C
                NOP
                NOP
                SETB SCL
                CALL dly_7us
                CLR SCL
                CALL dly_7us
                DJNZ R7,
write_loop
                SETB SDA
                NOP
                NOP
                NOP
                SETB SCL
                NOP
                NOP
                MOV C, SDA
                CLR SCL
                RET
;*****
;*****
read_byte:   MOV R7,#8
                SETB SDA
read_loop:   NOP
                SETB SCL
                CALL dly_7us
                SETB SDA
                NOP
                NOP
                MOV C, SDA
                RLC A
                NOP
                NOP
                CLR SCL
                CALL dly_5us
                DJNZ R7, read_loop
                MOV data_Read, A
                RET
;*****
;*****
write_time_out: CALL
small_delay
                RET
;*****
;*****
DLY_7US:     NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                NOP
                RET
;*****
;*****
DLY_5US:     NOP
                NOP
                NOP
                RET
;*****
;*****
get_address: call
inc_address ; if address = 0000h,
use the stored nvm address
pointer in 4093
                MOV A,
address_lo
                ORL A,
address_hi
                JNZ skip1
                CLR mem_full
                SETB new_mem
                RET
;*****
;*****
skip1:       CLR mem_full
                CLR new_mem
                MOV A,
address_hi
                CJNE
A,#high(INTERVAL_ADDRESS), chk1
                MOV A,
address_lo
                CJNE
A,#low(INTERVAL_ADDRESS), chk2
back1:      MOV
address_hi,#0

```

```

MOV
address_lo,#0
SETB mem_full
CLR new_mem
back2: RET
chk1: JNC back1
RET
chk2: JNC back2
RET
;*****
*****
send_Data: CLR ti
; cleared
MOV sbuf, A
JNB TI,$
CLR TI
call
small_delay
RET
;*****
*****
SMALL_DELAY: MOV TEMP1,#20
; cleared
SMALL_LOOP: MOV temp2,#0
RELOAD_A: NOP
NOP
NOP
DJNZ
TEMP2,RELOAD_a ; CHANGED
THIS HERE
DJNZ temp1,
small_loop
RET
;*****
*****
connect_host: CALL read_port
; cleared
JB
timeout,exit_CONNECT
CJNE A, #"$",
exit_CONNECT
CALL READ_PORT
JB
timeout,exit_CONNECT
CJNE A,#"0",
exit_CONNECT
CALL read_port
JB timeout,
exit_CONNECT
;*****
*****
chk1a ; cleared
CALL
set_time_DATE
SETB error_led
RET
;*****
*****
CHK1a: CJNE A,#"2",
CHK2a ; cleared
CALL dump_Data
SETB error_led
RET
;*****
*****
chk2a: CJNE A,#"3",
chk3a ; cleared
call init_nvm
setb error_led
ret
chk3a:
exit_connect: RET
;*****
*****
serial_isr: CLR ti
; cleared
JNB ri,
exit_serial2
call
connect_host
MOV
DPTR,#START_UP
PUSH dpl
PUSH dph
exit_Serial2: RETI
;*****
*****
init_nvm: Call
init_pointer
JC
EXIT_INIT_NVM
CLR mem_full
MOV
DPTR,#INIT_NVM_CMD
CALL WRITE_CMD
CLR A
CALL SEND_DATA
RET
exit_init_nvm: MOV
DPTR,#INIT_NVM_CMD2
CALL WRITE_CMD
MOV A, ERROR
CALL SEND_DATA
CALL GET_eERROR
RET
INIT_NVM_CMD: DB "$03",0
INIT_NVM_CMD2: DB "$E3",0
;*****
*****
init_pointer: MOV
address_hi,#high(pointer_address)

```

```

MOV
address_lo, #low(pointer_address)
MOV
DATA_2_WRITE, #OFFH
call write_nvm
JC
error_init_pointer
CALL
INC_ADDRESS
CALL WRITE_NVM
JC
error_init_pointer
RET
error_init_pointer: MOV
error, #6
RET

;*****
;*****
set_time_date: MOV
R3, #host_time_Date_lenght
call read_host
JNC SKIP_SET
JMP HOST_eRROR

SKIP_sET: MOV R0, #BUFFER
MOV R1, #SECONDS
MOV COUNT, #7
CALL BIN_2_HEX
MOV R0, #seconds
MOV
address_hi, #0
MOV
ADDRESS_LO, #SECONDS_address
MOV COUNT, #7
CALL WRITE_TIME
JC
SET_TIME_DATE_FAIL

TIME_UPDATE_PASS: MOV DPTR, #CMD1
CALL WRITE_CMD
CLR A
CALL SEND_data
RET

;*****
;*****
set_time_Date_fail: MOV
DPTR, #CMD2
CALL WRITE_CMD
MOV A, ERROR
CALL SEND_data
CALL GET_eRROR
RET

CMD1: DB "$01", 0
CMD2: DB "$E1", 0
;*****
;*****
WRITE_CMD: CLR A

MOV A, @A+DPTR
JZ
EXIT_WRITE_CMD
CALL SEND_DATA
INC DPTR
SJMP WRITE_CMD
EXIT_WRITE_CMD: RET
;*****
;*****
START_TIMER2: MOV
COUNT1_RELOAD, #100
MOV
COUNT2_RELOAD, #100
MOV
COUNT3_RELOAD, #4
CALL
START_TIMER
RET
;*****
;*****
BIN_2_HEX: MOV A, @R0
MOV B, #10
DIV AB
SWAP A
ORL A, B
MOV @R1, A
INC R0
INC R1
DJNZ COUNT,
BIN_2_HEX
RET
;*****
;*****
read_host: MOV R0, #BUFFER
GET_LOOP: CALL READ_PORT
jb timeout,
fuck_this
MOV @R0, A
INC R0
DJNZ
R3, get_loop
clr c
ret
fuck_this: setb c
ret
;*****
;*****
read_port: CLR timeout
; use hardware timeout
generator here!!!!
CALL
START_TIMER2
PORT_LOOP: JBC RI, GO_READ
JNB TIMEOUT,
PORT_LOOP
SETB TIMEOUT
RET

```





```

;*****
;*****
go_init:      MOV secondsS,#0
              MOV minutesS,#0
              MOV hourS,#0
              MOV day_Week,#3
              MOV day,#01h
              MOV month,#01h
              MOV year,#09h
              MOV control,#0
              MOV
address_lo,#seconds_Address
              MOV R0,#secondsS
              MOV count,#8
              acall
write_time
              jc exit_init2
              acall
write_sig_byte
              jc exit_init2
              CLR C
              RET
exit_init2:   acall
ERROR_INIT_TIMER
              RET
;*****
;*****
write_time:   MOV
data_2_write,@R0
              call write_rtcc
              jc
exit_Write_time
              INC address_lo
              INC R0
              DJNZ
count,write_time
              RET
exit_Write_time: MOV ERROR,#07H
              RET
;*****
;*****
write_sig_byte: MOV
DPTR,#sig_byte_msg
              MOV
address_lo,#sig_byte_Address
              acall write_sig
              RET
;*****
;*****
write_Sig:    CLR A
              MOV C
A,@a+dptr
              JZ
exit_write_Sig
              MOV
data_2_write,A
              write_rtcc      call
              exit_write_sig  jc
              address_lo      INC
              write_Sig       INC DPTR
              exit_Write_Sig: JMP
              ;*****
              ;*****
              sig_byte_msg:   DB
              "DS1307",0
              ;*****
              ;*****
              compare_sig_byte: MOV
              R0,#buffer+sig_byte_offset
              MOV A,@R0
              XRL A,#"D"
              JNZ EXIT
              INC R0
              MOV A,@R0
              XRL A,#"S"
              JNZ EXIT
              INC R0
              MOV A,@R0
              XRL A,#"1"
              JNZ EXIT
              INC R0
              MOV A,@R0
              XRL A,#"3"
              JNZ EXIT
              INC R0
              MOV A,@R0
              XRL A,#"0"
              JNZ EXIT
              INC R0
              MOV A,@R0
              XRL A,#"7"
              JNZ EXIT
              CLR C
              RET
EXIT:         SETB C
              RET
;*****
;*****
dump_NVM:    call
load_pointer
              JC error_dump1
              JB new_mem,
error_dump2
              JB mem_full,
skip_dump1
              MOV DPH,
ADDRESS_HI
              MOV
DPL,ADDRESS_LO
              SJMP dump1

```

```

;*****
;*****
skip_dump1:      MOV DPTR,#4092

DUMP1:          MOV
ADDRESS_LO,#0   MOV
ADDRESS_HI,#0   MOV
DUMP1_LOOP:     CALL READ_NVM
                JC ERROR_DUMP1
                MOV A,
DATA_READ       MOV B,#16
                DIV AB
                CALL
CONVERT_2_ASCII CALL SEND_DATA
                MOV A, B
                CALL
CONVERT_2_aSCII CALL SEND_DATA
                CALL
INC_ADDRESS     CALL DEC_DPTR
                MOV A, DPH
                ORL A, DPL
                JNZ DUMP1_LOOP
                CLR C
                RET
;*****
;*****
ERROR_DUMP1:    SETB C
                MOV ERROR,#4
                RET
error_dump2:    SETB C
                MOV error,#9
                RET
;*****
;*****
DUMP_DATA:     MOV
DPTR,#CMD_STATUS2
                CALL WRITE_CMD
                CALL DUMP_NVM
                JC
ERROR_DUMP_DATA RET
;*****
;*****
ERROR_DUMP_DATA: MOV A,error
                CALL SEND_DATA
                CALL GET_eERROR
                RET
CMD_STATUS:    DB "$E2",0

CMD_STATUS2:   DB "$02",0
;*****
;*****
clear_error:    SETB led_red
                SETB led_orange
                RET
;*****
;*****
get_Error:     call
clear_error    MOV A, ERROR
                MOV
                DPTR,#ERROR_TABLE
                CLR C
                RLC A
                JMP @A+DPTR
;*****
;*****
ERROR_TABLE:   AJMP
TEMP_WRITE_ERROR
                AJMP
TIME_DATE_LOAD_eERROR
                AJMP
POINTER_LOAD_ERROR
                AJMP
POINTER_STORE_ERROR
                AJMP
DATA_DUMP_ERROR
                AJMP
NVM_INIT_ERROR
                AJMP
POINTER_INIT_ERROR
                AJMP
TIME_WRITE_ERROR
                ajmp
host_Error2    ajmp
new_mem_error  ;
                ajmp
mem_full_error
;*****
;*****
TEMP_WRITE_ERROR:
pointer_load_Error:
data_dump_Error:
nvm_init_error:
pointer_init_error:
pointer_Store_Error:  clr
led_Red              CLR
led_green            SETB
led_orange           RET
;*****
;*****

```

```

;*****
*****
TIME_DATE_LOAD_ERROR:
TIME_WRITE_ERROR:
TIMER_INIT_ERROR:      CLR
led_orange              SETB
led_red                 CLR
led_green               CLR
                        ret
;*****
*****

```

```

;*****
*****
new_mem_error:
host_error2:
mem_full_error:        CLR
led_Red                CLR
led_orange             CLR
led_green              CLR
                        RET

```