ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)

ORGANIZATION OF THE ISLAMIC CONFERENCE (OIC)

DEPARTMENT OF MECHANICAL & CHEMICAL ENGINEERING (MCE)

MICROCONTROLLER BASED TEMPERATURE SENSOR AND DATA LOGGER

Prepared by:

Kazi Ahsan Uddin (081404)

Yeasir Arafat (081407)

Supervised by:

Prof. Dr. A.K.M. Sadrul Islam

Ph.D. (London), M.Sc. Engg. (Mech.), B.Sc. Engg. (Mech) BUET

Islamic University of Technology (IUT)

Organization of Islamic Cooperation

Board bazar, Dhaka

This thesis is submitted to the

DEPARTMENT OF MECHANICAL & CHEMICAL ENGINEERING (MCE)

ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)

By:

KaziAhsanUddin (081404)

Yeasir Arafat (081407)

IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF SCIENCE IN MECHANICAL ENGINEERING

Acknowledgement

The real spirit of achieving a goal is through the way of excellence and discipline. We would have never succeeded in completing our task without the cooperation, encouragement and help provided to us by various personalities.

First of all, we render our gratitude to the ALMIGHTY who bestowed self-confidence, ability and strength in us to complete this work. Without His grace this would never come to be today's reality.

With deep sense of gratitude, we express our sincere thanks to our esteemed and worthy Supervisor **Prof. Dr. A. K. M. Sadrul Islam, department of Mechanical and Chemical Engineering, Islamic University of Technology (IUT), Bangladesh,** for his valuable guidance in carrying out this work under his effective supervision, encouragement, enlightenment and cooperation. Most of the novel ideas and solutions found in this thesis are the result of our numerous stimulating discussions. His feedback and editorial comments were also invaluable for writing of this thesis.

We shall be failing in our duties if we do not express our deep sense of gratitude towards Md. Hamidur Rahman, assistant professor of the Department of Mechanical and chemical engineering and Mr. Golam Sarowar, assistant professor of the Department of Electrical and Electronic Engineering of Islamic university of technology (IUT), who have been a constant source of inspiration for us throughout this work. We are also thankful to all the staff members of the Department for their fullcooperation and help.

Our greatest thanks are to all who wished us success especially our parents, our brothers and sisters whose support and care make us stay on earth. We would also like to thank our friend Towfiq, from MCE'08 who helped us a lot in completing this work. And we also thank all those who helped us directly and indirectly in completion of this work.

List of abbreviations

ALU	Arithmetic and logical unit
AT	Atmel
CLK	Clock
CPU	Central Processing Unit
EPROM	Erasable and programmable read only memory
EEPROM	Electrically erasable programmable read only Memory
GND	Ground
I/O	Input/output
IC	Integrated circuit
LCD	Liquid Crystal Display
MCU	Microcontroller unit
MHZ	Megahertz
RAM	Random access memory
ROM	Read only memory
RTC	Real time clock
RTD	Resistive temperature device
XTAL	Crystal

Abstract

The average temperature during summer season in Bangladesh is quite high. The food industries & houses are exposed to excess heating not only due to the cooking temperature but also by cooking appliances and there's always problem of high atmospheric temperature. Moreover, the kitchens are not designed or positioned in proper manner. The sole idea of this project is to help create an efficient temperature measuring and acquisition system for domestic and industrial spaces and to know about the temperature at different times. The acquired data can later be utilized to design these areas with proper maintenance for optimum thermal comfort.

With the progress of technology, the processes are getting more and more complex. Due to this increase in complexity, for efficient analysis of process the number ofparameters needed for data acquisition also increases. Data Acquisition is simply thecollection of information about a system or process. It is the process of gathering data in an automated fashion from analog and digital measurement sources such as sensors and devices under test. Before the computer age, most data was recorded manually or on strip chart recorders. Many new generation data acquisition systems have been designed due to emergence of microcontroller that enables real-time gathering, analysis, logging and viewing of data. To fulfill these requirements the need for an improved, efficient and up to date data logger is increasing day by day.

In this thesis, a data logger for specific application has been designed. Data loggers have an on-board memory that is large enough to hold data that is recorded over a longer period of time. Data loggers are provided with real time clocks to record the date and time of acquisition. The system works on the famous atmega16 microcontroller of AVR family. The system is designed and developed to measure the temperature with the help of temperature sensors and the result is showed and stored in a computer device. During the testing, it is verified that there is continuous and correct acquisition of data. It is also verified that the data is sequentially stored in memory. The focus of design is on portability and low power consumption for battery operated applications.

CONTENTS

Chapter 1: Introduction

Temperature is the ever-changing parameter because of exposition to huge arrayof stimuli from their environment. It can be measured via a diverse array of sensors. Allof them infer temperature by sensing some change in a physical characteristic. One must be careful when measuring temperature to ensure that the measuring instrument(thermometer, thermocouple, etc.) is really the same temperature as the material that isbeing measured. Under some conditions heat from the measuring instrument can cause a temperature gradient, so the measured temperature is different from the actualtemperature of the system. In such a case the measured temperature will vary not onlywith the temperature of the system, but also with the heat transfer properties of thesystem.

The processes to collect, analyze and store the data for later use is called logging.It is a process to record events during a test or measurement with the use of a system or a product. The human brain and its memory, the nature's creation, no doubt is the best data logging mechanism. Where there is the need to collect information faster than a human, data loggers can possibly collect the information and in cases where accuracy is essential. A data logger is a device that can be used to store and retrieve the data. Data logging also implies the control of how sensor collects and analyzes the data. It is commonly used in scientific experiments and in monitoring systems. Data loggers automatically make a record of the readings of the instruments located at different parts of plant. The type of information recorded is determined by the user. Their advantage is that they can operate independently of a computer and they are available in various shapes and sizes. The range includes simple economical single channel fixed function loggers to more powerful programmable devices capable of handling hundreds of inputs.

The objective of this work is to use data logging for temperature measurement. Inorder to meet the above requirements, a low cost, versatile and computer based data logger is designed.

Chapter 2: Literature Review

This chapter describes the introduction to temperature sensors,microcontroller,data loggers and literature survey.

2.1 Introduction to data loggers

The data logger is an invaluable tool to collect and analyze experimental data,having the ability to clearly present real time analysis with sensors and probes able torespond to parameters that are beyond the normal range available from the mosttraditional equipment. The differences between various data loggers are based on theway that data is recorded and stored.

2.1.1 Definition of Data Loggers

Data logger is an electronic device that automatically records, scans and retrievesthe data with high speed and greater efficiency during a test or measurement, at any partof the plant with time. The type of information recorded is determined by the user i.e.whether temperature, relative humidity, light intensity, voltage, pressure or shock is to berecorded, therefore it can automatically measures electrical output from any type oftransducer and log the value. A data logger works with sensors to convert physicalphenomena and stimuli into electronic signals such as voltage or current. These electronic signals are then converted into binary data. The binary data is then easily analyzed bysoftware and stored on memory for post process analysis.

2.1.2 Characteristics of Data Loggers:

Data loggers possess the following characteristics [8]:

1.) **Modularity**: Data loggers can be expanded simply and efficiently wheneverrequired, without any interruption to the working system.

2.) **Reliability and Ruggedness**: They are designed to operate continuously withoutinterruption even in the worst industrial environments.

3.) **Accuracy**: The specified accuracy is maintained throughout the period of use.

4.) **Management Tool**: They provide simple data acquisition, and present the results inhandy form.

5.) **Easy to use**: These communicate with operators in a logical manner, are simple inconcept, and therefore easy to understand, operate and expand.

2.1.3 Operation of data logger:

The ability to take sensor measurements and store the data for future use is, bydefinition, a characteristic of a data logger. However, a data-logging application rarelyrequires only data acquisition and storage. Inevitably, the ability to analyze and presentthe data to determine results and make decisions based on the logged data is needed. Acomplete data-logging application typically requires most of the elements illustratedbelow [8].

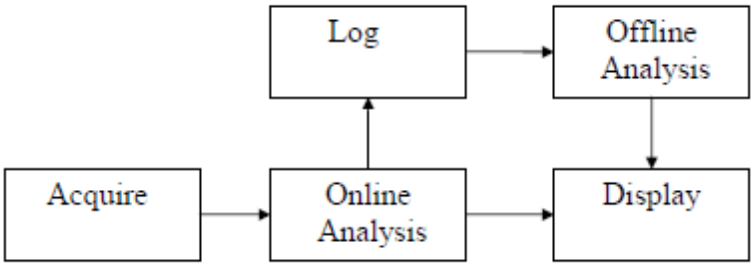

Figure1: Block diagram of data logger.

1.) **Acquire** – This step includes your sensors and data logger hardware as well asconversion of physical phenomena into digital signals.

2.) **Online analysis** – This step includes any analysis that is likely to be done before storing the data. A common example of this is converting the voltage measurement tomeaningful scientific units, such as degree Celsius. These complex calculations anddata compression are completed before logging the data. Every data logging software application should complete this conversion from binary value to voltage and theconversion from voltage to scientific units.

3.) **Log** – This step refers to the storage of analyzed data including any formattingrequired for the data files.

4.) **Offline Analysis** - This step includes any analysis that is to be done after storing thedata. A common example is looking for trends in historical data or data reduction.

5.) **Displaying, reporting** - This step includes the creation of any reports that are neededto make to present data and displaying the data. However, this can also present datastraight from online analysis. This represents the ability to monitor and view the dataas acquired and analyzed in addition to simply viewing historical data. i.e. it shouldhave the following components:

-Hardware to digitize what is to be logged including sensors, signal conditioning,and analog-to-digital conversion hardware.

- Long-term data storage.

- Data-logging software for data acquisition, analysis, and presentation.

2.1.4 Advantages of Data Loggers:

1.) Data Loggers don't interfere with the users in performing their tasks.

2.) They can operate independently of a computer and they are available in variousshapes and sizes.

3.) The range of data loggers varies from simple channel inputs to multichannel devices.

2.1.5 Applications of Data Loggers:

They can be used in the following applications such as:

1.) In unattended recording at weather stations to record parameters like temperature,wind speed / direction, solar radiation and relative humidity.

2.) For hydrographic recording of water flow, water pH, water conductivity, water leveland water depth.

3.) In the recording of soil moisture levels.

4.) To record gas pressure and to monitor tank levels.

5.) In transportation monitoring, troubleshooting, educational science, quality studies,field studies and general research.

6.) Remote collection of recorded data and alarming or unusual parameters are possiblewith the help of data loggers where these are connected to modems and cellularphones.

2.2 Literature survey:

Dr. Saul Greenburg [10] has described the concept of logging and how logging is done isin detail. Logging is a process to record events with the use of data loggers during a testor field use of a system or a product. Logging is one of the usability methods that can and should be used to gather more supplementary information as an integral part of theiterative design of the usability engineering cycle. Logging has the major advantagecompared with other usability methods of not interfering with the users in theirperforming their tasks. Users can basically ignore the log and use the system in exactlythe way they would anyway.

H S kalsi[8] has detailed the concept of data loggers and its basic operation is described.
A data logger is a comprehensive and highly advanced data acquisition system. It is madeversatile and flexible, to render it suitable for widely varying applications, specificrequirements being met simply by setting up a suitable program. It can measure electricaloutput from any type of transducer and log the value automatically.

S.J.Perez, M.A.Calva, R.Castañeda[11] described a microcontroller-based datalogging system to record temperature and relative humidity for acoustic measurementapplications. The system is simple to use, requires no additional hardware and allows theselection of amount of data and the time intervals between them. The collected data caneasily be exported to a PC computer via a serial port.

Chapter 3: System Design and Implementation:

For the design and development of the system, the methodology used involves thesoftware and hardware implementation. The actual implementation of the systeminvolves the following steps:

1.) **System Definition**: Broad definition of system hardware includingmicrocontroller and its interface with display, programming, memory, keypad etc.
2.) **Circuit Design**: Selection of AVR microcontroller and other interfacingdevices, as per system definition. Design of hardware circuit and its testing onlaboratory kits with some simple microcontroller software routines.
3)**Hardware Modifications**: Making any hardware changes found necessaryafter the initial hardware tests, to produce a revised circuit board schematicdiagram and layout.
4) **Software Design**: Developing software for the system, allocating memoryblocks as per functionality, coding and testing.
5)**Integration and Final Testing**: Integrating the entire hardware andsoftware modules and its final testing for data logging operation.
Thus the complete design is divided into two parts:
1.) Hardware Implementation.
2.) Software Implementation.

3.1 Hardware Implementation:

It involves the details of the set of design specifications. The hardware designconsists of, the selection of system components as per the requirement, the details of subsystemsthat are required for the complete implementation of the system and fullhardware schematics for the layout. Design of the circuit and its testing has beencarried out. It involves the component selection, component description and hardwaredetails of the system designed.
1.) Component selection and description.
2.) Hardware details of the system designed.

3.1.1 Component description:

Temperature measurement using microcontroller based data logger includes the followingcomponents:

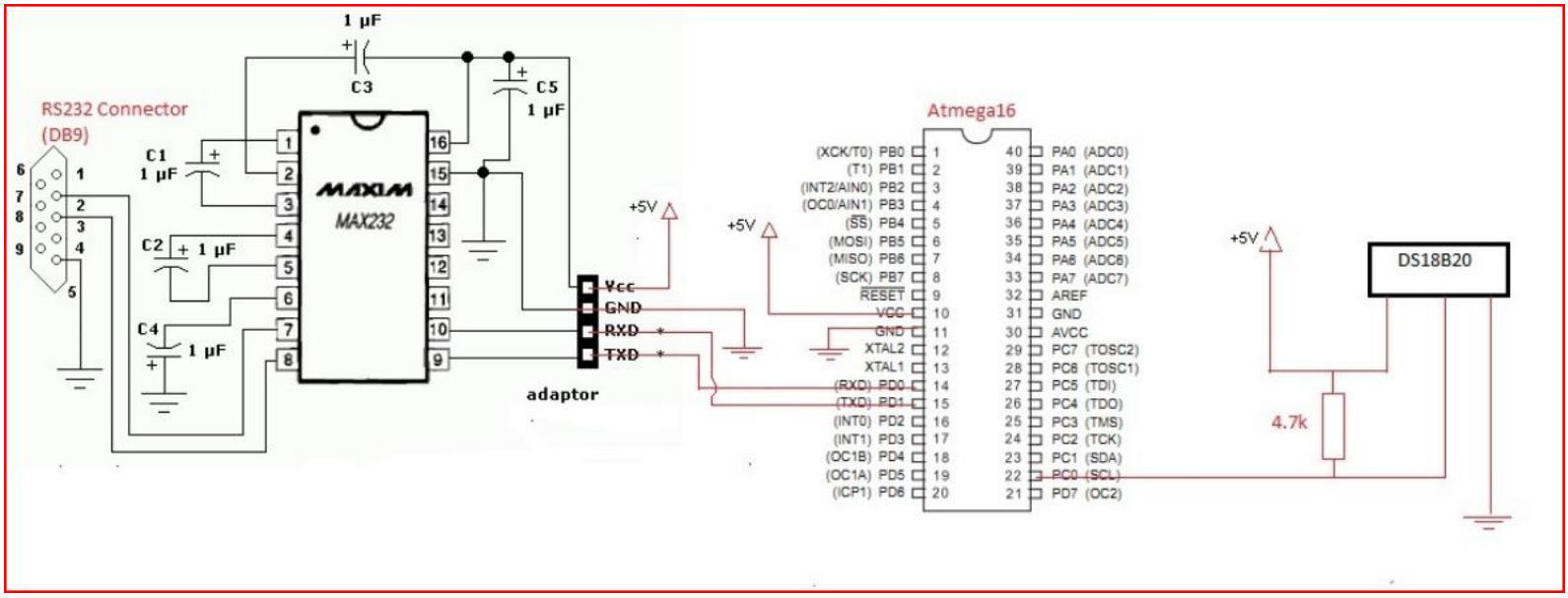

Figure2: Final Circuit Diagram

3.1.1.1 Temperature Sensor:

For measuring the temperature, the choice of sensor is of utmost importance.

The sensors are used in many fields includes Thermocouples, Resistive temperaturedevices (RTDs and thermistors) and bimetallic devices. The factors for the selection ofsensor that we take into account includes the inherent accuracy for durability, range ofoperation, susceptibility to external noise influences, ease of maintenance andinstallation, handling during installation (delicacy), ease of calibration, and type ofenvironment it will be used in.

The temperature sensor used for this purpose is DS18b20 because of the following features:

(a)Features of DS18b20:

FEATURES-

- ❖ Unique 1-Wire® Interface Requires Only One Port Pin for Communication
- ❖ Each Device has a Unique 64-Bit Serial Code Stored in an On-Board ROM
- ❖ Multidrop Capability Simplifies Distributed Temperature-Sensing Applications
- ❖ Requires No External Components
- ❖ Can Be Powered from Data Line; Power Supply Range is 3.0V to 5.5V
- ❖ Measures Temperatures from -55°C to +125°C (-67°F to +257°F)
- ❖ ±0.5°C Accuracy from -10°C to +85°C
- ❖ Thermometer Resolution is User Selectable from 9 to 12 Bits
- ❖ Converts Temperature to 12-Bit Digital Word in 750ms (Max)

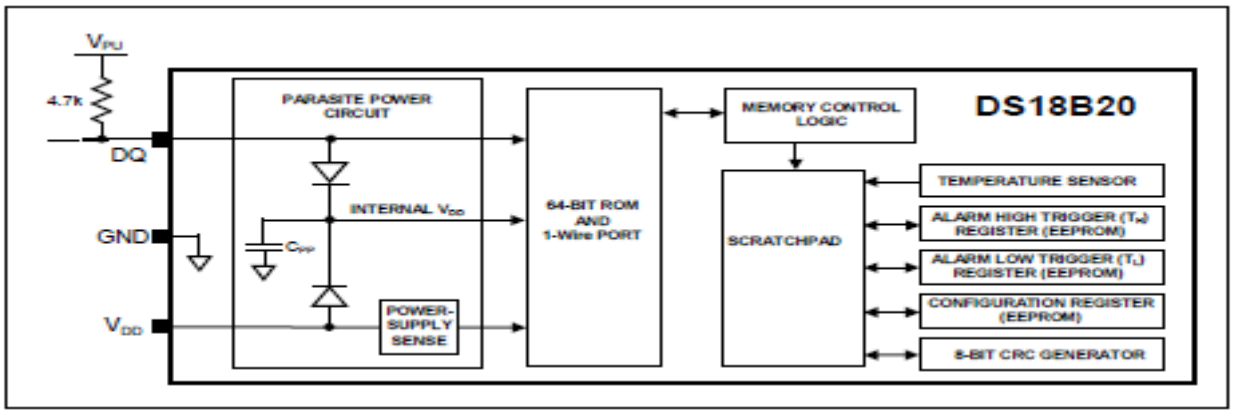

Figure 3: Block diagram for DS18b20

(b) OPERATION—MEASURING TEMPERATURE:

The core functionality of the DS18B20 is its direct-to-digital temperature sensor. The resolution of the temperature sensor is user-configurable to 9, 10, 11, or 12 bits, corresponding to increments of 0.5°C, 0.25°C, 0.125°C, and 0.0625°C, respectively. The default resolution at power-up is 12-bit. The DS18B20 powers up in a low-power idle state. To initiate a temperature measurement and A-to-D conversion, the master must issue a Convert T [44h] command. Following the conversion, the resulting thermal data is stored in the 2-byte temperature register in the scratchpad memory and the DS18B20 returns to its idle state. If the DS18B20 is powered by an external supply, the master can issue "read time slots" (see the *1-Wire Bus System* section) after the Convert T command and the

DS18B20 will respond by transmitting 0 while the temperature conversion is in progress and 1 when the conversion is done. If the DS18B20 is powered with parasite power, this notification technique cannot be used since the bus must be pulled high by a strong pullup during the entire temperature conversion. The bus requirements for parasite power are explained in detail in the *Powering the DS18B20* section.

The DS18B20 output temperature data is calibrated in degrees Celsius; for Fahrenheit applications, a lookup table or conversion routine must be used. The temperature data is stored as a 16-bit sign-extended two's complement number in the temperature register (see **Figure 2**). The sign bits (S) indicate if the temperature is positive or negative: for positive numbers S = 0 and for negative numbers S = 1. If the DS18B20 is configured for 12-bit resolution, all bits in the temperature register will contain valid data. For 11-bit resolution, bit 0 is undefined. For 10-bit resolution, bits 1 and 0 are undefined, and for 9-bit resolution bits 2, 1, and 0 are undefined. Table 1 gives examples of digital output data and the corresponding temperature reading for 12-bit resolution conversions.

TEMPERATURE (°C)	DIGITAL OUTPUT (BINARY)	DIGITAL OUTPUT (HEX)
+125	0000 0111 1101 0000	07D0h
+85*	0000 0101 0101 0000	0550h
+25.0625	0000 0001 1001 0001	0191h
+10.125	0000 0000 1010 0010	00A2h
+0.5	0000 0000 0000 1000	0008h
0	0000 0000 0000 0000	0000h
-0.5	1111 1111 1111 1000	FFF8h
-10.125	1111 1111 0101 1110	FF5Eh
-25.0625	1111 1110 0110 1111	FE6Fh
-55	1111 1100 1001 0000	FC90h

*The power-on reset value of the temperature register is +85°C.

Figure 4: Temperature/Data Relationship Table.

(c)POWERING THE DS18B20:

The DS18B20 can be powered by an external supply on the VDD pin, or it can operate in "parasite power" mode, which allows the DS18B20 to function without a local external supply. Parasite power is very useful for applications that require remote temperature sensing or that is very space constrained. Block diagram shows the DS18B20's parasite-power control circuitry, which "steals" power from the 1-Wire bus via the DQ pin when the bus is high. The stolen

charge powers the DS18B20 while the bus is high, and some of the charge is stored on the parasite power capacitor (CPP) to provide power when the bus is low. When the DS18B20 is used in parasite power mode, the VDD pin must be connected to ground.

In parasite power mode, the 1-Wire bus and CPP can provide sufficient current to the DS18B20 for most operations as long as the specified timing and voltage requirements are met. However, when the DS18B20 is performing temperature conversions or copying data from the scratchpad memory to EEPROM, the operating current can be as high as 1.5mA. This current can cause an unacceptable voltage drop across the weak 1-Wire pullup resistor and is more current than can be supplied by CPP. To assure that the DS18B20 has sufficient supply current, it is necessary to provide a strong pullup on the 1-Wire bus whenever temperature conversions are taking place or data is being copied from the scratchpad to EEPROM. This can be accomplished by using a MOSFET to pull the bus directly to the rail as shown in Figure 4. The 1-Wire bus must be switched to the strong pullup within 10μs (max) after a Convert T [44h] or Copy Scratchpad [48h] command is issued, and the bus must be held high by the pullup for the duration of the conversion (tCONV) or data transfer (tWR = 10ms). No other activity can take place on the 1-Wire bus while the pullup is enabled to the VDD pin, as shown in **Figure 6**. The advantage of this method is that the MOSFET pullup is not required, and the 1-Wire bus is free to carry other traffic during the temperature conversion time.

The use of parasite power is not recommended for temperatures above +100°C since the DS18B20 may not be able to sustain communications due to the higher leakage currents that can exist at these temperatures. For applications in which such temperatures are likely, it is strongly recommended that the DS18B20 be powered by an external power supply.

In some situations the bus master may not know whether the DS18B20s on the bus are parasite powered or powered by external supplies. The master needs this information to determine if the strong bus pullup should be used during temperature conversions. To get this information, the master can issue a Skip ROM [CCh] command followed by a Read Power Supply [B4h] command followed by a “read time slot”. During the read time slot, parasite powered DS18B20s will pull the bus low, and externally powered DS18B20s will let the bus remain high. If the bus

is pulled low, the master knows that it must supply the strong pullup on the 1-Wire bus during temperature conversions.

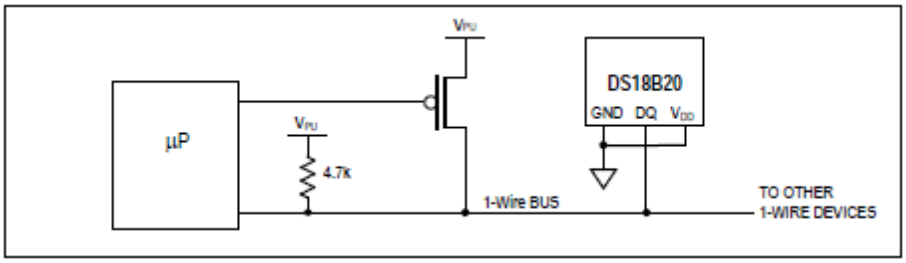

Figure 5: Supplying the Parasite-Powered DS18b20 during Temperature Conversions

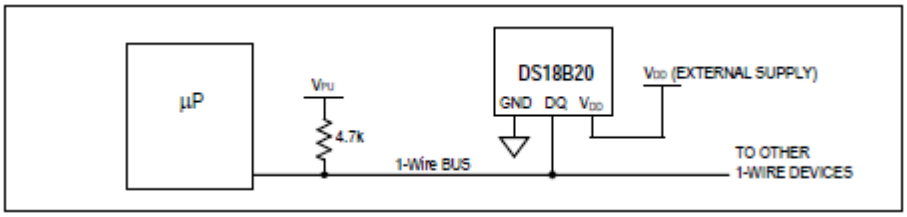

Figure 6:Powering the DS18B20 with an External Supply

(d)HARDWARE CONFIGURATION:

The 1-Wire bus has by definition only a single data line. Each device (master or slave) interfaces to the data line via an open-drain or 3-state port. This allows each device to "release" the data line when the device is not transmitting data so the bus is available for use by another device. The 1-Wire port of the DS18B20 (the DQ pin) is open drain with an internal circuit equivalent to that shown in **Figure 7**.

The 1-Wire bus requires an external pullup resistor of approximately 5kΩ; thus, the idle state for the 1-Wire bus is high. If for any reason a transaction needs to be suspended, the bus MUST be left in the idle state if the transaction is to resume. Infinite recovery time can occur between bits so long as the 1-Wire bus is in the inactive (high) state during the recovery period. If the bus is held low for more than 480µs, all components on the bus will be reset.

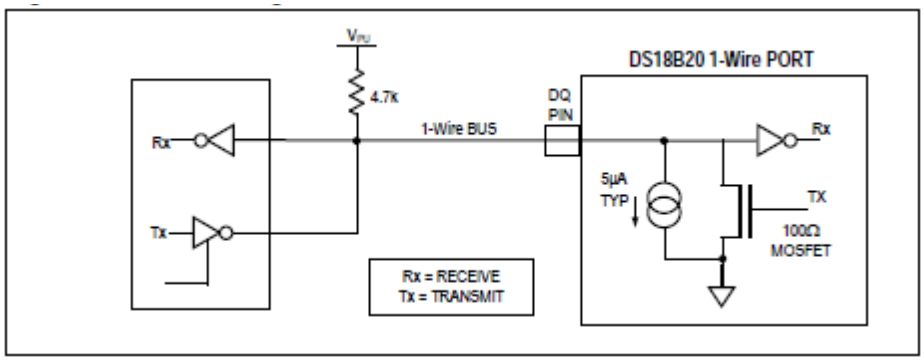

Figure 7: Hardware Configuration

(e)INITIALIZATION:

All transactions on the 1-Wire bus begin with an initialization sequence. The initialization sequence consists of a reset pulse transmitted by the bus master followed by presence pulse(s) transmitted by the slave(s). The presence pulse lets the bus master know that slave devices (such as the DS18B20) are on the bus and are ready to operate. Timing for the reset and presence pulses is detailed in the *1-Wire Signaling* section

.

(f)ROM COMMANDS:

After the bus master has detected a presence pulse, it can issue a ROM command. These commands operate on the unique 64-bit ROM codes of each slave device and allow the master to single out a specific device if many are present on the 1-Wire bus. These commands also allow the master to determine how many and what types of devices are present on the bus or if any device has experienced an alarm condition. There are five ROM commands, and each command is 8 bits long. The master device must issue an appropriate ROM command before issuing a DS18B20 function command. A flowchart for operation of the ROM commands is shown in **Figure 8.**

(g)READ ROM [33h] :

This command can only be used when there is one slave on the bus. It allows the bus master to read the slave's 64-bit ROM code without using the Search ROM procedure. If this command is used when there is more than one slave present on the bus, a data collision will occur when all the slaves attempt to respond at the same time.

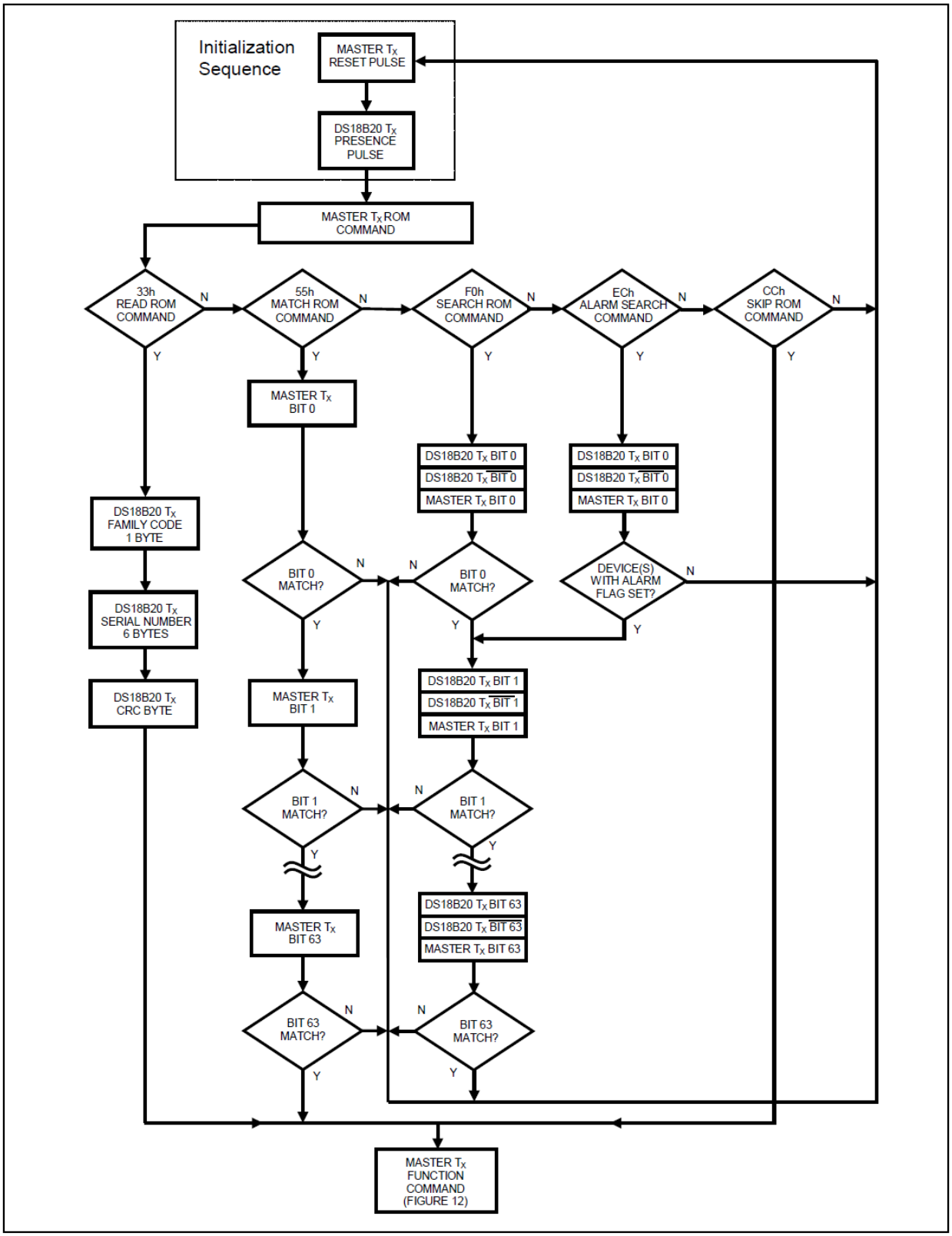

Figure 8: ROM Commands Flowchart

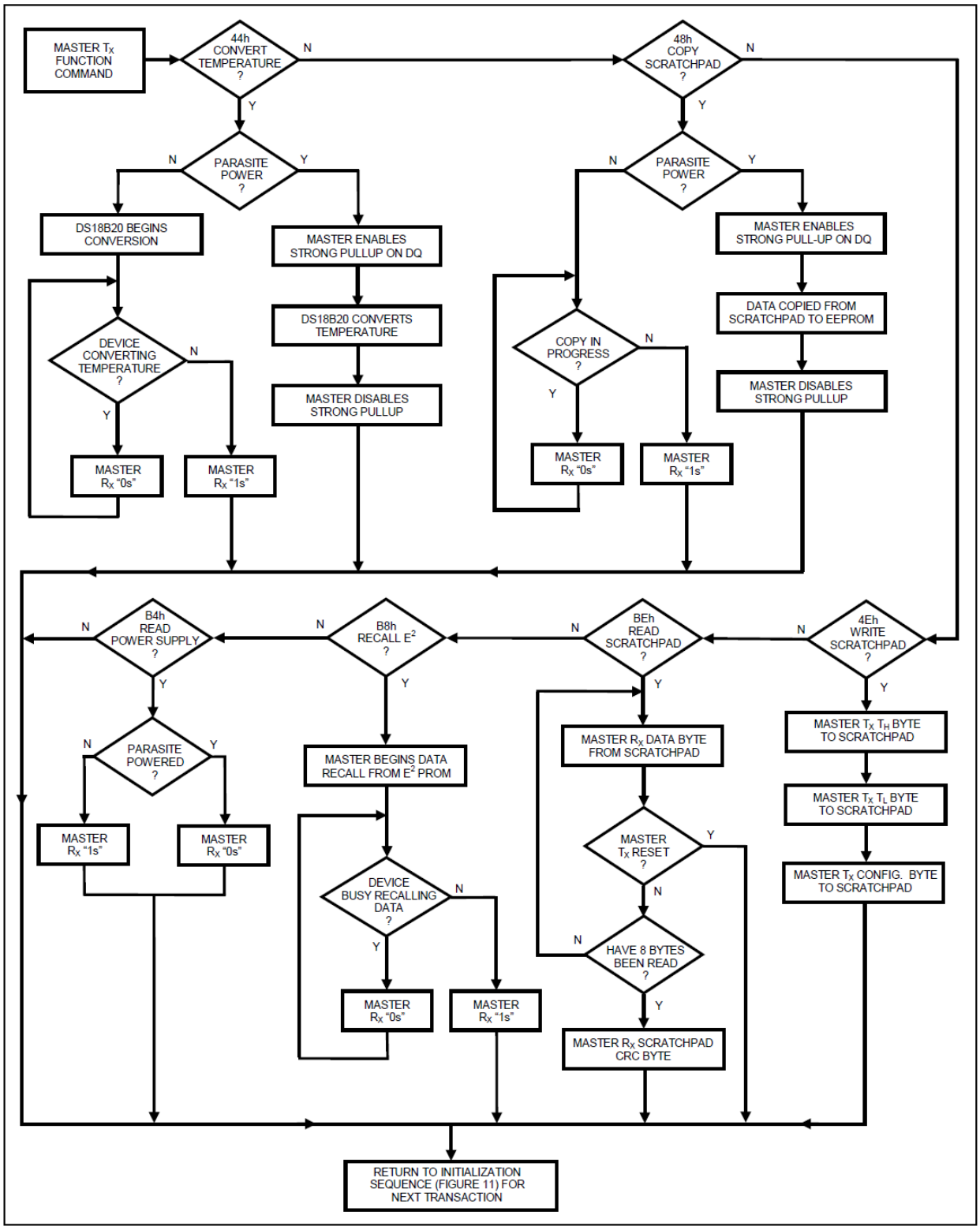

Figure 9: DS18B20 Function Commands Flowchart

(h)ONE-WIRE SIGNALING:

The DS18B20 uses a strict 1-Wire communication protocol to ensure data integrity. Several signal types are defined by this protocol: reset pulse, presence pulse, write 0, write 1, read 0, and read 1. The bus master initiates all these signals, with the exception of the presence pulse.

(i) INITIALIZATION PROCEDURE—RESET AND PRESENCE PULSES:

All communication with the DS18B20 begins with an initialization sequence that consists of a reset pulse from the master followed by a presence pulse from the DS18B20. This is illustrated in **Figure 10**. When the DS18B20 sends the presence pulse in response to the reset, it is indicating to the master that it is on the bus and ready to operate.

During the initialization sequence the bus master transmits (Tx) the reset pulse by pulling the 1-Wire bus low for a minimum of 480µs. The bus master then releases the bus and goes into receive mode (Rx). When the bus is released, the 5kΩ pullup resistor pulls the 1-Wire bus high. When the DS18B20 detects this rising edge, it waits 15µs to 60µs and then transmits a presence pulse by pulling the 1-Wire bus low for 60µs to 240µs.

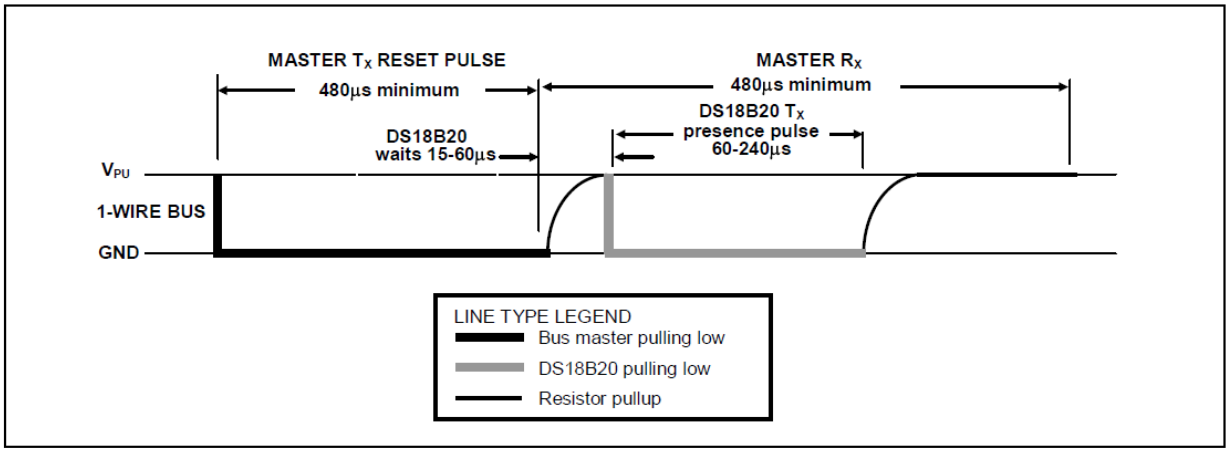

Figure 10: Initialization Timing

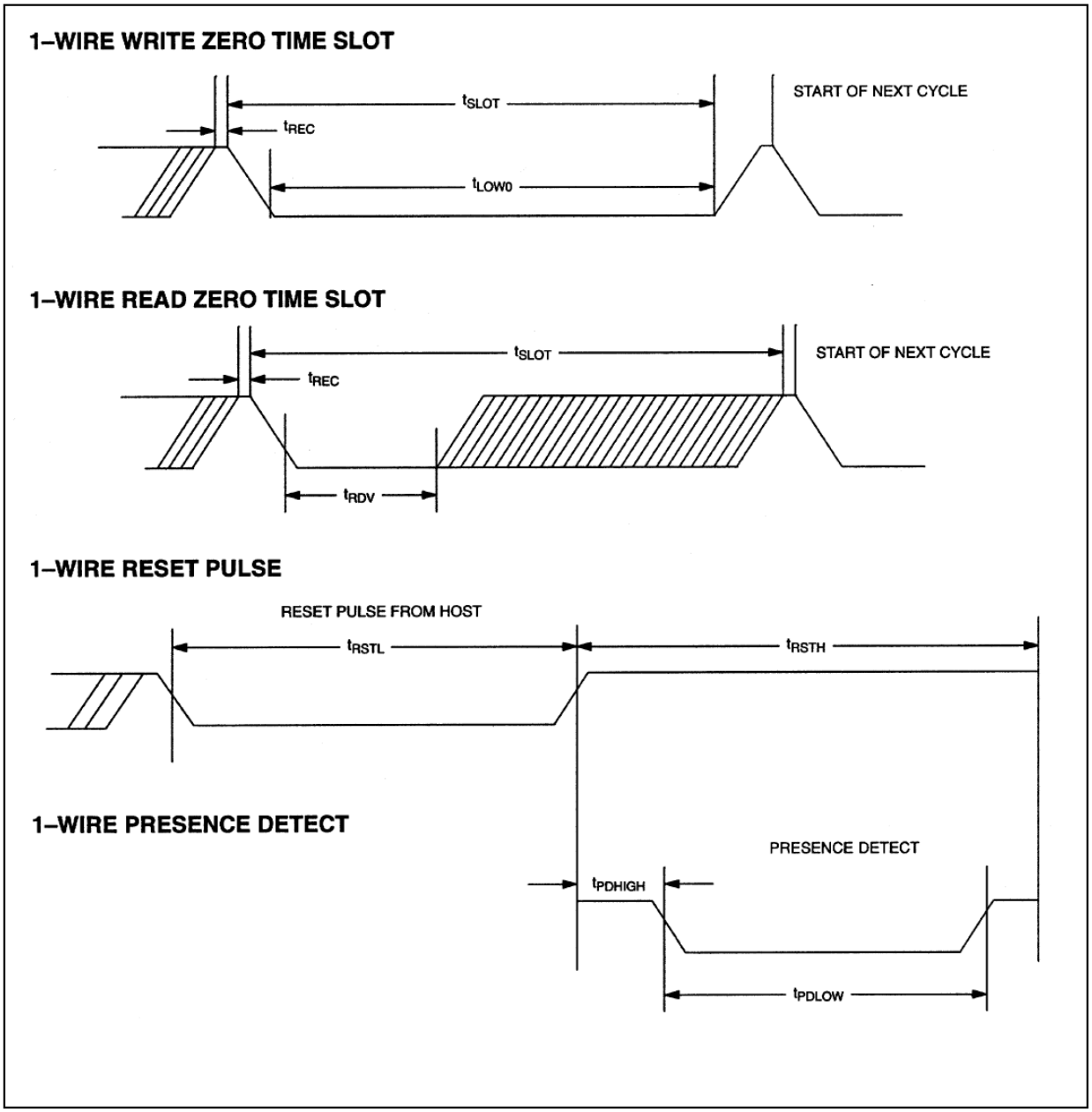

Figure 11: Timing diagrams

3.1.1.2 Microcontroller chip:

Criteria for choosing a microcontroller

1.) The first and foremost criterion for choosing a microcontroller is that it must meetthe task at hand efficiently and cost effectively. In analyzing the needs of amicrocontroller-based project, it

is seen whether an 8-bit, 16-bit or 32-bit microcontrollercan best handle the computing needs of the task most effectively. Among the otherconsiderations in this category are:

(a) Speed – What is the highest speed that the microcontroller supports?

(b) Packaging – Does it come in 40-pin DIP (dual inline package) or a QFP (quad flatpackage), or some other packaging format? This is important in terms of space,assembling, and prototyping the end product.

(c) Power consumption – This is especially critical for battery-powered products.

(d) The number of I/O pins and the timer on the chip.

(f) How easy it is to upgrade to higher –performance or lower consumption versions.

(g) Cost per unit – this is important in terms of the final cost of the product in which amicrocontroller is used.

2.) The second criterion in choosing a microcontroller is how easy it is to develop products around it. Key considerations include the availability of an assembler, debugger, a code – efficient compiler, technical support.

(a)Reasons behind choosing Atmega16:

Features-

• High-performance, Low-power Atmel® AVR® 8-bit Microcontroller

• Advanced RISC Architecture

– 131 Powerful Instructions – Most Single-clock Cycle Execution

– 32 x 8 General Purpose Working Registers

– Fully Static Operation

– Up to 16 MIPS Throughput at 16 MHz

– On-chip 2-cycle Multiplier

• High Endurance Non-volatile Memory segments

– 16 Kbytes of In-System Self-programmable Flash program memory

– 512 Bytes EEPROM

– 1 Kbyte Internal SRAM

– Write/Erase Cycles: 10,000 Flash/100,000 EEPROM

– Data retention: 20 years at 85°C/100 years at 25°C

– Optional Boot Code Section with Independent Lock Bits

In-System Programming by On-chip Boot Program

True Read-While-Write Operation

– Programming Lock for Software Security

• JTAG (IEEE std. 1149.1 Compliant) Interface

– Boundary-scan Capabilities According to the JTAG Standard

– Extensive On-chip Debug Support

– Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface

• Peripheral Features

– Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes

– One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode

– Real Time Counter with Separate Oscillator

– Four PWM Channels

– 8-channel, 10-bit ADC

8 Single-ended Channels

7 Differential Channels in TQFP Package Only

2 Differential Channels with Programmable Gain at 1x, 10x, or 200x

– Byte-oriented Two-wire Serial Interface

– Programmable Serial USART

– Master/Slave SPI Serial Interface

– Programmable Watchdog Timer with Separate On-chip Oscillator

– On-chip Analog Comparator

• Special Microcontroller Features

– Power-on Reset and Programmable Brown-out Detection

– Internal Calibrated RC Oscillator

– External and Internal Interrupt Sources

– Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby

• I/O and Packages

– 32 Programmable I/O Lines

– 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF

• Operating Voltages

– 4.5V - 5.5V for

• Speed Grades

– 0 - 16 MHz for

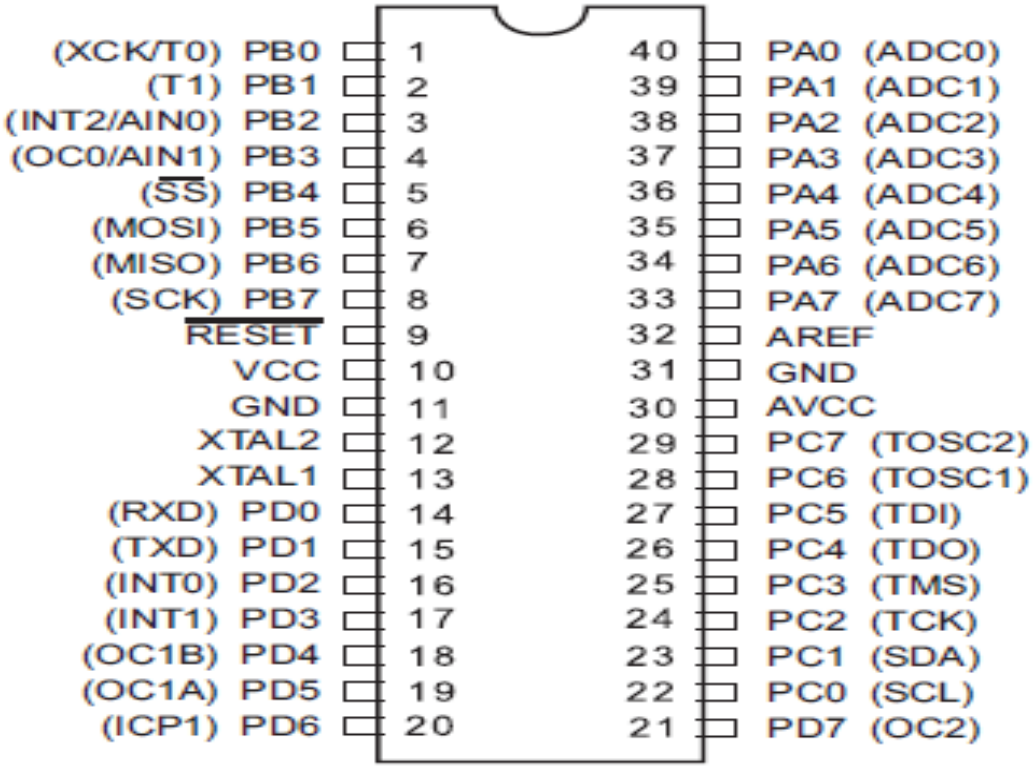

Figure 12: Pinout ATmega16

(b)Pin Descriptions:

[VCC Digital supply voltage.

GND Ground.]

Port A (PA7..PA0) Port A serves as the analog inputs to the A/D Converter.Port A also serves as an 8-bit bi-directional I/O port, if the A/D Converter is not used. Port pinscan provide internal pull-up resistors (selected for each bit). The Port A output buffers have symmetricaldrive characteristics with both high sink and source capability. When pins PA0 to PA7are used as inputs and are externally pulled low, they will source current if the internal pull-up resistors are activated. The Port A pins are tri-stated when a reset condition becomes active,even if the clock is not running.

Port B (PB7..PB0) Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). ThePort B output buffers have symmetrical drive characteristics with both high sink and sourcecapability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active,even if the clock is not running.

Port C (PC7..PC0) Port C is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). ThePort C output buffers have symmetrical drive characteristics with both high sink and sourcecapability. As inputs, Port C pins that are externally pulled low will source current if the pull-upresistors are activated. The Port C pins are tri-stated when a reset condition becomes active,even if the clock is not running. If the JTAG interface is enabled, the pull-up resistors on pinsPC5(TDI), PC3(TMS) and PC2(TCK) will be activated even if a reset occurs.

Port D (PD7..PD0) Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). ThePort D output buffers have symmetrical drive characteristics with both high sink and sourcecapability. As inputs, Port D pins that are externally pulled low will source current if the pull-upresistors are activated. The Port D pins are tri-stated when a reset condition becomes active,even if the clock is not running.

RESET Input. A low level on this pin for longer than the minimum pulse length will generate areset, even if the clock is not running.

XTAL1 Input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

XTAL2 Output from the inverting Oscillator amplifier.

AVCC AVCC is the supply voltage pin for Port A and the A/D Converter. It should be externally connectedto VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCCthrough a low-pass filter.

AREF AREF is the analog reference pin for the A/D Converter.

(c)AVR CPU Core:

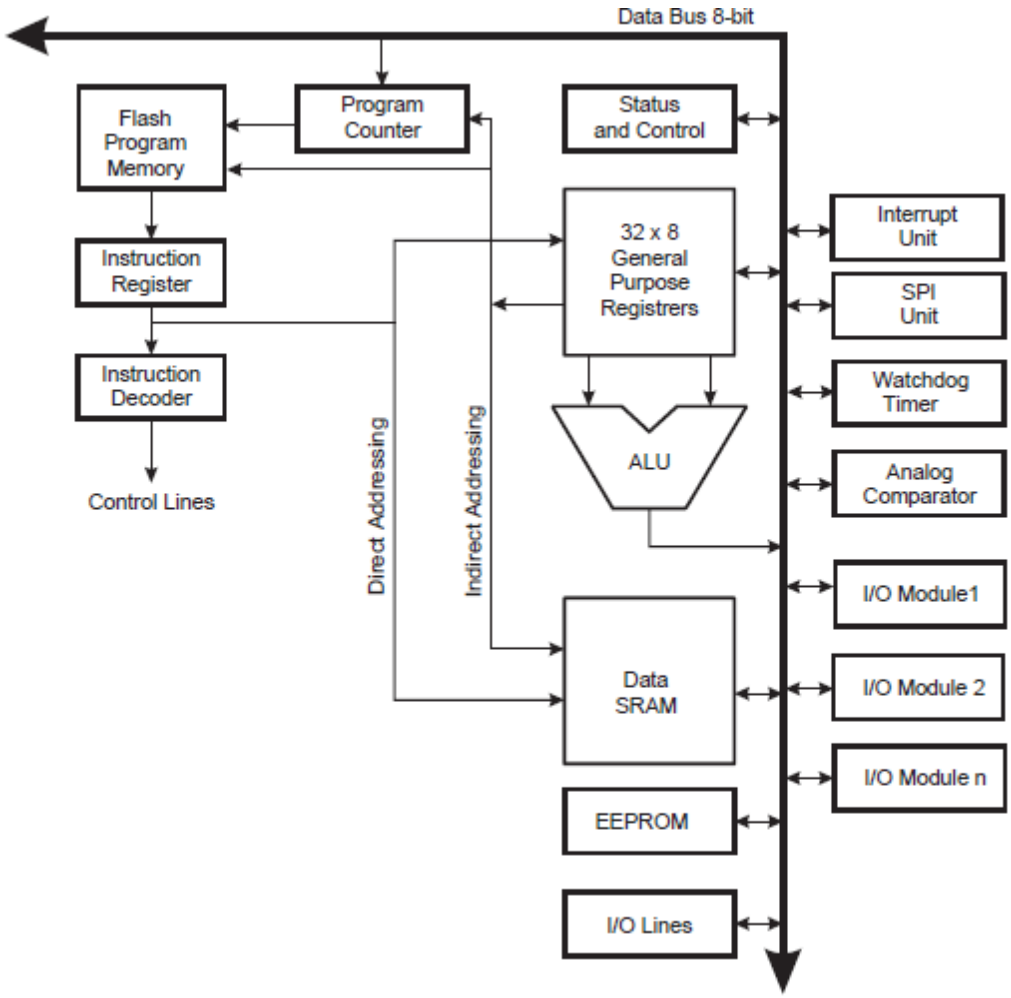

Figure 13: Block Diagram of the AVR MCU Architecture

(d)USART:

The Universal Synchronous and Asynchronous serial Receiver and Transmitter (USART) is a highly flexible serial communication device. The main features are:

- **Full Duplex Operation (Independent Serial Receive and Transmit Registers)**
- **Asynchronous or Synchronous Operation**
- **Master or Slave Clocked Synchronous Operation**
- **High Resolution Baud Rate Generator**
- **Supports Serial Frames with 5, 6, 7, 8, or 9 Data Bits and 1 or 2 Stop Bits**
- **Odd or Even Parity Generation and Parity Check Supported by Hardware**
- **Data Overrun Detection**
- **Framing Error Detection**
- **Noise Filtering Includes False Start Bit Detection and Digital Low Pass Filter**
- **Three Separate Interrupts on TX Complete, TX Data Register Empty, and RX Complete**
- **Multi-processor Communication Mode**
- **Double Speed Asynchronous Communication Mode**

Overview-

A simplified block diagram of the USART transmitter is shown in the figure below. CPU accessible I/O registers and I/O pins are shown in bold.

The dashed boxes in the block diagram separate the three main parts of the USART (listed from the top): Clock Generator, Transmitter and Receiver. Control Registers are shared by all units.

The clock generation logic consists of synchronization logic for external clock input used by synchronous Slave operation, and the baud rate generator. The XCK (Transfer Clock) pin is only used by Synchronous Transfer mode. The Transmitter consists of a single write buffer, a serial Shift Register, parity generator and control logic for handling different serial frame formats. The write buffer allows a continuous transfer of data without any delay between frames. The

Receiver is the most complex part of the USART module due to its clock and data recovery units. The recovery units are used for asynchronous data reception. In addition to the recoveryunits, the receiver includes a parity checker, control logic, a Shift Register and a two levelreceive buffer (UDR). The receiver supports the same frame formats as the transmitter, and can detect frame error, data overrun and parity errors.

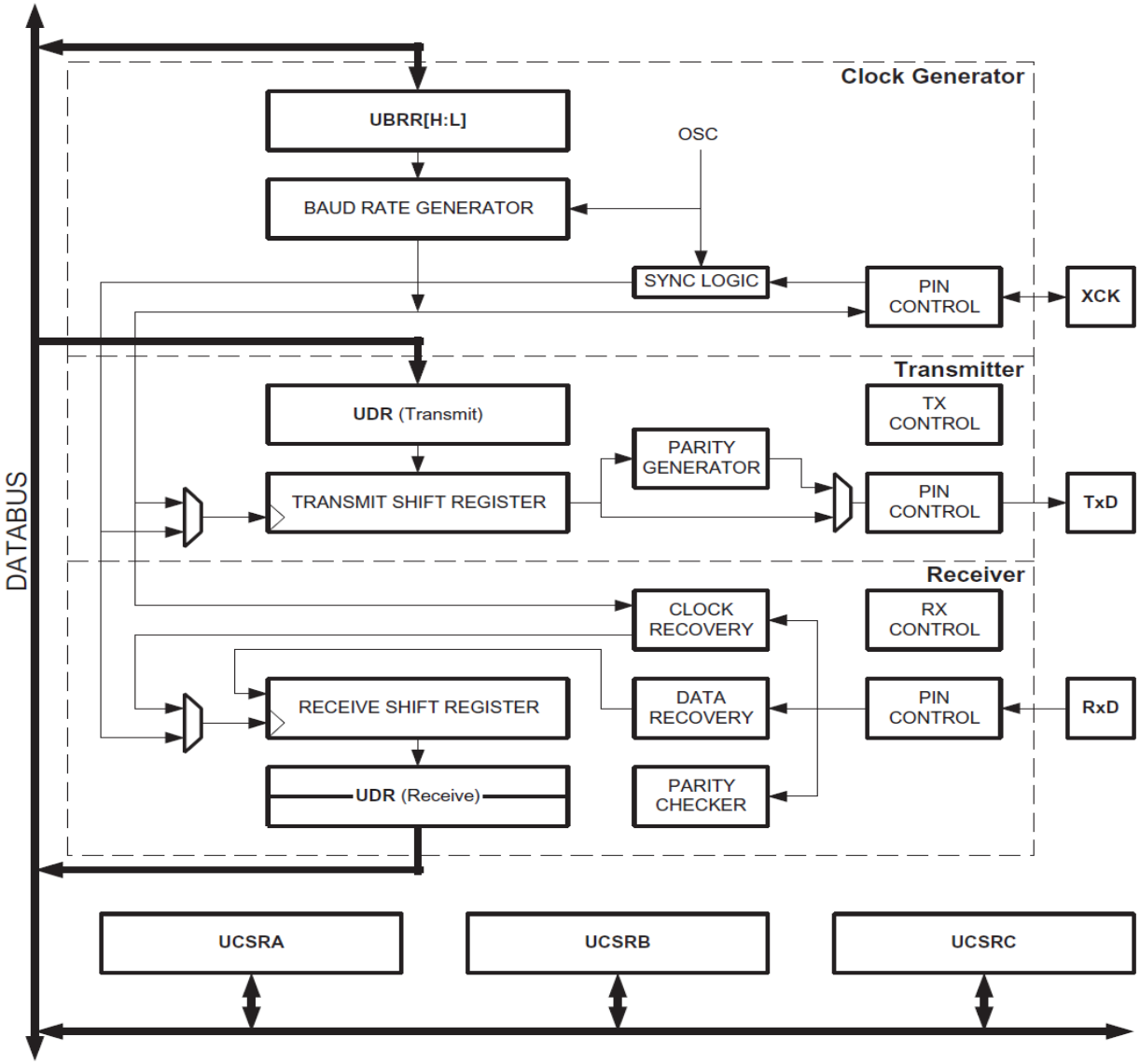

Figure 14: Block diagram for USART

(e)AVR USART vs. AVR:

1)UART – Compatibility-

The USART is fully compatible with the AVR UART regarding:

• Bit locations inside all USART Registers

• Baud Rate Generation

• Transmitter Operation

• Transmit Buffer Functionality

• Receiver Operation

However, the receive buffering has two improvements that will affect the compatibility in some special cases:

• A second Buffer Register has been added. The two Buffer Registers operate as a circular FIFO buffer. Therefore the UDR must only be read once for each incoming data! More important is the fact that the Error Flags (FE and DOR) and the 9th data bit (RXB8) are buffered with the data in the receive buffer. Therefore the status bits must always be read before the UDR Register is read. Otherwise the error status will be lost since the buffer state is lost.

• The receiver Shift Register can now act as a third buffer level. This is done by allowing the received data to remain in the serial Shift Register if the Buffer Registers are full, until a new start bit is detected. The USART is therefore more resistant to Data OverRun(DOR) error conditions. The following control bits have changed name, but have same functionality and register location:

• CHR9 is changed to UCSZ2

• OR is changed to DOR

2) Clock Generation: The clock generation logic generates the base clock for the Transmitter and Receiver. The USART supports four modes of clock operation: Normal Asynchronous, Double Speed Asynchronous, Master Synchronous and Slave Synchronous mode. The UMSEL bit in USART Control and Status Register C (UCSRC) selects between asynchronous and synchronous operation. Double Speed (Asynchronous mode only) is controlled by the U2X found in the UCSRA Register. When using Synchronous mode (UMSEL = 1), the Data Direction Register for the XCK pin (DDR_XCK) controls whether the clock source is internal

(Master mode) or external (Slave mode). The XCK pin is only active when using Synchronous mode.

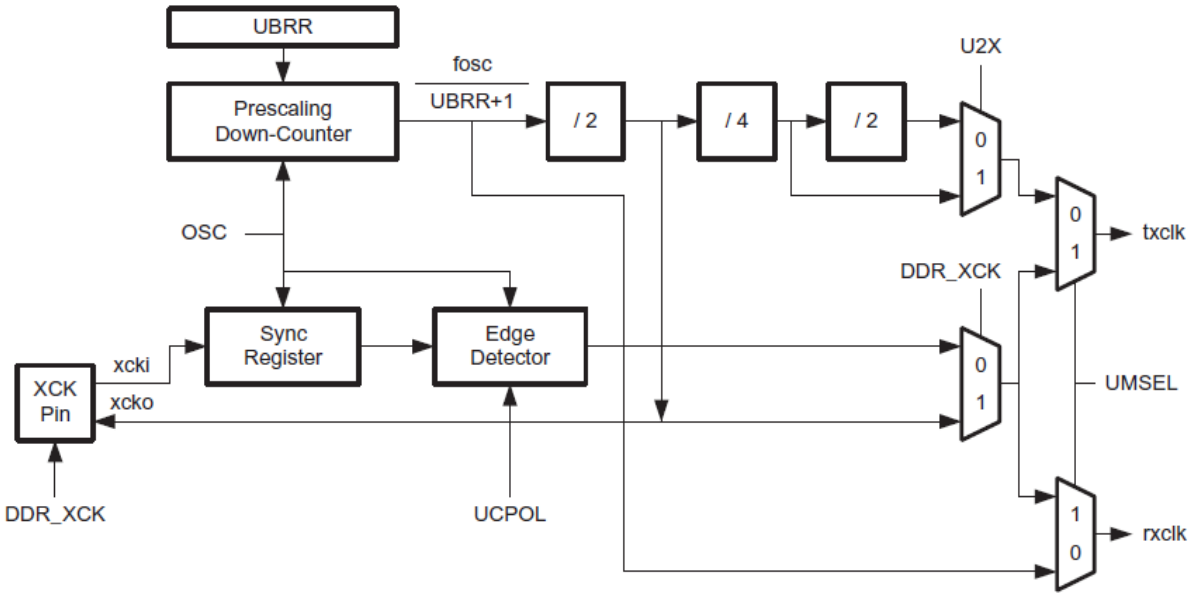

Figure15: Clock Generation logic block diagram

Signal description:

txclkTransmitter clock (Internal Signal).

rxclkReceiver base clock (Internal Signal).

xckiInput from XCK pin (Internal Signal). Used for synchronous Slave operation.

xckoClock output to XCK pin (Internal Signal).Used for synchronous Master operation.

foscXTAL pin frequency (System Clock).

3) Internal Clock Generation – The Baud Rate Generator:

Internal clock generation is used for the asynchronous and the synchronous Master modes of operation. The description in this section refers to **Clock Generation Logic diagram above.** The USART Baud Rate Register (UBRR) and the down-counter connected to it function as a programmableprescaler or baud rate generator. The down-counter, running at system clock (fosc), is loaded with the UBRR value each time the counter has counted down to zero or when the UBRRL Register is written. A clock is generated each time the counter reaches zero. This

clock is the baud rate generator clock output (= fosc/(UBRR+1)). The Transmitter divides the baud rate generator clock output by 2, 8 or 16 depending on mode. The baud rate generator output is used directly by the receiver's clock and data recovery units. However, the recovery units use a state machine that uses 2, 8 or 16 states depending on mode set by the state of the UMSEL, U2X and DDR_XCK bits.

Table 16contains a table of equations for calculating the baud rate (in bits per second) and for calculating the UBRR value for each mode of operation using an internally generated clock source.

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{OSC}}{16(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{OSC}}{8(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{OSC}}{2(UBRR+1)}$	$UBRR = \frac{f_{OSC}}{2BAUD} - 1$

Table 16: Equations for Calculating Baud Rate Register Setting

[Note: The baud rate is defined to be the transfer rate in bit per second (bps).]

BAUD Baud rate (in bits per second, bps)

fOSC System Oscillator clock frequency

UBRR Contents of the UBRRH and UBRRL Registers, (0 - 4095)

(f) Analog to Digital Converter:

Features-

• 10-bit Resolution

• 0.5 LSB Integral Non-linearity

• ±2 LSB Absolute Accuracy

• 13 µs- 260 µs Conversion Time

• Up to 15 kSPS at Maximum Resolution

• 8 Multiplexed Single Ended Input Channels

• 7 Differential Input Channels

• 2 Differential Input Channels with Optional Gain of 10x and 200x

• Optional Left adjustment for ADC Result Readout

• 0 - VCC ADC Input Voltage Range

• Selectable 2.56V ADC Reference Voltage

• Free Running or Single Conversion Mode

• ADC Start Conversion by Auto Triggering on Interrupt Sources

• Interrupt on ADC Conversion Complete

• Sleep Mode Noise Canceller

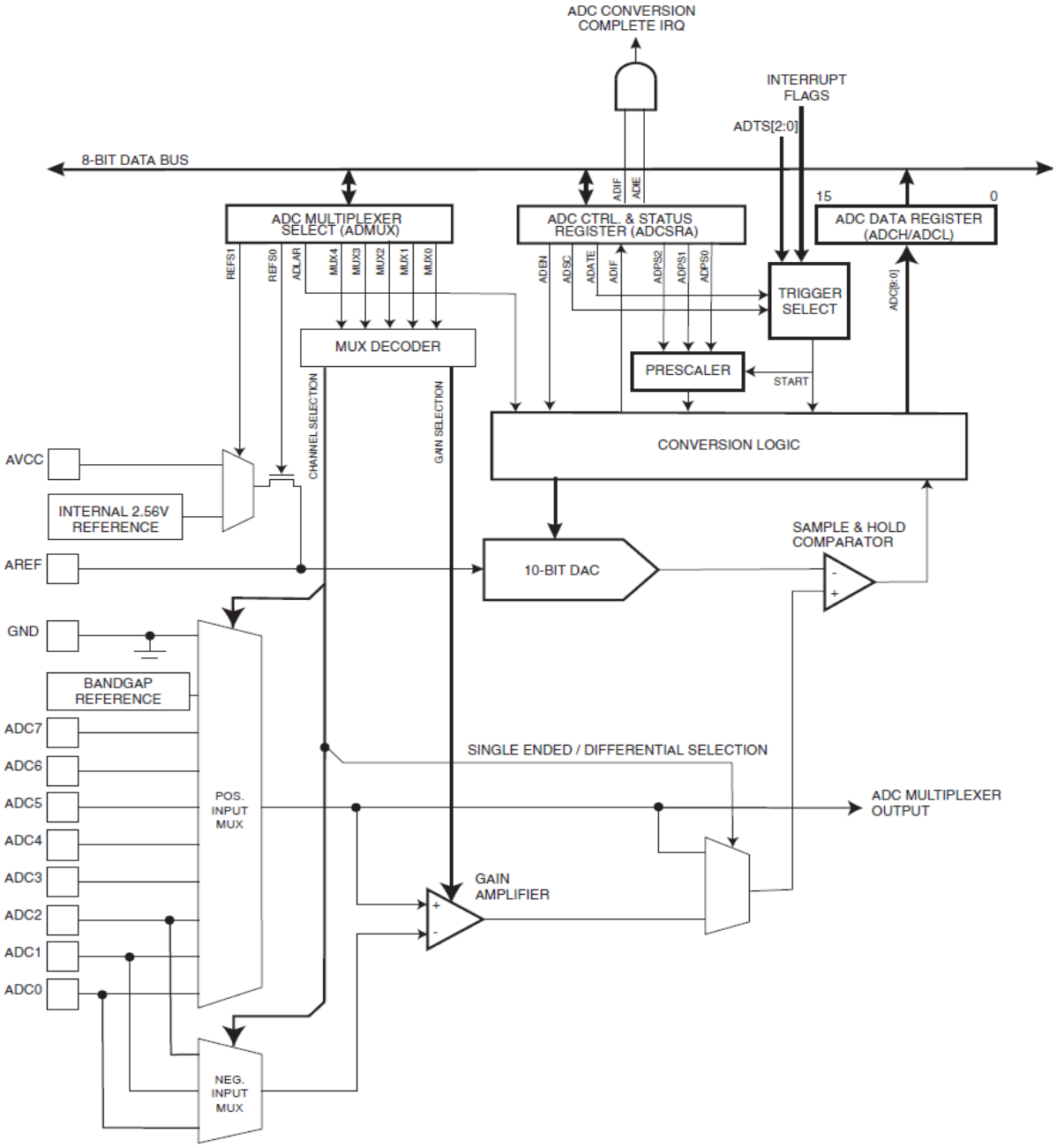

Figure 17: ADC block schematic diagram

The ATmega16 features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows 8 single-ended voltage inputs constructed from the pins of Port A. The single-ended voltage inputs refer to 0V (GND).

The device also supports 16 differential voltage input combinations. Two of the differential inputs (ADC1, ADC0 and ADC3, ADC2) are equipped with a programmable gain stage, providing amplification steps of 0 dB (1x), 20 dB (10x), or 46 dB (200x) on the differential input voltage before the A/D conversion. Seven differential analog input channels share a common negative terminal (ADC1), while any other ADC input can be selected as the positive input terminal. If 1x or 10x gain is used, 8-bit resolution can be expected. If 200x gain is used, 7-bit resolution can be expected. The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion.

The ADC has a separate analog supply voltage pin, AVCC. AVCC must not differ more than ±0.3V from VCC.Internal reference voltages of nominally 2.56V or AVCC are provided On-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance.
The ADC converts an analog input voltage to a 10-bit digital value through successive approximation.

The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, AVCC or an internal 2.56V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register. The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity.

The analog input channel and differential gain are selected by writing to the MUX bits in ADMUX. Any of the ADC input pins, as well as GND and a fixedband gap voltage reference, can be selected as single ended inputs to the ADC. A selection of ADC input pins can be selected as positive and negative inputs to the differential gain amplifier.

If differential channels are selected, the differential gain stage amplifies the voltage difference between the selected input channel pair by the selected gain factor. This amplified value then becomes the analog input to the ADC. If single ended channels are used, the gain amplifier is bypassed altogether.

The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC does not consume power when ADEN is cleared, so it is recommended to switch off the ADC before entering powersaving sleep modes.

The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX.

If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the Data Registers belongs to the same conversion. Once ADCL is read, ADC access to Data Registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL Registers is re-enabled.

The ADC has its own interrupt which can be triggered when a conversion completes. When ADC access to the Data Registers is prohibited between reading of ADCH and ADCL, the interrupt will trigger even if the result is lost.

1)Starting a Conversion:

A single conversion is started by writing a logical one to the ADC Start Conversion bit, ADSC. This bit stays high as long as the conversion is in progress and will be cleared by hardware when the conversion is completed. If a different data channel is selected while a conversion is in progress, the ADC will finish the current conversion before performing the channel change.

Alternatively, a conversion can be triggered automatically by various sources. Auto Triggering is enabled by setting the ADC Auto Trigger Enable bit, ADATE in ADCSRA. The trigger source is selected by setting the ADC Trigger Select bits, ADTS in SFIOR (see description of the ADTS bits for a list of the trigger sources). When a positive edge occurs on the selected trigger signal, the ADC prescaler is reset and a conversion is started. This provides a method of starting conversions at fixed intervals. If the trigger signal still is set when the conversion completes, a new conversion will not be started. If another positive edge occurs on the trigger signal during conversion, the edge will be ignored. Note that an Interrupt Flag will be set even if the specific interrupt is disabled or the global interrupt enable bit in SREG is cleared. A conversion can thus be triggered without causing an interrupt. However, the Interrupt Flag must be cleared in order to trigger a new conversion at the next interrupt event.

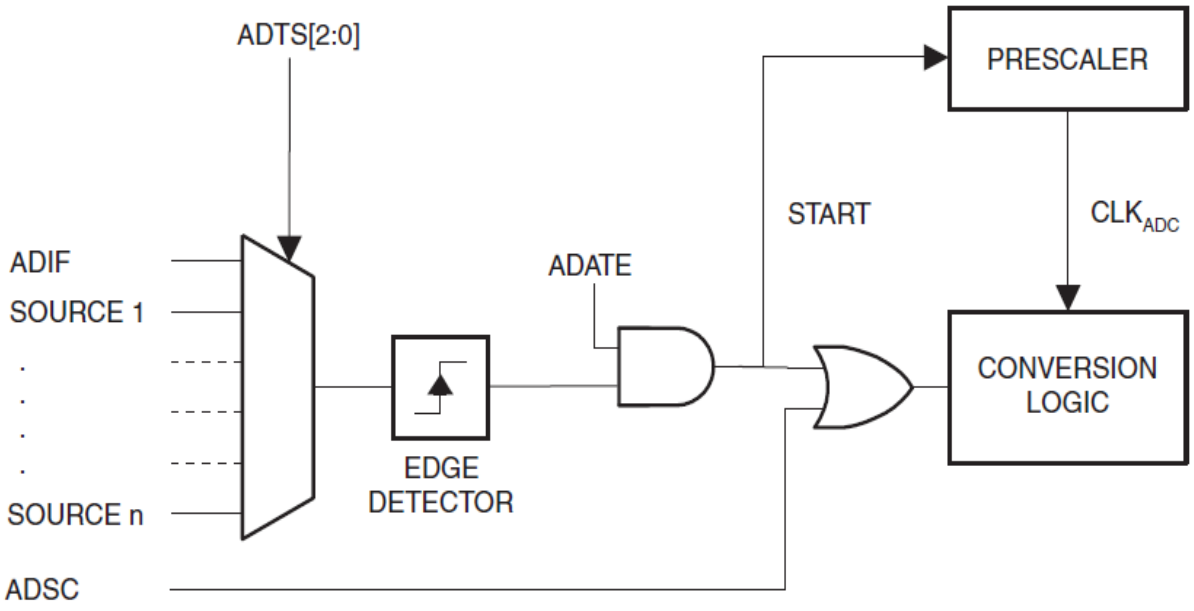

Figure 18: ADC auto trigger logic

Using the ADC Interrupt Flag as a trigger source makes the ADC start a new conversion as soon as the ongoing conversion has finished. The ADC then operates in Free Running mode, constantly sampling and updating the ADC Data Register. The first conversion must be started by writing a logical one to the ADSC bit in ADCSRA. In this mode the ADC will perform successive conversions independently of whether the ADC Interrupt Flag, ADIF is cleared or not.

If Auto Triggering is enabled, single conversions can be started by writing ADSC in ADCSRA to one. ADSC can also be used to determine if a conversion is in progress. The ADSC bit will be read as one during a conversion, independently of how the conversion was started.

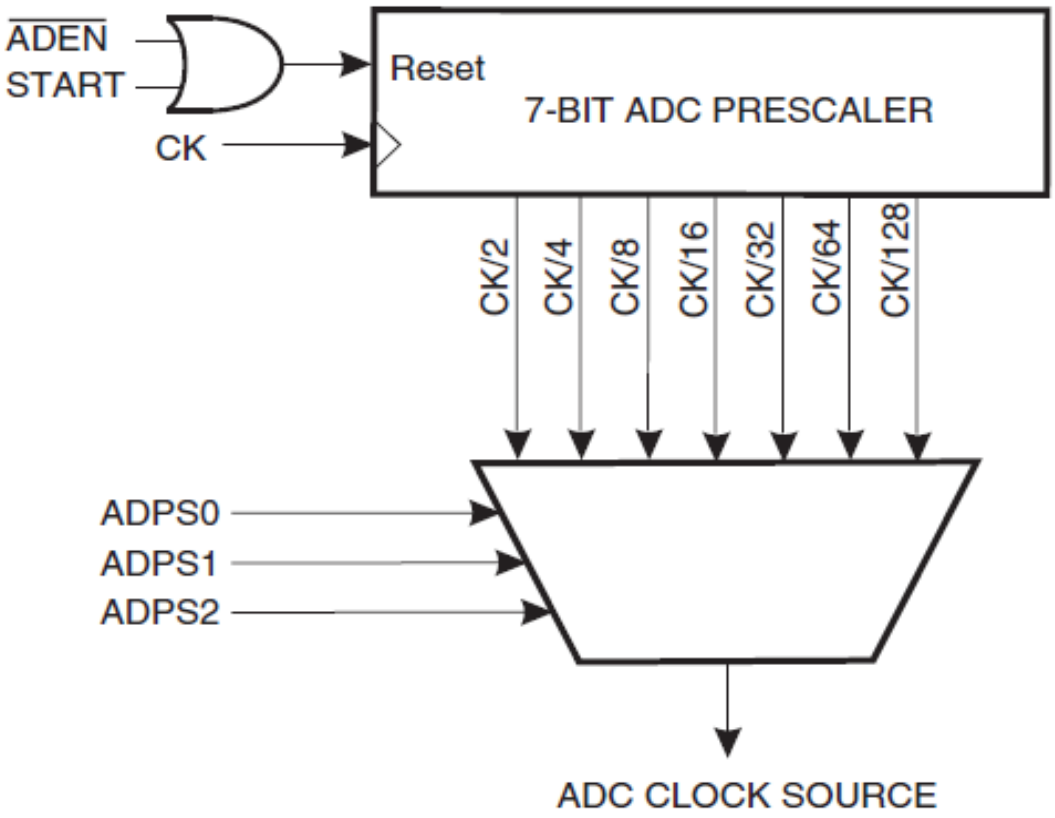

Figure 19:ADC prescaler

By default, the successive approximation circuitry requires an input clock frequency between 50 kHz and 200 kHz to get maximum resolution. If a lower resolution than 10 bits is needed, the input clock frequency to the ADC can be higher than 200 kHz to get a higher sample rate. The ADC module contains a prescaler, which generates an acceptable ADC clock frequency from any CPU frequency above 100 kHz. The prescaling is set by the ADPS bits in ADCSRA. The prescaler starts counting from the moment the ADC is switched on by setting the ADEN bit in ADCSRA. The prescaler keeps running for as long as the ADEN bit is set, and is continuously reset when ADEN is low. When initiating a single ended conversion by setting the ADSC bit in ADCSRA, the conversion starts at the following rising edge of the ADC clock cycle.

A normal conversion takes 13 ADC clock cycles. The first conversion after the ADC is switched on (ADEN in ADCSRA is set) takes 25 ADC clock cycles in order to initialize the analog circuitry.

The actual sample-and-hold takes place 1.5 ADC clock cycles after the start of a normal conversion and 13.5 ADC clock cycles after the start of a first conversion. When a conversion is complete, the result is written to the ADC Data Registers, and ADIF is set. In single conversion mode, ADSC is cleared simultaneously. The software may then set ADSC again, and a new conversion will be initiated on the first rising ADC clock edge. When Auto Triggering is used, the prescaler is reset when the trigger event occurs. This assures a fixed delay from the trigger event to the start of conversion. In this mode, the sample-and-hold takes place 2 ADC clock cycles after the rising edge on the trigger source signal. Three additional CPU clock cycles are used for synchronization logic. When using Differential mode, along with Auto triggering from a source other than the ADC Conversion Complete, each conversion will require 25 ADC clocks. This is because the ADC must be disabled and re-enabled after every conversion.

In Free Running mode, a new conversion will be started immediately after the conversion completes, while ADSC remains high.

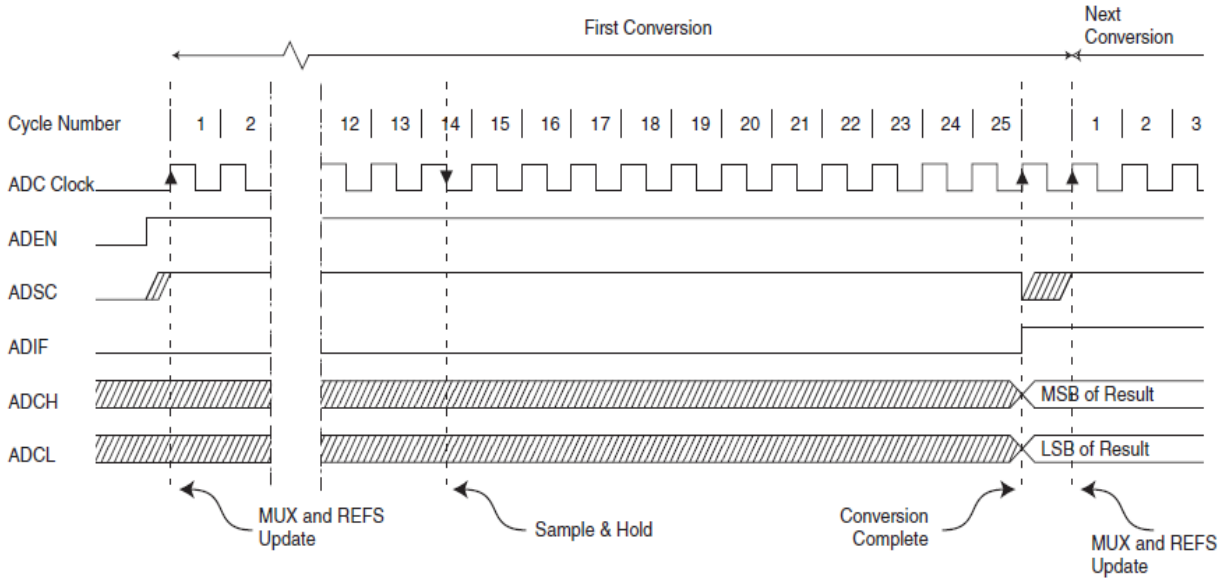

Figure 20: ADC timing diagram,First conversion(single conversion mode)

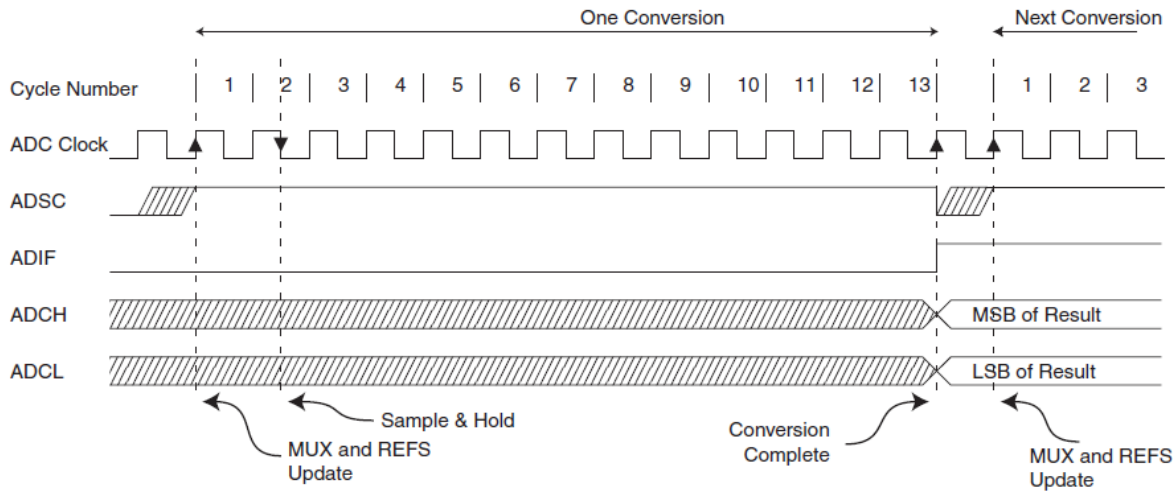

Figure 21: ADC timing diagram,single conversion.

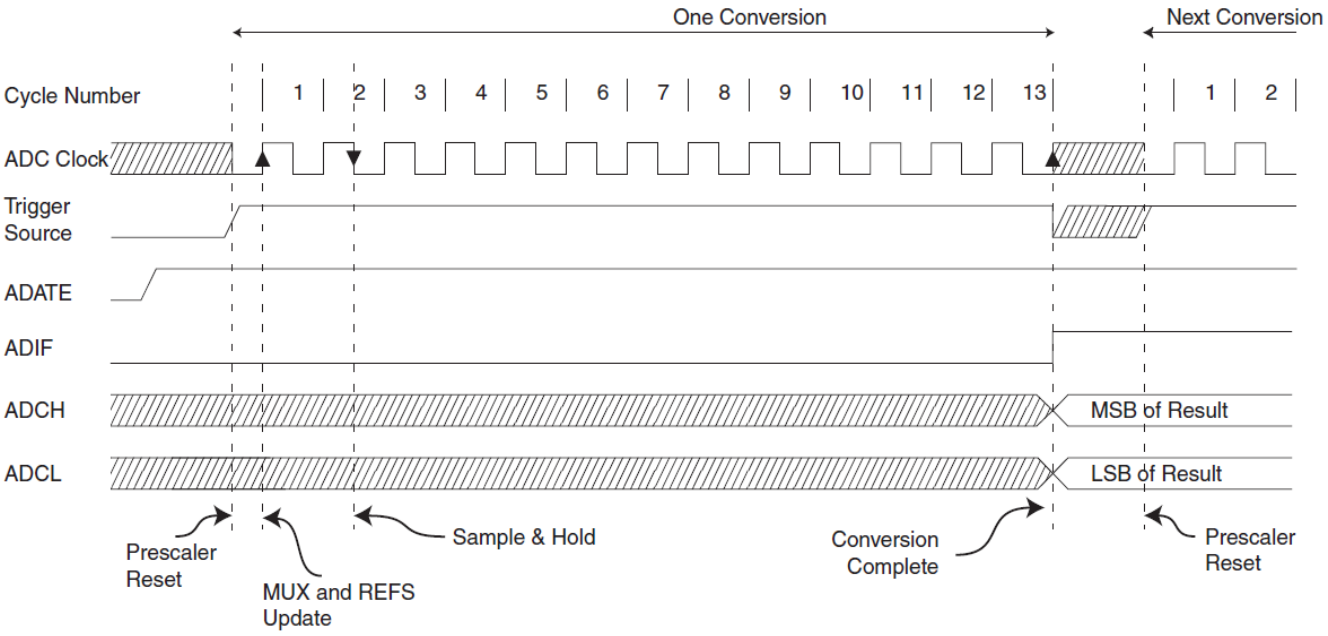

Figure 21: ADC Timing Diagram, Auto Triggered Conversion

Condition	Sample & Hold (Cycles from Start of Conversion)	Conversion Time (Cycles)
First conversion	13.5	25
Normal conversions, single ended	1.5	13
Auto Triggered conversions	2	13.5
Normal conversions, differential	1.5/2.5	13/14

Table 22: ADC Conversion Time

2) Analog Input Circuitry:

The Analog Input Circuitry for single ended channels is illustrated in figure below. An analog source applied to ADCn is subjected to the pin capacitance and input leakage of that pin, regardless of whether that channel is selected as input for the ADC. When the channel is selected, the source must drive the S/H capacitor through the series resistance (combined resistance in the input path).

The ADC is optimized for analog signals with an output impedance of approximately 10 kΩor less. If such a source is used, the sampling time will be negligible. If a source with higher impedance is used, the sampling time will depend on how long time the source needs to charge the S/H capacitor, with can vary widely. The user is recommended to only use low impedant sources with slowly varying signals, since this minimizes the required charge transfer to the S/H capacitor.

If differential gain channels are used, the input circuitry looks somewhat different, although source impedances of a few hundred kΩor less is recommended.
Signal components higher than the Nyquist frequency (fADC/2) should not be present for either kind of channels, to avoid distortion from unpredictable signal convolution. The user is advised to remove high frequency components with a low-pass filter before applying the signals as inputs to the ADC.

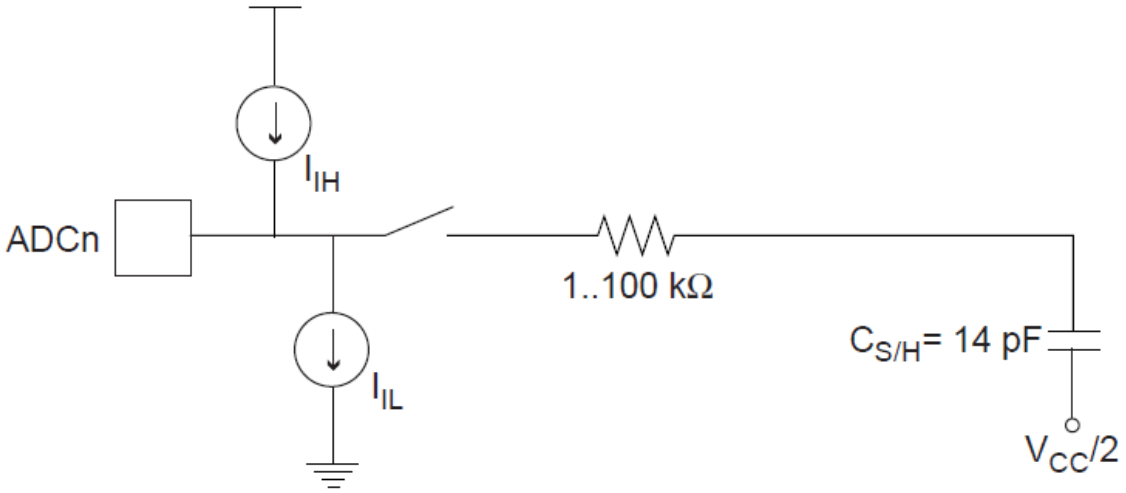

Figure 23: Analog input circuitry

3) Analog Noise Canceling Techniques:

Digital circuitry inside and outside the device generates EMI which might affect the accuracy of analog measurements. If conversion accuracy is critical, the noise level can be reduced by applying the following techniques:

1. Keep analog signal paths as short as possible. Keep them well away from high speed switching digital tracks.
2. The AVCC pin on the device should be connected to the digital VCC supply voltage via an LC network as shown in **Figure 24.**
3. Use the ADC noise canceler function to reduce induced noise from the CPU.
4. If any ADC port pins are used as digital outputs, it is essential that these do not switch while a conversion is in progress.

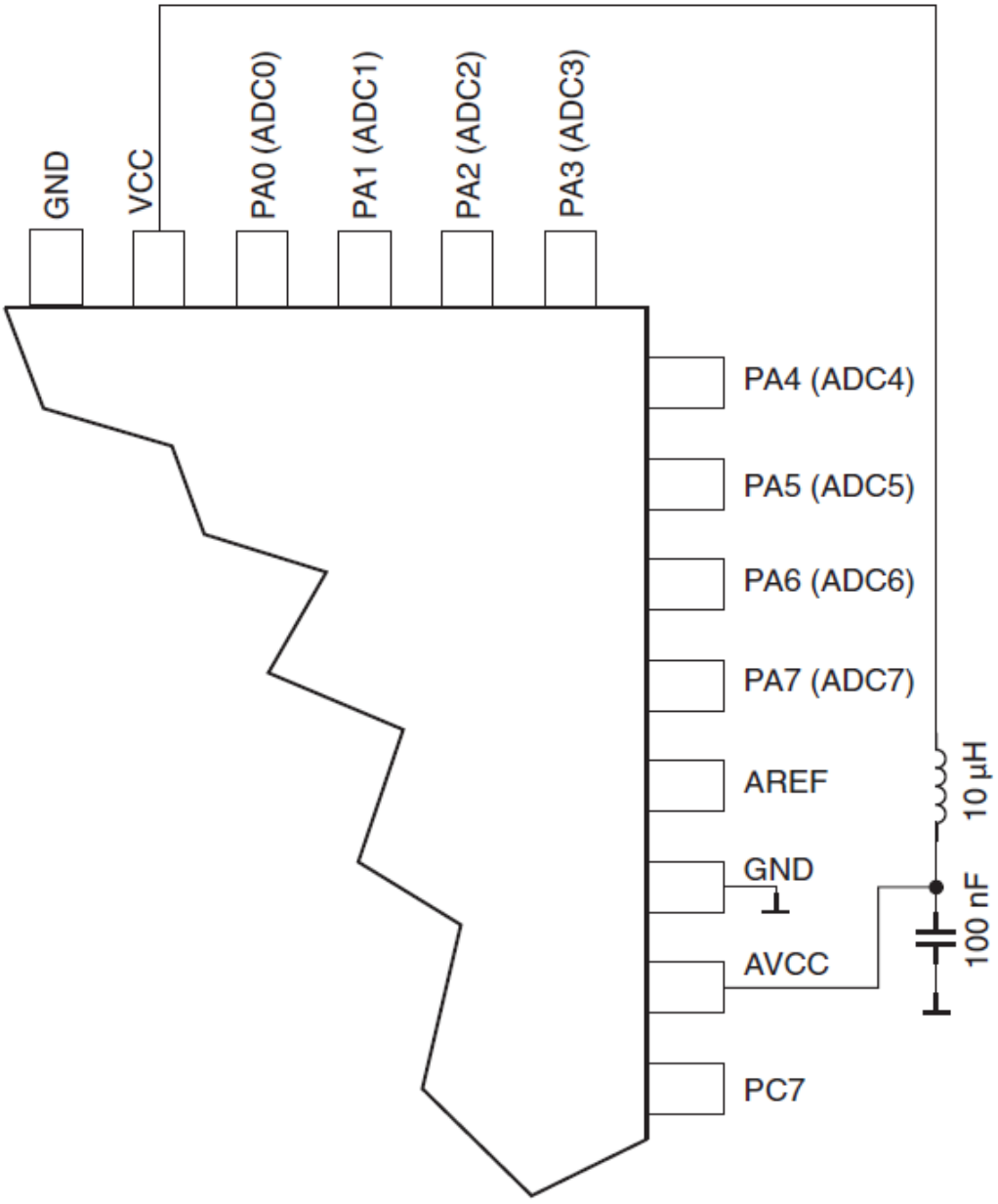

Figure 24:ADC Power Connections

4) ADC Accuracy Definitions:

An n-bit single-ended ADC converts a voltage linearly between GND and VREF in 2n steps (LSBs). The lowest code is read as 0, and the highest code is read as 2n-1.

Several parameters describe the deviation from the ideal behavior:

• Offset: The deviation of the first transition (0x000 to 0x001) compared to the ideal transition (at 0.5 LSB). Ideal value: 0 LSB.

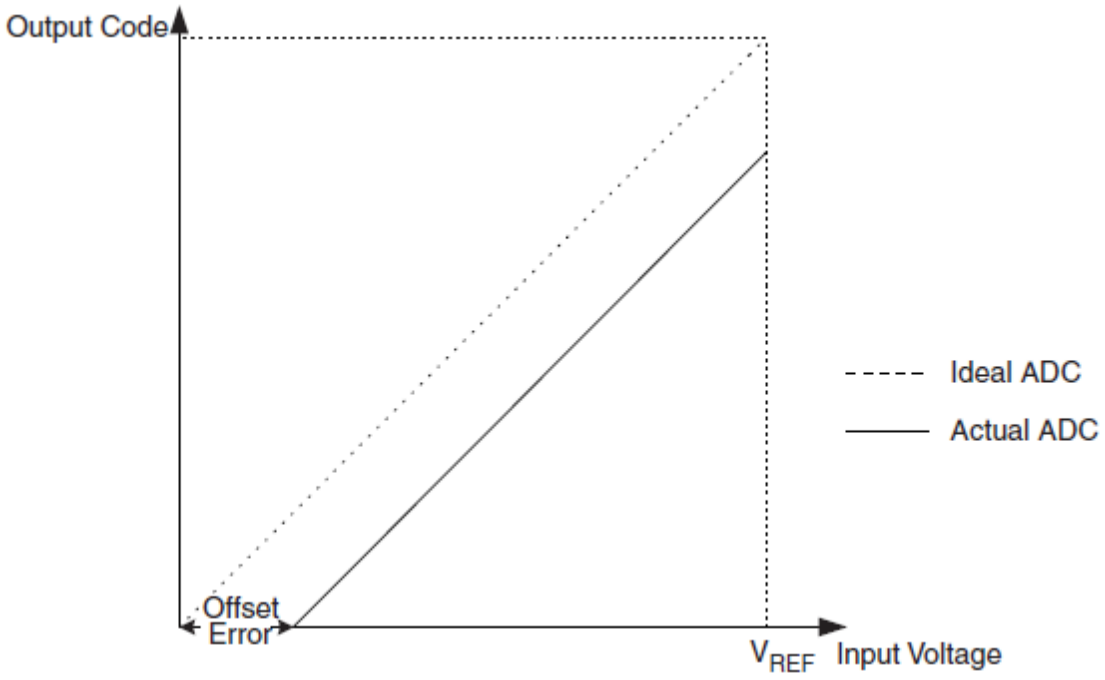

Figure 25: Offset error

• Gain Error: After adjusting for offset, the Gain Error is found as the deviation of the last transition (0x3FE to 0x3FF) compared to the ideal transition (at 1.5 LSB below maximum). Ideal value: 0 LSB

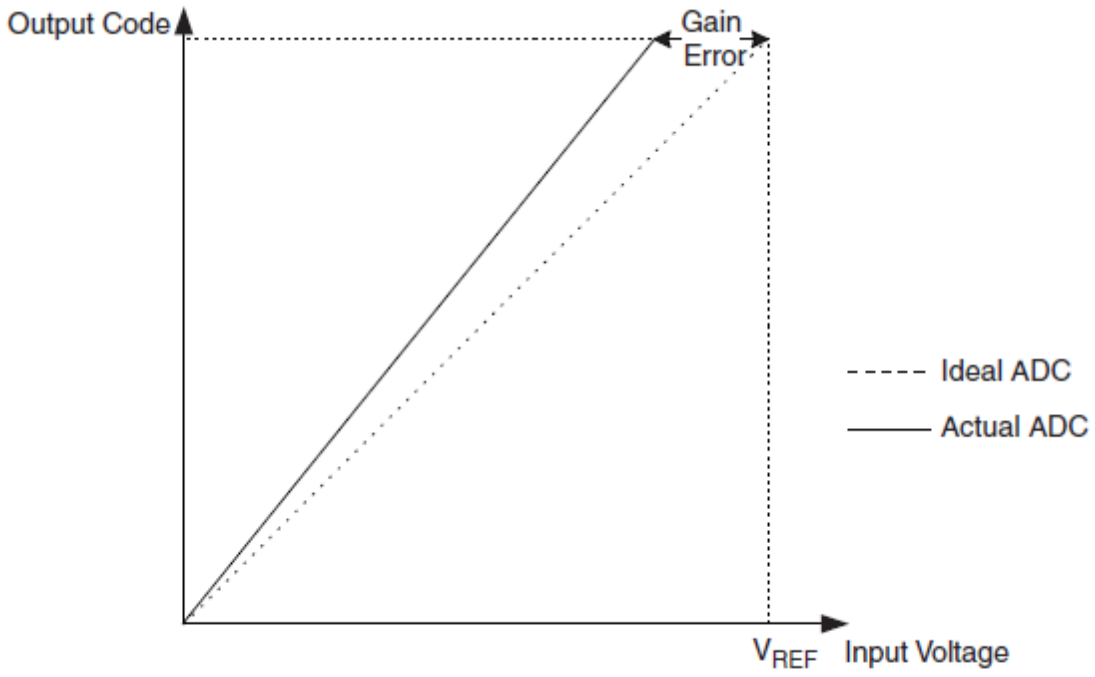

Figure 26: Gain error

• Integral Non-linearity (INL): After adjusting for offset and gain error, the INL is the maximum deviation of an actual transition compared to an ideal transition for any code. Ideal value: 0 LSB.

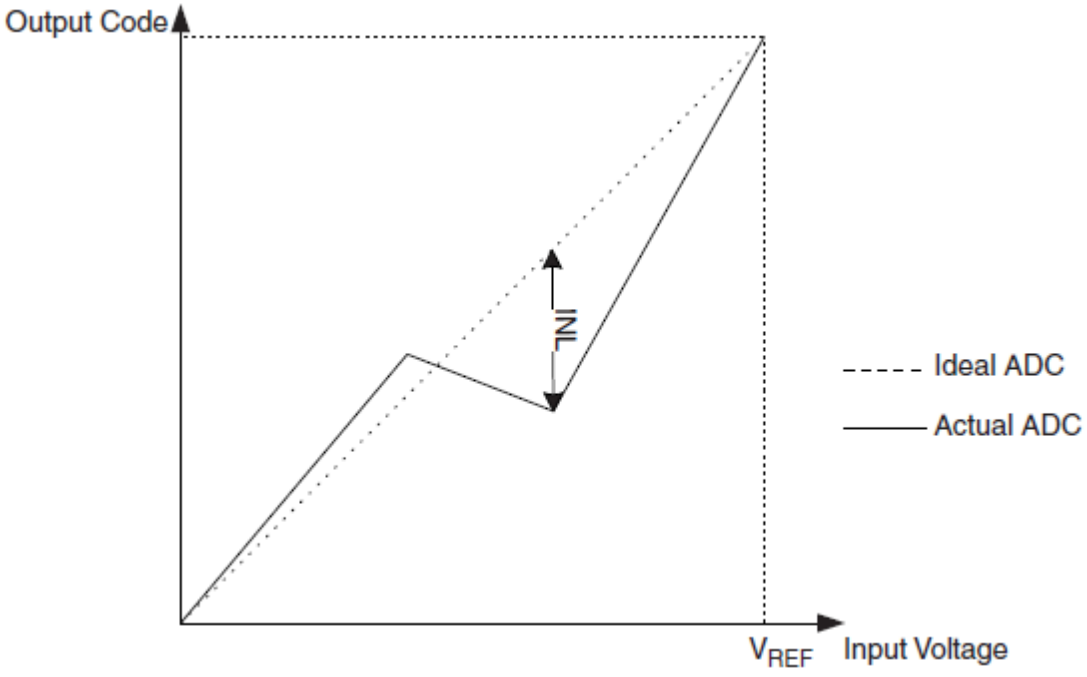

Figure 27: Integral Non-linearity

• Differential Non-linearity (DNL): The maximum deviation of the actual code width (the interval between two adjacent transitions) from the ideal code width (1 LSB). Ideal value: 0 LSB.

• Quantization Error: Due to the quantization of the input voltage into a finite number of codes, a range of input voltages (1 LSB wide) will code to the same value. Always ±0.5 LSB.

• Absolute Accuracy: The maximum deviation of an actual (unadjusted) transition compared to an ideal transition for any code. This is the compound effect of Offset, Gain Error, Differential Error, Non-linearity, and Quantization Error. Ideal value: ±0.5 LSB.

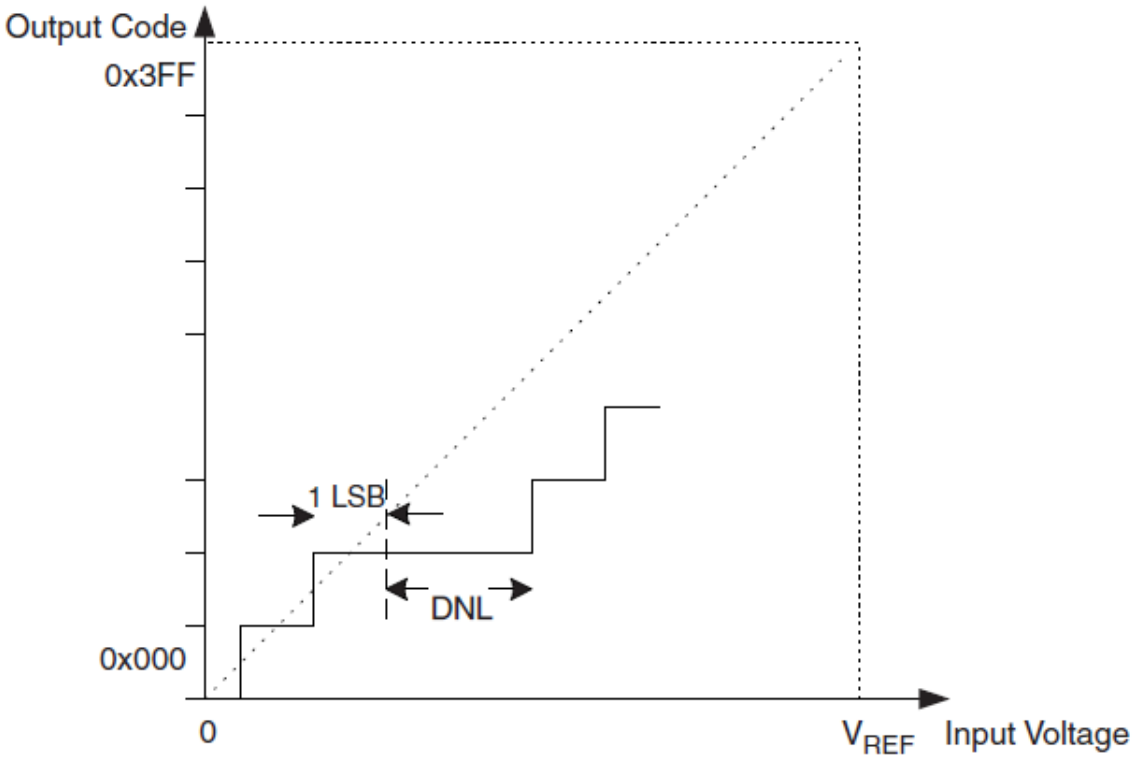

Figure 28: Differential Non-linearity

5) ADC Conversion Result:

After the conversion is complete (ADIF is high), the conversion result can be found in the ADC Result Registers (ADCL, ADCH).

For single ended conversion, the result is

$$ADC = \frac{V_{IN} \cdot 1024}{V_{REF}}$$

where VIN is the voltage on the selected input pin and VREF the selected voltage reference.

If differential channels are used, the result is

$$ADC = \frac{(V_{POS} - V_{NEG}) \cdot GAIN \cdot 512}{V_{REF}}$$

Where, VPOS is the voltage on the positive input pin, VNEG the voltage on the negative input pin,

GAIN the selected gain factor, and VREF the selected voltage reference. The result is presented in two's complement form, from 0x200 (-512d) through 0x1FF (+511d). Note that if the user wants to perform a quick polarity check of the results, it is sufficient to read the MSB of the result.

(ADC9 in ADCH), If this bit is one, the result is negative, and if this bit is zero, the result is positive.

Figure 29shows the decoding of the differential input range.

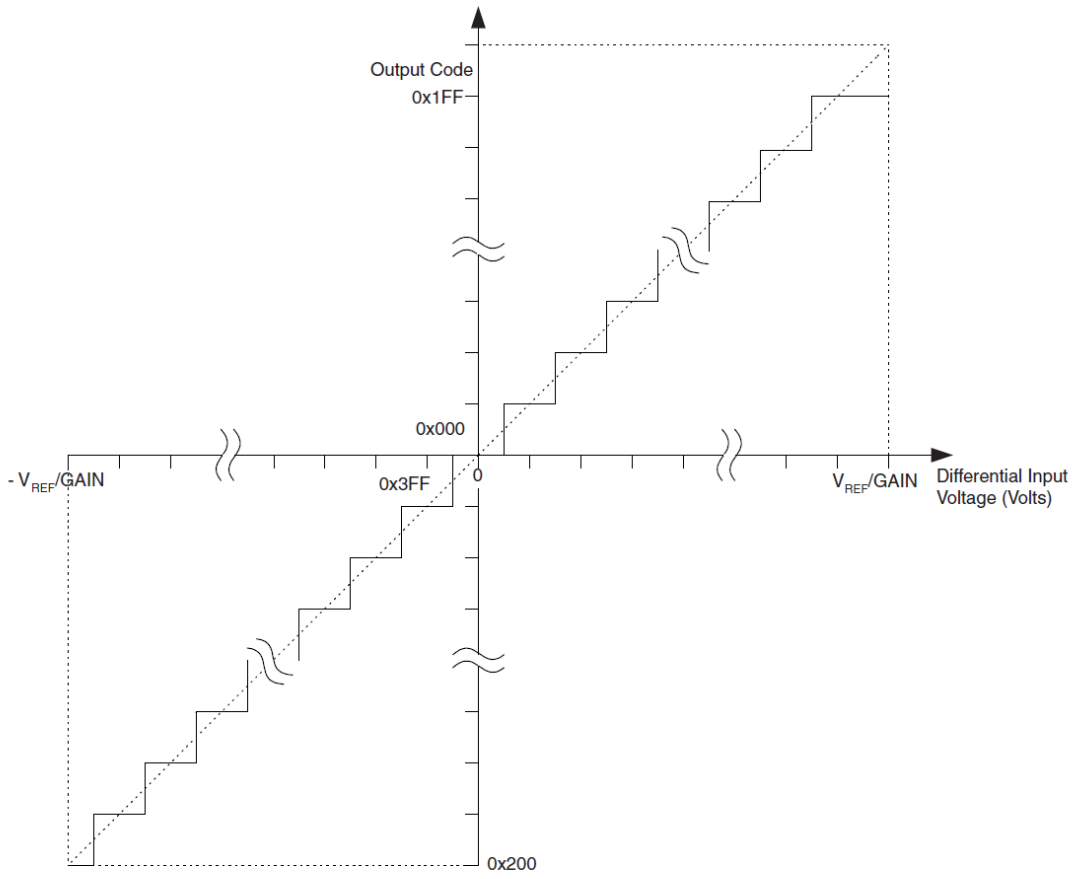

Figure 29: Differential Measurement Range

V_{ADCn}	Read code	Corresponding Decimal Value
V_{ADCm} + V_{REF}/GAIN	0x1FF	511
V_{ADCm} + 511/512 V_{REF}/GAIN	0x1FF	511
V_{ADCm} + 510/512 V_{REF}/GAIN	0x1FE	510
...	...	...
V_{ADCm} + 1/512 V_{REF}/GAIN	0x001	1
V_{ADCm}	0x000	0
V_{ADCm} - 1/512 V_{REF}/GAIN	0x3FF	-1
...	...	...
V_{ADCm} - 511/512 V_{REF}/GAIN	0x201	-511
V_{ADCm} - V_{REF}/GAIN	0x200	-512

Table 30: Correlation between Input Voltage and Output Codes

(g) Fuse Settings:

Fuse Settings done to change oscillator to external 8MHz crystal.

Reasons for choosing 8MHz Crystal Value

8 MHz is the highest accommodated by atmega16L-8PU micro controller.

8 MHz External Crystal is chosen because it gives fairly low error i.e. +/-0.2 at baud rates of 9600 and 19200.

Baud Rate (bps)	f_{osc} = 8.0000 MHz				f_{osc} = 11.0592 MHz				f_{osc} = 14.7456 MHz			
	U2X = 0		U2X = 1		U2X = 0		U2X = 1		U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error	UBRR	Error
2400	207	0.2%	416	-0.1%	287	0.0%	575	0.0%	383	0.0%	767	0.0%
4800	103	0.2%	207	0.2%	143	0.0%	287	0.0%	191	0.0%	383	0.0%
9600	51	0.2%	103	0.2%	71	0.0%	143	0.0%	95	0.0%	191	0.0%
14.4k	34	-0.8%	68	0.6%	47	0.0%	95	0.0%	63	0.0%	127	0.0%
19.2k	25	0.2%	51	0.2%	35	0.0%	71	0.0%	47	0.0%	95	0.0%
28.8k	16	2.1%	34	-0.8%	23	0.0%	47	0.0%	31	0.0%	63	0.0%
38.4k	12	0.2%	25	0.2%	17	0.0%	35	0.0%	23	0.0%	47	0.0%
57.6k	8	-3.5%	16	2.1%	11	0.0%	23	0.0%	15	0.0%	31	0.0%
76.8k	6	-7.0%	12	0.2%	8	0.0%	17	0.0%	11	0.0%	23	0.0%
115.2k	3	8.5%	8	-3.5%	5	0.0%	11	0.0%	7	0.0%	15	0.0%
230.4k	1	8.5%	3	8.5%	2	0.0%	5	0.0%	3	0.0%	7	0.0%
250k	1	0.0%	3	0.0%	2	-7.8%	5	-7.8%	3	-7.8%	6	5.3%
0.5M	0	0.0%	1	0.0%	–	–	2	-7.8%	1	-7.8%	3	-7.8%
1M	–	–	0	0.0%	–	–	–	–	0	-7.8%	1	-7.8%
Max (1)	0.5 Mbps		1 Mbps		691.2 Kbps		1.3824 Mbps		921.6 Kbps		1.8432 Mbps	

Table 31: UBRR settings table

1) Fuse settings for the atmega16 microcontroller used in our work was found from the fuse calculator:

The settings are given below as shown in the images:

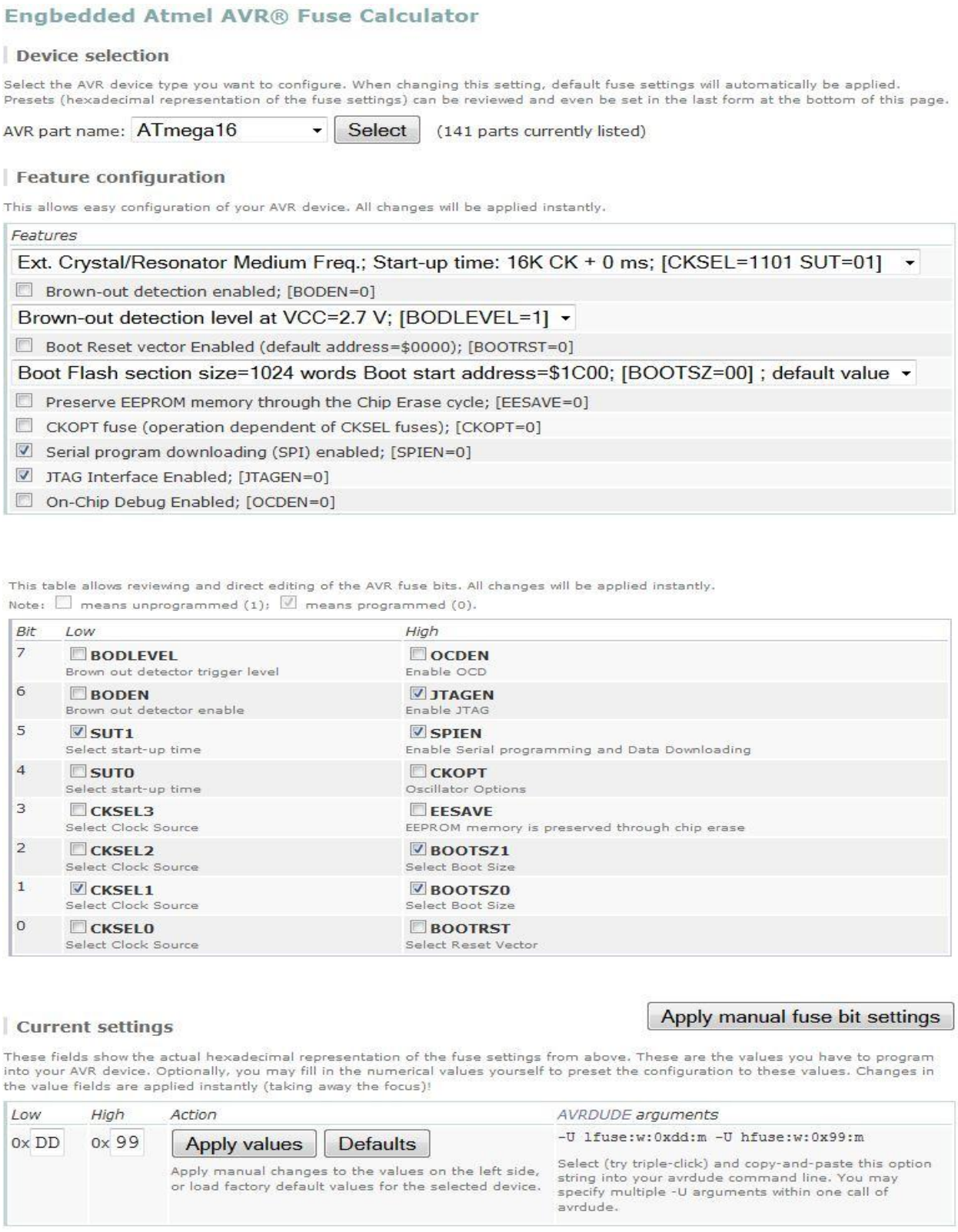

Figure 32: AVR fuse calculator

Chapter 4: Coding

Programming language C is used to code and to embed the code onto the microcontroller winAVR software is used.

WinAVR™ is a suite of executable, open source software development tools for the Atmel AVR series of RISC microprocessors hosted on the Windows platform. It includes the GNU GCC compiler for C and C++.

WinAVRMakefile Settings:

```
MCU Type: Atmega16
Programmer: usbasp *
Port: usb
F_CPU = 8000000 *
* Manually changed
```

the 'Main file name is set without .c extension and everything else is set as default values.

The coding that is punched by winAVR to the microcontroller for single sensor is given below

```
#include <avr\io.h>
#include <util\delay.h>
#include <stdio.h>


#define F_CPU    8000000UL       //Your clock speed in Hz (8Mhz here)
#define LOOP_CYCLES      8       //Number of cycles that the loop takes
#define us(num) (num/(LOOP_CYCLES*(1/(F_CPU/1000000.0))))

#define USART_BAUDRATE 9600
#define BAUD_PRESCALE (((F_CPU/(USART_BAUDRATE*16UL)))-1)

#define THERM_CMD_CONVERTTEMP 0x44
#define THERM_CMD_RSCRATCHPAD 0xbe
#define THERM_CMD_WSCRATCHPAD 0x4e
#define THERM_CMD_CPYSCRATCHPAD 0x48
#define THERM_CMD_RECEEPROM 0xb8
#define THERM_CMD_RPWRSUPPLY 0xb4
#define THERM_CMD_SEARCHROM 0xf0
#define THERM_CMD_READROM 0x33
#define THERM_CMD_MATCHROM 0x55
#define THERM_CMD_SKIPROM 0xcc
#define THERM_CMD_ALARMSEARCH 0xec
/* Thermometer Connections (At your choice) */
#define THERM_PORT PORTB
#define THERM_DDR DDRB
#define THERM_PIN PINB
```

```
#define THERM_DQ PB0
/* Utils */
#define THERM_INPUT_MODE() THERM_DDR&=~(1<<THERM_DQ)
#define THERM_OUTPUT_MODE() THERM_DDR|=(1<<THERM_DQ)
#define THERM_LOW() THERM_PORT&=~(1<<THERM_DQ)
#define THERM_HIGH() THERM_PORT|=(1<<THERM_DQ)
#define THERM_DECIMAL_STEPS_12BIT 625 //.0625

#define sbi(x,y) x |= _BV(y)                    //set bit
#define cbi(x,y) x &= ~(_BV(y))                 //clear bit



inline __attribute__((gnu_inline)) void therm_delay(uint16_t delay){
while(delay--) asm volatile("nop");
}



uint8_t therm_reset(){
uint8_t i;

//Pull line low and wait for 480uS
THERM_LOW();
THERM_OUTPUT_MODE();
therm_delay(us(480));


//Release line and wait for 60uS
THERM_INPUT_MODE();
therm_delay(us(60));


//Store line value and wait until the completion of 480uS period
i=(THERM_PIN & (1<<THERM_DQ));
therm_delay(us(420));


//Return the value read from the presence pulse (0=OK, 1=WRONG)
return i;
}


voidtherm_write_bit(uint8_t bit){
//Pull line low for 1uS
THERM_LOW();
THERM_OUTPUT_MODE();
therm_delay(us(1));


//If we want to write 1, release the line (if not will keep low)
if(bit) THERM_INPUT_MODE();
//Wait for 60uS and release the line
therm_delay(us(60));


THERM_INPUT_MODE();
```

```
}


uint8_t therm_read_bit(void){
uint8_t bit=0;
//Pull line low for 1uS
THERM_LOW();
THERM_OUTPUT_MODE();
therm_delay(us(1));


//Release line and wait for 14uS
THERM_INPUT_MODE();
therm_delay(us(14));


//Read line value
if(THERM_PIN&(1<<THERM_DQ)) bit=1;
//Wait for 45uS to end and return read value
therm_delay(us(45));


return bit;
}



////FOR READ/WRITE A BYTE/////////////////////////
uint8_t therm_read_byte(void){
uint8_t i=8, n=0;
while(i--){

//Shift one position right and store read value

n>>=1;
n|=(therm_read_bit()<<7);
}
return n;
}


voidtherm_write_byte(uint8_t byte){
uint8_t i=8;
while(i--){
//Write actual bit and shift one position right to make the next bit ready

therm_write_bit(byte&1);
byte>>=1;
}
}



voidUSART_Init( unsigned intubrr)
{
/* Set baud rate *///
UBRRH = (BAUD_PRESCALE>>8);
```

```
UBRRL = BAUD_PRESCALE;
/* Enable receiver and transmitter */
UCSRB = (1<<RXEN)|(1<<TXEN);
/* Set frame format: 8data, 1stop bit */
UCSRC = (1<<URSEL)|(1<<UCSZ0)|(1<<UCSZ1);
}

voidUSART_Transmit(char stringchar)
{
/* Wait for empty transmit buffer */
while ( !( UCSRA & (1<<UDRE)) )
;
/* Put data into buffer, sends the data */
UDR = stringchar;
}


voidUSART_TxString(const char StringPtr[])
{
while (*StringPtr != 0x00)
   {
USART_Transmit(*StringPtr);
StringPtr++;
   }
}


unsigned char USART_Receive( void )
{
/* Wait for data to be received */
while ( !(UCSRA & (1<<RXC)) )
;
/* Get and return received data from buffer */
return UDR;
}




void main()
{
        const char *buffer[20];
        uint8_t temperature[2];
        int8_t digit;
        uint16_t decimal;

USART_Init (BAUD_PRESCALE);

while(1){

// Buffer length must be at least 12bytes long! ["+XXX.XXXX C"]
        uint8_t temperature[2];
        int8_t digit;
        uint16_t decimal;

//Reset, skip ROM and start temperature conversion
```

```
        therm_reset();
        therm_write_byte(THERM_CMD_SKIPROM);
        therm_write_byte(THERM_CMD_CONVERTTEMP);

//Wait until conversion is complete
        while(!therm_read_bit());

//Reset, skip ROM and send command to read Scratchpad
                therm_reset();
                therm_write_byte(THERM_CMD_SKIPROM);
                therm_write_byte(THERM_CMD_RSCRATCHPAD);

//Read Scratchpad (only 2 first bytes)
                temperature[0]=therm_read_byte();
                temperature[1]=therm_read_byte();
                therm_reset();

//Store temperature integer digits and decimal digits
                digit=temperature[0]>>4;
                digit|=(temperature[1]&0x7)<<4;

//Store decimal digits
                decimal=temperature[0]&0xf;
                decimal*=THERM_DECIMAL_STEPS_12BIT;

//Format temperature into a string [+XXX.XXXX C]

        sprintf(buffer, "%+d.%04u C", digit, decimal);

//Transmit the temperature string to PC via UART connection
        USART_TxString(buffer);
}


}
```

4.1 Showing output on 16*2 LCD:

Before making microcontroller to pc connection, we developed a a test run circuit with a 16*2 LCD to check whether it works accurately or not.We used one temperature sensor instead of many for the test run. The coding was written in C programming language. The figure shows the whole circuit for LCD.

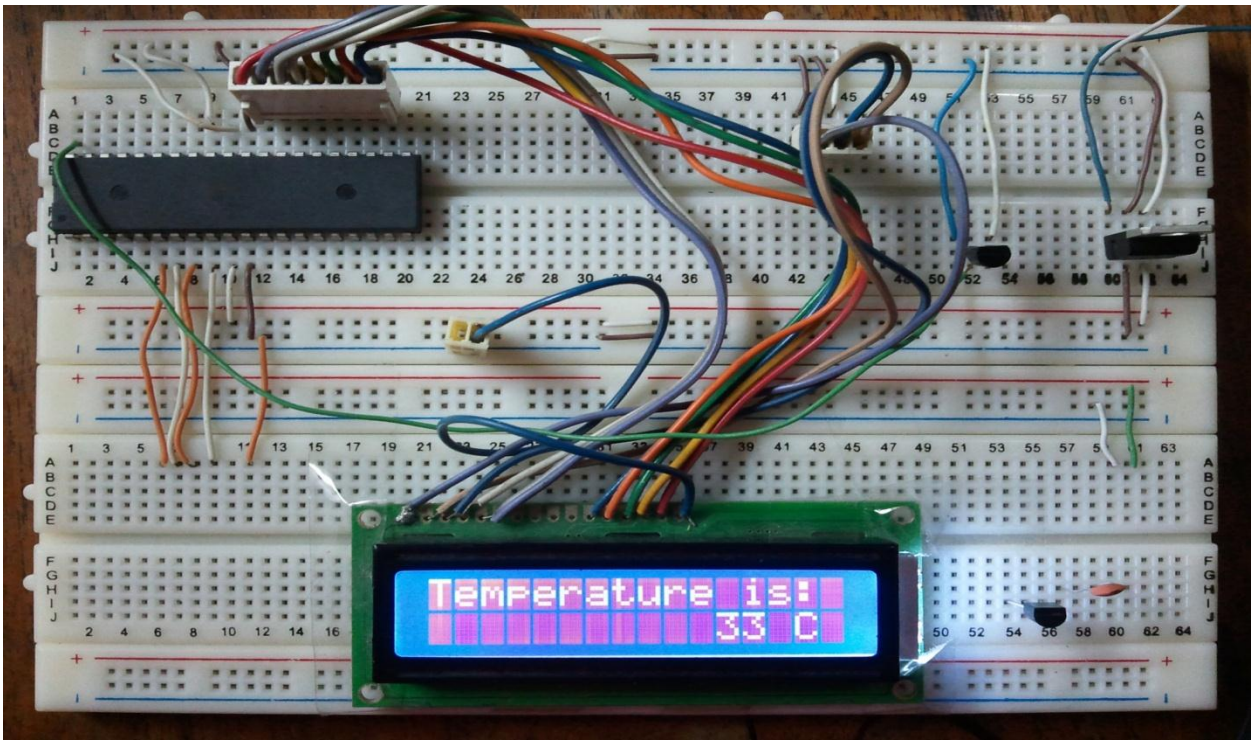

Figure 33: Temperature showing on LCD

Chapter 5: Bridging connection (microcontroller to PC)

One of the tools we use most when debugging our projects is serial input/output. Serial is very easy to implement, and it allows you to send/receive any data you need from your microcontroller to a computer's serial port so it can be viewed using a terminal emulator. These two devices are compatible from a software perspective; however you can't just hook a microcontroller up to a computer because the hardware interfaces are not compatible.

Most microcontrollers these days have built in UARTs (universally asynchronous receiver/transmitter) that can be used to receive and transmit data serially. UARTs transmit one bit at a time at a specified data rate (i.e. 9600bps, 115200bps, etc.). The atmega16 microcontroller used in this project also uses the same methodology.This method of serial communication is sometimes referred to as **TTL serial** (transistor-transistor logic). Serial communication at a TTL level will always remain between the limits of **0V and Vcc**, which is often 5V or 3.3V. A logic high ('1') is represented by Vcc, while a logic low ('0') is 0V.

The serial port on computer complies with the**RS-232** (Recommended Standard 232) telecommunications standard. RS-232 signals are similar to your microcontroller's serial signals in that they transmit one bit at a time, at a specific baud rate, with or without parityand/or stop bits. The two differ solely at a hardware level. By the RS-232 standard a logic high ('1') is represented by a negative voltage – anywhere from -3 to -25V – while a logic low ('0') transmits a positive voltage that can be anywhere from +3 to +25V. On most PCs these signals swing from **-13 to +13V**.

The more extreme voltages of an RS-232 signal help to make it less susceptible to noise, interference, and degradation. This means that an RS-232 signal can generally travel longer physical distances than their TTL counterparts, while still providing a reliable data transmission.

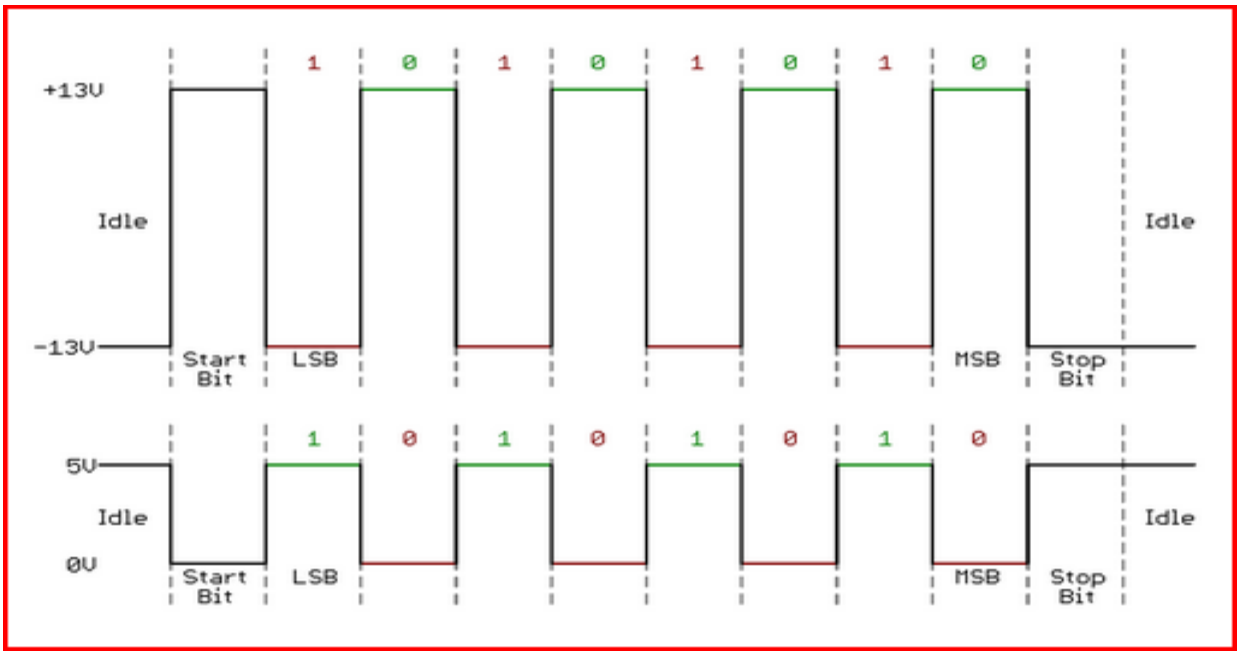

Figure 34: Timing diagram showing both a TTL (bottom) and RS-232 signal.

The problem lies in interfacing these two signals. To connect these two ports we not only have to **invert** the signals, but you also have to deal with regulating the potentially harmful RS-232 voltages to something that won't destroy a microcontroller's serial pins. There are a handful of solutions to this problem of voltage converting and inverting. The most common and easiest solution is just plugging a MAX-232 in between the two devices which also was used in our project :

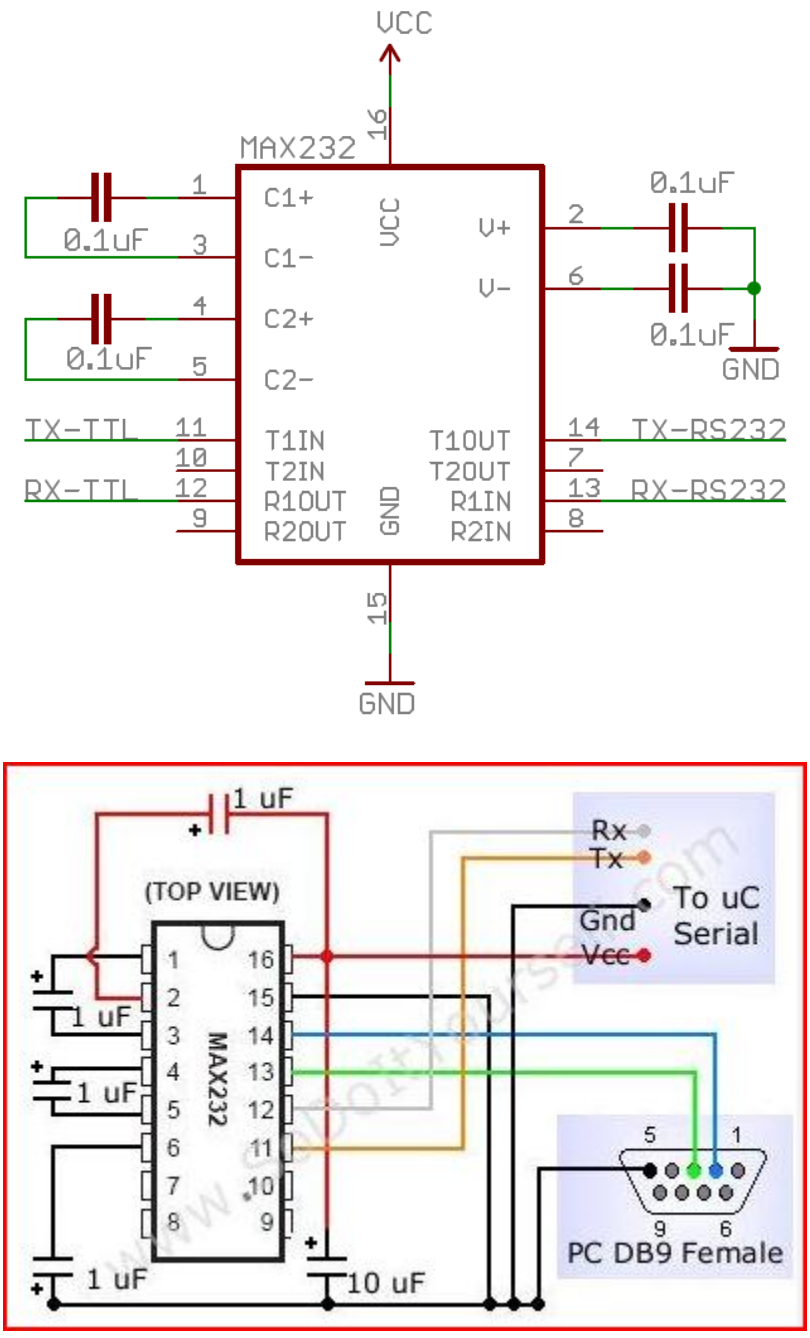

Figure 35: circuit diagram and connections for MAX232

After connecting the designed circuit to the pc by DB9 converter, Bray's terminal uses some commands to show the temperature output. The image shows the settings and output tab. Pressing the 'Connect' button initializes the process according to the configurations set.

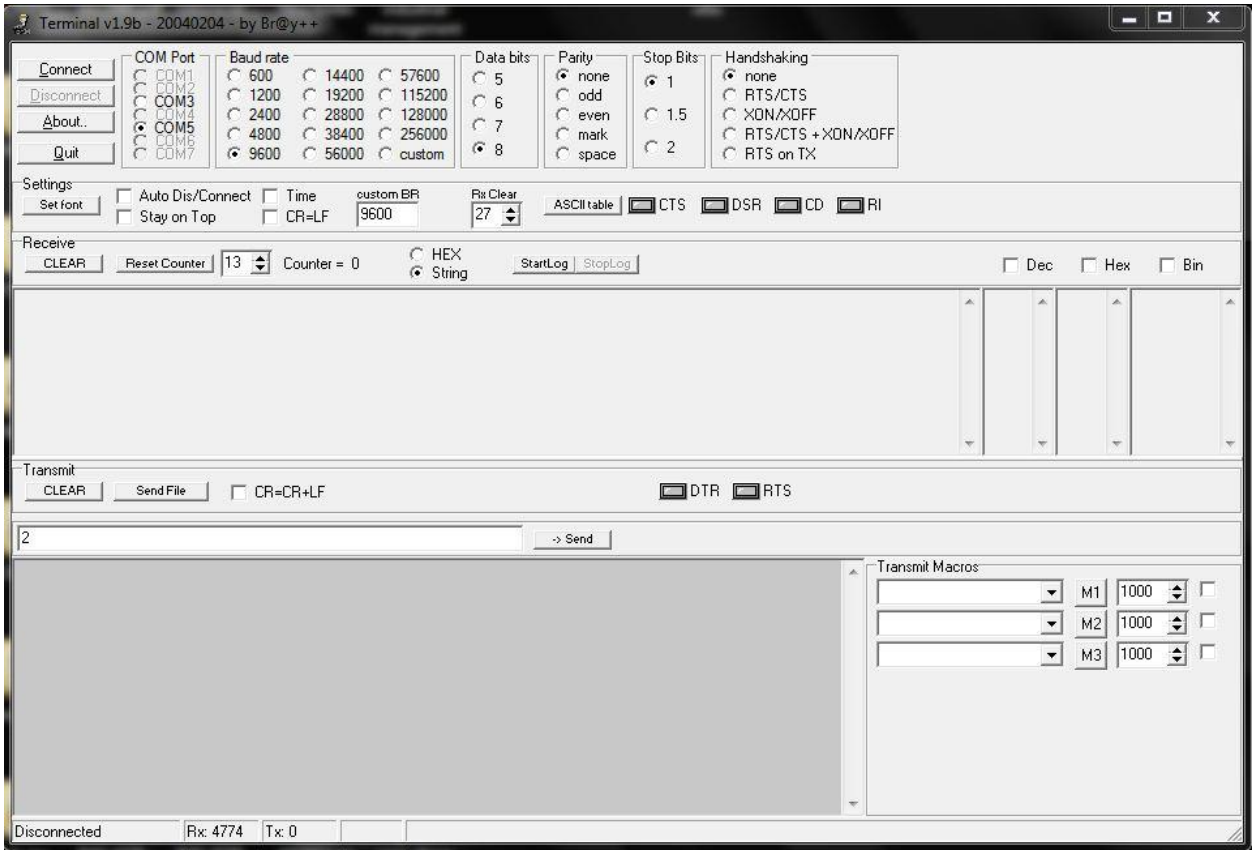

Figure 36: Bray's Terminal

Chapter 6: Conclusion and Future Scope

6.1 Conclusion:

The data logger is an invaluable tool to collect and analyze experimental data,having the ability to clearly present real time results, with sensors and probes able to respond to parameters that are beyond the normal range available from most traditional equipment. Data loggers used for measuring the temperature might have certain limitations in terms of speed, memory and cost.

In this work, an attempt has been done to design a data logger, which is of less cost, portable, very low power consumption, self contained. It is an efficient data logger,which works in real time mode. The reduced number of channels also makes the system simple. This system can be used for multiple sensors which needs a few minor changes in the settings.

A step-by-step approach in designing a Microcontroller based system for temperature measurement has been followed. According to the study and analysis of various parts of the system, a design has been carried out.

6.2 Future Scope:

1.) The performance of microcontroller based temperature data logger has been found on the expected lines. However, there exists a scope for further improvement in its speed, number of channels, power consumption, and PC interface software for post data analysis.

2.) The number of analog channels can be increased to monitor more sensor outputs.

3.) The low power requirement of this data logger makes it easy to use. The device can be made to perform better by providing the power supply with the help of battery source which can be rechargeable or non-rechargeable, to reduce the requirement of main AC power.

4.) This system can be connected to communication devices such as modems, cellular phones, or satellite terminal to enable the remote collection of recorded data or alarming of certain parameters. The new system will email information based upon a regular schedule of based upon alarms.

5.) Moreover, system can be made user friendly by interfacing it with user friendly software and thus can support the post process analysis. There lies the scope to make the system application specific.

6.) The system can also be modified to change the scan time of the channels.

APPENDIX- I

REFERENCES

[1] Sagarkumar S. Badhiye, Dr. P. N. Chatur, B. V. Wakode, "Data logger system: A survey",proceedings of National Conference on Emerging Trends in Computer Science & Information Technology (NCETCSIT-2011),India.

[2] http:// http://www.sparkfun.com/tutorials/215

[3] http://www.atmel.com/Images/doc2466.pdf

[4] http://teslabs.com/openplayer/docs/docs/other/ds18b20_pre1.pdf

[5] http://www.engbedded.com

[6] http/www.loggingtutorial/dr.saulgreenburg/htm

[8] H S kalsi, "Electronic instrumentation", Tata McGraw-Hill Ltd., New Delhi, 1999.

[9] www.academic.hws.edu/geo/logger/paper/loggerpaper.pdf.

[10] http/www.loggingtutorial/dr.saulgreenburg/htm

[11] S. J. Perez, M. A. Calva, R. Castaneda, "A microcontroller based data logging System" Instrumentation and Development Vol. 3 Nr. 8, copyright 1997, Journal of the Mexican Society of Instrumentation.

APPENDIX-II

LIST OF FIGURES AND TABLES