MICROCONTROLLER BASED DIGITAL CLOCK

This project report is submitted to the Department of Electrical & Electronic Engineering (EEE), Daffodil International University, Bangladesh, in Partial fulfillment of the requirements for the Degree of "Bachelor of Science in Electrical & Electronic Engineering".

Supervised By

Md. Mahmudur Rahman

Senior Lecturer

Department of Electrical & Electronic Engineering (EEE)

Daffodil International University

Prepared By

Md. Tamzidul Islam **ID: 102-33-231**

Md. Ferdous Wahid **ID: 102-33-227**

DAFFODIL INTERNATIONAL UNIVERSITY

DHAKA, BANGLADESH

08 October 2013

DECLARATION

We hereby declare that, this project has been done by us under the supervision of **Md. Mahmudur Rahman, Senior Lecturer, Department of EEE,** Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma.

Supervised by:

_____

Md. Mahmudur Rahman

Senior Lecturer

Department of EEE

Daffodil International University

Submitted by:

Md. Tamzidul Islam
ID: 102-33-231
Department of EEE
Daffodil International University

Md. Ferdous Wahid
ID: 102-33-227
Department of EEE
Daffodil International University

ACKNOWLEDGEMENT

First we express our heartiest thanks and gratefulness to almighty Allah for his divine blessing makes us possible to complete this project successfully.

We are grateful and wish our profound indebtedness to supervisor **Md. Mahmudur Rahman, Senior Lecturer,** Department of EEE Daffodil International University, Dhaka. His endless patience , scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior draft and correcting them at all stage have made it possible to complete this project.

We would like to express our heartiest gratitude to **Dr. M. Shamsul Alam**, Professor, and Head, Department of EEE, for his kind help to finish our project and also to other faculty members of EEE department of Daffodil International University.

Finally, we must acknowledge with due respect the constant support and patients of our parents.

_ _ _ _ _ ***Author***

ABSTRACT

A digital clock is a type of clock that displays the time digitally, i.e. in ciphers, as opposed to an analog clock, where the time is displayed by hands. Usually, digital clocks are associated with electronic drives.

Digital clocks typically use the 50 or 60 hertz oscillator of AC power or crystal oscillator as in a quartz clock to keep time. Internationally digital clocks display the hours of the day in24 hour format but in our country a more commonly used hour sequence is 12 hour format (with some indication of AM or PM).

In this project to represent the time, Seven-segment Display has been used for each of four digit which shows hours and minutes and in order to represent seconds LED's are used for each and every seconds . Moreover, as they run on electricity, most digital clocks must be reset every time they are moved or the power is cut off. To reduce the problem, here a battery backup has been used to maintain the time during power outages. In this project we have used RTC (Real Time Clock).

Two switches have been used, one for entering the time and second is for time increment. This digital clock is inexpensive device that makes product designs more popular.

This project uses regulated 5V & 1A power supply. 7805 three terminal voltage regulator is used for voltage regulation. Bridge type full wave rectifier is used to rectify AC output of secondary of 230/6V step down transformer.

Dedicated

To

Our Beloved Parents

&

Honorable Teachers

TABLE OF CONTENT

LIST OF FIGURES

FIGURES **PAGE**

LIST OF TABLES

TABLES **PAGE**

Chapter 1

1.1 Introduction:

A digital clock is a type of clock that displays the time digitally, i.e. in numerals or other symbols, as opposed to an analog clock, where the time is indicated by the positions of rotating hands.

In this project, we have built a digital clock with 12 hour count time. The clock runs from 00:00 to 11:59 and then back to 00:00. Our display has four digits, two digits for minutes and two for hour. The specialty of this clock is that it has very low power consumption and condensed layout.

Every 10 minutes, we add one on the ten's digit of minute. After every 6 such cycles, we add one on the unit's digit of hour, and every 10 hours, we add one on tens digit of hour. As soon as it reaches 1 on the ten's digit of hour and 2 on unit's digit of hour, both the ten's and the unit's digit of hour are reset to 0. So, in practice, we do not see 12:00 displayed. We rather see a transition from 11:59 to 00:00.

We have used a Atmega8 Microcontroller and a DS1307 IC. Also used 8MHz crystal oscillator, here we used AC power source so we take aback up power from a battery. The outputs would be connected to LEDs and would make the LED glow for a high output and vice-versa for a low output. We make a seven segment display by using LEDs.

The requirements are:

- Use LED 7 segment displays.
- Display time.
- Buttons to set the time.
- Fit in a small red translucent box.

Digital clocks are often associated with electronic drives, but the "digital" description refers only to the display, not to the drive mechanism. (Both analog and digital clocks can be driven either mechanically or electronically, but "clockwork" mechanisms with digital displays are rare.)

1.2 The History:

First digital pocket watch was the invention of Austrian engineer Josef Pallweber who created his "jump-hour" mechanism in 1883. Instead of a conventional dial the jump-hour featured two windows in an enamel dial, through which the hours and minutes are visible on rotating discs. The second hand remained conventional. By 1885 Pall Weber's mechanism was already on the market in pocket watches by Corte Bert and IWC; arguably contributing to the subsequent rise and commercial success of IWC. The principles of Pall Weber's jump-hour movement later appeared in wristwatches by the 1920s (Corte Bert) and still used today (Chronoswiss Digiteur). While the original inventor didn't have a watch brand at the time, his name has since been resurrected by a newly established watch manufacturer.

Plato clocks used a similar idea but different layout. These spring-wound pieces consisted of a glass cylinder with a column inside, affixed to which were small digital cards with numbers printed on them, which flipped as time passed. The Plato clocks were introduced at the St. Louis World Fair in 1904, produced by Ansonia Clock Company. Eugene Fitch of New York patented the clock design in 1903.

The earliest patent for a digital alarm clock was registered by D.E Protzmann and others on October 23, 1956 in the United States. D.E Protzmann and his associates also patented another digital clock in 1970, which was said to use a minimal amount of moving parts. Two side-plates held digital numerals between them, while an electric motor and cam gear outside controlled movement.

In 1970, the first digital wristwatch with an LED display was mass-produced. Called the Pulsar, and produced by the Hamilton Watch Company, this watch was hinted at

two years prior when the same company created a prototype digital watch for Kubrick's 2001: A Space Odyssey. Throughout the 1970s, despite the initial hefty cost of digital watches, the popularity of said devices steadily rose.

Over the years, many different types of digital alarm clocks have been developed. In Soviet Russia the 7-segment digital clocks were known as Elektronika 7.

Chapter 2

2.1 Design and Development

This project creates a clock which displays the time on seven segment displays. The project fits in a small box. The circuit uses seven outputs and four common connections (com1, com2, com3, com4) for the segments. To allow the setting of the time two buttons are provided. The first selects the digit and the second increments the digits. The order is hours, minutes, second. When the setting mode is active only the digits being set are displayed, all the others are shown as '--'. Before creating the program we listed the steps the program requires:

- Initialize the PIC and set all digits to zero.
- Start the timer interrupts to count the seconds elapsed.
- When seconds reach 60 increment minutes and reset seconds.
- When minutes reach 60 increment hours and reset minutes.
- When hours reach 12 increments and reset hours.
- When the first button is pressed stop updating the clock, reset all digits to – apart from the hours. If the button is pressed again set hours to – and display the minutes. Each press moves through the digits.
- After the first button has been pressed the second button increments the number on the digits. When the digit reaches its intended maximum it returns back to its lowest value. For example, hours range from 00 to 12. When the digit increments to 12 it returns to 00.

First version only in the program and this is reflected in the flow charts below. Based on these steps we created a flow diagram to help create the program. The Main Flow Diagram constitutes the main sub program. After setting up the ports and starting the interrupt the main sub program polls the buttons. The interrupt flow diagram now shows only the seconds being updated. Not shown in the flow charts, code is also required to display the digits and to read the inputs. we found in use that the clock

gained time as the day went on. After a few days the time difference was noticeable. I calculated that every day 16 seconds were being gained. I decided to setup a new counter that was incremented each second and it would be used to lose a second at the correct moment. I needed to lose 1 second every 24 / 16 = 1.5 hours. This was 5400 seconds (1.5x60x60). When the counter reached 5400 seconds the seconds counter was decreased by 1 and the counter was reset. The clock now keeps very good time.

Outputs are used to turn the 7 segment and decimal points on and off. The displays are common four anodes so a port set low turns the segment on and a port set high turns it off.

Each of the digits is turned on sequentially. The segments for that digit are only on while that digit is on

2.2 Flow chart:

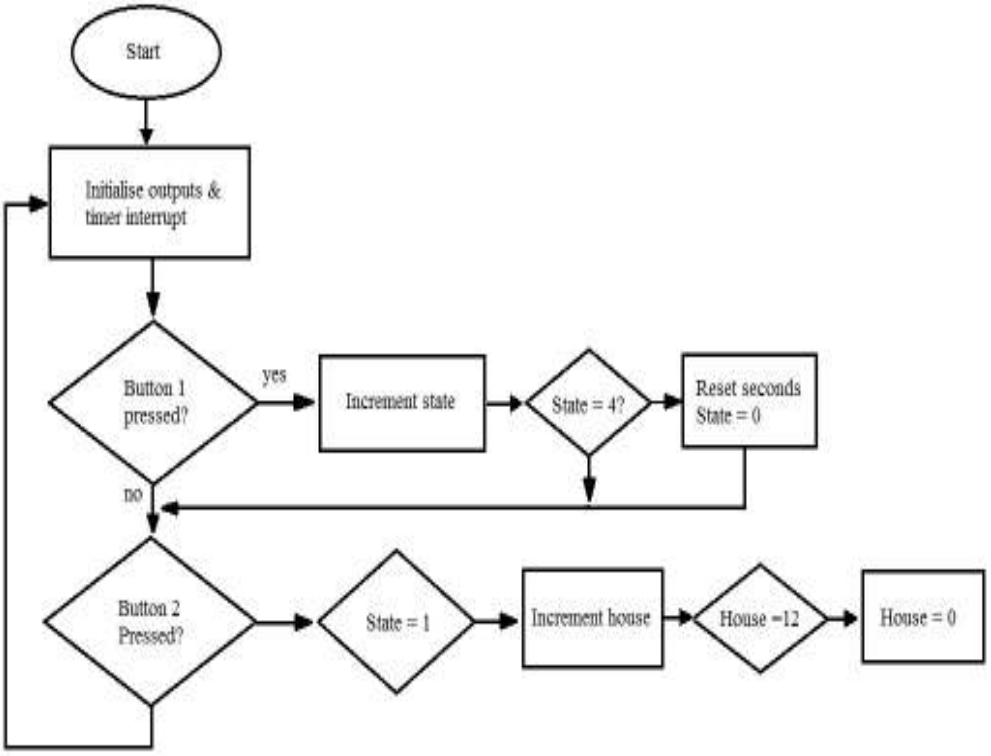

Figure-2.1: Flow Chart

2.3 Circuit Diagram:

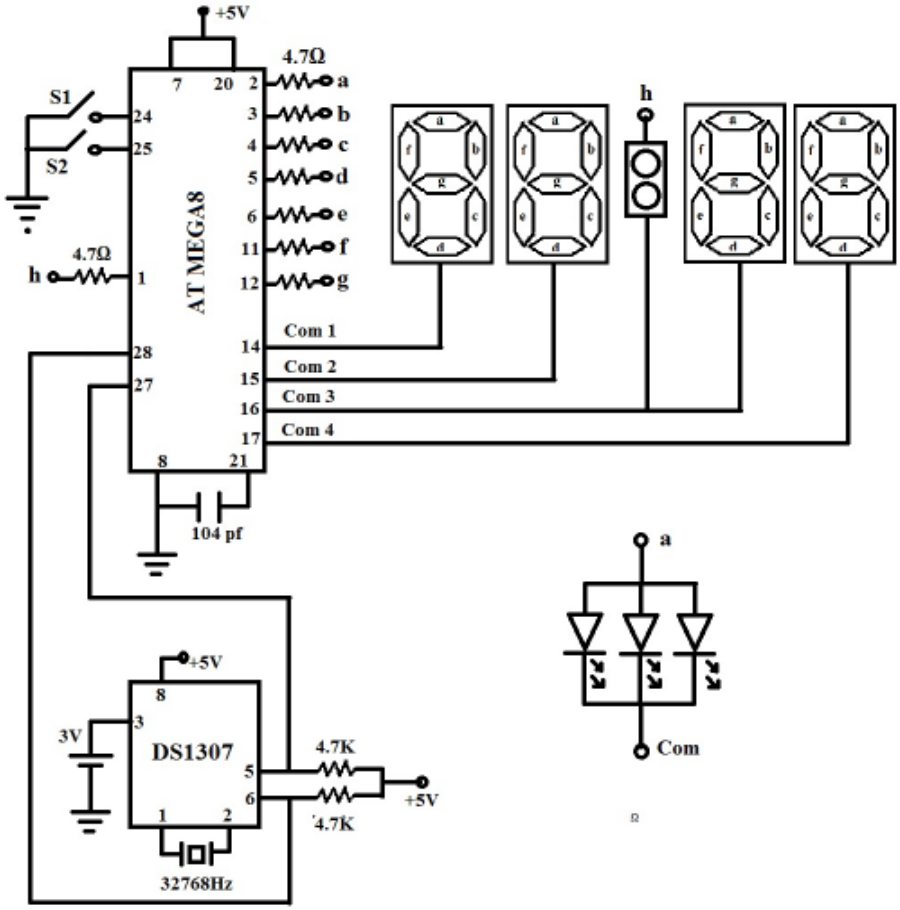

Figure-2.2: Circuit Diagram

Chapter 3

3.1 MICROCONTROLLER AT MEGA8:

3.1.1 Features:

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

• Advanced RISC Architecture

– 130 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

– 8K Bytes of In-System Self-programmable Flash program memory

– 512 Bytes EEPROM

– 1K Byte 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

• Peripheral Features

– Two 8-bit Timer/Counters with Separate Prescaler, one Compare Mode

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

– Real Time Counter with Separate Oscillator

– Three PWM Channels

– 8-channel ADC in TQFP and QFN/MLF package

Eight Channels 10-bit Accuracy

– 6-channel ADC in PDIP package

Six Channels 10-bit Accuracy

– 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

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

• I/O and Packages

– 23 Programmable I/O Lines

– 28-lead PDIP, 32-lead TQFP, and 32-pad QFN/MLF

• Operating Voltages

– 2.7 - 5.5V (ATmega8L)

– 4.5 - 5.5V (ATmega8)

• Speed Grades

– 0 - 8 MHz (ATmega8L)

– 0 - 16 MHz (ATmega8)

• Power Consumption at 4 MHz, 3V, 25°C

– Active: 3.6 mA

– Idle Mode: 1.0 mA

– Power-down Mode: 0.5 µA

3.1.2 Pin Configurations:

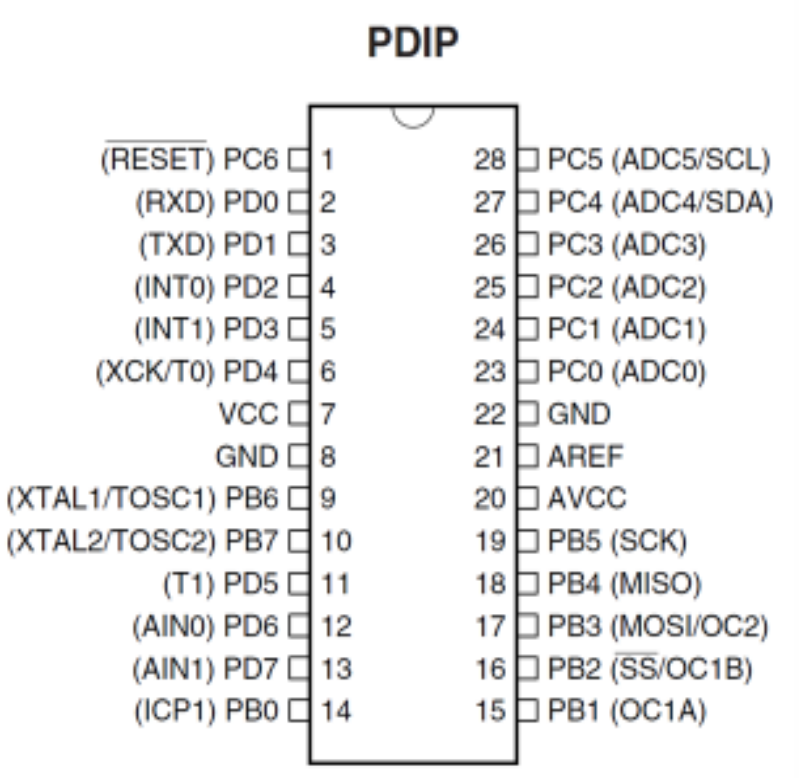

Figure-3.1: Pin Configuration of PIC AT MEGA8

3.1.3 Overview:

The ATmega8 is a low-power CMOS 8-bit microcontroller based on the AVR RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega8 achieves throughputs approaching 1 MIPS per MHz, allowing the system designed to optimize power consumption versus processing speed.

3.1.4 Block Diagram:

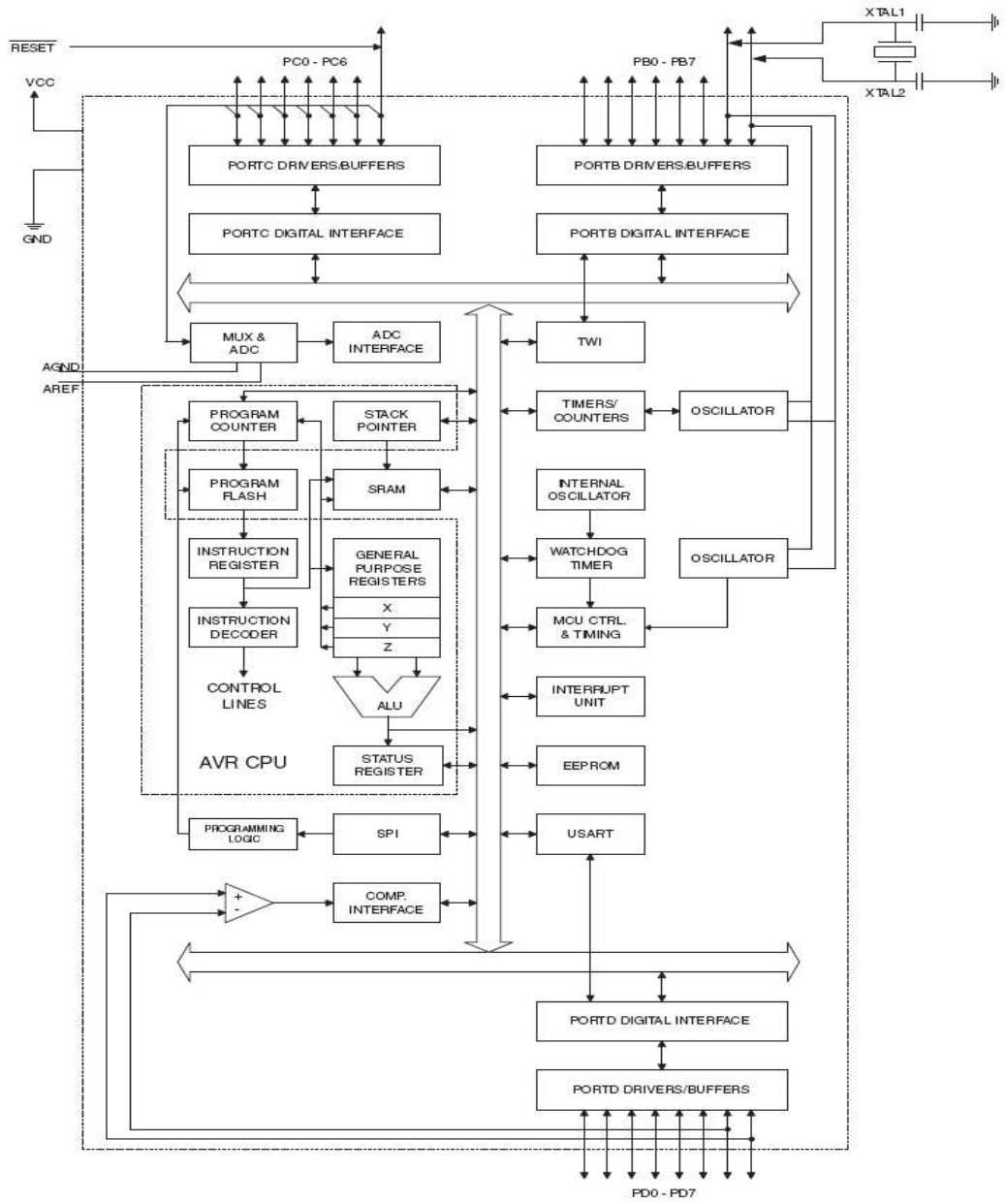

Figure-3.2: Block Diagram of PIC AT MEGA8

The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers. The ATmega8 provides the following features: 8K bytes of In-System Programmable Flash with Read-While-Write capabilities, 512 bytes of EEPROM, 1K byte of SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte oriented Two wire Serial Interface, a 6-channel ADC (eight channels in TQFP and QFN/MLF packages) with 10-bit accuracy, a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and five software selectable power saving modes. The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next Interrupt or Hardware Reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping. The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low-power consumption. The device is manufactured using Atmel's high density non-volatile memory technology. The Flash Program memory can be reprogrammed In-System through an SPI serial interface, by a conventional non-volatile memory programmer, or by an On-chip boot program running on the AVR core. The boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash Section will continue to run while the Application Flash Section is updated, providing true Read-While-Write operation. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega8 is a powerful microcontroller that provides a highly-flexible and cost-effective solution to many embedded control applications. The ATmega8 AVR is supported with a full suite of program and system development tools, including C compilers, macro assemblers, program debugger/simulators, In-Circuit Emulators, and evaluation kits.

3.2 Pin Descriptions:

VCC: Digital supply voltage.

GND: Ground.

Port B (PB7..PB0) XTAL1/XTAL2/TOSC1/ TOSC2:

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. 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.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit. Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier. If the Internal Calibrated RC Oscillator is used as chip clock source, PB7..6 is used as TOSC2..1 input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

Port C (PC5..PC0) :

Port C is an 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port C output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

PC6/RESET:

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C. If the RSTDISBL Fuse is un-programmed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a Reset.

Port D (PD7..PD0):

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

RESET:

Reset input. A low level on this pin for longer than the minimum pulse length will generate a reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a reset.

AVCC:

AVCC is the supply voltage pin for the A/D Converter, Port C (3..0), and ADC (7..6). It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter. Note that Port C (5..4) use digital supply voltage, VCC.

AREF:

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

ADC7..6 (TQFP and QFN/MLF Package Only):

In the TQFP and QFN/MLF package, ADC7..6 serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

3.3 AVR CPU Core:

3.3.1 Introduction:

This section discusses the AVR core architecture in general. The main function of the CPU core is to ensure correct program execution. The CPU must therefore be able to access memories, perform calculations, control peripherals, and handle interrupts.

3.3.2 Architectural Overview:

In order to maximize performance and parallelism, the AVR uses a Harvard architecture – with separate memories and buses for program and data. Instructions in the Program memory are executed with a single level pipelining. While one instruction is being executed, the next instruction

is pre-fetched from the Program memory. This concept enables instructions to be executed in every clock cycle. The Program memory is In-System Reprogrammable Flash memory.

Block Diagram of the AVR MCU Architecture:

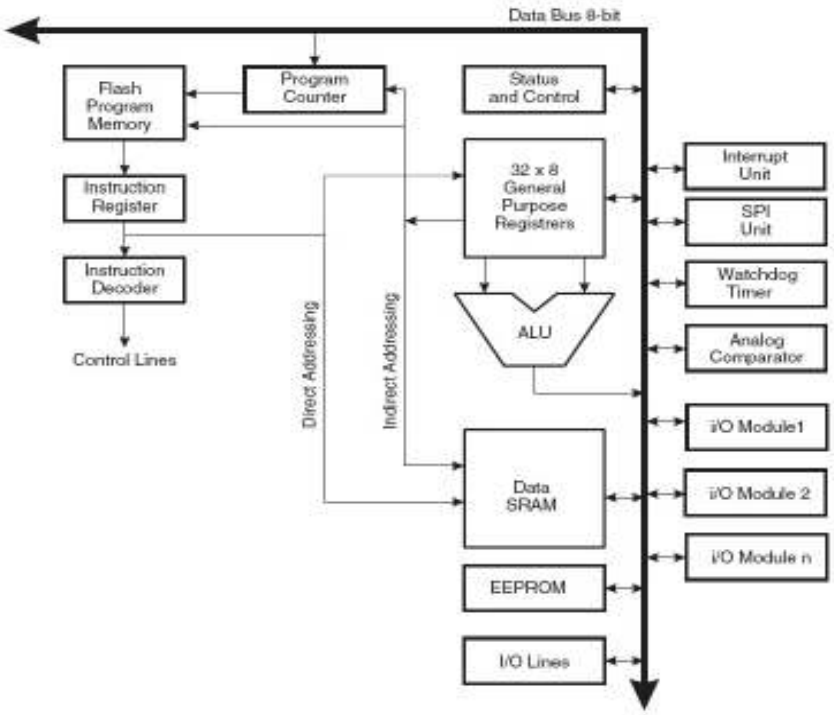

Figure-3.3: Block Diagram of the AVR MCU

3.3.3 Arithmetic Logic Unit – ALU:

The high-performance AVR ALU operates in direct connection with all the 32 general purpose working registers. Within a single clock cycle, arithmetic operations between general purpose registers or between a register and an immediate are executed. The ALU operations are divided into three main categories – arithmetic, logical, and bit-functions. Some implementations of the architecture also provide a powerful multiplier supporting both signed/unsigned multiplication and fractional format. See the "Instruction Set" section for a detailed description.

3.3.4 Status Register:

The Status Register contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations. Note that the Status Register is updated after all ALU operations, as specified in the Instruction Set Reference. This will in many cases remove the need for using the dedicated compare instructions, resulting in faster and more compact code. The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

3.3.5 General Purpose Register File:

The Register File is optimized for the AVR Enhanced RISC instruction set. In order to achieve the required performance and flexibility, the following input/output schemes are supported by the

Register File:

• One 8-bit output operand and one 8-bit result input.

• Two 8-bit output operands and one 8-bit result input.

• Two 8-bit output operands and one 16-bit result input.

• One 16-bit output operand and one 16-bit result input.

3.3.6 The X-register, Y-register and Z-register:

The registers R26..R31 have some added functions to their general purpose usage. These registers are 16-bit address pointers for indirect addressing of the Data Space.

3.3.7 Stack Pointer:

The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls. The Stack Pointer Register always points to the top of the Stack. Note that the Stack is implemented as growing from higher memory locations to lower memory locations. This implies that a Stack PUSH command decreases the Stack Pointer.

3.3.8 Instruction Execution Timing:

This section describes the general access timing concepts for instruction execution. The AVR CPU is driven by the CPU clock clk CPU, directly generated from the selected clock source for the chip. No internal clock division is used.

3.3.9 Reset and Interrupt Handling:

The AVR provides several different interrupt sources. These interrupts and the separate Reset Vector each have a separate Program Vector in the Program memory space. All interrupts are assigned individual enable bits which must be written logic one together with the Global Interrupt Enable bit in the Status Register in order to enable the interrupt. Depending on the Program Counter value, interrupts may be automatically disabled when Boot Lock Bits BLB02 or BLB12 are programmed. This feature improves software security. The lowest addresses in the Program memory space are by default defined as the Reset and Interrupt Vectors. The list also determines the priority levels of the different interrupts. The lower the address the higher is the priority level. RESET has the highest priority, and next is INT0 – the External Interrupt Request 0. The Interrupt Vectors can be moved to the start of the boot Flash section by setting the Interrupt Vector Select (IVSEL) bit in the General Interrupt Control Register (GICR). The Reset Vector can also be moved to the start of the boot Flash section by programming the BOOTRST Fuse.

3.4 Memories:

This section describes the different memories in the ATmega8. The AVR architecture has two main memory spaces, the Data memory and the Program Memory space. In addition, the ATmega8 features an EEPROM Memory for data storage. All three memory spaces are linear and regular.

3.4.1 I/O Memory:

All ATmega8 I/Os and peripherals are placed in the I/O space. The I/O locations are accessed by the IN and OUT instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O Registers within the address range 0x00 - 0x1F are directly bit accessible

using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions. Refer to the instruction set section for more details. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O Registers as data space using LD and ST instructions, 0x20 must be added to these addresses. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written. Some of the Status Flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers 0x00 to 0x1F only.

3.5 System Clock and Clock Options:

3.5.1 System Clock and Clock Options:

All of the clocks need not be active at a given time. In order to reduce power consumption, the clocks to modules not being used can be halted by using different sleep modes. The clock systems are detailed in Figure3.4

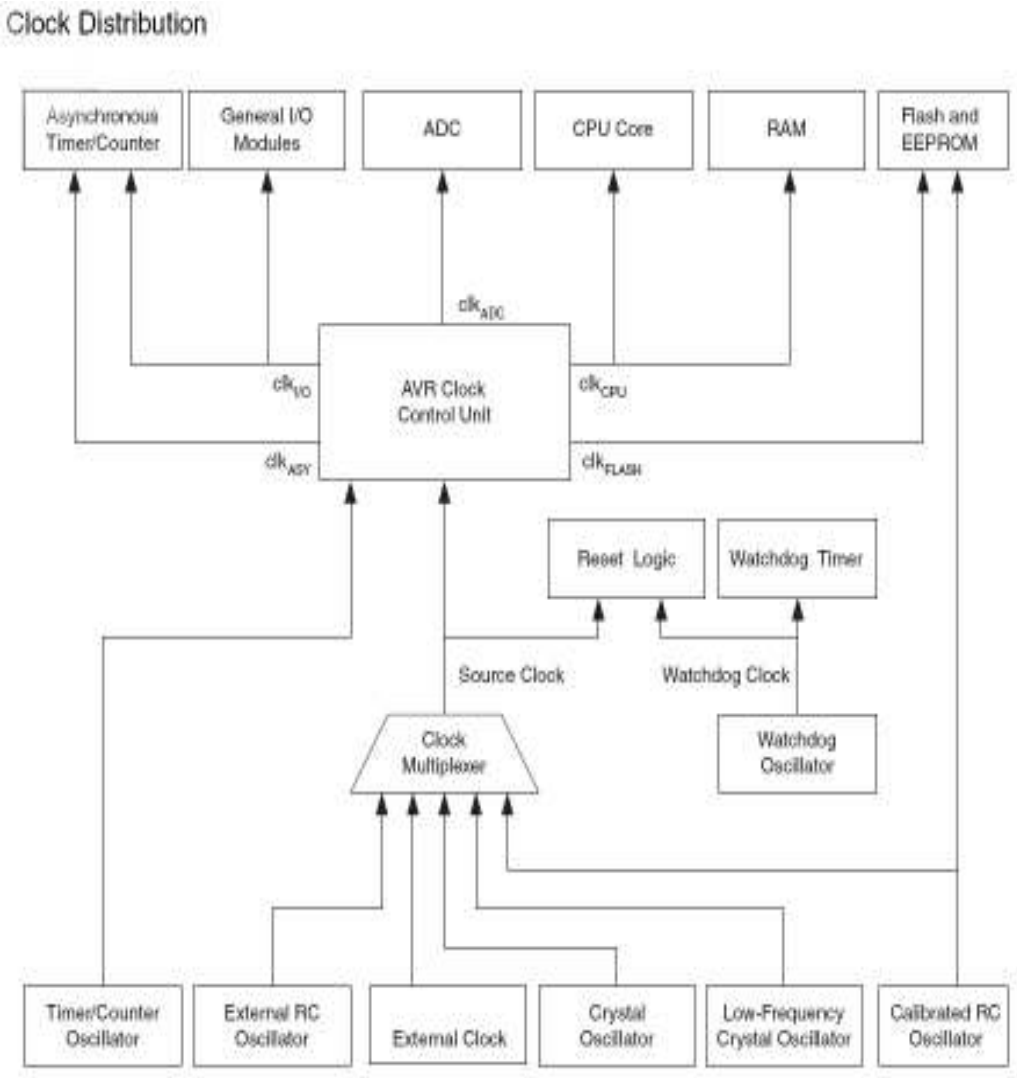

Figure-3.4: The Clock System

3.5.2 CPU Clock – clk CPU:

The CPU clock is routed to parts of the system concerned with operation of the AVR core. Examples of such modules are the General Purpose Register File, the Status Register and the Data memory holding the Stack Pointer. Halting the CPU clock inhibits the core from performing general operations and calculations.

3.5.3 I/O Clock – clk I/O:

The I/O clock is used by the majority of the I/O modules, like Timer/Counters, SPI, and USART. The I/O clock is also used by the External Interrupt module, but note that some external interrupts are detected by asynchronous logic, allowing such interrupts to be detected even if the I/O clock is halted. Also note that address recognition in the TWI module is carried out asynchronously when clk I/O is halted, enabling TWI address reception in all sleep modes.

3.5.4 Flash Clock – clk Flash:

The Flash clock controls operation of the Flash interface. The Flash clock is usually active simultaneously with the CPU clock.

3.6 Crystal Oscillator:

XTAL1 and XTAL2 are input and output, respectively, of an inverting amplifier which can be configured for use as an On-chip Oscillator. Either a quartz crystal or a ceramic resonator may be used. The CKOPT Fuse selects between two different Oscillator amplifier modes. When CKOPT is programmed, the Oscillator output will oscillate a full rail-to rail swing on the output. This mode is suitable when operating in a very noisy environment or when the output from XTAL2 drives a second clock buffer. This mode has a wide frequency range. When CKOPT is un-programmed, the Oscillator has a smaller output swing. This reduces power consumption considerably. This mode has a limited frequency range and it cannot be used to drive other clock buffers. For resonators, the maximum frequency is 8 MHz with CKOPT un-programmed and 16 MHz with CKOPT programmed. C1 and C2 should always be equal for both crystals and resonators. The optimal value of the capacitors depends on the crystal or resonator in use, the amount of stray capacitance, and the electromagnetic noise of the environment. Some initial guidelines for choosing capacitors for use with crystals are given in Table. For ceramic resonators, the capacitor values given by the manufacturer should be used.

3.6.1 Figure:

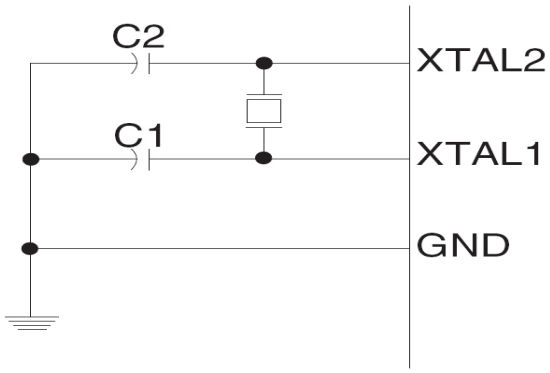

Figure-3.5: Connection of Crystal oscillator

3.6.2 Table:

CKOPT	CKSEL3..1	Frequency Range(MHz)	Recommended Range for Capacitors C1 and C2 for Use with Crystals (pF)
1	101	0.4 - 0.9	–
1	110	0.9 - 3.0	12 - 22
1	111	3.0 - 8.0	12 - 22
0	101, 110, 111	1.0 ≤	12 - 22

Table-3.1: Crystal oscillator details

3.6.3 Low-frequency Crystal Oscillator:

To use a 32.768 kHz watch crystal as the clock source for the device, the Low-frequency Crystal Oscillator must be selected by setting the CKSEL Fuses to “1001”. The crystal should be connected as shown in Figure-6. By programming the CKOPT Fuse, the user can enable internal capacitors on XTAL1 and XTAL2, thereby removing the need for external capacitors. The internal capacitors have a nominal

value of 36 pF. When this Oscillator is selected, start-up times are determined by the SUT Fuses as shown in Table-2.

Table:

SUT1...0	Start-up Time from Power-down and Power-save	Additional Delay From Reset (VCC = 5.0V)	Recommended Usage
00	1K CK	4.1 ms	Fast rising power or BOD enabled
01	1K CK	65ms	Slowly rising power
10	32K CK	65ms	Stable frequency at start-up
11	Reserved		

Table-3.2: SUT1...0 Details

3.6.4 Timer/Counter Oscillator:

For AVR microcontrollers with Timer/Counter Oscillator pins (TOSC1 and TOSC2), the crystal are connected directly between the pins. By programming the CKOPT Fuse, the user can enable internal capacitors on XTAL1 and XTAL2, thereby removing the need for external capacitors. The Oscillator is optimized for use with a 32.768 kHz watch crystal. Applying an external clock source to TOSC1 is not recommended. Note: The Timer/Counter Oscillator uses the same type of crystal oscillator as Low-Frequency Oscillator and the internal capacitors have the same nominal value of 36 pF.

3.7 Power Management and Sleep Modes:

Sleep modes enable the application to shut down unused modules in the MCU, thereby saving power. The AVR provides various sleep modes allowing the user to tailor the power consumption to the application's requirements. To enter any of the five sleep modes, the SE bit in MCUCR must be written to logic one and a SLEEP instruction must be executed. The SM2, SM1, and SM0 bits in the MCUCR Register select which sleep mode (Idle, ADC Noise Reduction, Power-down, Power-save, or Standby) will be activated by the SLEEP instruction. If an enabled interrupt occurs while the MCU is in a sleep mode, the MCU wakes up. The MCU is then halted for four cycles in addition to the start-up time; it executes the interrupt routine, and resumes execution from the instruction following SLEEP. The contents of the Register File and SRAM are unaltered when the device wakes up from sleep. If a reset occurs during sleep mode, the MCU wakes up and executes from the Reset Vector. Note that the Extended Standby mode present in many other AVR MCUs has been removed in the ATmega8, as the TOSC and XTAL inputs share the same physical pins.

3.7.1 Standby Mode:

When the SM2..0 bits are 110 and an external crystal/resonator clock option is selected, the SLEEP instruction makes the MCU enter Standby mode. This mode is identical to Power-down with the exception that the Oscillator is kept running. From Standby mode, the device wakes up in 6 clock cycles.

3.7.2 Minimizing Power Consumption:

There are several issues to consider when trying to minimize the power consumption in an AVR controlled system. In general, sleep modes should be used as much as possible, and the sleep mode should be selected so that as few as possible of the device's functions are operating. All functions not needed should be disabled. In particular, the following modules may need special consideration when trying to achieve the lowest possible power consumption.

3.8 Analog-to-Digital Converter (ADC):

If enabled, the ADC will be enabled in all sleep modes. To save power, the ADC should be disabled before entering any sleep mode. When the ADC is turned off and on again, the next conversion will be an extended conversion.

3.9 System Control and Reset:

3.9.1 Resetting the AVR:

During Reset, all I/O Registers are set to their initial values, and the program starts execution from the Reset Vector. If the program never enables an interrupt source, the Interrupt Vectors are not used, and regular program code can be placed at these locations. This is also the case if the Reset Vector is in the Application section while the Interrupt Vectors are in the boot section or vice versa. The I/O ports of the AVR are immediately reset to their initial state when a reset source goes active. This does not require any clock source to be running. After all reset sources have gone inactive, a delay counter is invoked, stretching the internal reset. This allows the power to reach a stable level before normal operation starts. The time-out period of the delay counter is defined by the user through the CKSEL Fuses

3.9.2 Reset Sources:

The ATmega8 has four sources of Reset:

• Power-on Reset. The MCU is reset when the supply voltage is below the Power-on Reset threshold (VPOT).

• External Reset. The MCU is reset when a low level is present on the RESET pin for longer than the minimum pulse length.

• Watchdog Reset. The MCU is reset when the Watchdog Timer period expires and the Watchdog is enabled.

• Brown-out Reset. The MCU is reset when the supply voltage VCC is below the Brown-out Reset threshold (VBOT) and the Brown-out Detector is enabled.

3.9.3 External Reset:

An External Reset is generated by a low level on the RESET pin. Reset pulses longer than the minimum pulse width will generate a reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a reset. When the applied signal reaches the Reset Threshold Voltage – VRST on its positive edge, the delay counter starts the MCU after the time-out period tTOUT has expired.

3.10 Unconnected pins:

If some pins are unused, it is recommended to ensure that these pins have a defined level. Even though most of the digital inputs are disabled in the deep sleep modes as described above, floating inputs should be avoided to reduce current consumption in all other modes where the digital inputs are enabled (Reset, Active mode and Idle mode). The simplest method to ensure a defined level of an unused pin, is to enable the internal pull-up. In this case, the pull-up will be disabled during reset. If low power consumption during reset is important, it is recommended to use an external pull-up or pull-down. Connecting unused pins directly to VCC or GND is not recommended, since this may cause excessive currents if the pin is accidentally configured as an output.

3.11 Operation:

The counting direction is always up (incrementing), and no counter clear is performed. The counter simply overruns when it passes its maximum 8-bit value (MAX = 0xFF) and then restarts from the bottom (0x00). In normal operation the Timer/Counter Overflow Flag (TOV0) will be set in the same timer clock cycle as the TCNT0 becomes zero. The TOV0 Flag in this case behaves like a ninth bit, except that it is only set, not cleared. However, combined with the timer overflow interrupt that automatically clears the TOV0 Flag, the timer resolution can be increased by software. A new counter value can be written anytime.

3.12 Compatibility:

The 16-bit Timer/Counter has been updated and improved from previous versions of the 16-bit AVR Timer/Counter. This 16-bit Timer/Counter is fully compatible with the earlier version regarding:

• All 16-bit Timer/Counter related I/O Register address locations, including Timer Interrupt Registers.

• Bit locations inside all 16-bit Timer/Counter Registers, including Timer Interrupt Registers.

• Interrupt Vectors.

The following control bits have changed name, but have same functionality and register location:

• PWM10 is changed to WGM10.

• PWM11 is changed to WGM11.

• CTC1 is changed to WGM12.

The following bits are added to the 16-bit Timer/Counter Control Registers:

• FOC1A and FOC1B are added to TCCR1A.

• WGM13 is added to TCCR1B.

The 16-bit Timer/Counter has improvements that will affect the compatibility in some special cases.

3.13 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.

Figure-3.6 shows a block diagram of the clock generation logic.

Figure:

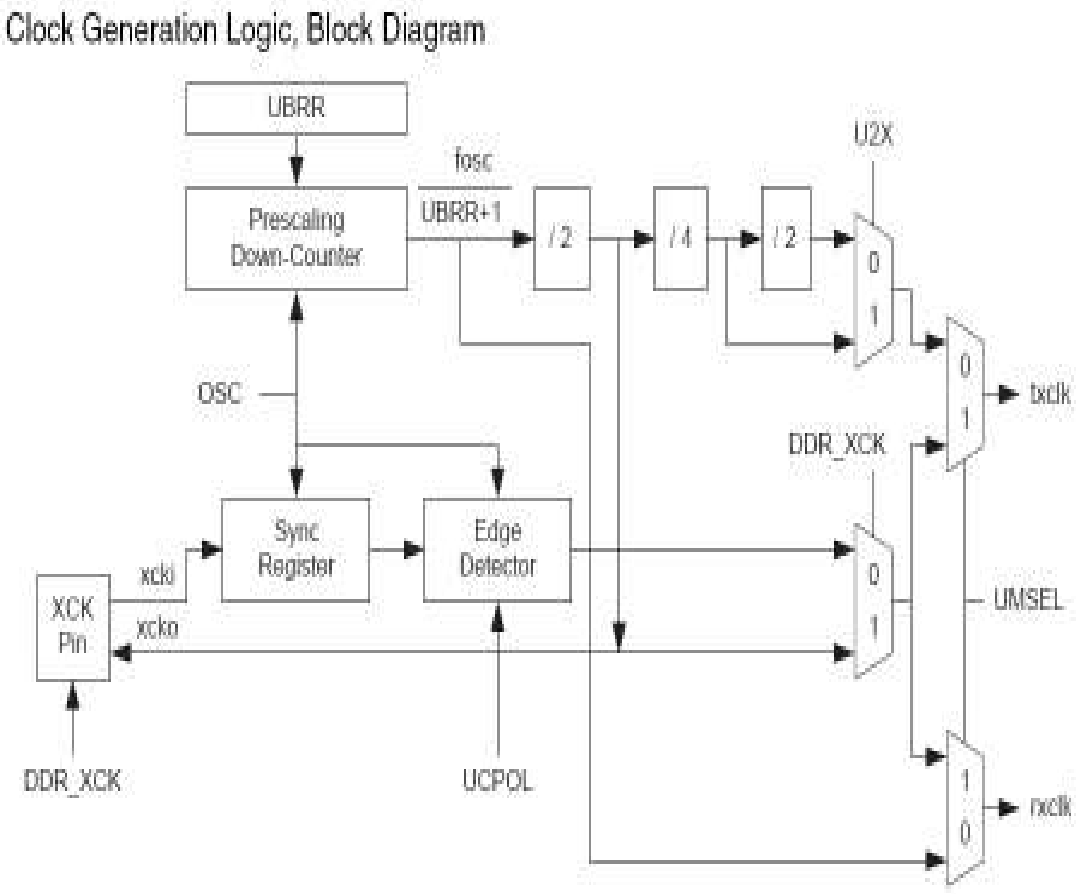

Figure-3.6: Clock Generation Logic

Chapter 4

4.1 IC DS 1307:

4.1.1 Features:

- Real-time clock (RTC) counts seconds, minutes, hours, date of the month, month, day of the week, and year with leap-year compensation valid up to 2100
- 56-byte, battery-backed, nonvolatile (NV) RAM for data storage
- Two-wire serial interface
- Programmable square wave output signal
- Automatic power-fail detect and switch circuitry
- Consumes less than 500nA in battery backup mode with oscillator running
- Optional industrial temperature range:

  -40°C to +85°C
- Available in 8-pin DIP or SOIC
- Underwriters Laboratory (UL) recognized

4.1.2 Ordering Information:

- DS1307 8-Pin DIP (300-mil)
- DS1307Z 8-Pin SOIC (150-mil)
- DS1307N 8-Pin DIP (Industrial)
- DS1307ZN 8-Pin SOIC (Industrial)

4.2 Pin Assignment

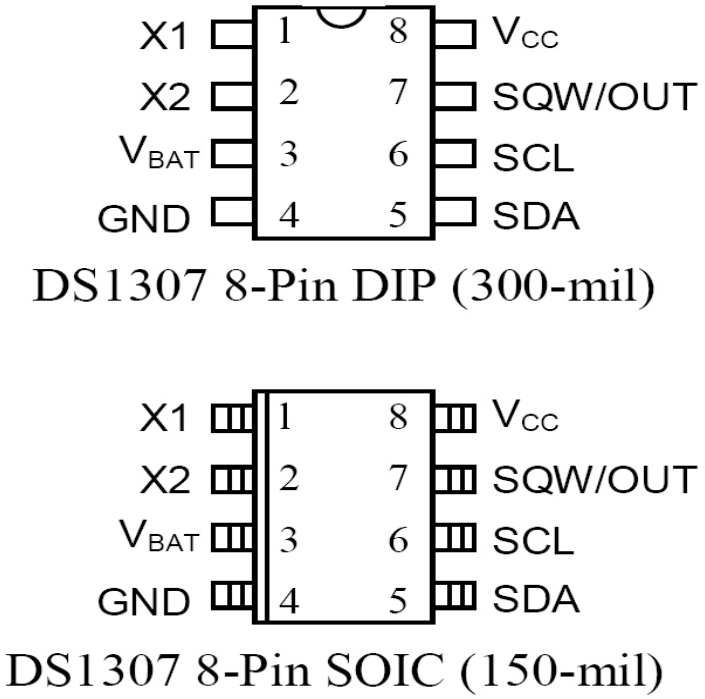

Figure-4.1: Pin Assignment of DS1307

4.2.1 Pin Description:

VCC - Primary Power Supply

X1, X2 - 32.768kHz Crystal Connection

VBAT - +3V Battery Input

GND - Ground

SDA - Serial Data

SCL - Serial Clock

SQW/OUT - Square Wave/Output Driver

4.3 Description:

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.

4.4 Typical Operating Circuit:

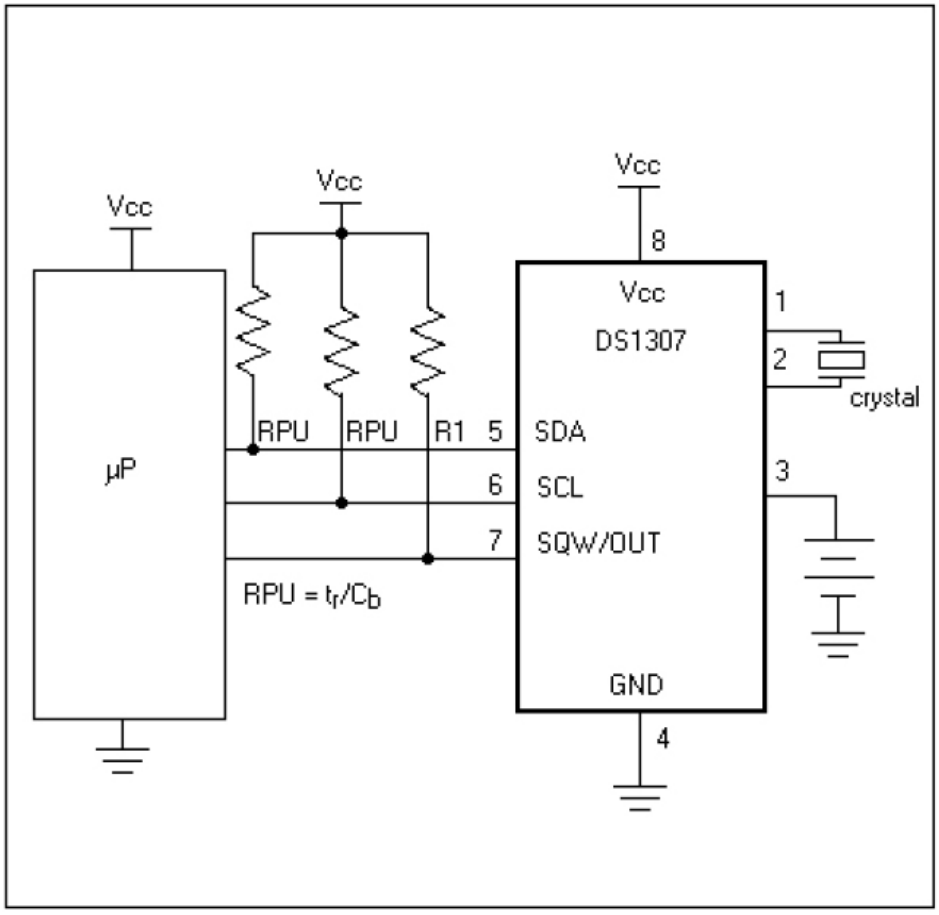

Figure-4.2: Typical Operating Circuit

4.4.1 Operation:

The DS1307 operates as a slave device on the serial bus. Access is obtained by implementing a START condition and providing a device identification code followed by a register address. Subsequent registers

can be accessed sequentially until a STOP condition is executed. When VCC falls below 1.25 x VBAT the device terminates an access in progress and resets the device address counter. Inputs to the device will

not be recognized at this time to prevent erroneous data from being written to the device from an out of tolerance system. When VCC falls below VBAT the device switches into a low-current battery backup mode. Upon power-up, the device switches from battery to VCC when VCC is greater than VBAT + 0.2V and recognizes inputs when VCC is greater than 1.25 x VBAT. The block diagram in Figure 1 shows the

main elements of the serial RTC.

4.4.2 DS1307 Block Diagram:

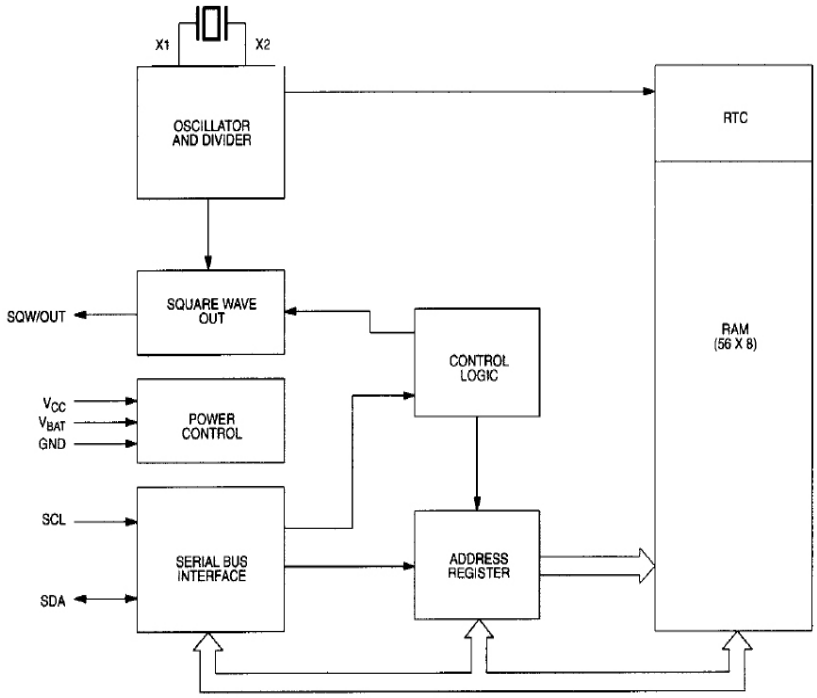

Figure-4.3: Block Diagram of DS1307

4.5 Signal Description:

VCC, GND:

DC power is provided to the device on these pins. VCC is the +5V input. When 5V is applied within normal limits, the device is fully accessible and data can be written and read. When a 3V battery is connected to the device and VCC is below 1.25 x VBAT, reads and writes are inhibited. However, the timekeeping function continues unaffected by the lower input voltage. As VCC falls below VBAT the RAM and timekeeper are switched over to the external power supply (nominal 3.0V DC) at VBAT.

VBAT:

Battery input for any standard 3V lithium cell or other energy source. Battery voltage must be held between 2.0V and 3.5V for proper operation. The nominal write protect trip point voltage at which access to the RTC and user RAM is denied is set by the internal circuitry as 1.25 x VBAT nominal. A lithium battery with 48mAhr or greater will back up the IC DS1307 for more than 10 years in the absence of power at 25ºC. UL recognized to ensure against reverse charging current when used in conjunction with a lithium battery.

SCL (Serial Clock Input):

SCL is used to synchronize data movement on the serial interface.

SDA (Serial Data Input /Output):

SDA is the input/output pin for the 2-wire serial interface. The SDA pin is open drain which requires an external pull-up resistor.

SQW/OUT (Square Wave/Output Driver):

When enabled, the SQWE bit set to 1, the SQW/OUT pin outputs one of four square wave frequencies (1Hz, 4kHz, 8kHz, 32kHz). The SQW/OUT pin is open drain and requires an external pull-up resistor. SQW/OUT will operate with either Vcc or Vbat applied.

X1, X2 :

Connections for a standard 32.768kHz quartz crystal. The internal oscillator circuitry is designed for operation with a crystal having a specified load capacitance (CL) of 12.5pF. The DS1307 can also be driven by an external 32.768kHz oscillator. In this configuration, the X1 pin is connected to the external oscillator signal and the X2 pin is floated.

RECOMMENDED LAYOUT FOR CRYSTAL:

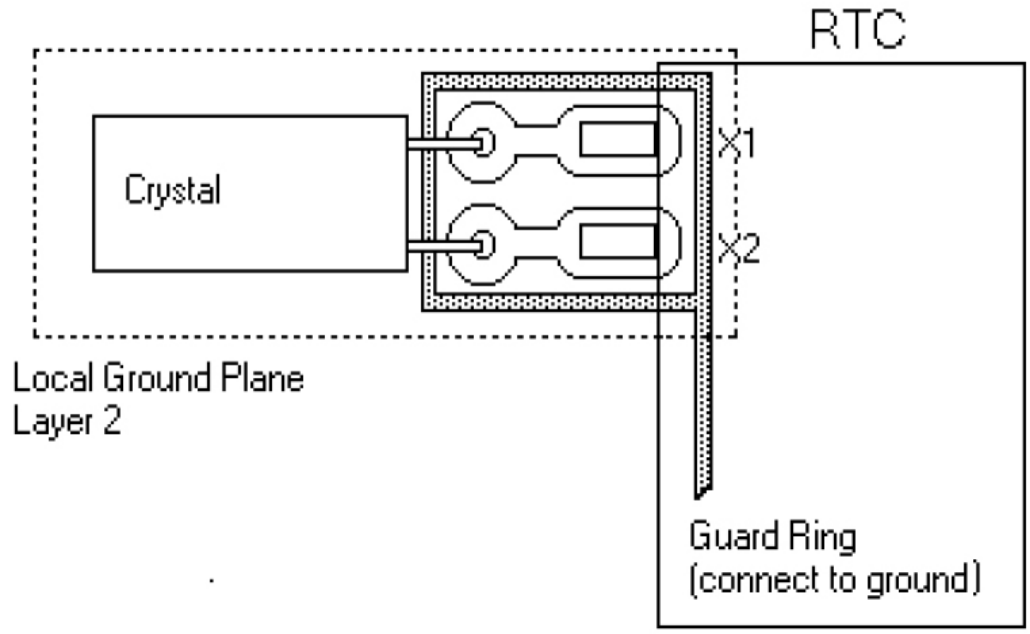

Figure-4.4: Layout for Crystal

4.6 Clock Accuracy:

The accuracy of the clock is dependent upon the accuracy of the crystal and the accuracy of the match between the capacitive load of the oscillator circuit and the capacitive load for which the crystal was trimmed. Additional error will be added by crystal frequency drift caused by temperature shifts. External circuit noise coupled into the oscillator circuit may result in the clock running fast.

4.6.1 RTC and RAM Address Map:

The address map for the RTC and RAM registers of the DS1307 is shown in Figure 2. The RTC registers are located in address locations 00h to 07h. The RAM registers are located in address locations 08h to 3Fh. During a multi-byte access, when the address pointer reaches 3Fh, the end of RAM space, it wraps around to location 00h, the beginning of the clock space.

4.6.2 DS1307 Address Map:

Figure-4.5: Address Map of DS1307

4.7 Clock:

The time information is obtained by reading the appropriate register bytes. The RTC registers are illustrated in Figure 4.6. The time and calendar are set or initialized by writing the appropriate register bytes. The contents of the time and calendar registers are in the BCD format. Bit 7 of register 0 is the clock halt (CH) bit. When this bit is set to a 1, the oscillator is disabled. When cleared to a 0, the oscillator is enabled.

The DS1307 can be run in either 12-hour or 24-hour mode. Bit 6 of the hours register is defined as the 12- or 24-hour mode select bit. When high, the 12-hour mode is selected. In the 12-hour mode, bit 5 is the AM/PM bit with logic high being PM. In the 24-hour mode, bit 5 is the second 10 hour bit (20- 23 hours).

On a 2-wire START, the current time is transferred to a second set of registers. The time information is read from these secondary registers, while the clock may continue to run. This eliminates the need to reread the registers in case of an update of the main registers during a read.

4.8 DS1307 Timekeeper Register:

	BIT7							BIT0	
00H	CH	10 SECONDS			SECONDS				00–59
	0	10 MINUTES			MINUTES				00–59
	0	12/24	10 HR A/P	10 HR	HOURS				01–12 00–23
	0	0	0	0	0	DAY			1–7
	0	0	10 DATE		DATE				01–28/29 01–30 01–31
	0	0	0	10 MONTH	MONTH				01–12
	10 YEAR				YEAR				00–99
07H	OUT	0	0	SQWE	0	0	RS1	RS0	

Figure-4.6: Time keeper Resistor of DS1307

4.8.1 Control Register:

The DS1307 control register is used to control the operation of the SQW/OUT pin.

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
Out	0	0	SQWE	0	0	RS1	RS0

Table-4.1: Control Resister

4.8.2 OUT (Output control):

This bit controls the output level of the SQW/OUT pin when the square wave output is disabled. If SQWE = 0, the logic level on the SQW/OUT pin is 1 if OUT = 1 and is 0 if OUT = 0.

4.8.3 SQWE (Square Wave Enable):

This bit, when set to a logic 1, will enable the oscillator output. The frequency of the square wave output depends upon the value of the RS0 and RS1 bits. With the square wave output set to 1Hz, the clock registers update on the falling edge of the square wave.

4.8.4 RS (Rate Select) :

These bits control the frequency of the square wave output when the square wave output has been enabled. Table 1 lists the square wave frequencies that can be selected with the RS bits.

4.9 2- Wires Serial Data Bus:

The DS1307 supports a bi-directional, 2-wire bus and data transmission protocol. A device that sends data onto the bus is defined as a transmitter and a device receiving data as a receiver. The device that controls the message is called a master. The devices that are controlled by the master are referred to as slaves. The bus must be controlled by a master device that generates the serial clock (SCL), controls the bus access, and generates the START and STOP conditions. The DS1307 operates as a

slave on the 2- wire bus. A typical bus configuration using this 2-wire protocol is show in Figure 4.7.

4.9.1 Typical 2-Wire Bus Configuration:

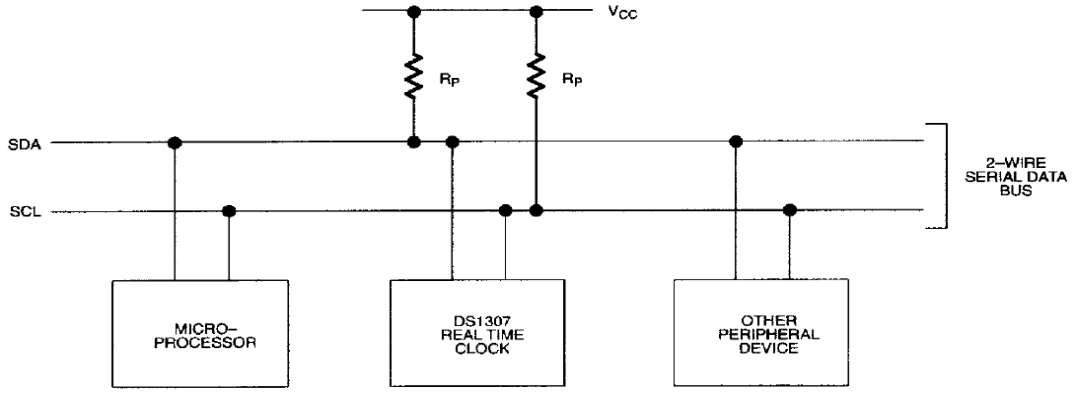

Figure-4.7: Bus Configuration

4.10 Notes:

- ICCS specified with VCC = 5.0V and SDA, SCL = 5.0V.
- VCC = 0V, VBAT = 3V.
- After this period, the first clock pulse is generated.
- A device must internally provide a hold time of at least 300ns for the SDA signal (referred to the VIHMIN of the SCL signal) in order to bridge the undefined region of the falling edge of SCL.
- The maximum tHD: DAT has only to be met if the device does not stretch the LOW period (tLOW) of the
- SCL signal.
- CB – Total capacitance of one bus line in pF.
- ICCA – SCL clocking at max frequency = 100kHz.
- VPF measured at VBAT = 3.0V.

Chapter 5

5.1 Construction

5.1.1 Parts List

This project was constructed using Vero strip board, as it is quick to work with. A parts list is shown below:

Description	Quantity
Vero Strip Board	3 pc (7cm*9cm)(22cm*10cm)(5cm*9cm)
PIC MEGA8	1 pc
DS 1307	1 pc
32768Hz Crystal Oscillator	1 pc
4K7 Resistor	3 pc
4.7Ω Resistor	8 pc
104 pf Capacitor	3 pc
4700µF25V Capacitor	1 pc
1N4007 Diode	4 pc
L7805CV	1 pc
Lithium Cell Battery	1 pc
Push Switch	2 pc
Red LED	73 pc
220V/6V 1A Transformer	1 pc
28 Pin DIL Socket	1 pc
8 Pin DIL Socket	1 pc

Table-5.1: Parts List of Project

I used a crystal as the timing source because of its accuracy. However I did notice that a certain amount of seconds were gained during the day.

This project is constructed on three circuit boards. The top one contains the displays. There is not enough room on the boards to link the displays on the top so the displays are linked together on the copper side using stranded insulated wire.

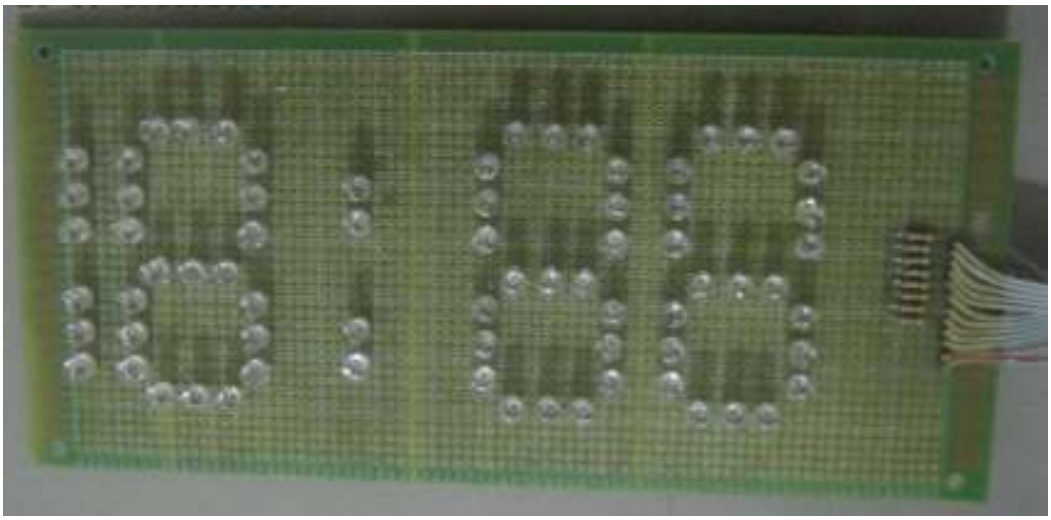

Figure-5.1: Display Section

This is back part of display board where we connect these segment each others with insulated wire.

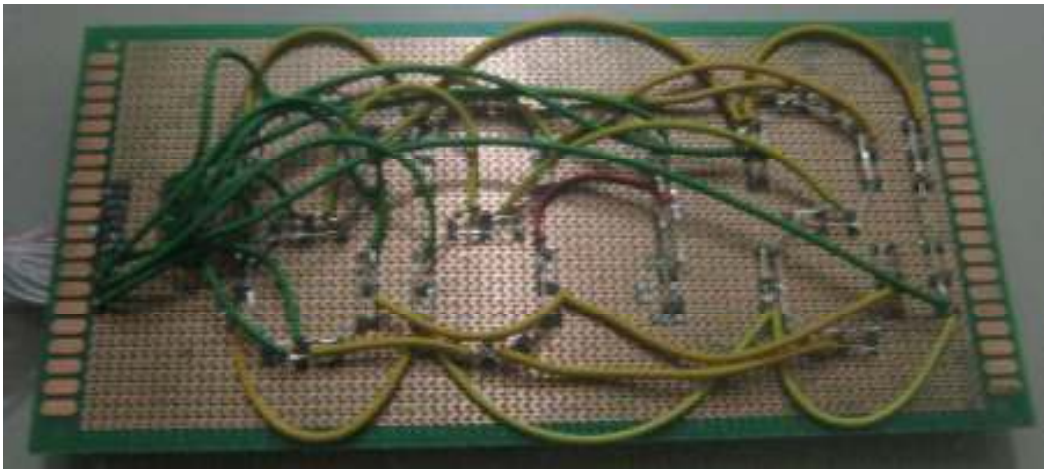

Figure-5.2: Connection of Display Section

Once the top board was complete I started on the lower board. This contains all other components including the programmed PIC MEGA8, DS1307, Battery, Capacitor, Switches and the transistors. Usually the resistors are soldered first as they are less prone to damage by excess heat. Finally solder in the rest of the components.

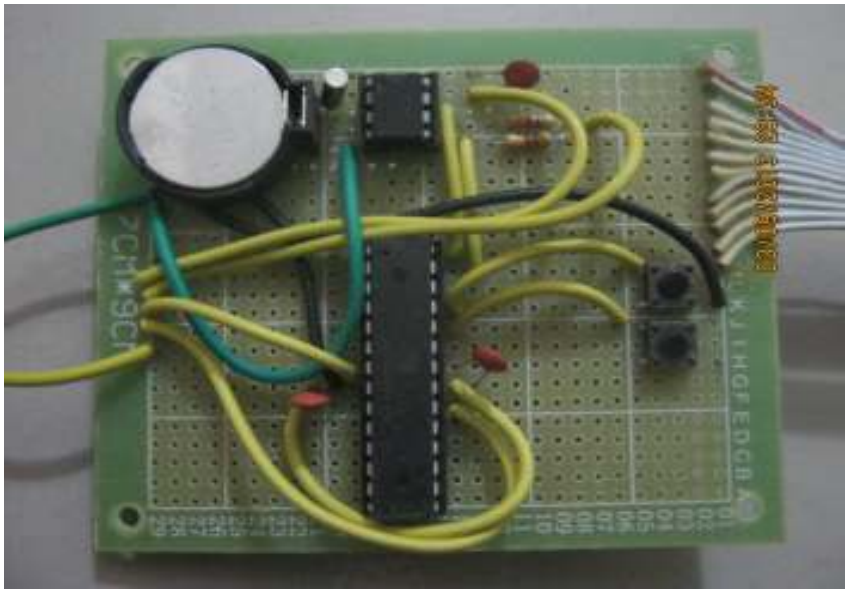

Figure-5.3: Control Section

The other board contains the bridge circuit including 4700µF25V capacitor, 1N4007 diode and L7805CV controller. The transistor is separate from all circuit board,

Figure-5.4: Bridge Section

The three boards are gapped and then connected each others.

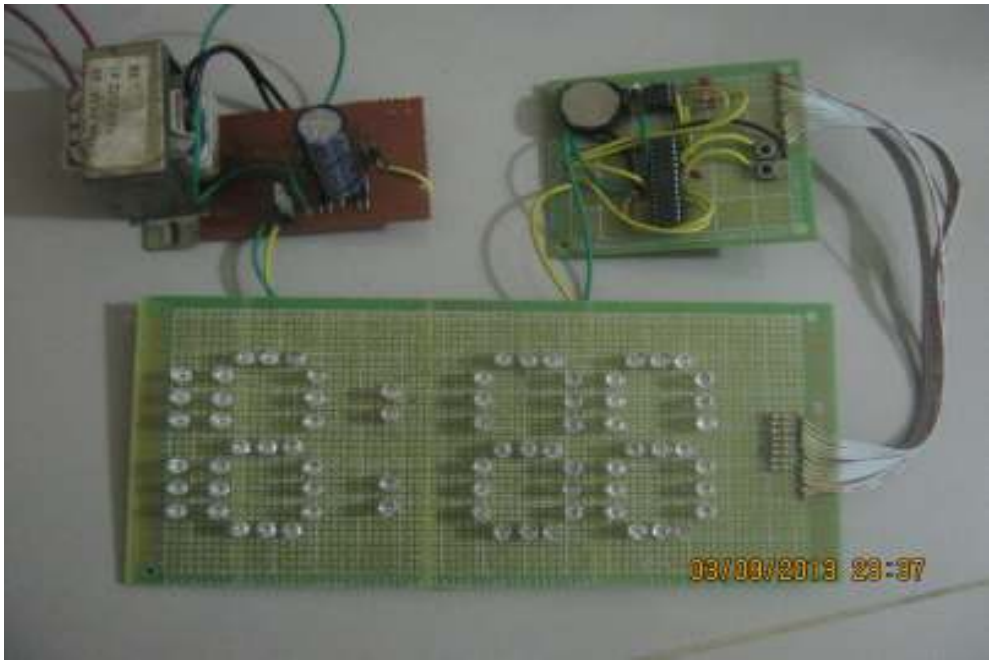

Figure-5.5: The Completed Project

Before plugging in the IC's the other components should be tested. This is done using the following procedure.

1. Connect 5v Power supply
2. Measure the voltage and polarity of the pins that supply power to the IC's.
3. Connect the Segment pins to PIC MEGA8

All that was left to do was to plug in the PIC MEGA8 and connect the 5V supply to test. Set the time to an accurate clock and check periodically to see if it gains or loose time.

The display section

Figure-5.6: Display Output

5.2 Input Power System

The digital clock operate in DC power source, so we decide input power

Supply source is Bridge Rectifier (e.g. ac to dc converter). That's why we make a bridge rectifier. Here needed a 4700µF25V capacitor and 1N4007 diode. Then we supply ac power throw a 220V/6V transformer. Finally we connect a L7805CV controller to ensure 5V supply to the system.

Figure-5.7: Bridge Circuit

Chapter 6

6.1 Conclusion

We have designed a unique PIC and seven segments based intelligent Digital clock successfully. Its required power is very low. To make it user friendly we have kept different button for second, minute, hour to operate the device.

The accuracy of the clock is dependent upon the accuracy of the crystal and the accuracy of the match between the capacity load of the oscillator circuit and the capacitive load for which the crystal was trimmed. Additional error will be added by crystal frequency drift caused by temperature shifts. External circuit noise coupled into the oscillator circuit may result in the running fast.

This project provided significant information that will enable to build a complete Digital Clock. Overall we are very hopeful about our proposed device

Program Code in C++:

```
/*
 mega8 pinout
        _____ +---------------+
      pc6/reset |  1          28 | pc5
          pd0   |  2             27 | pc4
          pd1   |  3          26 | pc3
          pd2   |  4          25 | pc2
          pd3   |  5          24 | pc1
          pd4   |  6          23 | pc0
          VCC |  7          22 | GND
          GND |  8          21 | AREF
     pb6/xtal1|  9          20 | AVCC
     pb7/xtal2| 10          19 | ISP/pb5
          pd5   | 11          18 | ISP/pb4
          pd6   | 12          17 | ISP/pb3
          pd7   | 13          16 | pb2
          pb0   | 14          15 | pb1
                +---------------+
*/
// Includes
#include <inttypes.h>
```

```
#include <avr/interrupt.h>
#include "avr\iom8.h"
#include <avr/io.h>
#include <util/delay.h>
#include <avr/eeprom.h>
#include <avr/wdt.h>
#include <compat/twi.h>
#include <avr/signal.h>

// constants
#define FCPU        8000000     /* CPU speed */

uint8_t         min;
uint8_t         hour;
uint8_t         sec;
uint8_t         ms_125;
uint8_t         delay=2;                    //delay variable for display refersh rate control
uint8_t         key_time;                   // Save & Exit from sub menu
uint8_t         blink;
uint8_t         seconds,minutes,hours,day,date,month,year,am_pm;
uint8_t          key,key_time_out, time_set;
```

```
#define DS1307ADR 104
#definesg1      PB0
#define sg2     PB1
#define sg3     PB2
#define sg4     PB3
#define set     PC1
#defineup       PC2
#definedn       PC3                         //Look up table for seven segment driver

uint8_t                                 segment[43]                                 =
{0x3F,0x06,0x5B,0x4F,0x66,0x6D,0x7D,0x07,0x7F,0x6F,0x80,0x00,0x77,0x7C,0x5
8,0x39,0x5E,0x79,0x71,0x3D,0x76,0x74,0x30,0x1E,0x70,0x28,0x55,0x54,0x5C,0x7
3,0x6B,0x50,0x64,0x78,0x3E,0x1C,0x2A,0x6A,0x49,0x6E,0x52,0x10,0x40};
void Timer0_init(void)
{
        TCNT0=0x06;
        TCCR0|=_BV(CS02);               //prescaller 1024 interrupts/s
        TIMSK|=_BV(TOIE0);              //Enable Timer0 Overflow interrupts
        TIFR|=_BV(TOV0);          //Enable Timer0 Overflow flag
}
ISR(TIMER0_OVF_vect)
{
        TIMSK&=~_BV(TOIE0);             //Enable Timer0 Overflow interrupts
        TCNT0=0x06;                 // Reload value for next count
```

```
        ms_125++;
        if(ms_125>=125) {ms_125 = 0; key_time++;}
//display();
        TIMSK|=_BV(TOIE0);                  //Enable Timer0 Overflow interrupts
        wdt_reset();                        // Reset WDT
}
void twi_init(void)
{
        TWSR = 0x00;
        //TWSR |= (1 << TWPS1) | (1 << TWPS0);  Prescaler=1
        TWBR = 152;                         // SCL=100KHz @ 8 MHz
        //TWCR |= (1 << TWEN);     //TWI aktivieren
}
void twi_start(void)
{
        TWCR = (1 << TWSTA) | (1 << TWEN) | (1 << TWINT);
        while(!(TWCR & (1 << TWINT)));
}
void twi_writeadr(uint8_t adr, uint8_t wr)     //W=0 R= alles außer 0
{
        adr=(DS1307ADR << 1);
        if(wr)
```

```
	adr |= 1;			//Beim lesen SLA+R übertragen ansonsten LSB auf 0 lassen (SLA+W)
	TWDR=adr;
	TWCR = (1 << TWEN) | (1 << TWINT);
	while(!(TWCR & (1 << TWINT)));
}
void twi_writedata(uint8_t data)
{
	TWDR=data;
	TWCR = (1 << TWEN) | (1 << TWINT);
	while(!(TWCR & (1 << TWINT)));
}
void twi_stop(void)
{
	TWCR = (1 << TWEN) | (1 << TWSTO) | (1 << TWINT);
	while(TWCR & (0 << TWSTO));
}
uint8_t twi_readdata(void)
{
	uint8_t data;
	TWCR = (1 << TWEN) | (1 << TWINT);
	while(!(TWCR & (1 << TWINT)));
	data = TWDR;
```

```
        return data;
}
void ds1307_writereg(uint8_t adr,uint8_t data)
{
        twi_start();
        twi_writeadr(DS1307ADR,TW_WRITE);
        twi_writedata(adr);
        twi_writedata(data);
        twi_stop();
}
uint8_t ds1307_readreg(uint8_t adr)
{
        uint8_t data;
        //_delay_us(60);
        twi_start();
        twi_writeadr(DS1307ADR,TW_WRITE);
        twi_writedata(adr);
        twi_stop();
        _delay_us(60);
        twi_start();
        twi_writeadr(DS1307ADR,TW_READ);
        data=twi_readdata();
        twi_stop();
```

```
        return data;
}
void ds1307_gettime(void)
{
        seconds = ds1307_readreg(0);
        minutes = ds1307_readreg(1);
        hours = ds1307_readreg(2);
        day = ds1307_readreg(3);
        date = ds1307_readreg(4);
        month = ds1307_readreg(5);
        year = ds1307_readreg(6);
}
void display(void)
{
        if(blink<3)
{
                ////////////////////// Hour ////////////////////////////
        hours = ds1307_readreg(2);
        if(blink == 1)
{
        if(ms_125<90)
                                {
```

```
                PORTB &=~_BV(sg1);                      //Turn    on
Digit 1

                if((hours >> 4) & 0x01) PORTD |= segment[1];

                else PORTD |= segment[11];  // show null

                _delay_ms(delay);

                PORTB |=_BV(sg1);
                                        //Turn off Digit 1

                PORTD = 0x00;

                _delay_us(150);


                PORTB &=~_BV(sg2);
                                                //Turn on Digit 2

                PORTD |= segment[hours & 0x0f];

                _delay_ms(delay);

                PORTB |=_BV(sg2);
                                        //Turn off Digit 2

                PORTD = 0x00;

                _delay_us(150);

                }

        else

                {

                PORTB &=~_BV(sg1);      //Turn on Digit 1

                PORTD |= segment[11];       // show null

                _delay_ms(delay);
```

```
                PORTB |=_BV(sg1);  //Turn off Digit 1
                PORTD = 0x00;
                _delay_us(150);
                PORTB &=~_BV(sg2);          //Turn on Digit 2
                PORTD |= segment[11];
                _delay_ms(delay);
                PORTB |=_BV(sg2);  //Turn off Digit 2
                PORTD = 0x00;
                _delay_us(150);
                }
        }
else
        {
        PORTB &=~_BV(sg1);                  //Turn on Digit 1
        if((hours >> 4) & 0x01) PORTD |= segment[1];
        else PORTD |= segment[11];              // show null
        _delay_ms(delay);
        PORTB |=_BV(sg1);              //Turn off Digit 1
        PORTD = 0x00;
        _delay_us(150);
        PORTB &=~_BV(sg2);                  //Turn on Digit 2
        PORTD |= segment[hours & 0x0f];
        _delay_ms(delay);
```

```
                PORTB |=_BV(sg2);            //Turn off Digit 2
                PORTD = 0x00;
                _delay_us(150);
                }

//////////////////////////////// minutes //////////////////////
        minutes = ds1307_readreg(1);
        if(blink==2)
                {
                if(ms_125<90)
                        {
                        PORTB &=~_BV(sg3);              //Turn on Digit 3
                        PORTD |= segment[minutes >> 4];
                        _delay_ms(delay);
                        PORTB |=_BV(sg3);           //Turn off Digit 3
                        PORTD = 0x00;
                        _delay_us(150);
                        PORTB &=~_BV(sg4);              //Turn on Digit 4
                        PORTD |= segment[minutes & 0x0f];
                        _delay_ms(delay);
                        PORTB |=_BV(sg4);           //Turn off Digit 4
                        PORTD = 0x00;
                        _delay_us(150);
```

```
            }
        else
            {
            PORTB &=~_BV(sg3);                  //Turn on Digit 3
            PORTD |= segment[11];
            _delay_ms(delay);
            PORTB |=_BV(sg3);               //Turn off Digit 3
            PORTD = 0x00;
            _delay_us(150);
            PORTB &=~_BV(sg4);                  //Turn on Digit 4
            PORTD |= segment[11];
            _delay_ms(delay);
            PORTB |=_BV(sg4);               //Turn off Digit 4
            PORTD = 0x00;
            _delay_us(150);
            }
        }
    else
        {
        PORTB &=~_BV(sg3);                  //Turn on Digit 3
        PORTD |= segment[minutes >> 4];
        _delay_ms(delay);
        PORTB |=_BV(sg3);               //Turn off Digit 3
```

```
                PORTD = 0x00;
                _delay_us(150);
                PORTB &=~_BV(sg4);                  //Turn on Digit 4
                PORTD |= segment[minutes & 0x0f];
                _delay_ms(delay);
                PORTB |=_BV(sg4);             //Turn off Digit 4
                PORTD = 0x00;
                _delay_us(150);
                }
        }
else
        {
        ////////////////////////// seconds ////////////////////
        seconds = ds1307_readreg(0);
        PORTB &=~_BV(sg1);                          //Turn on Digit 5
        PORTD |= segment[42];
        _delay_ms(delay);
        PORTB |=_BV(sg1);                   //Turn off Digit 5
        PORTD &= 0x80;
        _delay_us(150);
        PORTB &=~_BV(sg2);                          //Turn on Digit 6
        PORTD |= segment[42];
        _delay_ms(delay);
```

```
                PORTB |=_BV(sg2);                    //Turn off Digit 6
                PORTD &= 0x80;
                _delay_us(150);
                PORTB &=~_BV(sg3);                   //Turn on Digit 5
                PORTD |= segment[(seconds>>4) & 0x07];
                _delay_ms(delay);
                PORTB |=_BV(sg3);                    //Turn off Digit 5
                PORTD &= 0x80;
                _delay_us(150);
                PORTB &=~_BV(sg4);                   //Turn on Digit 6
                PORTD |= segment[seconds & 0x0f];
                _delay_ms(delay);
                PORTB |=_BV(sg4);                    //Turn off Digit 6
                PORTD &= 0x80;
                _delay_us(150);
                }
        }

// CPU Initialization
void init_cpu(void)
        {
        DDRB = 255;             // oooooooo
        PORTB = 255;                    // iiiiiiii
```

```
	DDRC = 0b11100001;		// -iiioiii
	//DDRC |= _BV(PC6);
	PORTC = 0b00001110;		// -iiiiiii
	DDRD = 255;			// oooiiiii
	PORTD = 0;			// iiiiiooo
	}
void key_check(void)
	{
	key = 0;
	// if set key pressed
	if(bit_is_clear(PINC, set))
		{
		loop_until_bit_is_set(PINC, set);
		_delay_ms(250);
		key = 1;
		key_time = 0;
		}
	// if up key pressed
	if(bit_is_clear(PINC, up))
		{
		loop_until_bit_is_set(PINC, up);
		_delay_ms(250);
		key = 2;
```

```
                key_time = 0;
                }
        // if down key pressed
        if(bit_is_clear(PINC, dn))
                {
                loop_until_bit_is_set(PINC, dn);
                _delay_ms(250);
                key = 3;
                key_time = 0;
                }
        }
// Main program
int main( void )
{
        init_cpu();
        wdt_enable(WDTO_2S);
        Timer0_init();
        sei();                          // enable global intertupt
        twi_init();
        _delay_us(50);
        ds1307_writereg(0x07,0x10);
        _delay_us(50);
        _delay_ms(500);
```

```
while(1)// forever
{
display();
if((key_time>=30) &&(blink<3)) {blink = 0; time_set = 0;}
                        // key time out for 30 seconds
if(key == 1) {time_set = 1; key_time = 0; blink ++; if(blink==4) blink=1;}
if(key==1 && blink==3 && key_time>=30) {blink = 0; time_set = 0;}
// hour set
if(time_set==1 && blink==1 && key==2)
        {
        key_time = 0;
        hours = ds1307_readreg(2);
        hour = (((hours >> 4) & 0x01)*10) + (hours & 0x0f);
        hour ++;
        if(hour==12)
                {
                if(am_pm == 16) am_pm = 32;
                else am_pm = 16;
                }
        if(hour>12) hour -= 12;
        hour = ((hour/10)<<4) + (hour%10);
        if(am_pm==16) hour |= 0x40;
        else hour |= 0x60;
```

```
        ds1307_writereg(2,hour);
        }
// minutes set up
if(time_set==1 && blink==2 && key==2)
        {
        key_time = 0;
        minutes = ds1307_readreg(1);
        min = ((minutes >> 4)*10) + (minutes & 0x0f);
        min ++;
        if(min>59) min = 0;
        min = ((min/10)<<4) + (min%10);
        ds1307_writereg(1,min);
        }
// minutes set down
if(time_set==1 && blink==2 && key==3)
        {
        key_time = 0;
        minutes = ds1307_readreg(1);
        min = ((minutes >> 4)*10) + (minutes & 0x0f);
        if(min>0) min --;
        if(min==0) min = 59;
        min = ((min/10)<<4) + (min%10);
        ds1307_writereg(1,min);
```

```
                }
        }
        return 0;
}Ω
```

REFERENCE

[1] wineyard.in/wp-content/uploads/2012/12/WYARD9.pdf

[2] http://www.pelnet.co.uk/elect/projects/clock/Clock_a.pdf

[3] http://en.wikipedia.org/wiki/Digital_clock

[4] www.ee.duke.edu/~jmorizio/ece261/F09/projects/clk.pdf

[5] https://www.hni.uni-paderborn.de/uploads/media/doc2486_01.pdf

[6] datasheets.maximintegrated.com/en/ds/DS1307.pdf