

Microcontroller Based Portable Paramedic Blood Warmer for Transfusion

**Submitted in partial fulfillment of the requirements for the award
of Bachelor of Engineering degree in**

ELECTRONICS AND COMMUNICATION ENGINEERING

BY

M.AMRUTHAVALLI (Reg no: 38130702)

M.SHANKARI MALAKAR (Reg no: 38130705)



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

SCHOOL OF ELECTRICAL AND ELECTRONICS ENGINEERING

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600 119**

MAY-2022



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with "A" grade by NAAC

Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai 600119

www.sathyabama.ac.in

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of M.AMRUTHAVALLI (Reg no:38130702), M.SHANKARIMALAKAR (Reg no:38130705) who carried out the project entitled **"MICROCONTROLLER BASED PORTABLE PARAMEDIC BLOOD WARMER FOR TRANSFUSION"** Under our supervision from December 21 to April 22

Internal Guide

(Dr.S.EMALDA ROSLIN)

External Guide

(Dr.R.PANDIAN)

Head of the Department

Submitted for Viva voce Examination held on 2.5.2022

Internal Examiner

External Examiner

DECLARATION

M.AMRUTHAVALLI, and M.SHANKARI MALAKAR , hereby declare that the Project Report entitled “**Microcontroller Based Portable Paramedic Blood Warmer for Transfusion**” done by us under the guidance of **Dr.P.KAVIPRIYA** at **SATHYABAMA INSTITUTE OF SCIENCEAND TECHNOLOGY** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering Degree in **ELECTRONICS AND COMMUNICATIONENGINEERING**.

DATE:

SIGNATURE OF THECANDIDATE

PLACE:

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of SATHYABAMA for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. N. M. NANDITHA**, Dean, School of ELECTRICAL AND ELECTRONICS and **Dr.S.RAVI** Head of the Department, Dept. of ELECTRONICS AND COMMUNICATION for providing necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide Dr. **P.KAVIRIYA** for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of ELECTRONICS AND COMMUNICATION ENGINEERING who were helpful in many ways for the completion of the project.

ABSTRACT

Main aim of this project is to heat the stored blood by using warmer based on the patient's body temperature. Temperature maintenance is the major consideration for the effective and safe handling of the patient. Mistakes regarding to it can lead to life-threatening condition for the patient. The normal body temperature of human is 37.5 °C. The blood from blood bank is stored at lower temperature about 2 °C – 6 °C. Here two temperature sensor are been used where one is to detect the body temperature and another sensor to detect the blood bank temperature. Therefore we introduce microcontroller based portable paramedic blood warmer which aims to decrease the time for warming the blood by providing controlled heating. Both sensors send the output to microcontroller, inbuilt with ADC (Analog-to- Digital Converter). Peltier is a device that can be used either for heating or for cooling .It can also be used as a temperature controller. Relay driver is used to control the vibration due to abnormal condition of the patient. The output is given to LCD display unit which monitors the heat. To avoid hypothermic adverse effects in the patient body while transfusion, real time comparison of blood bag temperature and patient body temperature and accordingly heating is provided. Although blood warmer has advantages of no contamination, easy manipulation, portability, clear digital temperature indication, with fast result it has also limitations and challenges like inability to cool the heated blood and using restriction. The aspects of component specifications, accuracy, dynamicity and appearance may differ from the actual device.

CONTENTS

CHAPTER NO	TITLE	PAGE NO
1.	INTRODUCTION	12
2.	LITERATURE SURVERY	13
2.1	EXISTING SYSTEM	13
2.2	MICROCONTROLLER BASED PORTABLE PARAMEDIC BLOOD WARMER FOR TRANSFUSION	13
2.3	BLOOD INFUSION WARMER	14
2.4	INTRODUCING TWO NEW DEVICES FOR BLOOD WARMING	14
2.5	AN ELECTRONIC BARRIER SYSTEM TO IMPROVE BLOOD TRANSFUSION SAFETY	15
3.	PROPOSED SYSTEM	16
3.1	BLOCK DIAGRAM	16
4.	HARDWARE DESCRIPTION	17
4.1	POWER SUPPLY UNIT	17
4.2	INTRODUCTION	17
4.3	PIN DIAGRAM	18
4.3.1	ATmega328pinout	19
4.4	BLOCK DIAGRAM	19

4.5	PIN DESCRIPTION	20
4.6	ARCHITECHURE DIAGRAM	21
4.7	RESET AND INTERRUPT HANDLING	25
4.8	MEMORY DESCRIPTION	26
4.9	REGISTERS	28
4.10	ATmega328 and Arduino	29
4.11	UART	29
4.12	LIQUID CRYSTAL DISPLAY (LCD)	43
	4.12.1 INTRODUCTION	43
4.13	INSTRUCTION AND DATA REGISTER	46
4.14	COMMANDS AND INSTRUCTION SET	46
4.15	TEMPERATURE SENSOR	47
	4.15.1 LM35: Precision Centigrade Temperature Sensors	47
	4.15.2 GENERAL DESCRIPTION	47
4.16	THERMOELECTRIC EFFECT	49
4.17	WHAT IS THE PELTIER EFFECT?	49
4.18	Code	50
4.19	PELTIER MODULE	54
4.20	VIBRATION SENSOR	55

4.21	CIRCUIT DIAGRAM	57
4.22	RELAY	59
4.23	SOLENOID VALVE	60
	4.23.1 OPERATING PRINCIPLES	60
	4.23.2 Principles of operation	60
4.24	HEART BEAT SENSOR	61
	4.24.1 SPECIFICATION	61
5.	SOFTWARE REQUIREMENT	62
	5.1 ARDUINO IDE – 1.8.5	63
	5.2 Why Arduino?	63
	5.3 EMBEDDED C	65
6.	RESULT	69
7.	CONCLUSION	72
8.	REFERENCES	73

LIST OF ABBREVIATIONS

UART- UNIVERSEL ASYNCHRONOUS RECEIVER/TRANSMITTER

**EEPROM - ELECTRICALLY ERASABLE PROGRAMMABLE READ ONLY
MEAMORY**

SRAM - STATIC RANDOM ACCESS MEMORY

ADC - ANALOG-TO-DIGITAL CONVERTER

ALU - ARITHMETIC LOGIC UNIT

PC- PROGRAM COUNTER

ROM - READ ONLY MEMORY

LPM - LOAD PROGRAM MEMORY

GP - GENERAL PURPOSE

TX - TRANSMITTING DATA

RX - RECEIVING DATA

USB - UNIVERSL SERIAL BUS

SPI - SERIAL PERIPHERAL INTERFACE

MOSI - MASTER SLAVE IN

MISO - MASTER IN SLAVE OUT

I2C- INTER-INTEGRATED CIRCUIT

LCD - LIQUID CRYSTAL DISPLAY

CRT- CATHODE RAY TUBE

ITO - INDIUM TIN OXIDE

IR - INSTRUCTION REGISTER

BPM - BEATS PER MINUTE

LIST OF TABLES:

TABLE NO	TITLE	PAGE NO
4.12	PIN DETAIL OF LCD	45
4.14	LCD INSTRUCTION COMMANDS	47

LIST OF FIGURES

FIG NO	TITLE	PAGE NO
---------------	--------------	----------------

4.11	Hardware connection diagram for UART	30
4.11	GPIO0 and GPIO1 are serial RX and TX.	30
	Any GPIO pin can be used as serial RC and TX with the software serial library.	
4.11	UART data packet visualized/electric imp	31
4.11	Hardware connection diagram for SPI	33
4.11	Multiple-slave connected to a single master	34
4.11	CPOL and CPHA settings visualized	36
4.11	The MOSI, MISO and SCK pins are broken out on the ICSP headers as well as on GPIO11, GPIO12, and GPIO13	37
4.11	I2C hardware connection diagram	39
4.11	What the address and data portions of the communication.	39
4.11	I2C (Wire) SDA is Analog pin 4(A4) SCL is Analog pin 5(A5)	40
4.15	Full-Range centigrade Temperature sensor	49
5.3.1	Connection of Blood Infusion Warmer	66
5.3.2	Manual Value of Blood Infusion Warmer	67
5.3.3	Auto Mode of Blood Infusion Warmer	68
5.3.4	Total Hardware	71

CHAPTER-1

INTRODUCTION:

The normal human body temperature is 37.5°C. The blood is stored at lower temperature to lower the metabolic rate and bacterial growth. Temperature maintenance is the major consideration for the effective and safe handling of the patient. Mistakes regarding to it can lead to life-threatening condition for the patient. Warmed blood is required the following clinical circumstances: (i) An adult receiving frequently more than 50 ml/kg/h blood; (ii) A child receiving more than 15 ml/kg/h blood; (iii) Blood exchange in neonates; (iv) a patient with active cold agglutinin; (v) sever forms of nocturnal paroxysmal hemoglobinuria; (vi) blood transfusion from central veins (cold blood could cause arrhythmias and cardiac arrest).The procedure is minimally invasive, portable and time saving and hence useful in emergencies Patients who have major trauma have chances of getting affected by Hypothermia. The major factors that hypothermia contributes are severity of injury, and anesthesia/sedation agents. Unwarmed blood and crystalloid infusion results in coagulopathy, alterations in response to pharmacological agents, cardiac arrhythmias which results in life loss. Keeping these factors in mind it is always important to warm the Intravenous fluid before administering the patients.After a unit of blood is collected, it is prepared and then cooled for storage in the blood bank. When many units of cooled blood are given in a short time, body temperature may drop to threatening level, a condition called hypothermia. Hypothermia is due to the use of anesthetics and cold intravenous fluids. Also, maintaining a surgical patient's normal body temperature has been shown to reduce infection, speed healing, shorten hospital stays, and diminish the chance of serious heart injury. The motive is to measure the temperature accurately.

CHAPTER-2

LITERATURE SURVEY:

2.1 EXISTING SYSTEM:

There are several methods to warm IV fluids and bloods are currently available. These include immersing coiled IV tubing in a water bath, microwaving the bag of fluid to be infused, adding heated saline to blood to be infused, passing the IV tubing through a heating block or through a plastic tube warmed with forced air, passing the IV tubing through a conductive surface interfaced with a counter current heated water bath, magnetic induction, pre-warming fluids in a convection oven or in a microwave oven. But all these methods consume time and are not safe. To overcome this we introduce Microcontroller based portable paramedic blood warmer for transfusion.

2.2 MICROCONTROLLER BASED PORTABLE PARAMEDIC BLOOD WARMER FOR TRANSFUSION

In this paper involves warming of blood using water bath which takes 15-20min to warm the cold blood. Therefore we introduce microcontroller based portable paramedic blood warmer which aims to decrease the time for warming the blood by providing controlled heating. LM35 precision centigrade temperature sensor is used to measure blood bag temperature. It sends the output to microcontroller, inbuilt with ADC (Analog-to-Digital Converter). Hot air drier is used to heat the blood to meet the required temperature difference which is controlled using PIC microcontroller. Thus the temperature is maintained at 37°C. The output is given to LCD display unit which monitors the heat. This is a novel method as it helps to ensure that there are no aftershocks from the patient as the difference in temperatures can be quite deadly to an extent that it can lead to death sometimes.

2.3 BLOOD INFUSION WARMER

Temperature maintenance is the prime consideration for the effective and safe handling of the patient. As the normal body temperature of human is 37.5 deg. Celsius so it is very dangerous to directly infuse cool blood in patient. To avoid hypothermic adverse effects in the patient body while transfusion, real time comparison of blood bag temperature and patient body temperature and accordingly heating is provided. Although blood warmer has advantages of no contamination, easy manipulation, portability, clear digital temperature indication, with fast result it has also limitations and challenges like inability to cool the heated blood and using restriction. The aspects of component specifications, accuracy, dynamicity and appearance may differ from the actual device. But, this project is sufficient to provide functional simulation to the actual real-time warmer for blood infusion. As the blood is precious so water or dextrose will be used for this project instead of blood.

2.4 INTRODUCING TWO NEW DEVICES FOR BLOOD WARMING

Blood and blood products should be kept in cold temperatures to avoid hemolysis and preserve sterility and functional integrity of blood components. Patients needing massive transfusion require the blood to be warmed at bedside to avoid cardiac and general hypothermia. Hypothermia due to massive and rapid transfusion in ill and injured patients is associated with adverse side effects which are very difficult to be reversed. Here in, we describe two inventions of portable warming devices for blood and blood products and fluid solutions. These devices have several advantages including reduced contact with blood and blood products, instant preparation of blood for injections, decreased damage to blood and blood products, precise controlling of heating process, avoiding physical and chemical changes in quality of blood and blood products and also avoiding hypersensitivity symptoms due to injections.

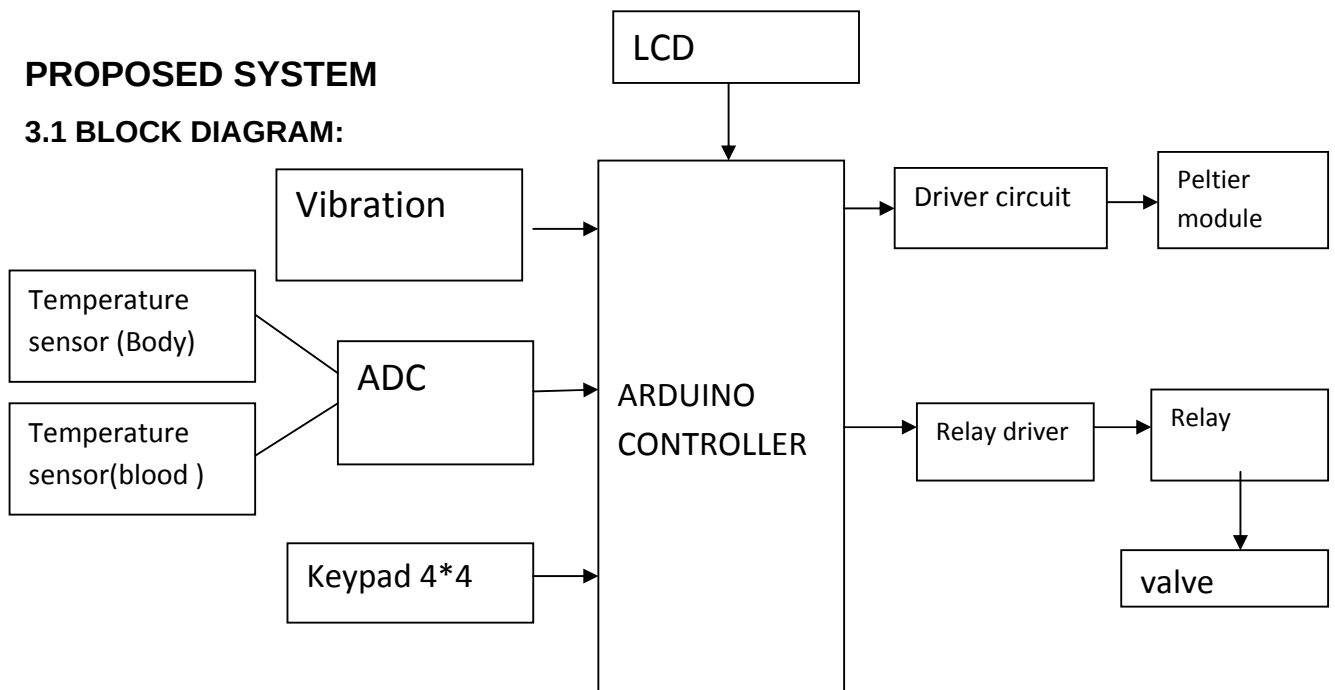
2.5 AN ELECTRONIC BARRIER SYSTEM TO IMPROVE BLOOD TRANSFUSION SAFETY

In blood transfusion procedures, it is necessary to guarantee that each patient actually receives the blood unit which has been assigned to him, so that mistransfusions are avoided. For this reason, before starting the administration, a cross-check involving the patient and the transfusion unit codes must be carried out at bedside. To perform these tests in a safe and fast way, several electronic solutions based on barcode labels and RFID tags have been adopted in the last few years. However, most reliable results are still obtained by means of barrier systems, i.e. mechanical devices that prevent the use of units until the cross-check is positive. In this paper, we present a microcontroller-based barrier system which can easily be integrated with common barcode/RFID-based solutions, so that an even higher safety degree is achievable together with simple and fast utilization. The system provides blood unit temperature monitoring and event logging capabilities, can be controlled by means of a wireless connection, and is powered by a high capacity Li-ion battery, which ensures long operating cycles. Moreover, it features a simple human interface, and appears rather small and light, thus resulting definitely suitable for the clinical settings. Once implemented, the system has been successfully validated by means of a Lab VIEW interface emulating a typical medical facility set-up.

CHAPTER 3:

PROPOSED SYSTEM

3.1 BLOCK DIAGRAM:

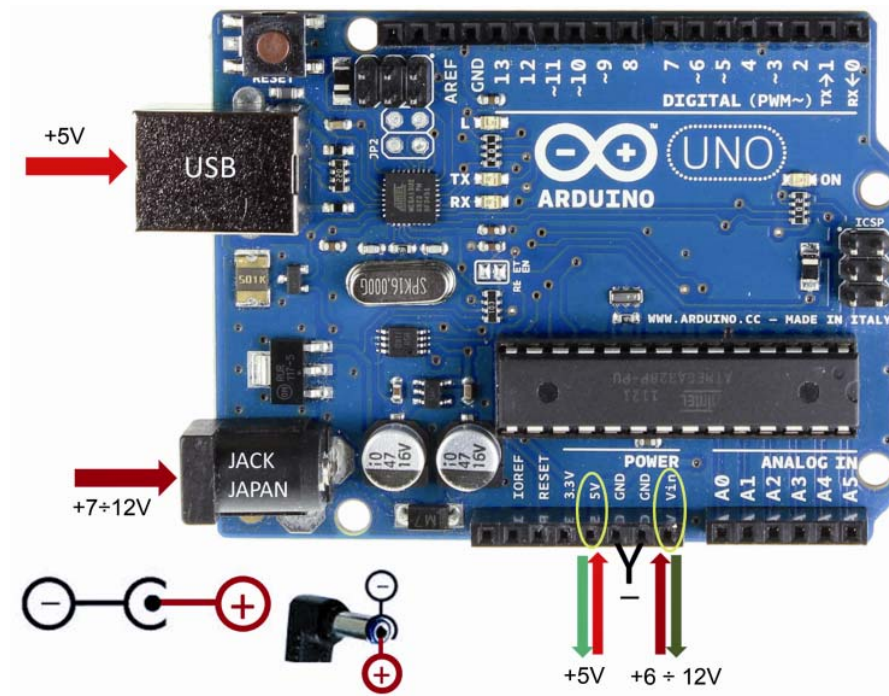


Our approach we are using two LM35 temperature sensors to detect the body temperature and blood temperature. Vibration sensor is used to find the any abnormal condition like fits stage while transfusion of blood. A PIC microcontroller is used which is connected to LCD for display. Both sensors send the output to microcontroller, inbuilt with ADC (Analog-to- Digital Converter).Peltieris a device that can be used either for heating or for cooling .It can also be used as a temperature controller. Relay driver is used to control the vibration due to abnormal condition of the patient. During abnormal situations the transfusion is halted as detected by the vibration sensor. The overall process is done by automatically or manually. For manual operation we can use keypad.

CHAPTER 4:

HARDWARE DESCRIPTION

4.1 POWER SUPPLY UNIT:



- Must be a DC adapter (i.e. it has to put out DC, not AC);
- should be between 9V and 12V DC (see note below);
- must be rated for a minimum of 250mA current output, although you will likely want something more like 500mA or 1A output, as it gives you the current necessary to power a servo or twenty LEDs if you want to.
- Must have a 2.1mm power plug on the Arduino end, and
- The plug must be "centre positive", that is, the middle pin of the plug has to be the + connection.

4.2 INTRODUCTION

ATmega-328 is basically an Advanced Virtual RISC (AVR) micro-controller. It supports the data up to eight (8) bits. ATmega-328 has 32KB internal builtin memory. This micro-controller has a lot of other characteristics.

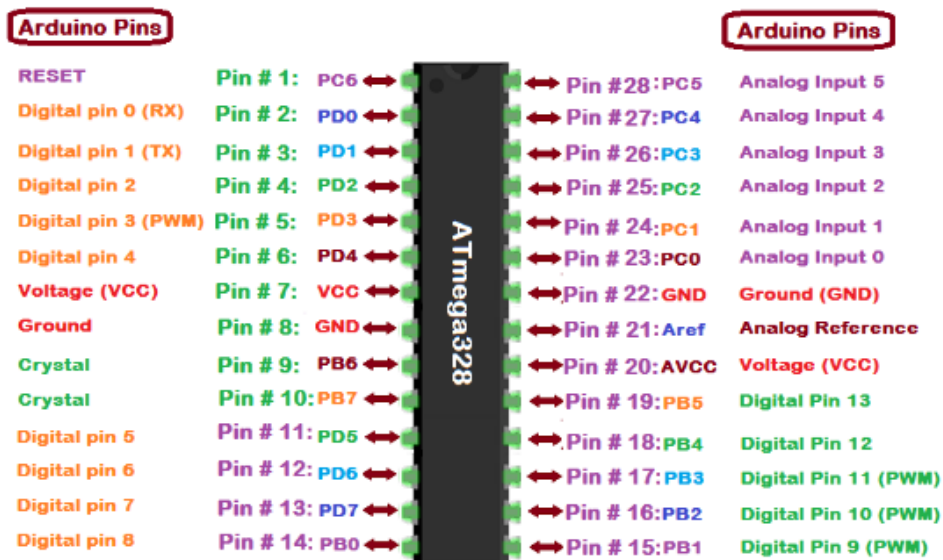
ATmega 328 has 1KB Electrically Erasable Programmable Read Only Memory (EEPROM). This property shows if the electric supply supplied to the micro-controller is removed, even then it can store the data and can provide results after providing it with the electric supply. Moreover, ATmega-328 has 2KB Static Random Access Memory (SRAM). Other characteristics will be explained later. ATmega 328 has several different features which make it the most popular device in today's market. These features consist of advanced RISC architecture, good performance, low power consumption, real timer counter having separate oscillator, 6 PWM pins, programmable serial USART, programming lock for software security, throughput up to 20 MIPS etc. ATmega-328 is mostly used in Arduin **ATmega328** is an eight (8) bit micro-controller. It can handle the data sized of up to eight (8) bits. It is an AVR based micro-controller. Its built in internal memory is around 32KB. It operates ranging from 3.3V to 5V. It has an ability to store the data even when the electrical supply is removed from its biasing terminals. Its excellent features include the cost efficiency, low power dissipation, programming lock for security purposes, real timer counter with separate oscillator.

4.3 PIN DIAGRAM

Atmega328

(PCINT14/ $\overline{\text{RESET}}$) PC6	1	28	PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0	2	27	PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1	3	26	PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2	4	25	PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3	5	24	PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4	6	23	PC0 (ADC0/PCINT8)
VCC	7	22	GND
GND	8	21	AREF
(PCINT6/XTAL1/TOSC1) PB6	9	20	AVCC
(PCINT7/XTAL2/TOSC2) PB7	10	19	PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1) PD5	11	18	PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6	12	17	PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7	13	16	PB2 ($\overline{\text{SS}}$ /OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0	14	15	PB1 (OC1A/PCINT1)

ATmega328 Pinout



4.4 BLOCK DIAGRAM

Port C (PC[5:0]) Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC[5:0] 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 unprogrammed, 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. The various special features of Port C are elaborated in the Alternate Functions of Port C section.

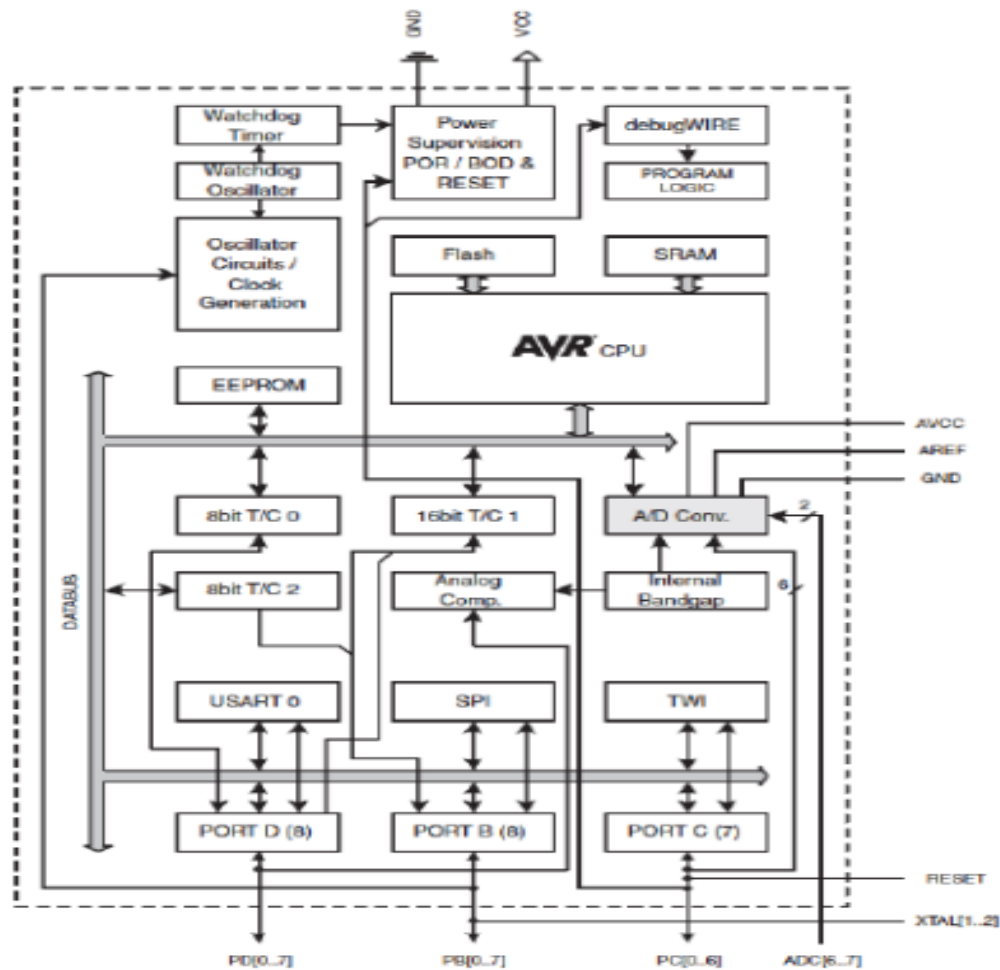
Port D (PD[7:0]) Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if 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.

AVCC AVCC is the supply voltage pin for the A/D Converter, PC[3:0], and PE[3:2]. 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 PC[6:4] use digital supply voltage, VCC.

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

ADC[7:6] (TQFP and VFQFN Package Only) In the TQFP and VFQFN package, ADC[7:6] serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

4.6 ARCHITECTURE DIAGRAM



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. The fast-access Register File contains 32 x 8-bit general purpose working registers with a single clock cycle access time. This allows single-cycle Arithmetic Logic Unit (ALU) operation. In a typical ALU operation, two operands are output from the Register File, the operation is executed, and the result is stored back in the Register File – in one clock cycle. Six of the 32 registers can be used as three 16-bit indirect address register pointers for Data Space addressing – enabling efficient address calculations. One of these address pointers can also be used as an address pointer for look up tables in Flash program memory.

These added function registers are the 16-bit X-, Y-, and Z-register, described later in this section. The ALU supports arithmetic and logic operations between registers or between a constant and a register. Single register operations can also be executed in the ALU. After an arithmetic operation, the Status Register is updated to reflect information about the result of the operation. Program flow is provided by conditional and unconditional jump and call instructions, able to directly address the whole address space.

Most AVR instructions have a single 16-bit word format. Every program memory address contains a 16- or 32-bit instruction. Program Flash memory space is divided in two sections, the Boot Program section and the Application Program section. Both sections have dedicated Lock bits for write and read/write protection. The SPM instruction that writes into the Application Flash memory section must reside in the Boot Program section.

During interrupts and subroutine calls, the return address Program Counter (PC) is stored on the Stack. The Stack is effectively allocated in the general data SRAM, and consequently the Stack size is only limited by the total SRAM size and the usage of the SRAM. All user programs must initialize the SP in the Reset routine (before subroutines or interrupts are executed). The Stack Pointer (SP) is read/write accessible in the I/O space. The data SRAM can easily be accessed through the five different addressing modes supported in the AVR architecture. The memory spaces in the AVR architecture are all linear and regular memory maps. A flexible interrupt module has its control registers in the I/O space with an additional Global Interrupt Enable bit in the Status Register. All interrupts have a separate Interrupt Vector in the Interrupt Vector table. The interrupts have priority in accordance with their Interrupt Vector position. The lower the Interrupt Vector address, the higher the priority. The I/O memory space contains 64 addresses for CPU peripheral functions as Control Registers, SPI, and other I/O functions. The I/O Memory can be accessed directly, or as the Data Space locations following those of the Register File, 0x20 - 0x5F. In addition, this device has Extended I/O space from 0x60 - 0xFF in SRAM where only the ST/STS/STD and LD/LDS/LDD instructions can be used.

ALU – Arithmetic Logic Unit 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 Instruction Set Summary section for a detailed description.

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. 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.

Status Register When addressing I/O Registers as data space using LD and ST instructions, the provided offset must be used. When using the I/O specific commands IN and OUT, the offset is reduced by 0x20, resulting in an I/O address offset within 0x00 - 0x3F. Name: SREG Offset: 0x5F Reset: 0x00 Property: When addressing as I/O Register: address offset is 0x3F

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Global Interrupt Enable The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The Ibit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

Copy Storage The Bit Copy instructions BLD (Bit Load) and BST (Bit Store) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

Half Carry Flag The Half Carry Flag H indicates a Half Carry in some arithmetic operations. Half Carry Flag is useful in BCD arithmetic. See the Instruction Set Description for detailed information.

Sign Flag, $S = N \oplus V$ The S-bit is always an exclusive or between the Negative Flag N and the Two's Complement Overflow Flag V. See the Instruction Set Description for detailed information.

Two's Complement Overflow Flag The Two's Complement Overflow Flag V supports two's complement arithmetic. See the Instruction Set Description for detailed information.

Negative Flag The Negative Flag N indicates a negative result in an arithmetic or logic operation. See the Instruction Set Description for detailed information.

Zero Flag The Zero Flag Z indicates a zero result in an arithmetic or logic operation. See the Instruction Set Description for detailed information.

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 is implemented as growing from higher to lower memory locations. The Stack Pointer Register always points to the top of the Stack. The Stack Pointer points to the data SRAM Stack area where the Subroutine and Interrupt Stacks are located. A Stack PUSH command will decrease the Stack Pointer. The Stack in the data SRAM must be defined by the program before any subroutine calls are executed or interrupts are enabled. Initial Stack Pointer value equals the last address of the internal SRAM and the Stack Pointer must be set to point above start of the SRAM. See the table for Stack Pointer details.

4.7 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. They have determined priority levels: 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 IVSEL bit in the MCU Control Register (MCUCR). The Reset Vector can also be moved to the start of the Boot Flash section by programming the BOOTRST Fuse. When an interrupt occurs, the Global Interrupt Enable I-bit is cleared and all interrupts are disabled. The user software can write logic one to the I-bit to enable nested interrupts. All enabled interrupts can then interrupt the current interrupt routine. The I-bit is automatically set when a Return from Interrupt instruction – RETI – is executed. There are basically two types of interrupts: The first type is triggered by an event that sets the Interrupt Flag. For these interrupts, the Program Counter is vectored to the actual Interrupt Vector in order to execute the interrupt handling routine, and hardware clears the corresponding Interrupt Flag. Interrupt Flags can also be cleared by writing a logic one to the flag bit position(s) to be cleared. If an interrupt condition occurs while the corresponding interrupt enable bit is cleared, the Interrupt Flag will be set and remembered until the interrupt is enabled, or the flag is cleared by software. Similarly, if one or more interrupt conditions occur while the Global Interrupt Enable bit is cleared, the corresponding Interrupt Flag(s) will be set and remembered until the Global Interrupt Enable bit is set, and will then be executed by order of priority. The second type of interrupts will trigger as long as the interrupt condition is present. These interrupts do not necessarily have Interrupt Flags. If the interrupt condition disappears before the interrupt is enabled, the interrupt will not be triggered. When the AVR exits from an interrupt, it will always return to the main program and execute one more instruction before any pending interrupt is served. The Status Register is not automatically stored when entering an interrupt routine, nor restored when returning from an interrupt routine. This must be

handled by software. When using the CLI instruction to disable interrupts, the interrupts will be immediately disabled. No interrupt will be executed after the CLI instruction, even if it occurs simultaneously with the CLI instruction.

4.8 MEMORY DESCRIPTION

AT mega 328 has three types of memories e.g. EEPROM, SRAM etc.

Flash Memory has 32KB capacity. It has an address of 15 bits. It is a Programmable Read Only Memory (ROM). It is non volatile memory. The ATmega328/P contains 32Kbytes On-chip In-System Reprogrammable Flash memory for program storage. Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 16K x 16. For software security, the Flash Program memory space is divided into two sections - Boot Loader Section and Application Program Section in the device . The Flash memory has an endurance of at least 10,000 write/erase cycles. The ATmega328/P Program Counter (PC) is 14 bits wide, thus addressing the 16K program memory locations. The operation of Boot Program section and associated Boot Lock bits for software protection are described in detail in Boot Loader Support – Read-While-Write Self-Programming. Refer to Memory Programming for the description on Flash data serial downloading using the SPI pins. Constant tables can be allocated within the entire program memory address space, using the Load Program Memory (LPM) instruction. Timing diagrams for instruction fetch and execution are presented in Instruction Execution Timing

SRAM stands for Static Random Access Memory. It is a volatile memory i.e. data will be removed after removing the power supply. Device SRAM Memory is organized. The device is a complex microcontroller with more peripheral units than can be supported within the 64 locations reserved in the Opcode for the IN and OUT

instructions. For the Extended I/O space from 0x60 - 0xFF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used. The lower 2303 data memory locations address both the Register File, the I/O memory, Extended I/O memory, and the internal data SRAM. The first 32 locations address the Register File, the next 64 location the standard I/O memory, then 160 locations of Extended I/O memory, and the next 2K locations address the internal data SRAM.

The five different addressing modes for the data memory cover:

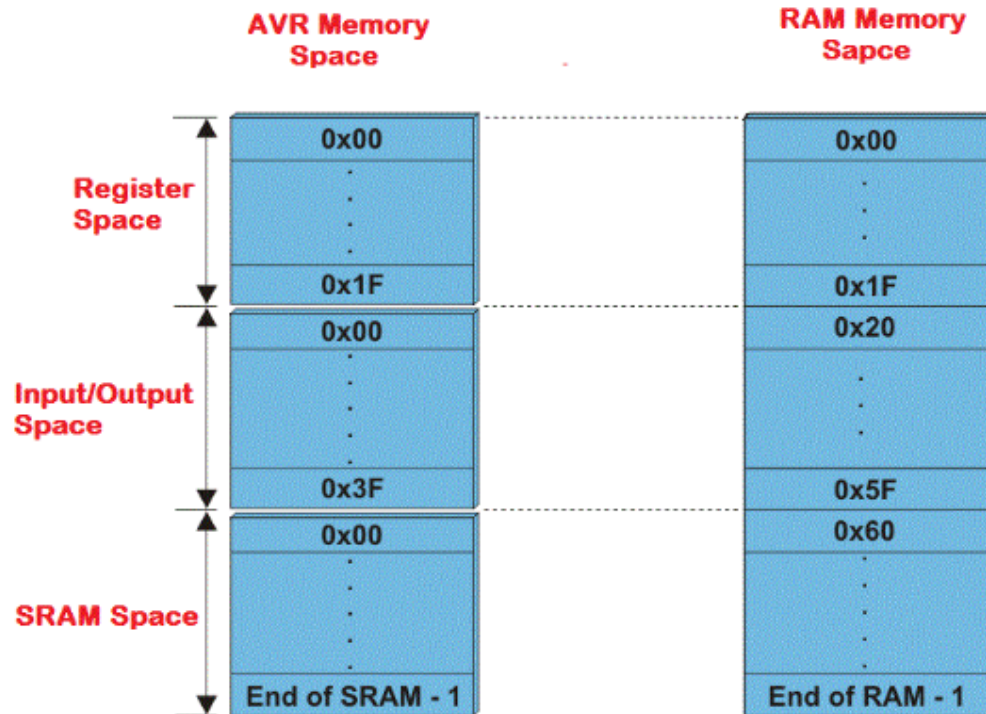
- Direct – The direct addressing reaches the entire data space.
- Indirect with Displacement – The Indirect with Displacement mode reaches 63 address locations from the base address given by the Y- or Z-register.
 - Indirect – In the Register File, registers R26 to R31 feature the indirect addressing pointer registers.
 - Indirect with Pre-decrement – The address registers X, Y, and Z are decremented.
 - Indirect with Post-increment – The address registers X, Y, and Z are incremented.

The 32 general purpose working registers, 64 I/O Registers, 160 Extended I/O Registers, and the 2K bytes of internal data SRAM in the device are all accessible through all these addressing modes.

EEPROM stands for Electrically Erasable Programmable Read Only Memory. It has a long term data. The ATmega328/P contains 1K bytes of data EEPROM memory. It is organized as a separate data space, in which single bytes can be read and written. The EEPROM has an endurance of at least 100,000 write/erase cycles. The access between the EEPROM and the CPU is described in the following, specifying the EEPROM Address Registers, the EEPROM Data Register, and the EEPROM

Control

Register.



4.9 REGISTERS

- ATmega-328 has thirty two (32) General Purpose (GP) registers.
- These all of the registers are the part of Static Random Access Memory (SRAM).

7	0	Addr.
R0		0x00
R1		0x01
R2		0x02
...		
R13		0x0D
R14		0x0E
R15		0x0F
R16		0x10
R17		0x11
...		
R26		0x1A
R27		0x1B
R28		0x1C
R29		0x1D
R30		0x1E
R31		0x1F

4.10 ATmega328 and Arduino

- ATmega-328 is the most micro-controller that is used while designing.
- ATmega-328 is the most important part of Arduino.
- The program is uploaded on the AVR micro-controller attached on Arduino.



4.11 UART

The first communication protocol we'll cover is Universal Asynchronous Receiver/Transmitter (UART). UART is a form of serial communication because data is transmitted as sequential bits (we'll get to this in a bit). The wiring involved with setting up UART communication is very simple: one line for transmitting data (TX) and one line for receiving data (RX). As you may expect, the TX line is used to for the data to send device, and the RX line is used to receive data. Together the TX and RX lines of a device using serial communication form a serial port through which communication can occur.

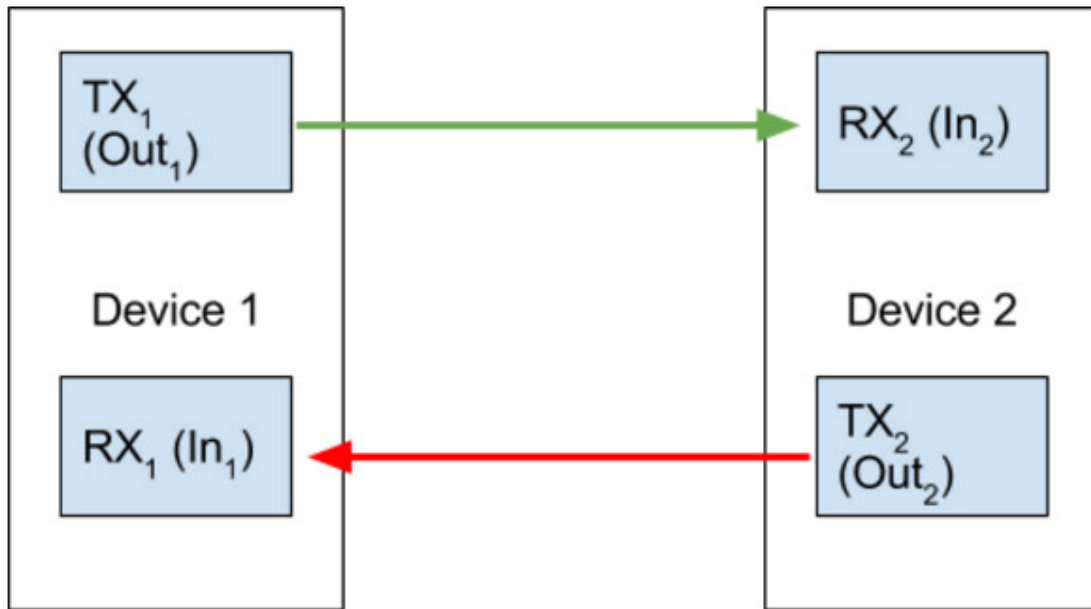


Figure: Hardware connection diagram for UART

The term UART actually refers to the onboard hardware that manages the packaging and translation of serial data. For a device to be able to communicate via the UART protocol, it must have this hardware! On the Arduino Uno, there is one serial port dedicated for communication with the computer the Arduino is connected to. That's right! USB, which stands for Universal Serial Bus, is a serial port! On the Arduino Uno, this USB connection is broken out through onboard hardware into two digital pins, GPIO 0 and GPIO 1, which can be used in projects that involve serial communication with electronics other than the computer.

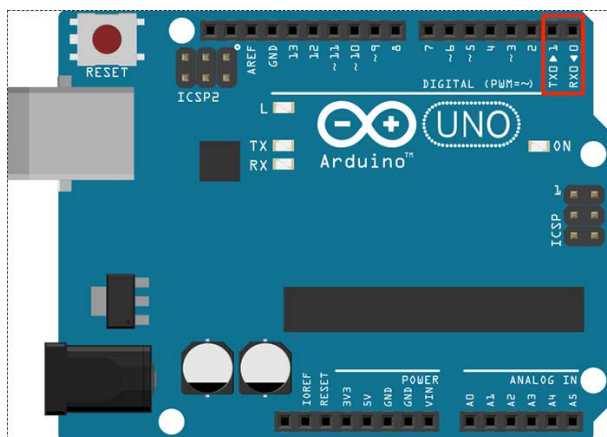


Figure: GPIO 0 and GPIO 1 are Serial RX and TX. Any GPIO pin can be used as Serial RX or TX with the Software Serial library

You can also use the Software SerialArduino library (SoftwareSerial.h) to use other GPIO pins as Serial RX and TX lines.

UART is called asynchronous because the communication does not depend on a synchronized clock signal between the two devices attempting to communicate with each other. Because the communication speed is not defined via this steady signal, the “sender” device cannot be certain that the “receiver” obtains the correct data. Therefore, the devices break data into fixed-size chunks to ensure that the data received is the same as the data that was sent.

The UART data packet looks something like this:

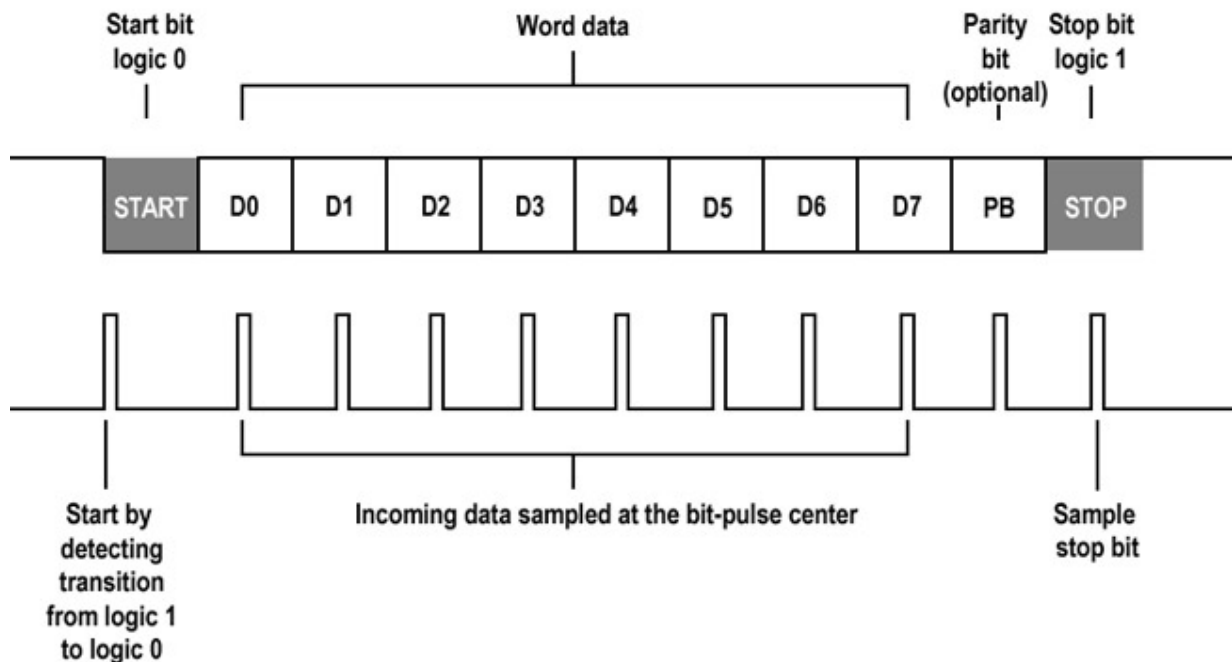


Figure 3. UART Data packet visualized / ©electric imp

Devices that communicate via UART send packets of pre-defined size that contain additional information regarding the start and end of the message and confirmation of whether the message was received correctly. For example, to begin communication, the transmitting device pulls the transmit line low, indicating the start of a data packet. Long-term, however, this means that UART is slower compared to

a synchronized form of communication because only a portion of the data transmitted is for the devices' applications (the rest is for the communication itself!)

When implementing UART serial communication on most embedded platforms such as the Arduino, the user does not have to deal with communication at the bit level. Instead, the platform often provides higher level software libraries that are the only aspect of the communication process the user has to deal with. In the Arduino platform, users can use the *Serial* and *SoftwareSerial* libraries to implement UART communication for their projects.

Here's a brief C++ reference for Arduino Serial and SoftwareSerial initialization and use.

Serial and SoftwareSerial Method	Purpose	Code	Explanation
<i>Constructor (SoftwareSerial only)</i>	Define the GPIO pins that will serve as the UART RX and TX lines	<code>SoftwareSerial comms(2, 3);</code>	Defines a serial connection with RX line on GPIO 2 and TX line on GPIO 3.
<i>Begin</i>	Define the baud rate (transmission speed) for the serial connection in range 4800 to 115200	<code>comms.begin(9600);</code>	Communication on the "comms" serial port will occur at 9600 baud.
<i>Print</i>	Write byte data converted into human-readable characters over serial connection	<code>comms.println("Hello World");</code>	Writes bytes equivalent to <i>Hello World</i> (human-readable characters)
<i>Write</i>	Write raw byte data over the serial connection	<code>comms.write(45);</code>	Writes byte with value 45.
<i>Available</i>	Evaluates to true when data is available over the serial connection	<code>if (comms.available())</code>	Enter <i>if</i> statement if there is data available to read over the serial connection
<i>Read</i>	Read data from the serial connection	<code>comms.read();</code>	Reads from serial

One important aspect to note about UART communication is that it is designed for communication between only two devices at a time. Because the protocol only sends bits indicating the start of a message, the message content, and the end of a message, there is no method of differentiating multiple transmitting and receiving devices on the same line. If more than one device attempts to transmit data on the same line, bus contention occurs, and the receiving devices will most likely receive garbage unusable data!

Furthermore, UART is half-duplex, which means that even though communication can occur bi-directionally, both devices cannot transmit data to each other at the same time. In a project where two Arduino's are communicating with each other via a serial connection, for example, this would mean that at a given time, only one of the Arduinos can be "talking" to the other. For most applications, however, this fact is relatively unimportant and not disadvantageous in any way.

SPI

The next communication protocol we'll cover is Serial Peripheral Interface (SPI). SPI is different from UART in several key ways:

- Synchronous
- Follows a master-slave model, where there is one master device and multiple slave devices
- More than two lines required for implementation

The hardware connection diagram for SPI is slightly more complicated, and looks something like this:

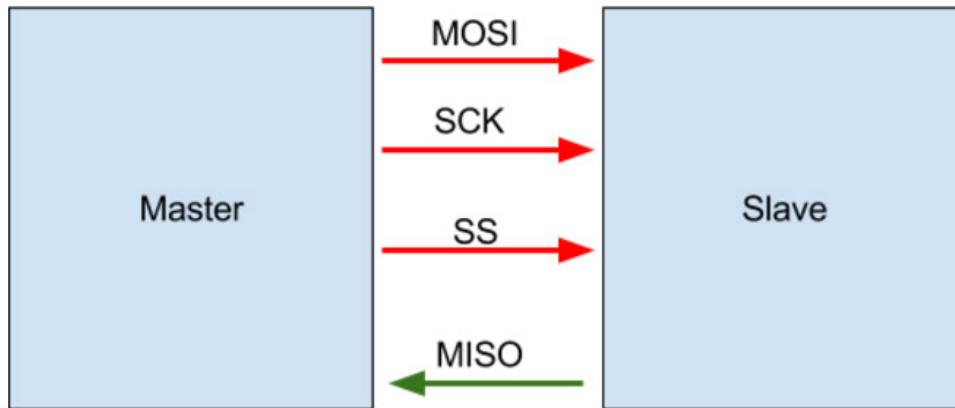


Figure 4. Hardware connection diagram for SPI

- **MOSI (“Master Out Slave In”)**: Data transmission line from master to slave
- **SCK (“Clock”)**: Clock line defining transmission speed and transmission start/end characteristics
- **SS (“Slave Select”)**: Line for master to select a particular slave to communicate with
- **MISO (“Master In Slave Out”)**: Data transmission line from slave to master

The first feature about SPI to note is that it follows a master-slave model. This means that the communication relies on defining one device as a master, and other devices as slaves. This creates a hierarchy between the devices signifying which device is effectively “in control” of the others. We’ll discuss this master-slave model shortly when we describe what a sample communication between a master and slave device might look like.

We noted earlier that multiple slaves can be connected to a single master device. The hardware diagram for such a system might look something like this:

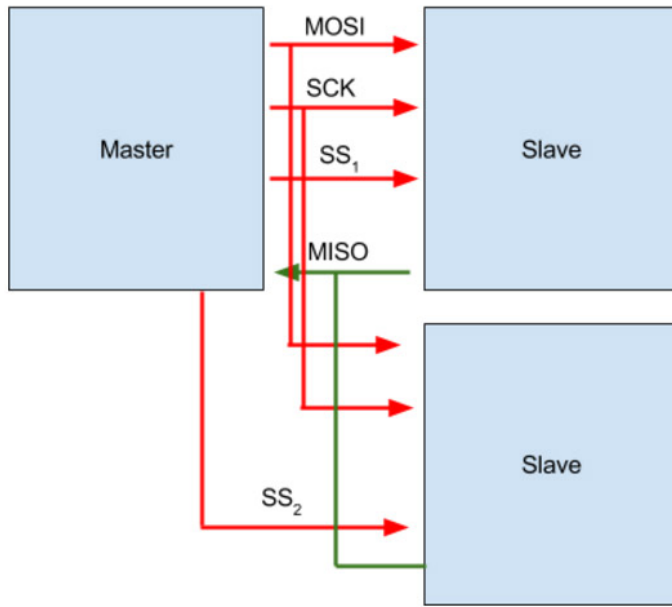


Figure 5. Multiple-slaves connected to a single master

SPI does not require a separate transmit and receive line for each slave that is connected to the master. One common receive line (MISO) and one common transmit line (MOSI), along with one common clock (SCK) line are connected between all the slaves and the master. The master device decides which slave it is communicating with through a separate SS line for each slave. This means that each additional slave that the master communicates with requires another GPIO pin on the master's side.

SPI is synchronous, which means that communication between master and slave is closely tied to a clock signal (a square wave of fixed-frequency) defined by the master device. Here we see one of the direct effects of the master-slave model. The master device drives the communication by specifying the communication speed through the clock signal, and the slaves respond by communicating at that speed. This definition of speed works for any communication speed the master decides on (within the maximum speed tolerable by the slave devices).

In SPI, two specific features of the clock signal determine the start and end of data transmission: the clock polarity (CPOL) and clock phase (CPHA). CPOL refers to the idle state (either low or high) of the clock signal. To conserve power, devices will put the clock line at the idle state when not communicating with any slaves, and the

two options available for this idle state are either low or high. CPHA refers to the edge of the clock signal upon which data is captured. A square wave has two edges (the rising edge and the falling edge), and depending on the CPHA setting, data is captured either on the rising edge or the falling edge.

There are four distinct combinations of CPOL and CPHA, that can be summarized in a table.

	CPHA = 0	CPHA = 1
	("first edge" of clock signal)	("second edge" of clock signal)
CPOL = 0 (idle 0)	• "SPI mode 0" • Data capture begins when clock goes from 0 (idle) to 1 for the first time. • Data capture occurs on the rising edge of the clock signal (0→1)	• "SPI mode 1" • Data capture begins when clock goes from 0 (idle) to 1 for the first time. • Data capture occurs on the falling edge of the clock signal (1→0)
CPOL = 1 (idle 1)	• "SPI mode 2" • Data capture begins when clock goes from 1 (idle) to 0 for the first time. • Data capture occurs on the rising edge of the clock signal (0→1)	• "SPI mode 3" • Data capture begins when clock goes from 1 (idle) to 0 for the first time. • Data capture occurs on the falling edge of the clock signal (1→0)

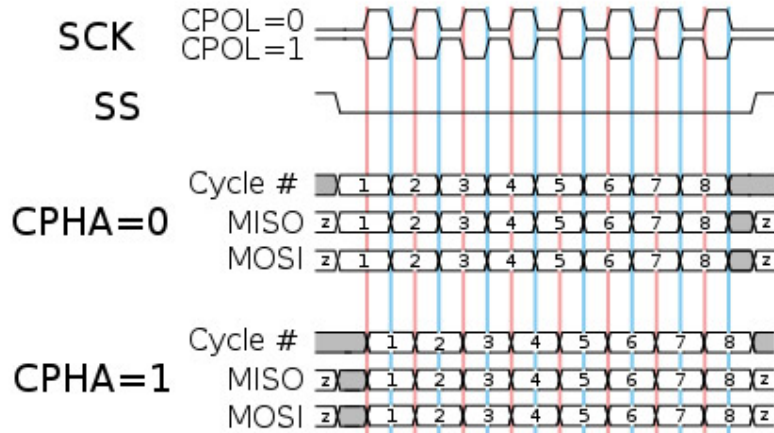


Figure 6. CPOL and CPHA settings visualized

Now we'll focus on SPI implementation on the Arduino using the Arduino as the master device (SPI.h). The SPI digital pin connections for SCK, MOSI, and MISO are predefined on Arduino boards. For the Arduino Uno, the connections are as follows:

- **SCK:** GPIO 13 or ICSP 3
- **MOSI:** GPIO 11 or ICSP 4
- **MISO:** GPIO 12 or ICSP 1
- **SS:** GPIO 10

Any digital pin can be also used for the SS pin. To select the device, this digital pin must be driven low.

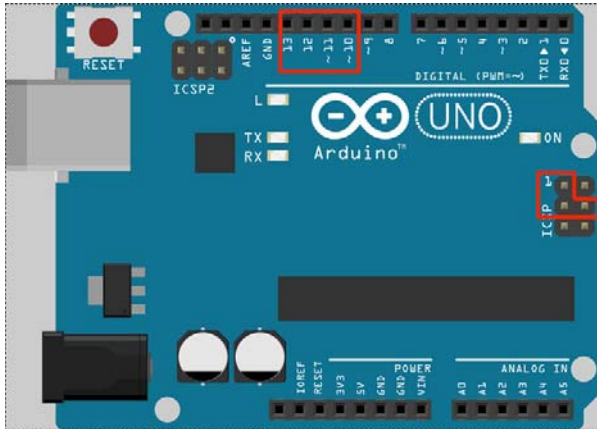


Figure 7. The MOSI, MISO, and SCK pins are broken out on the ICSP headers as well as on GPIO 11, GPIO 12, and GPIO 13

Here's a brief C++ reference for Arduino SPI initialization and use.

SPI Method	Purpose	Code	Explanation
Constructor	Define the clock speed, data bit order (most significant bit first or least significant bit first), and SPI mode	<code>SPI.beginTransaction(SPISettings(14000000, MSBFIRST, SPI_MODE0));</code>	Defines a SPI connection at 14MHz in SPI Mode 0 and data transmission occurring with the most significant bit first
	Select the slave device attached to this GPIO pin	<code>digitalWrite(10, LOW);</code>	Drive the GPIO pin low to select the slave device. Drive the pin high afterwards to de-select the slave.
Transfer	Transfer the byte to the selected slave	<code>SPI.transfer(0x00);</code>	Send a value of 0 as a byte
endTransaction	Ends the SPI transaction (should be called after a <code>digitalWrite(high)</code> is	<code>SPI.endTransaction();</code>	Ends the SPI transaction

called on the SS line)

SPI is full-duplex, which means that communication always occurs bidirectionally, even if the application only requires data transfer occur in one direction. SPI full-duplex data transfer is usually implemented with a shift register. (Check out the appendix for a tutorial on shift registers!) This means that as each bit is read, the bits before it are shifted over by one place. The foremost bit is subsequently popped off and sent back through the SPI connection to the other device!

I²C

Inter-integrated circuit (I²C), pronounced either “i-squared-c” or “i-two-c,” is the final communication protocol we’ll cover in this tutorial. Though its implementation is the most complicated of the three protocols, I²C addresses several drawbacks in the other communication protocols, giving it an advantage over the others in some applications. These include:

- The ability to connect multiple masters to multiple slaves
- Synchronicity (just like SPI), which means higher speed communication
- Simplicity: implementation only requires two wires and some resistors

At the hardware level, I²C is a two-wire interface—the only two wires required for an I²C connection are a data line (called *SDA*) and a clock line (called *SCL*). The data and clock lines are pulled high in their idle states, and when data needs to be sent over the connection, the lines are pulled low through some MOSFET circuits. For the purposes of this tutorial, we will not discuss the MOSFET operation inside I²C circuitry. The important fact to take away is that the system is *open-drain*, which means that the lines can only be driven low by the devices. Thus when using I²C in projects, it is crucial to include *pull-up resistors* (usually something like 4.7kΩ) to ensure that the line is actually pulled high in the idle state.

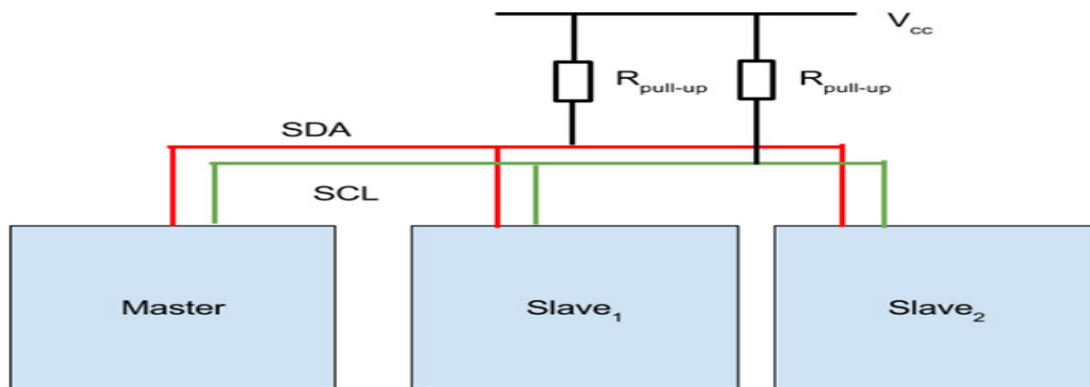


Figure 8. I²C hardware connection diagram

I²C is unique because it solves the issue of interfacing with multiple slave devices through addressing. Just like in SPI communication, I²C makes use of a master-slave model to establish the “hierarchy” of communication. However, instead of selecting slaves through separate digital lines, masters select slaves through their unique byte addresses. These addresses might look something like this: 0x1B. This means that connecting more slaves to a master will not require additional digital lines. As long as each slave has a unique address, the application will still be able to differentiate the slaves on these addresses alone. You can think of these addresses like names. To call a slave’s functionality, the master merely calls its name, and only that slave responds.

The address and data portions of the I²C communication line might look something like this.

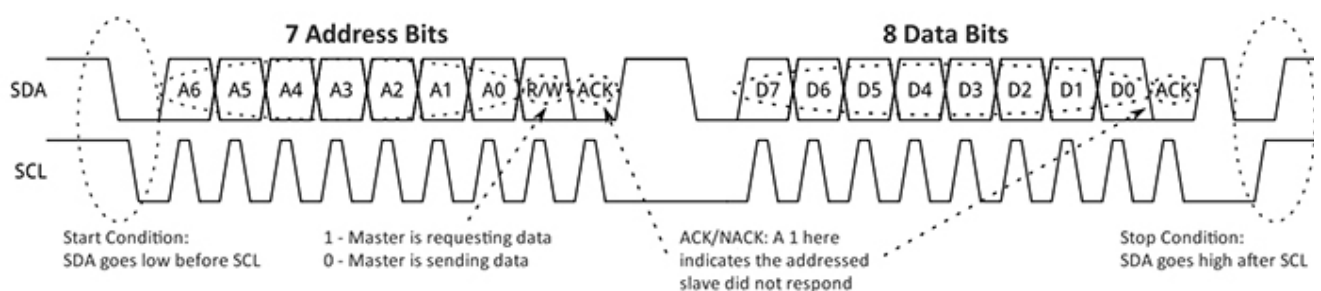


Figure 9. What the address and data portions of the communication

Notice the ACK and NACK bits on the communication line. These bits denote whether or not the addressed slave responds to the communication—a way of periodically checking whether or not communication is occurring as expected. These

bits, of course, are unrelated to the address or data bits being sent, but in the grand scheme of the communication, they add very little extra time to the communication compared to the many start and end bits and pauses that occur in protocols like UART.

I²C gives slaves freedom to decide what communication requests might look like. Different slaves require different bytes to be written over in different orders over the SDA line to write and request information from them. In some accelerometer modules, for example, bytes are written to denote which hardware register the master wants to read before any read requests are made. For these specifications, the user must refer to the slaves' datasheets for device addresses, register addresses, and device settings.

On the Arduino, I²C implementation occurs through the Wire library (Wire.h). The Arduino can be configured as either an I²C master or slave device. On the Arduino Uno, the connections are as follows:

- **SDA:** Analog Pin 4
- **SCL:** Analog Pin 5

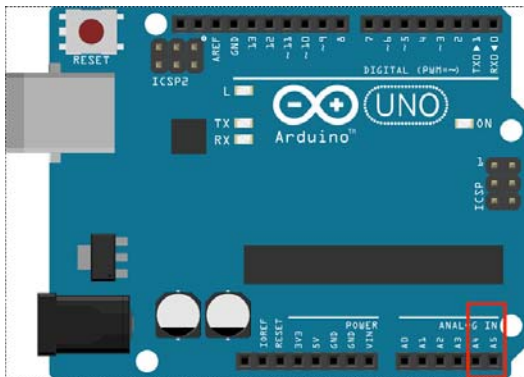


Figure 9. I²C (Wire) SDA is Analog Pin 4 (A4). SCL is Analog Pin 5 (A5).

Here's a brief C++ reference for Arduino I²C initialization and use.

I ² C Method	(Wire) Purpose	Code	Explanation
Begin	Initiate the library and join the I ² C bus as either a master or slave	<code>Wire.begin();</code>	Joins the I ² C bus as a master. If an address is specified as a parameter to the method, the Arduino joins the bus as a slave with that address.
begin Transmission	For Arduino configured as a I ² C master: initiate transmission with the slave that has the given address	<code>Wire.beginTransmission(0x68);</code>	Begins transmission with the slave that has the hexadecimal address 0x68
Write	Write the byte data over the I ² C bus	<code>Wire.write(0x6B);</code>	Writes byte data of value 0x6B over the I ² C bus
RequestFrom	Request a specified number of bytes from the slave with the	<code>Wire.requestFrom(0x68, 6, true);</code>	Requests 6 bytes from the slave that has the

given address; Has the option of releasing the I²C line or holding onto it for further communication

The requested bytes are put into a buffer to be subsequently read through *Wire.read()* calls.

Read bytes put into the buffer after transmissions from the slave device.

Read

Wire.read();

This method is called *after* a call to *Wire.requestFrom*

address 0x68. Releases the I²C line after the request is complete.

If the last parameter is *false*, the Arduino will hold onto the I²C line for further communication, not allowing other devices to communicate over it.

Reads one byte from the buffer (after a call to *requestFrom*) has been made. To read two bytes from the buffer, this method must be called twice:

Example:

```
Wire.requestFrom(0x68, 2, true);
```

```
Wire.read();
```

```
Wire.read();
```

EndTransmission	Ends the current transmission; Has the option of releasing the Wire.endTransmission(tr I ² C line or holding ue); onto it for further communication	End transmission and releases the I ² C line.
-----------------	---	--

Multiple masters can be connected over an I²C bus just by connecting their SDA and SCL lines to the bus's lines. However, only one master can communicate with slaves at a time, because having multiple devices attempting to communicate with each other would lead to bus contention. Similarly, communication cannot occur bidirectionally from master to slave and from slave to master at the same time because that would also lead to bus contention. This makes I²C half-duplex, just like UART!

APPLICATIONS

- A complete package including ATmega 328 and Arduino can be used in several different real life applications.
- It can be used in embedded systems projects.
- It can also be used in robotics.
- Quad-copter and even small aeroplanes can also be designed through it.
- Power monitoring and management systems can also be prepared using this device.

4.12 LIQUID CRYSTAL DISPLAY (LCD)

4.12.1 INTRODUCTION

A liquid crystal display (LCD) is a thin, flat electronic visual display that uses the light modulating properties of liquid crystals (LCs). LCs do not emit light directly. They are used in a wide range of applications, including computer monitors, television, instrument panels, aircraft cockpit displays, etc. LCDs have displaced cathode ray tube (CRT) displays in most applications. They are usually more

compact, lightweight, portable, less expensive, and more reliable.

Each pixel of an LCD typically consists of a layer of molecules aligned between two transparent electrodes and two polarizing filters, the axes of transmission of which are (in most of the cases) perpendicular to each other. With no actual liquid crystal between the polarizing filters, light passing through the first filter would be blocked by the second (crossed) polarizer. In most of the cases the liquid crystal has double refraction.

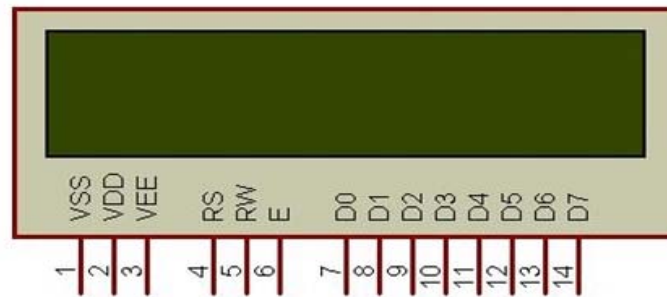
The surfaces of the electrodes that are in contact with the liquid crystal material are treated so as to align the liquid crystal molecules in a particular direction. Electrodes are made of a transparent conductor called Indium Tin Oxide (ITO).

Before applying an electric field, the orientation of the liquid crystal molecules is determined by the alignment at the surfaces of electrodes. In a twisted nematic device the surface alignment directions at the two electrodes are perpendicular to each other, and so the molecules arrange themselves in a helical structure, or twist. This reduces the rotation of the polarization of the incident light, and the device appears grey. If the applied voltage is large enough, the liquid crystal molecules in the center of the layer are almost completely untwisted and the polarization of the incident light is not rotated as it passes through the liquid crystal layer. This light will then be mainly polarized perpendicular to the second filter, and thus be blocked and the pixel will appear black. By controlling the voltage applied across the liquid crystal layer in each pixel, light can be allowed to pass through in varying amounts thus constituting different levels of gray.

PIN DESCRIPTION

The most common used LCDs found in the market today are 1 line, 2 line, or 4 line LCDs which have only one controller and support at most of 80 characters, whereas LCDs supporting more than 80 characters make use of 2HD44780

controllers. More LCDs with 1 controller has 14 pins and LCDs with 2 controller has 16pins.



Pin diagram of 2x16 LCD

FEATURES

- 5 x 8 dots with cursor.
- Built-in controller (KS 0066 or Equivalent)
- + 5V power supply (Also available for + 3V)
- 1/16 duty cycle
- B/L to be driven by pin 1, pin 2 or pin 15, pin 16 or A.K (LED)
- N.V. optional for + 3V power supply.

PIN NO.,	SYMBOL	FUNCTION
1.	Vss	GND
2.	Vdd	+3V or +5V
3.	Vo	Contrast Adjustment
4.	RS	H/L Register Select Signal
5.	R/ $\hat{W}$	H/L Read and Write Signal
6.	E	H→L Enable Signal
7.	DB0	H/L Data Bus Line
8.	DB1	H/L Data Bus Line
9.	DB2	H/L Data Bus Line
8.	DB3	H/L Data Bus Line
11.	DB4	H/L Data Bus Line

12.	DB5	H/L Data Bus Line
13.	DB6	H/L Data Bus Line
14.	DB7	H/L Data Bus Line
15.	A/Vee	+4.2V for LED/Negative Voltage Output
16.	K	Power Supply for B/L (0V)

Table 11.Pin details of LCD

DISPLAY CHARACTER ADDRESS CODE

Display Position	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
DD RAM Address	00	01														0F
DD RAM Address	40	41														4F

Address code for displaying character

BF- BUSY FLAG

Busy flag is an status indicator flag for LCD. When we send a command or data to the LCD for processing, this flag is set (BF=1) and as soon as the instruction is executed successfully this flag is cleared (BF=0). This is helpful in producing and exact amount of delay for the LCD processing. To read busy flag, the condition RS=0 and R/W=1 must be met and the MSB of the LCD data bus (D7) act as busy flag. When BF =1 means LCD is busy and will not accept next command or data and BF=0 means LCD is ready for the next command or data to process.

4.13 INSTRUCTION AND DATA REGISTER

There are two 8-bit registers in HD44780 controller Instruction and Data register. Instruction register corresponds to the register where you send commands to LCD e.g LCD shift command, LCD clear, LCD address etc. and Data register is used for storing data which is to be displayed on LCD. when send the enable signal of the LCD is asserted, the data on the pins is latched in to the data register and data is then moved automatically to the DDRAM and hence is displayed on the LCD.

Data register is not only used for sending data to DDRAM but also for CGRAM, the address where you want to send the data, is decided by the instruction you send to LCD. We will discuss more on LCD instruction set further in this chapter.

4.14 COMMANDS AND INSTRUCTION SET

Only the instruction register (IR) and the data register (DR) of the LCD can be controlled by the MCU. Before starting the internal operation of the LCD, control information is temporarily stored into these registers to allow interfacing with various MCUs, which operate at different speeds, or various peripheral control devices. The internal operation of the LCD is determined by signals sent from the MCU. These signals, which include register selection signal (RS), read/write signal (R/W), and the data bus (DB0 to DB7), make up the LCD instructions (Table 3).

No.	Instruction	Hex	Decimal
1	Function Set: 8-bit, 1 Line, 5x7 Dots	0x30	48
2	Function Set: 8-bit, 2 Line, 5x7 Dots	0x38	56
3	Function Set: 4-bit, 1 Line, 5x7 Dots	0x20	32
4	Function Set: 4-bit, 2 Line, 5x7 Dots	0x28	40
5	Entry Mode	0x06	6
6	Display off Cursor off (clearing display without clearing DDRAM content)	0x08	8
7	Display on Cursor on	0x0E	14
8	Display on Cursor off	0x0C	12
9	Display on Cursor blinking	0x0F	15
10	Shift entire display left	0x18	24
12	Shift entire display right	0x1C	30
13	Move cursor left by one character	0x10	16
14	Move cursor right by one character	0x14	20
15	Clear Display (also clear DDRAM content)	0x01	1

Commands

There are four categories of instructions that:

- Designate LCD functions, such as display format, data length, etc.
- Set internal RAM addresses
- Perform data transfer with internal RAM
- Perform the miscellaneous functions.

4.15 TEMPERATURE SENSOR

4.15.1 LM35: Precision Centigrade Temperature Sensors

4.15.2 GENERAL DESCRIPTION

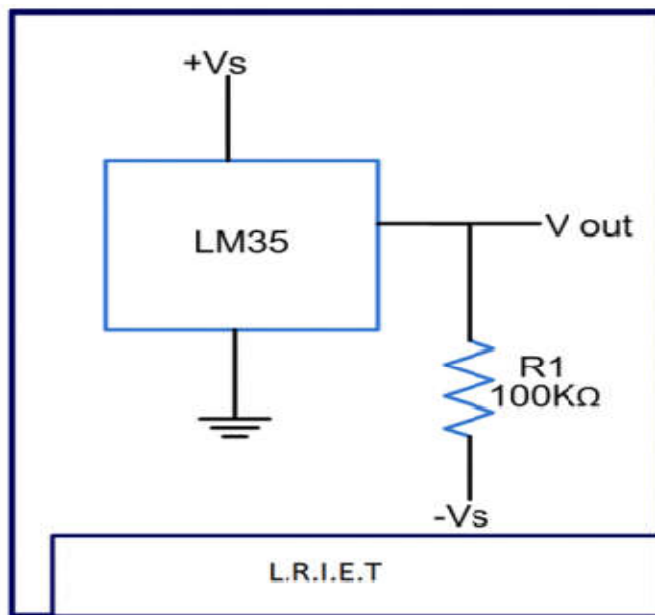
The LM35 series are precision integrated-circuit temperature sensors, whose Output voltage is linearly proportional to the Celsius (Centigrade) temperature. The LM35 thus has an advantage over linear temperature sensors calibrated in ° Kelvin, as the user is not required to subtract a large constant voltage from its output to Obtain convenient Centigrade scaling. The LM35 does not require any external Calibration or trimming to provide typical accuracies of $\pm 1/4^{\circ}\text{C}$ at room temperature and $\pm 3/4^{\circ}\text{C}$ over a full -55 to $+150^{\circ}\text{C}$ temperature range. Low cost is assured by trimming and calibration at the wafer level. The LM35's low output impedance, linear output, and precise inherent calibration make interfacing to readout or control circuitry especially easy. It can be used with single power supplies, or with plus and minus supplies. As it draws only $60\text{ }\mu\text{A}$ from its supply, it has very low self-heating, less than 0.1°C in still air. The LM35 is rated to operate over a -55° to $+150^{\circ}\text{C}$ temperature range, while the LM35C is rated for a -40° to $+110^{\circ}\text{C}$ range (-10° with improved accuracy). The LM35 series is available packaged in hermetic TO-46 transistor packages, while the LM35C, LM35CA, and LM35D are also available in the plastic TO-92 transistor package. The LM35D is also available in an 8-lead surface mount small outline package and a plastic TO-220 package.

FEATURES

- Calibrated directly in ° Celsius (Centigrade)
- Linear + 10.0 mV/°C scale factor
- 0.5°C accuracy guarantee able (at +25°C)
- Rated for full -55° to $+150^{\circ}\text{C}$ range
- Suitable for remote applications

- Low cost due to wafer-level trimming
- Operates from 4 to 30 volts
- Less than 60 μA current drain
- Low self-heating, 0.08°C in still air
- Nonlinearity only $\pm 1/4^\circ\text{C}$ typical
- Low impedance output, $0.1\ \Omega$ for 1 mA load.

TYPICAL APPLICATIONS



Full-Range Centigrade Temperature Sensor

4.16 THERMOELECTRIC EFFECT

The thermoelectric effect is the direct conversion of temperature differences to electric voltage and vice versa. A thermoelectric device creates voltage when there is a different temperature on each side. Conversely, when a voltage is applied to it, it creates a temperature difference. At the atomic scale, an applied temperature

gradient causes charge carriers in the material to diffuse from the hot side to the cold side.

4.17 WHAT IS THE PELTIER EFFECT?

When electric current is passed through a circuit consisting of two different conductors, a cooling effect is observed in one junction whereas another junction experiences a rise in temperature. This change in temperatures at the junctions is called the Peltier effect.

EXAMPLE: For example, when copper wire and bismuth wire are connected in an electric circuit, heat is generated at the point where current passes from copper to bismuth, and a drop in temperature occurs where the current passes from bismuth to copper. This effect is reversible in nature. The heating or cooling effect observed at a junction can be reversed by changing the direction of the current flow.

4.18 Code:

```
#include <LiquidCrystal.h>
#include <Keypad.h>
LiquidCrystal lcd(13,12,11,10,9,8);
const byte ROWS = 4;
const byte COLS = 3;
const int temp1 =A0;
const int temp2 =A1;
int t1v,t2v,f = 0;
char overf=0;
float value = 3035;
const int am =1;
const int val =7;
const int pel =6;
const int vib = A5;
//const int hbpin = 3;
char ckey, keyval[4]={0};
int am_count,a_count,m_count,i,diff,tset_f = 0;
unsigned int pulse,bpm,kval,time_count =0;
```

```
volatile int state = LOW;
char hexaKeys[ROWS][COLS] = {
  {'1','2','3'},
  {'4','5','6'},
  {'7','8','9'},
```

```

    {'*', '0', '#'}
};
byte rowPins[ROWS] = {5,A4,A3,A2};
byte colPins[COLS] = {2,3,4};
Keypad customKeypad = Keypad( makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);

void setup() {
  lcd.begin(16, 2);
  pinMode(am, INPUT_PULLUP);
  pinMode(pel, OUTPUT);
  pinMode(val, OUTPUT);
  digitalWrite(pel, LOW);
  digitalWrite(val, HIGH);
  lcd.print("Blood Infusion ");
  lcd.setCursor(0,1);
  lcd.print("Warmer");
  delay(2000);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Select The Mode");
}

void loop() {
  int v = analogRead(A5);
  if(v>1000){
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Vibration Detected");
    digitalWrite(val,LOW);
    delay(20000);
  }
  else{
    digitalWrite(val,HIGH);
  }

  //-----Mode Select-----
  if(digitalRead(am)){
    if(a_count==0){
      lcd.clear();
      lcd.setCursor(0,0);
      lcd.print("Auto Mode");
      delay(1000);
      a_count =1;
    }
    at();
  }
  if(!digitalRead(am)){
    if(m_count==0){
      lcd.clear();

```

```

    lcd.setCursor(0,0);
    lcd.print("Manual Mode");
    delay(500);
    m_count=1;
    }
    ml();

}

void at(){
    t1v = analogRead(temp1);
    t1v = t1v*4.88;
    t1v = t1v/10;
    //f = (t1v*1.8)+32;
    t1v = (t1v-32)*0.5555;
    t2v = analogRead(temp2);
    t2v = t2v*4.88;
    t2v = t2v/10;
    t2v = (t2v-32)*0.5555;
    delay(300);
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Auto Mode");
    lcd.setCursor(0,1);
    lcd.print("BdT=");
    lcd.print(t1v);
    lcd.setCursor(9,1);
    lcd.print("BLT=");
    lcd.print(t2v);

    diff = t1v-t2v;
    // diff = abs(diff);
    if(diff>1){
        digitalWrite(pel,HIGH);
        delay(500);
    }
    else{
        digitalWrite(pel,LOW);
        delay(500);
    }
    delay(500);

    void ml(){
        lcd.setCursor(0,0);
        lcd.print("Enter Temperature");
        lcd.setCursor(0,1);

```

```

lcd.print("Value:");
if(customKeypad.getKey()){
  do{
    char ckey = customKeypad.getKey();
    if(ckey){
      keyval[i] = ckey;
      lcd.setCursor(6+i,1);
      lcd.print(keyval[i]);
      delay(500);
      i++;
    }

  } while (i<2);

  kval = atoi(keyval);
  keyval[4] = {0};
  tset_f = 1;
}
if(tset_f == 1){
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Entered Temp=");
  lcd.setCursor(13,0);
  lcd.print(kval);
  delay(1000);
  t1v = analogRead(temp1);
  t1v = t1v*4.88;
  t1v = t1v/10;
  t1v = (t1v-32)*0.5555;
  t2v = analogRead(temp2);
  t2v = t2v*4.88;
  t2v = t2v/10;
  t2v = (t2v-32)*0.5555;
  //delay(100);
  //lcd.clear();
  lcd.setCursor(0,1);
  lcd.print("BdT=");
  lcd.print(t1v);
  lcd.setCursor(9,1);
  lcd.print("BLT=");
  lcd.print(t2v);
  char mdif1 = 0;
  // mdif1 = t1v - kval;
  // mdif1 = abs(mdif1);
  mdif1 = kval - t2v;
  //mdif1 = abs(mdif1);
  delay(1000);

```

```
if(mdif1>1 ){  
    digitalWrite(pel,HIGH);  
}  
else{  
    digitalWrite(pel,LOW);  
}  
}  
}
```

4.19 PELTIER MODULE



- 1) The module consists of two electrodes with good conductivity connected to semiconductor material.
- 2) Since they are dissimilar materials two thermocouple junctions are created.
- 3) Voltage is applied between the two electrodes to start an electrical current flow between the two junctions.
- 4) As the current flows, there is also flow of thermal energy from one junction to another.
- 5) Depending on the direction of current flow heat is either absorbed or transmitted

APPLICATIONS

The phenomenon behind the Peltier effect is used in the function of thermoelectric heat pumps and thermoelectric cooling devices. It is also used for cooling computers and other electronic equipment when other methods are not feasible.

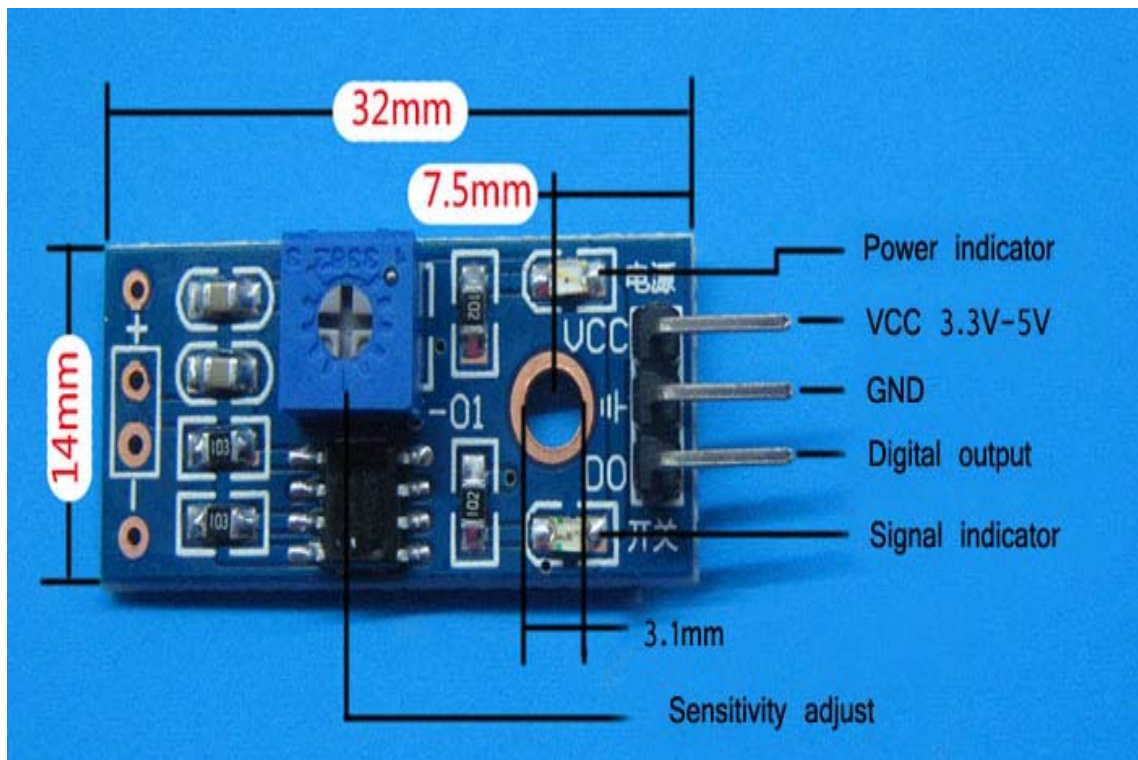
4.20 VIBRATION SENSOR

The Vibration Sensor Module Vibration Switch SW-420 is based on the vibration sensor SW-420 and Comparator LM393 to detect if there is any vibration that beyond the threshold. The threshold can be adjusted by the on-board potentiometer. When this no vibration, this module output logic LOW the signal indicate LED light, and vice versa.

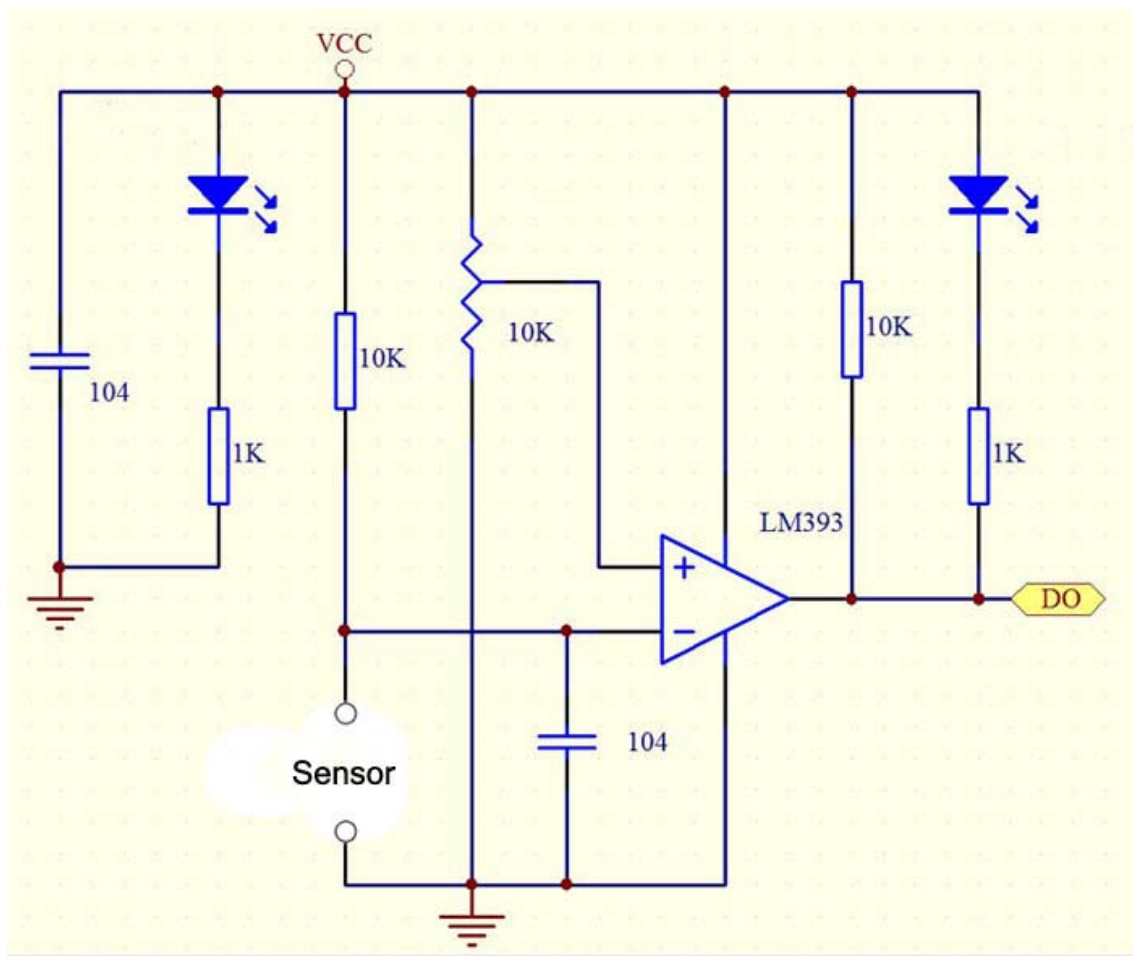


Features

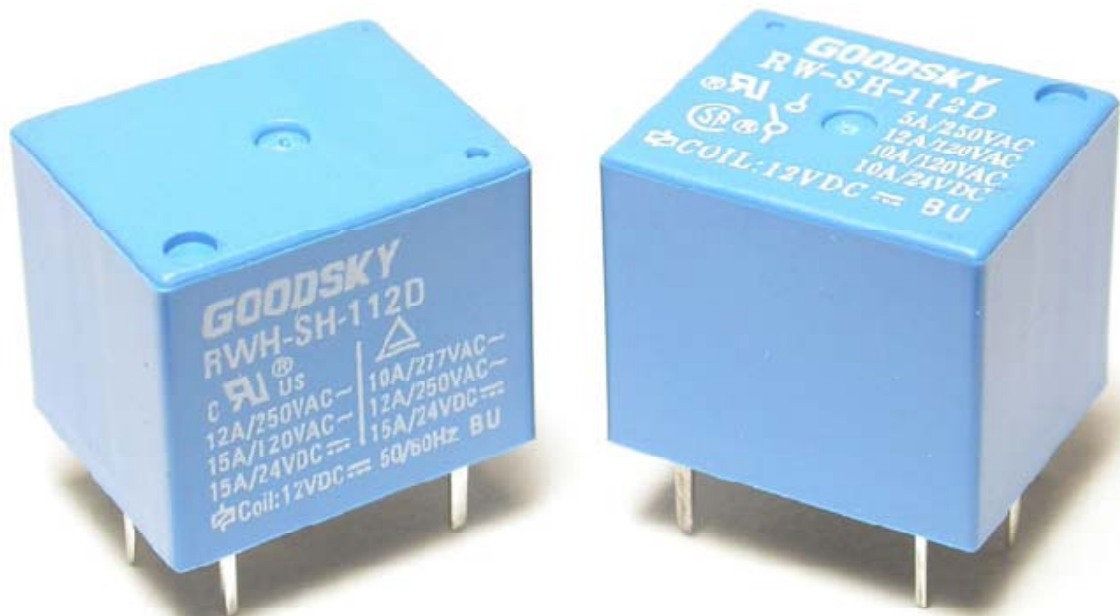
- Normally closed type vibration sensor SW-420
- The comparator output, signal clean, good waveform, driving ability is strong, for more than 15 ma
- The working voltage of 3.3V to 5V
- Output form: digital switch output (0 and 1)
- Has a fixed bolt hole, convenient installation
- Small board PCB size: 3.2cm x 1.4cm
- Use the LM393 wide voltage comparator



4.21 CIRCUIT DIAGRAM



4.22 RELAY



Main Feature

1. RW Series Relay covers switching capacity by 10A in spite of miniature size to comply with user's wide selection.
2. RWH is approved C-UL & TÜV safety standard.
3. The employment of suitable plastic materials is applied under high temperature condition and various chemical solutions.
4. Complete protective construction is designed for dust and soldering flux. If required, plastic sealed type is available for washing procedure.
5. 12A at 120VAC for RW & 12A at 240VAC for RWH are UL approved.

Application

Domestic Appliances, Office Machines, Audio Equipment, Coffee-Pots, Control units, etc.

4.23 SOLENOID VALVE:



4.23.1 OPERATING PRINCIPLES

The term “solenoid” does not refer to the valve itself, but to the operator and coil mounted on the moveable valve, also known as “pilot” or “magnetic actuator”. The term “solenoid” drives from the Greek “solen” which means “channel”. The coil in fact “channels” a strong magnetic force in the windings when the coil is energised, i.e. when electric current flows through it. The coil consists of capillary copper wire wound on a support reel. When electric current is fed into the coil, magnetic flow lines are generated, which are strongest in the centre of the coil.

This magnetic flow raises the moveable plunger in the coil until it brings it into contact with the pole piece. The valve body has an orifice through which the liquid or gas flows when the valve is open. The moveable plunger has an integral seat which, when the solenoid coil is energised, moves off the valve (direct operated) orifice or diaphragm (pilot operated) orifice opening the valve. When the coil is de-energised, a return spring

repositions the plunger in the original closing position on the valve or diaphragm orifice, thus cutting off the flow of the fluid.

4.23.2 Principles of operation

In compact solenoid valves, the solenoid coil is mounted directly on the enclosing tube, sealed and integral with the valve body. The moveable plunger is free to move in the closing tube and is normally held in position by a thrust (or return) spring. When the solenoid coil is energised, the plunger is attracted by the effect of the magnetic field and the seat, integral with the plunger, opens (or closes) the valve or the valve pilot

4.24 HEART BEAT SENSOR:

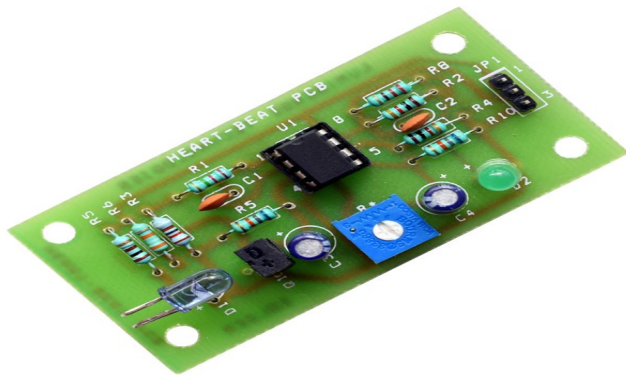
A person's heartbeat is the sound of the valves in his/her's heart contracting or expanding as they force blood from one region to another. The number of times the heart beats per minute (BPM), is the heart beat rate and the beat of the heart that can be felt in any artery that lies close to the skin is the pulse.

4.24.1 SPECIFICATION

Parameter	Value
Operating Voltage	+5V DC regulated
Operating Current	100 mA
Output Data Level	5V TTL level
Heart Beat detection	Indicated by LED and Output High Pulse
Light source	660nm Super Red LED

APPLICATIONS

- Digital Heart Rate monitor
- Patient Monitoring System
- Bio-Feedback control of robotics and applications



CHAPTER 5

5. SOFTWARE REQUIREMENT

5.1 ARDUINO IDE – 1.8.5

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. A worldwide community of makers - students, hobbyists, artists, programmers, and professionals - has gathered around this open-source platform, their contributions have added up to an incredible amount of accessible knowledge that can be of great help to novices and experts alike.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.

5.2 Why Arduino?

The Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux. Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get started with programming and robotics. Designers and architects build interactive prototypes, musicians and artists use it for installations and to experiment with new musical instruments. Makers, of course, use it to build many of the projects exhibited at the Maker Faire, for example. Arduino is a key tool to learn new things. Anyone - children, hobbyists, artists, programmers - can start tinkering just following the step by step instructions of a kit, or sharing ideas online with other members of the Arduino community.

There are many other microcontrollers and microcontroller platforms available for physical computing. Parallax Basic Stamp, Netmedia's BX-24, Phidgets, MIT's Handyboard, and many others offer similar functionality. All of these tools take the messy details of microcontroller programming and wrap it up in an easy-to-use package. Arduino also simplifies the process of working with microcontrollers, but it offers some advantage for teachers, students, and interested amateurs over other systems:

- **Inexpensive** - Arduino boards are relatively inexpensive compared to other microcontroller platforms. The least expensive version of the Arduino module can be assembled by hand, and even the pre-assembled Arduino modules cost less than \$50
- **Cross-platform** - The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems. Most microcontroller systems are limited to Windows.
- **Simple, clear programming environment** - The Arduino Software (IDE) is easy-to-use for beginners, yet flexible enough for advanced users to take advantage of as well. For teachers, it's conveniently based on the Processing programming

environment, so students learning to program in that environment will be familiar with how the Arduino IDE works.

- **Open source and extensible software** - The Arduino software is published as open source tools, available for extension by experienced programmers. The language can be expanded through C++ libraries, and people wanting to understand the technical details can make the leap from Arduino to the AVR C programming language on which it's based. Similarly, you can add AVR-C code directly into your Arduino programs if you want to.

- **Open source and extensible hardware** - The plans of the Arduino boards are published under a Creative Commons license, so experienced circuit designers can make their own version of the module, extending it and improving it. Even relatively inexperienced users can build the breadboard version of the module in order to understand how it works and save money.

5.3 EMBEDDED C:

Embedded c is a set of language extension for the C Programming language by the C Standards committee to address commonality issues that exist between C extension for different embedded systems. Historically embedded C programming requires nonstandard extensions to the C language in order to support exotic features such as fixed-point arithmetic, multiple distinct memory banks, and basic I/O operations.

Embedded C uses most of the syntax and semantics of standard C, e.g., main() function, variable definition, data type declaration, conditional statements(if, switch, case),loops(while, for),functions, arrays and strings, structures and union, bit operations, macros, etc.

During infancy years of microprocessor based systems, programs were developed using assemblers and fused into the EPROMs. there used to be no mechanism to find what the program was doing. LEDs, switches, etc. were used to check for correct

execution of the program .But they were too costly and were not quite reliable as well .As time progressed, use of microprocessor-specific assembly-only as the programming language reduced and embedded systems moved onto C as the embedded programming language of choice. C is the most widely used programming language for embedded processors/controllers.

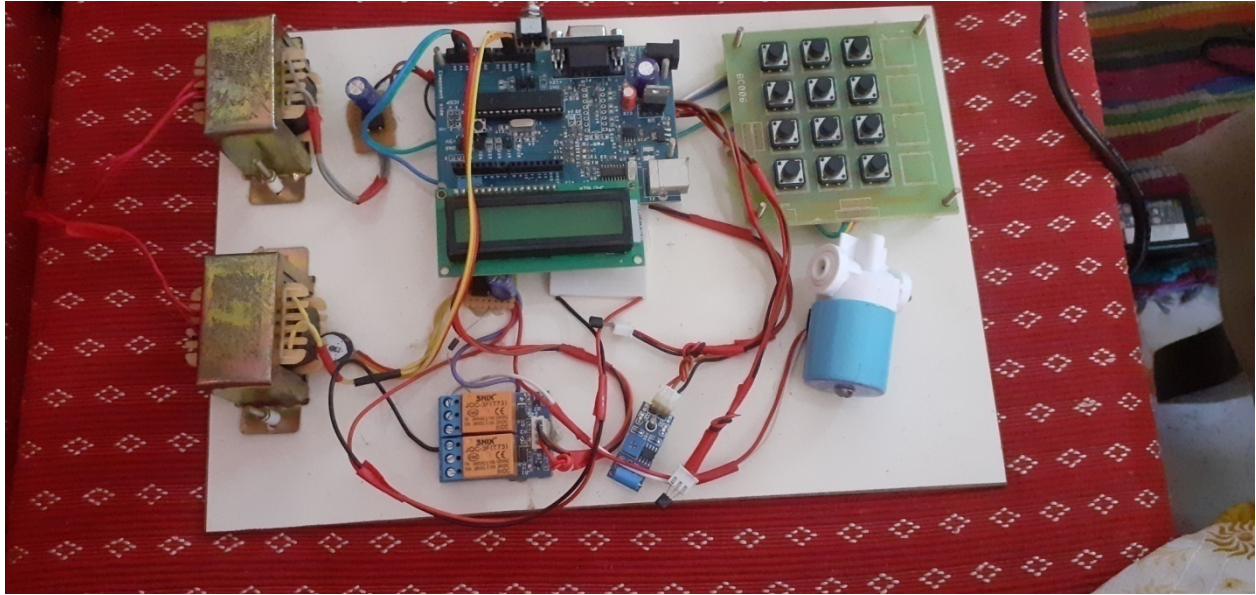


Fig 5.3.1: Connection of Blood Infusion Warmer

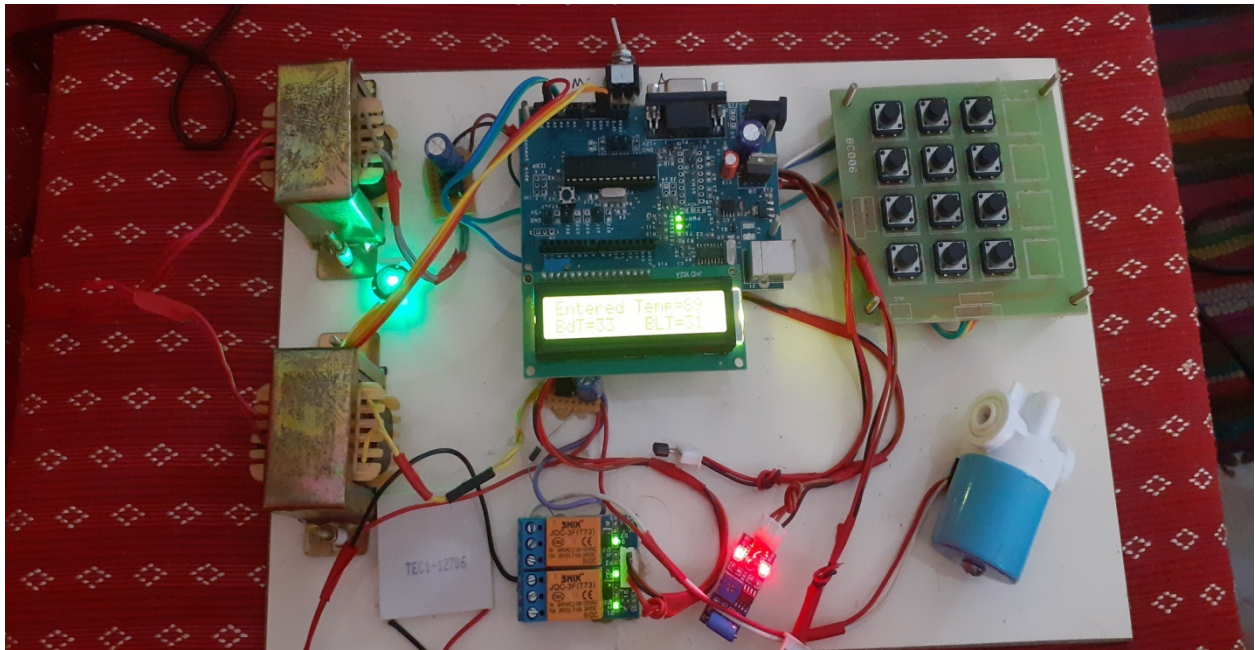


Fig 5.3.2: Manual Value of Blood Infusion Warmer

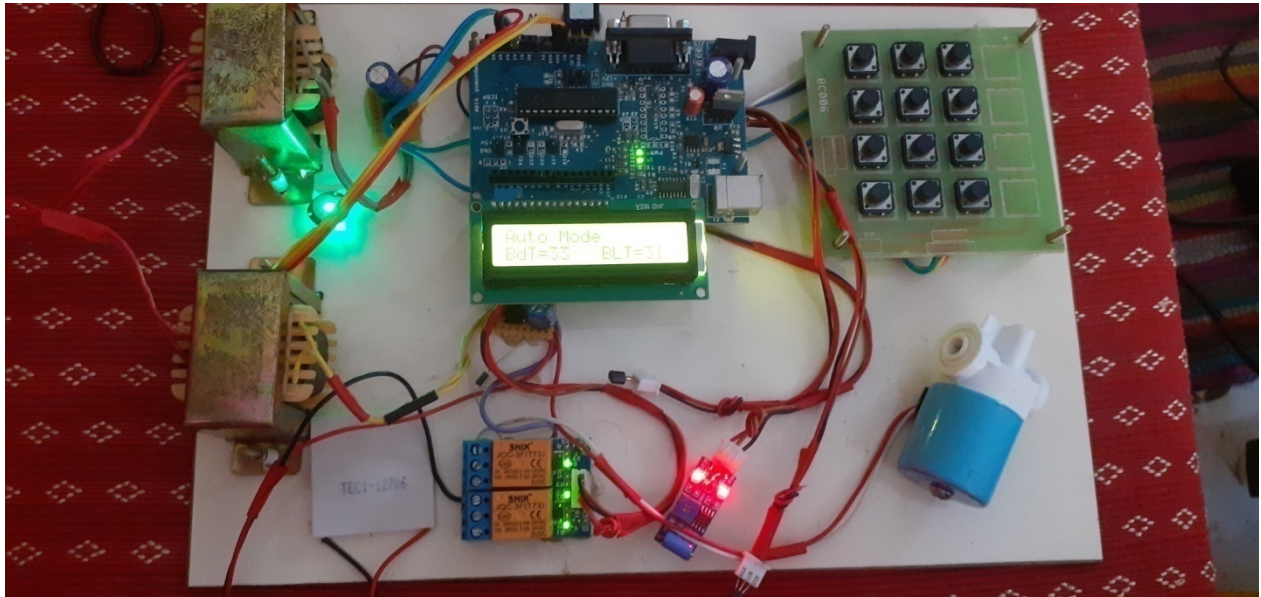


Fig 5.3.3: Auto Mode of Blood Infusion Warmer

CHAPTER 6

RESULT

The Blood Infusion Warmer is connecting between sensors with microcontroller. Main aim of this project is to heat the stored blood by using warmer based on the patient's body temperature.

Here two temperature sensor are been used where one is to detect the body temperature and another sensor to detect the blood bank temperature.

Therefore we introduce microcontroller based portable paramedic blood warmer which aims to decrease the time for warming the blood by providing controlled heating.

When electric current is passed through a circuit consisting of two different conductors, a cooling effect is observed in one junction whereas another junction experiences a rise in temperature. This change in temperatures at the junctions is called the Peltier effect.

When copper wire and bismuth wire are connected in an electric circuit, heat is generated at the point where current passes from copper to bismuth, and a drop in temperature occurs. Where the current passes from bismuth to copper. This effect is reversible in nature. The heating or cooling effect observed at a junction can be reversed by changing the direction of the current flow.

The LM35 is an integrated circuit sensor that can be used to measure temperature with an electrical output proportional to the temperature (in °C). You can measure temperature more accurately than using a thermistor. The sensor circuitry is sealed and not subject to oxidation, etc. The LM35 generates a higher output voltage than thermocouples and may not require that the output voltage be amplified.

A relay is an electrically operated switch. It consists of a set of input terminals for a single or multiple control signals, and a set of operating contact terminals. The switch may have any number of contacts in multiple contact forms, such as make contacts, break contacts, or combinations thereof.

Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online.

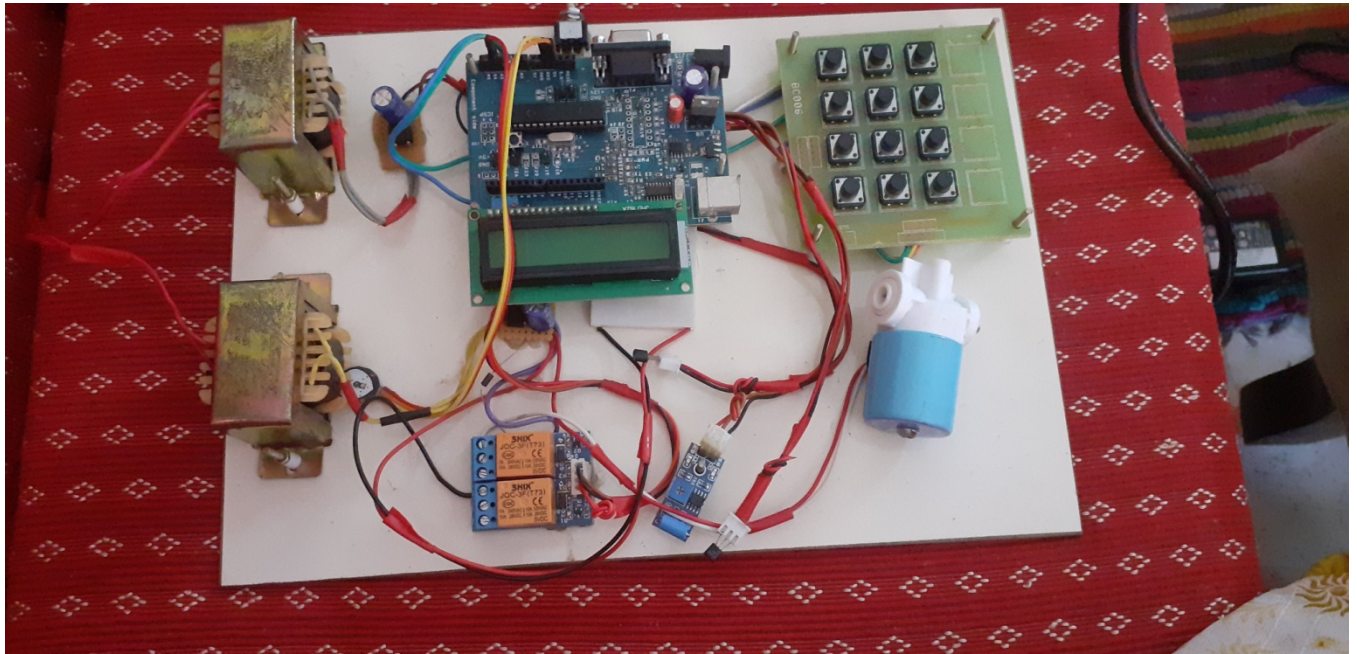


Fig 5.3.4 Total Hardware

CHAPTER 7

CONCLUSION

Main aim of this project is to heat the stored blood by using warmer based on the patient's body temperature. Therefore we introduce microcontroller based portable paramedic blood warmer which aims to decrease the time for warming the blood by providing controlled heating. Both sensors send the output to microcontroller, inbuilt with ADC (Analog-to-Digital Converter). Peltier is a device that can be used either for heating or for cooling. It can also be used as a temperature controller. Relay driver is used to control the vibration due to abnormal condition of the patient. The output is given to LCD display unit which monitors the heat. To overcome this we introduce Microcontroller based portable paramedic blood warmer for transfusion. But, this project is sufficient to provide functional simulation to the actual real-time warmer for blood infusion. As the blood is precious so water or dextrose will be used for this project instead of blood. As the current flows, there is also flow of thermal energy from one junction to another.

CHAPTER 8

REFERENCES

1. Benumof JL. Minimizing venous air embolism from reinfusion bags. *Anesthesiology* 1999; 91:1962.
2. Berzuini A, Bianco C, Paccapelo C, et al. Red cell bound antibodies and transfusion requirements in hospitalized patients with COVID-19. *Blood* 2020; published online June.
3. Bhatraju PK, Ghassemieh BJ, Nichols M, et al. Covid-19 in critically ill patients in the Seattle region—case series. *N Engl J Med* 2020; **382: 2012–22**.
4. Cai X, Ren M, Chen F, Li L, Lei H, Wang X. Blood transfusion during the COVID-19 outbreak. *Blood Transfus* 2020; **18: 79–82**.
5. DOI: 10.1182/ blood.2020006695. Koeckerling D, Pan D, Mudalige NL, Oyefeso O, Barker J. Blood transfusion strategies and ECMO during the COVID-19 pandemic. *Lancet Respir Med* 2020; **8:E40**.
6. Fan BE, Ong KH, Chan SSW, et al. Blood and blood product use during COVID-19 infection. *Am J Hematol* 2020; published online April 12. DOI:10.1002/ajh.25823.
7. Wise J. A third of covid-19 patients admitted to UK hospitals die. *BMJ* 2020; **369:m1794**.
8. WHO. COVID-19 global literature on corona virus disease. 2020. <https://search.bvsalud.org/global-literature-on-novel-coronavirus-2019-ncov/> (accessed March 23, 2020).
9. WHO. International Clinical Trials Registry Platform (ICTRP). <https://www.who.int/ictip/en/> (accessed April 3, 2020).
10. Guan WJ, Ni ZY, Hu Y, et al. Clinical characteristics of corona virus disease 2019 in China. *N Engl J Med* 2020; **382: 1708–20**.

