BRAIN MACHINE INTERFACE FOR WRIST MOVEMENT USING ROBOTIC ARM

APROJECTREPORT

Submitted by

AASHISH S GUNDESHA	**Reg.No.: 1110107002**
GOWTHAM S	**Reg.No.: 1110107030**
HARISH KUMAR C R	**Reg.No.: 1110107031**
KARTHIKEYAN P	**Reg.No.: 1110107041**

In partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION

ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE-641049

(An Autonomous Institution Affiliated to Anna University, Chennai)

APRIL 2015

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE-641049

(An Autonomous Institution Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report**"BRAIN MACHINE INTERFACE FOR WRIST MOVEMENT USING ROBOTIC ARM"**is the bonafide work of **AASHIAH S GUNDESHA, GOWTHAM S, HARISH KUMAR C R,KARTHIKEYAN P**who carried outtheproject workundermysupervision.

SIGNATURE **SIGNATURE**

Mr.R.Darwin	**Dr. Rajeswari Mariappan**
Assistant Professor,	Head of the Department,
Department of ECE,	Department of ECE,
Kumaraguru College of Technology,	Kumaraguru College of Technology,
Coimbatore-641049	Coimbatore-641049

The candidates with Register numbers **1110107002,1110107030,111010731**and **1110107041**are examined by us in the project viva-voce examination held on …………………….

INTERNAL EXAMINER EXTERNAL EXAMINER

ACKNOWLEDGEMENT

First we would like to express our praise and gratitude to the Lord, who has showered his grace and blessing enabling us to complete this project in an excellent manner.

We express our sincere thanks to our beloved Joint Correspondent,**Shri. Shankar Vanavarayar,** Kumaraguru College of Technology, we thank forhis kind support and for providing necessary facilities to carry out the work.

We would like to express our sincere thanks to our beloved Principal **Dr.R.S.Kumar**, Kumaraguru College of Technology, who encouraged us in each and every steps of the project work.

We would like to express our sincere thanks and deep sense of gratitude to our Head of the Department ,**Dr.Rajeswari Mariappan,M.E.,Ph.D.,**Department of Electronics and Communication Engineering, for her valuable suggestions and encouragement which paved way for the successful completion of the project work.

In particular, We wish to thank and everlasting gratitude to the Project Coordinator **Ms.Nagarathinam.S**,**M.E.,(Ph.D)**Assistant Professor SRG, Department of Electronics and Communication Engineering for her expert counseling.

We are greatly privileged to express our deep sense of gratitude to our guide **Mr.R.Darwin,M.E.,Assistant** Professor, Department of ECE, Kumaraguru College of Technology throughout the course of this project work and we wish to convey our deep sense of gratitude to all teaching and non-teaching staff members of ECE Department for their help and cooperation.

Finally,we thank our parents and our family members for giving us the moral support and abundant blessings in all of our activities and our dear friends who helped us to endure our difficult times with their unfailing support and warm wishes.

CONTENTS

ABSTRACT

This project discussed about a brain controlled robot based on Brain–computer interfaces (BCI).BCIs are systems that can bypass conventional channels of communication (i.e., muscles and thoughts) to provide direct communication and control between the human brain and physical devices by translating different patterns of brain activity into commands in real time. With these commands a mobile robot can be controlled. The intention of the project work is to develop a robot that can assist the disabled people in their daily life to do some work independent of others.

Here, we analyze the brain wave signals. Human brain consists of millions of interconnected neurons. The pattern of interaction between these neurons are represented as thoughts and emotional states. According to the human thoughts, this pattern will be changing which in turn produce different electrical waves. A muscle contraction will also generate a unique electrical signal. All these electrical waves will be sensed by the brain wave sensor and it will convert the data into packets and transmit through Bluetooth medium. Level analyzer unit (LAU) will receive the brain wave raw data and it will extract and process the signal using MATLAB platform. Then the control commands will be transmitted to the robot module to process. With this entire system, we can move a robot according to the human thoughts and it can be turned by blink muscle contraction.

LIST OF ABBREVATIONS

UART	Universal Asynchronous Receiver/Transmitter
SPI	Serial Peripheral Interface
SSP	System Service processor
GPIO	General Purpose Input/Output
BOD	Brown-Out Detect
RTC	Real-Time Clock
ADCR	ADC control register
ADDR	ADC Data Register
ADGDR	ADC Global Data Register
DTR	Data Terminal Ready
CTS	Clear to send
RTS	Request To Send
RISC	Reduced Instruction Set Computer
EEPROM	Electrically Erasable Programmable Read Only Memory
SRAM	Static Random Access Memory
LED	Light Emitting Diode
LCD	Liquid Crystal Display

RPM	Rotation per Minute
PWM	Pulse-width modulation
MEMS	Micro-Electro-Mechanical Systems
LDR	Light Dependant Resistor

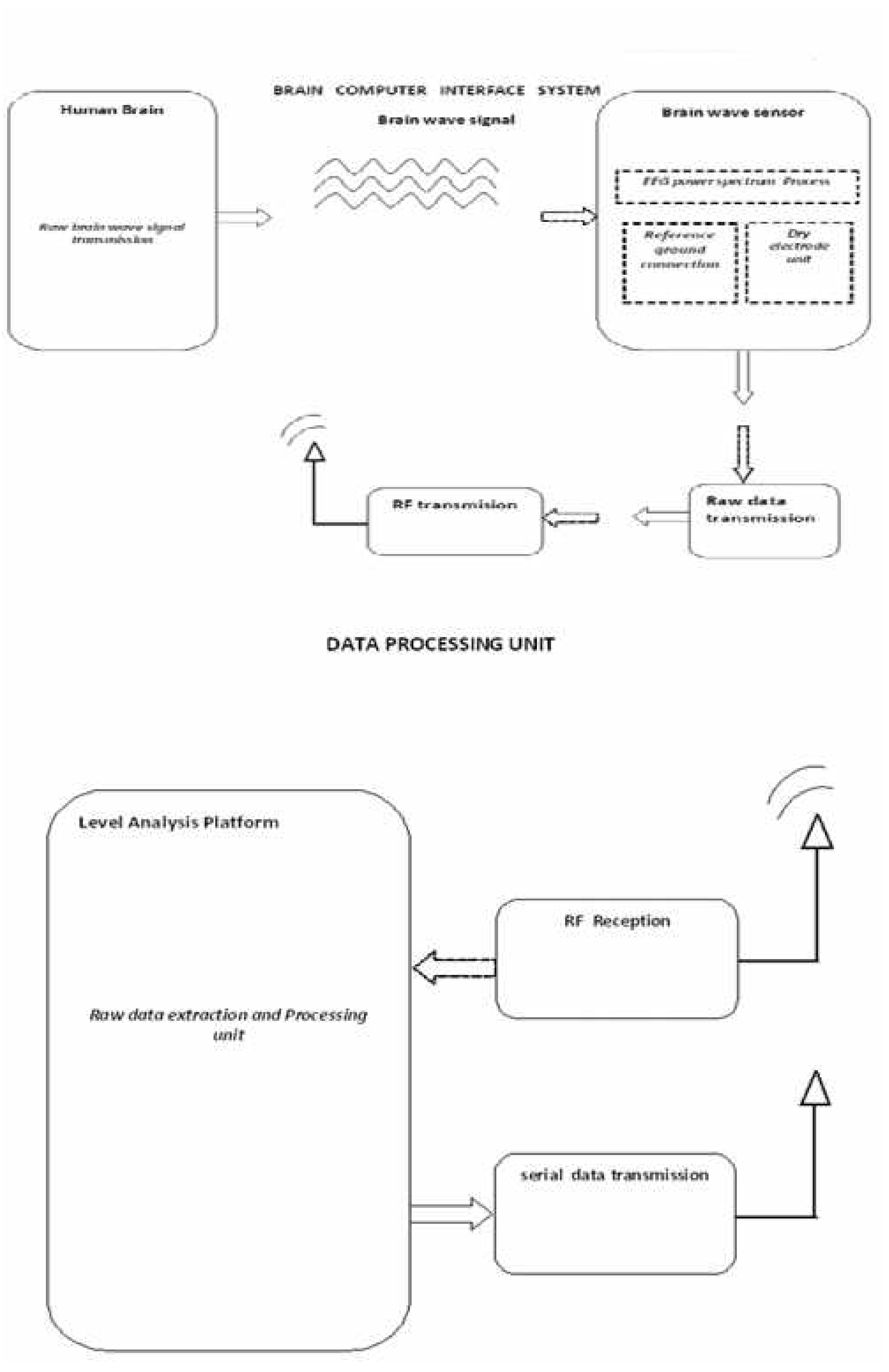
LIST OF FIGURES

INTRODUCTION

There has been a sudden surge in the research for the Brain Machine Interface (BMI) technology as a mode ofcommunication for the patients suffering from neurological disabilities. The discoverer of the existence of human EEGsignals was Hans Berger (1873–1941). In 1926, Berger started to use the more powerful Siemens double coil galvanometer (attaining asensitivity of 130 μV/cm). The first report of 1929 by Berger included the alpha rhythm as the major component of the EEG signals, and the alpha blocking response . Up to now, a lot of BCI systems have been developed for a variety of application purposes. For example,researchers at Graz university of technology have developed an EEG-based neuro-prosthesis by which a patient was able to grasp a simple object, and then release it after moving it from one place to another place , and a system integrated with functional electrical stimulation (FES) by which a tetraplegic patient could grasp a cylinder with the paralyzed hand through the control of beta oscillation signal . Also complex robotic devices have been successfully and reliably controlled by BCIs, by exploiting smart interaction designs, such as shared control. Mill´an's group has pioneered the use of shared control in neuroprosthetics, by taking the continuous estimation of the user's intentions and providing appropriate assistance to execute tasks safely and reliably . The basic principle for the working of BMI is the conversion of the neural signals into command for controlling external devices The acquisition of signals from brain can be in two ways Invasive or Non-invasive techniques. Non-invasive technique works by using external sensors where as Invasive technique works by implanting sensors inside or superficially on the brain. Invasive technique has surgical procedures for implanting sensors which may cause medical problems and are permanently placed inside the brain.In this paper EEG, a Non invasive technique has been used for recording the brain signals. Among the many invasive and noninvasive brain signal acquisition techniques, Electroencephalogram (EEG) is the most commonly adopted non-invasive method in BCI because of its high temporal resolution, ease of use, low cost and portability . The data is used to classify the movements. Figure 1. Flexion and extension wrist movements respectively The final result has been demonstrated on the Robotic Arm using ARM7 Micr

2.1BLOCK DIAGRAM

Brain wave sensor unit



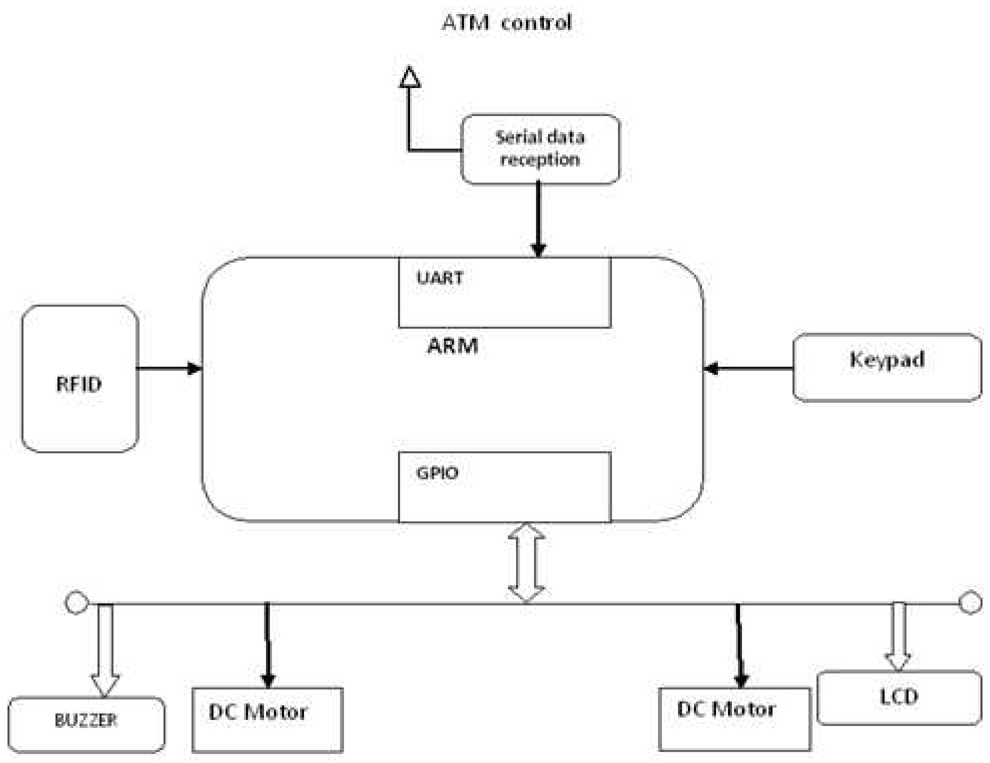


Fig 2.1. Brain wave sensor block diagram.

2.2HARDWARE DESCRIPTION

2.2.1MICROCONTROLLER

INTRODUCTION

The ARM micro-controllers are based on a 32/16 bit ARM7TDMI-S CPU core. They have real-time emulation and embedded trace support that combines the micro-controller with embedded high speed flash memory of 512 KB. A 128-bit wide memory interface and unique accelerator architecture enable 32-bit code execution at the maximum clock rate. For critical code size applications, the alternative 16-bit Thumb mode (16bit instruction set) reduces code by more than 30% with minimal performance penalty.

Due to their tiny size and low power consumption, LPC2148 are ideal for applications where miniaturization is a key requirement, such as access control systems and point-of-sale systems. It has serial communications interfaces ranging from a USB 2.0 Full Speed device, multiple UARTS, SPI, and SSP to I2Cs. It has on-chip SRAM of 8 KB up to 40 KB. This makes these devices very well suited for communication gateways and protocol converters, soft modems, voice recognition and low end imaging, providing both large buffer size and high processing power.

The ARM is a 32-bitreduced instruction set computer (RISC) instruction set architecture (ISA) developed by ARM Holdings. It was known as the Advanced RISC Machine, and before that as the Acorn RISC Machine. The ARM architecture is the most widely used 32-bit ISA in terms of numbers produced. They were originally conceived as a processor for desktop personal computers by Acorn Computers, a market now dominated by the x86 family used by IBM PC compatible and Apple Macintosh computers. The relative simplicity of ARM processors made them suitable for low power applications. This has made them dominant in the mobile and embedded electronics market as relatively low cost and small microprocessors and microcontrollers.

ARM processors are developed by ARM and by ARM licensees. Prominent ARM processor families developed by ARM Holdings include the ARM7, ARM9, ARM11 and Cortex. Notable ARM processors developed by licensees include DECStrong ARM, Freescalei.MX, Marvell (formerly Intel) X Scale, Nintendo, Vida Tetra, ST-EricssonNomadik, Qualcomm Snapdragon, the Texas InstrumentsOMAP product line, the Samsung Hummingbird and the Apple A4.

ARM Architecture

The architecture used in smart phones, personal digital assistants and other mobile devices is anything from ARMv5 in obsolete/low-end devices to ARM M-series in current high-end devices. X Scale and ARM926 processors are ARMv5TE, and are now more numerous in high-end devices than the Strong ARM, ARM9TDMI and ARM7TDMI based ARMv4 processors, but lower-end devices may use older cores with lower licensing costs. ARMv6 processors represented a step up in performance from standard ARMv5 cores, and are used in some cases, but Cortex processors (ARMv7) now provide faster and more power-efficient options than all those previous generations. Cortex-A targets applications processors, as needed by smart phones that previously used ARM9 or ARM11. Cortex-R targets real-time applications, and Cortex-M targets microcontrollers. ARM provides a summary of the numerous vendors who implement ARM cores in their design. KEIL also provides a somewhat newer summary of vendors of ARM based processors. ARM further provides a chart displaying an overview of the ARM processor lineup with performance and functionality versus capabilities for the more recent ARM7, ARM9, ARM11, Cortex-M, Cortex-R and Cortex-A device families.

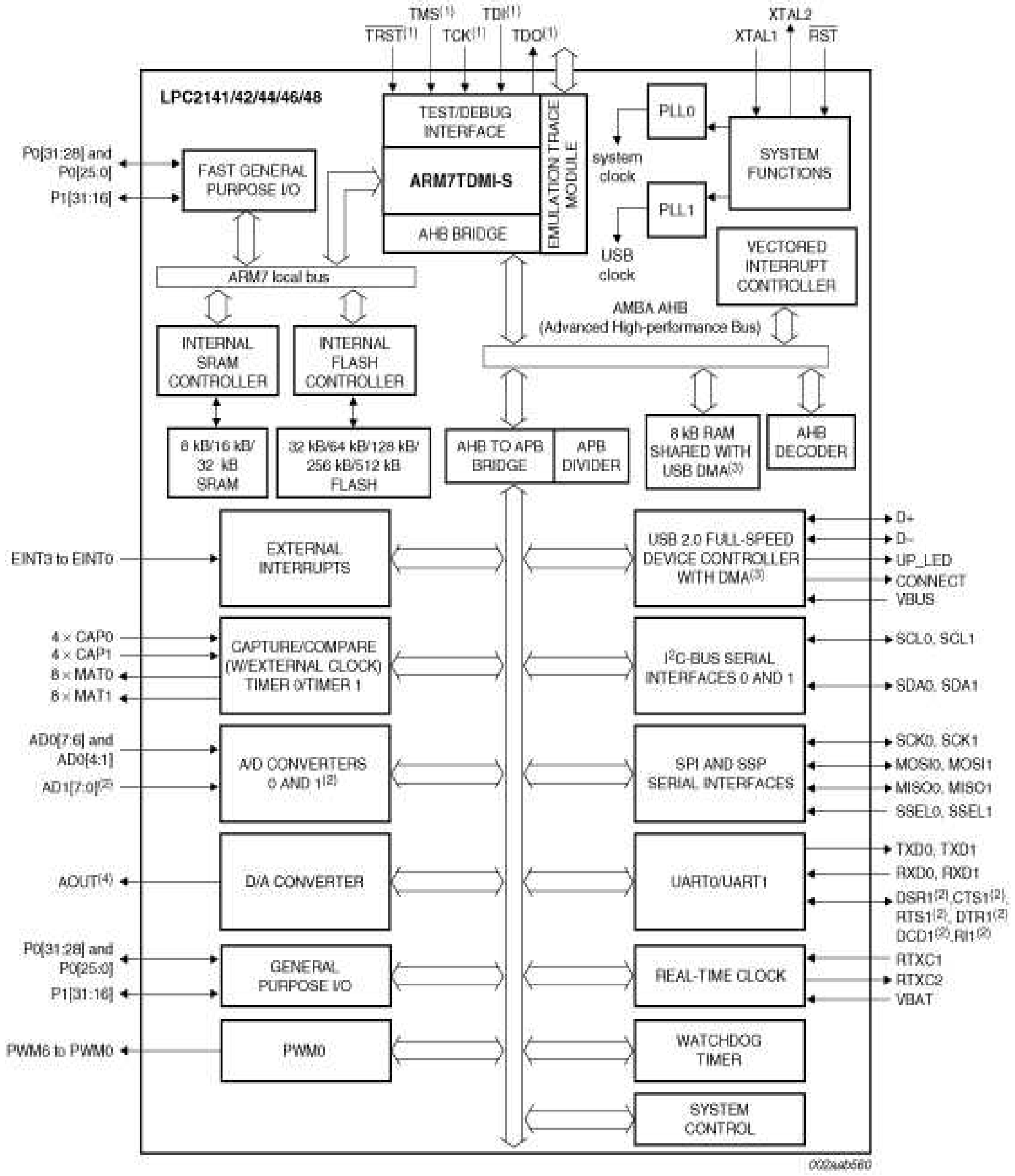


Fig 2.2 Architecture ARM Microcontrller

RISC features

The ARM architecture includes the following RISC features:

- Load/store architecture.
- No support for misaligned memory accesses (now supported in ARMv6 cores, with some exceptions related to load/store multiple word instructions).
- Uniform 16 × 32-bit register file.
- Fixed instruction width of 32 bits to ease decoding and pipelining, at the cost of decreased code density. Later, "the Thumb instruction set" increased code density.
- Mostly single-cycle execution.

Pipelines and other implementation issues

The ARM7 and earlier implementations have a three stage pipeline; the stages being fetch, decode, and execute. Higher performance designs, such as the ARM9, have deeper pipelines: Cortex-A8 has thirteen stages. Additional implementation changes for higher performance include a faster adder, and more extensive branch prediction logic. The difference between the ARM7DI and ARM7DMI cores, for example, was an improved multiplier (hence the added "M").

In ARM 7, a 3 stage pipeline is used. A 3 stage pipeline is the simplest form of pipeline that does not suffer from the problems such as read before written a pipeline, when one instruction is executed, second instruction is decoded and third instruction will be fetched. This is executed in a single cycle.

The ARM7TDMI implementation uses a Three-stage pipeline design. These three stages are

1. Instruction Fetch (F)
2. Instruction Decode (D)
3. Execute (E)

2.2.2 POWER SUPPLY UNIT

CIRCUIT DIAGRAM

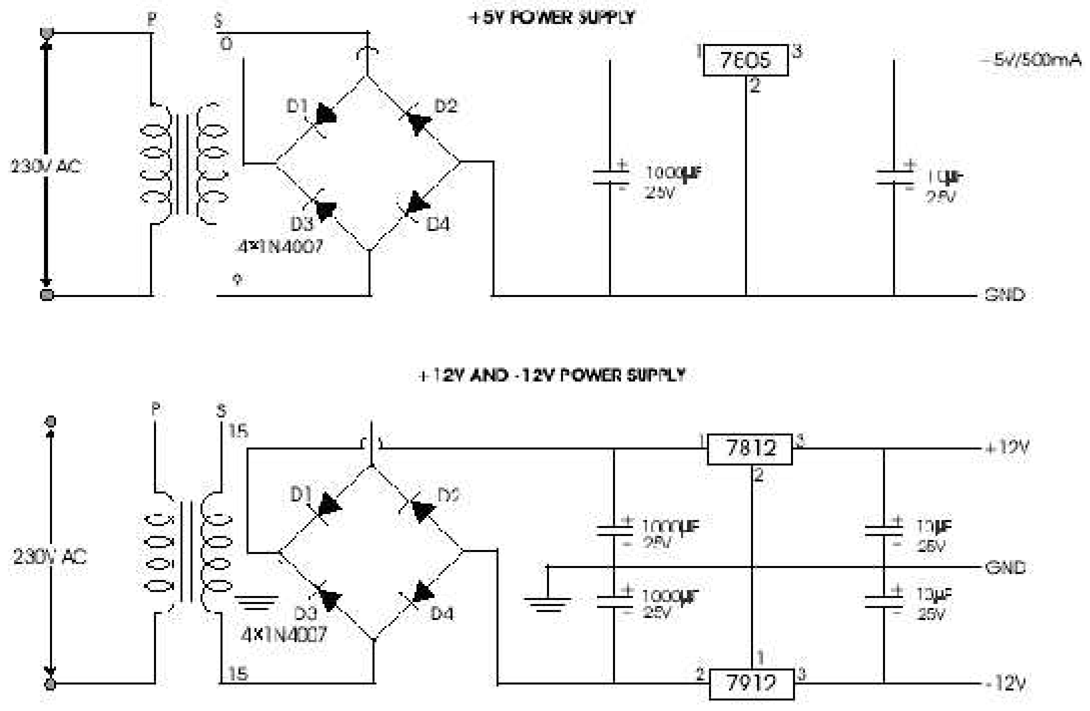


Fig.3.1 Circuit Diagram of Power Supply

WORKING PRINCIPLE

The AC voltage, typically 220V rms, is connected to a transformer, which steps that ac voltage down to the level of the desired DC output. A diode rectifier then provides a full-wave rectified voltage that is initially filtered by a simple capacitor filter to produce a dc voltage. This resulting dc voltage usually has some ripple or ac voltage variation.

A regulator circuit removes the ripples and also remains the same dc value even if the input dc voltage varies, or the load connected to the output dc voltage changes.

Block Diagram

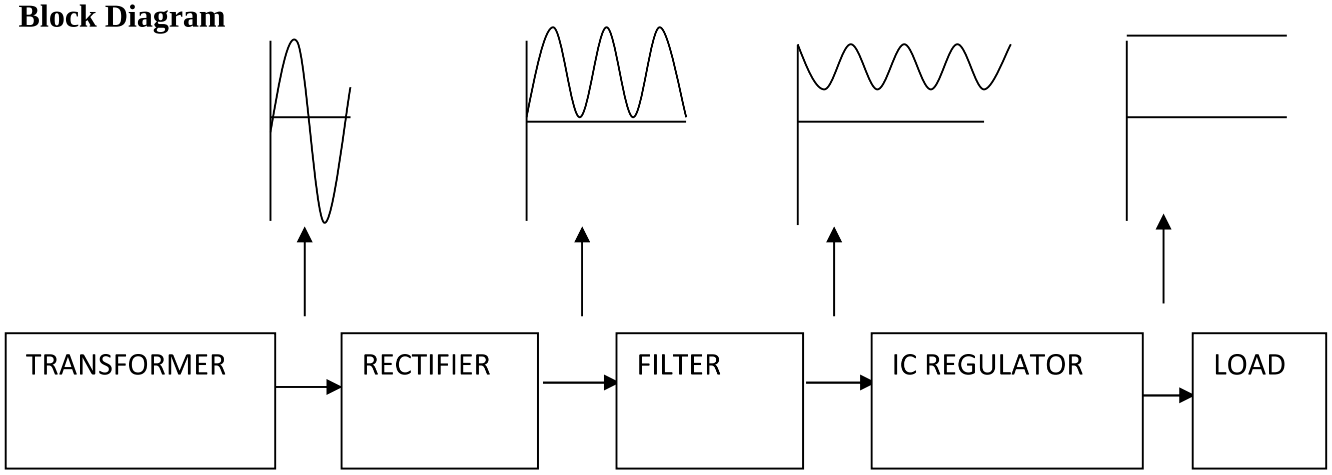


Fig 3.2. block diagram og puwer supply

TRANSFORMER

The potential transformer will step down the power supply voltage (0-230V) to (0-6V) level. Then the secondary of the potential transformer will be connected to the precision rectifier, which is constructed with the help of op–amp. The advantages of using precision rectifier are it will give peak voltage output as DC, rest of the circuits will give only RMS output.

BRIDGE RECTIFIER

When four diodes are connected as shown in figure, the circuit is called as bridge rectifier. The input to the circuit is applied to the diagonally opposite corners of the network, and the output is taken from the remaining two corners.

Let us assume that the transformer is working properly and there is a positive potential, at point A and a negative potential at point B. the positive potential at point A will forward bias D3 and reverse bias D4.

The negative potential at point B will forward bias D1 and reverse D2. At this time D3 and D1 are forward biased and will allow current flow to pass through them; D4 and D2 are reverse biased and will block current flow.

The path for current flow is from point B through D1, up through RL, through D3, through the secondary of the transformer back to point B. this path is indicated by the solid arrows. Waveforms (1) and (2) can be observed across D1 and D3.

One-half cycle later the polarity across the secondary of the transformer reverse, forward biasing 2 and D4 and reverse biasing D1 and D3. Current flow will now be from point A through D4, up through RL, through D2, through the secondary of T1, and back to point A. This path is indicated by the broken arrows. Waveforms (3) and (4) can be observed across D2 and D4. The current flow through RL is always in the same direction. In flowing through RL this current develops a voltage corresponding to that shown waveform (5). Since current flows through the load (RL) during both half cycles of the applied voltage, this bridge rectifier is a full-wave rectifier.

One advantage of a bridge rectifier over a conventional full-wave rectifier is that with a given transformer the bridge rectifier produces a voltage output that is nearly twice that of the conventional full-wave circuit.

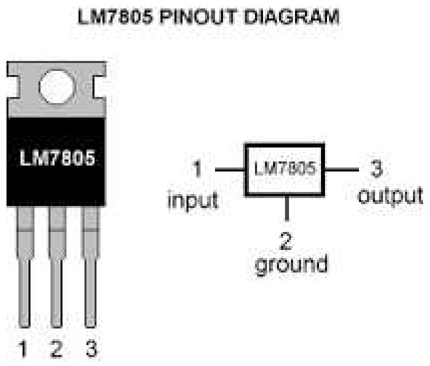
This may be shown by assigning values to some of the components shown in views A and B. assume that the same transformer is used in both circuits. The peak voltage developed between points X and y is 1000 volts in both circuits. In the conventional full-wave circuit shown—in view A, the peak voltage from the center tap to either X or Y is 500 volts.

Since only one diode can conduct at any instant, the maximum voltage that can be rectified at any instant is 500 volts.

The maximum voltage that appears across the load resistor is nearly-but never exceeds-500 v0lts, as result of the small voltage drop across the diode. In the bridge rectifier shown in view B, the maximum voltage that can be rectified is the full secondary voltage, which is 1000 volts. Therefore, the peak output voltage across the load resistor is nearly 1000 volts. With both circuits using the same transformer, the bridge rectifier circuit produces a higher output voltage than the conventional full-wave rectifier circuit.

IC VOLTAGE REGULATORS

Voltage regulators comprise a class of widely used ICs. Regulator IC units contain the circuitry for reference source, comparator amplifier, control device, and overload protection all in a single IC. IC units provide regulation of either a fixed positive voltage, a fixed negative voltage, or an adjustably set voltage. The regulators can be selected for operation with load currents from hundreds of milli amperes to tens of amperes, corresponding to power ratings from milli watts to tens of watts.



A fixed three-terminal voltage regulator has an unregulated dc input voltage, Vi, applied to one input terminal, a regulated dc output voltage, Vo, from a second terminal, with the third terminal connected to ground.

The series 78 regulators provide fixed positive regulated voltages from 5 to 24 volts. Similarly, the series 79 regulators provide fixed negative regulated voltages from 5 to 24 volts.

2.2.3 LCD DISPLAY

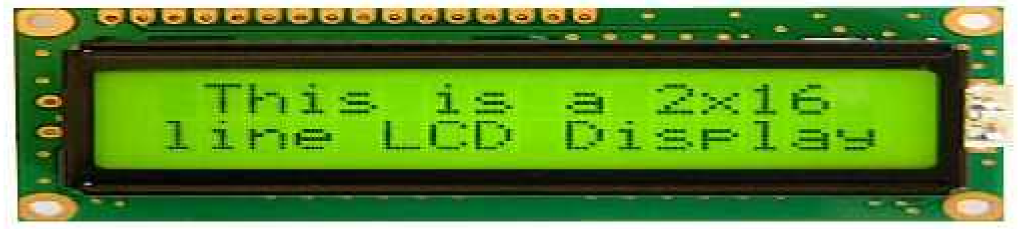
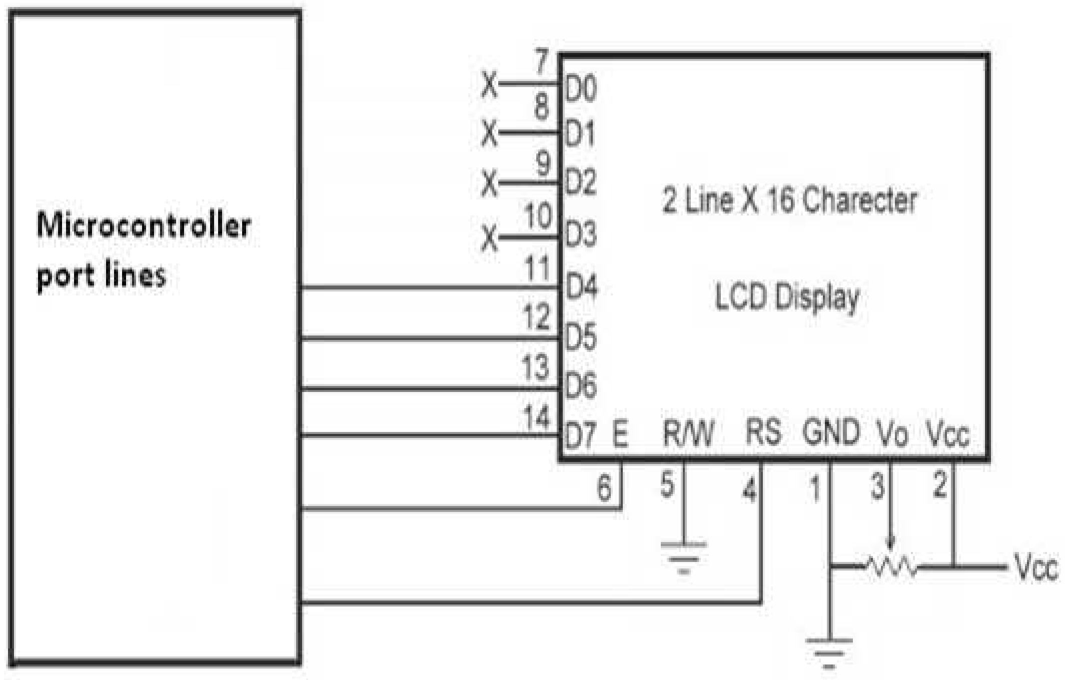


fig 4.1. 2*16 LCD Display

- The HD44780U dot-matrix liquid crystal display controller and driver LSI displays alphanumeric, Japanese kana characters, and symbols. It can be configured to drive a dot-matrix liquid crystal display under the control of a 4- or 8-bit microprocessor. Since all the functions such as display RAM, character generator, and liquid crystal driver, required for driving a dot-matrix liquid crystal display are internally provided on one chip, a minimal system can be interfaced with this controller/driver
- A single HD44780U can display up to one 8-character line or two 8-character lines

- The low power supply (2.7V to 5.5V) of the HD44780U is suitable for any portable battery-driven product requiring low power dissipation



LCD DISPLAY UNIT

A liquid crystal display (LCD) is an electro-optical amplitude modulator realized as a thin, flat display device made up of any number of color or monochrome pixels arrayed in front of a light source or reflector. It is often utilized in battery-powered electronic devices because it uses very small amounts of electric power.

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 liquid crystal between the polarizing filters, light passing through the first filter would be blocked by the second (crossed) polarizer.

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. This treATMent typically consists of a thin polymer layer that is unidirectional rubbed using, for example, a cloth. The direction of the liquid crystal alignment is then defined by the direction of rubbing. 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. 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. Because the liquid crystal material is birefringent, light passing through one polarizing filter is rotated by the liquid crystal helix as it passes through the liquid crystal layer, allowing it to pass through the second polarized filter. Half of the incident light is absorbed by the first polarizing filter, but otherwise the entire assembly is reasonably transparent.

The optical effect of a twisted nematic device in the voltage-on state is far less dependent on variations in the device thickness than that in the voltage-off state. Because of this, these devices are usually operated between crossed polarizers such that they appear bright with no voltage (the eye is much more sensitive to variations in the dark state than the bright state). These devices can also be operated between parallel polarizer, in which case the bright and dark states are reversed. The voltage-off dark state in this configuration appears blotchy, however, because of small variations of thickness across the device.

LCD pin descriptions

The LCD discussed in this section has 14 pins. The function of each pin is given in the table below.

V_{CC}, VSS, and V_{EE}

While V_{CC} and $_{VSS}$ provide +5V and ground, respectively, V_{EE} is used for controlling LCD contrast.

RS, Register Select

There are two very important registers inside the LCD. The RS pin is used for their selection as follows. If RS=0, the instruction command code register is selected, allowing the user to send a command such as clear display, cursor at home, etc. If RS=1 the data register is selected, allowing the user to send data to be displayed on the LCD.

R/W, Read/Write

R/W input allows the user to write information to the LCD or read information from it. R/W=1 when reading; R/W=0 when writing.

E, Enable

The enable pin is used by the LCD to latch information presented to its data pins. When data is supplied to data pins, a high-to-low pulse must be applied to this pin in order for the LCD to latch in the data present at the data pins. This pulse must be a minimum of 450ns wide.

D0-D7

The 8-bit data pins, D0-D7, are used to send information to the LCD or read the contents of the LCD's internal registers.

To display letters and numbers, we send ASCII codes for the letters A-Z, a-z, and numbers 0-9 to these pins while making RS=1.

There are also instruction command codes that can be sent to the LCD to clear the display or force the cursor to the home position or blink the cursor. Table lists the instruction command codes.

We also use RS=0 to check the busy flag bit to see if the LCD is ready to receive information. The busy flag is D7 and can be read when R/W=1 and RS=0, as follows: if R/W=1, RS=0. When D7=1 (busy flag=1), the LCD is busy taking care of internal operations and will not accept any new information.

2.2.4 SERIAL COMMUNICATION

INTRODUCTION

Serial communication is basically the transmission or reception of data one bit at a time. Today's computers generally address data in bytes or some multiple thereof. A byte contains 8 bits. A bit is basically either a logical 1 or zero. Every character on this page is actually expressed internally as one byte. The serial port is used to convert each byte to a stream of ones and zeroes as well as to convert a stream of ones and zeroes to bytes. The serial port contains a electronic chip called a Universal Asynchronous Receiver/Transmitter (UART) that actually does the conversion.

The serial port has many pins. We will discuss the transmit and receive pin first. Electrically speaking, whenever the serial port sends a logical one (1) a negative voltage is effected on the transmit pin. Whenever the serial port sends a logical zero (0) a positive voltage is affected. When no data is being sent, the serial port's transmit pin's voltage is negative (1) and is said to be in a MARK state. Note that the serial port can also be forced to keep the transmit pin at a positive voltage (0) and is said to be the SPACE orBREAK state. (The terms MARK and SPACE are also used to simply denote a negative voltage (1) or a positive voltage (0) at the transmit pin respectively).

When transmitting a byte, the UART (serial port) first sends a STARTBIT which is a positive voltage (0), followed by the data (general 8 bits, but could be 5, 6, 7, or 8 bits) followed by one or two STOP Bits which is a negative (1) voltage. The sequence is repeated for each byte sent. Figure 1 shows a diagram of what a byte transmission would look like.

At this point you may want to know what the duration of a bit is. In other words, how long does the signal stay in a particular state to define a bit. The answer is simple. It is dependent on the baud rate. The baud rate is the number of times the signal can switch states in one second. Therefore, if the line is operating at 9600 baud, the line can switch states 9,600 times per second. This means each bit has the duration of 1/9600 of a second or about 100µsec.

when transmitting a character there are other characteristics other than the baud rate that must be known or that must be setup. These characteristics define the entire interpretation of the data stream.   The first characteristic is the length of the byte that will be transmitted. This length in general can be anywhere from 5 to 8 bits.

The second characteristic is parity. The parity characteristic can be even, odd, mark, space, or none. If even parity, then the last data bit transmitted will be a logical 1 if the data transmitted had an even amount of 0 bits. If odd parity, then the last data bit transmitted will be a logical 1 if the data transmitted had an odd amount of 0 bits. If MARK parity, then the last transmitted data bit will always be a logical 1. If SPACE parity, then the last transmitted data bit will always be a logical 0. If no parity then there is no parity bit transmitted.

The third characteristic is the amount of stop bits. This value in general is 1 or 2. Assume we want to send the letter 'A' over the serial port. The binary representation of the letter 'A' is 01000001. Remembering that bits are transmitted from least significant bit (LSB) to most significant bit (MSB), the bit stream transmitted would be as follows for the line characteristics 8 bits, no parity, 1 stop bit and 9600 baud.

LSB (0 1 0 0 0 0 0 1 0 1) MSB

The above represents (Start Bit) (Data Bits) (Stop Bit). To calculate the actual byte transfer rate simply divide the baud rate by the number of bits that must be transferred for each byte of data. In the case of the above example, each character requires 10 bits to be transmitted for each character. As such, at 9600 baud, up to 960 bytes can be transferred in one second.

The above discussion was concerned with the "electrical/logical" characteristics of the data stream. We will expand the discussion to line protocol. Serial communication can be half duplex or full duplex. Full duplex communication means that a device can receive and transmit data at the same time. Half duplex means that the device cannot send and receive at the same time. It can do them both, but not at the same time. Half duplex communication is all but outdated except for a very small focused set of applications.

Half duplex serial communication needs at a minimum two wires, signal ground and the data line. Full duplex serial communication needs at a minimum three wires, signal ground, transmit data line, and receive data line. The RS232 specification governs the physical and electrical characteristics of serial communications. This specification defines several additional signals that are asserted (set to logical 1) for information and control beyond the data signal

These signals are the Carrier Detect Signal (CD), asserted by modems to signal a successful connection to another modem, Ring Indicator (RI), asserted by modems to signal the phone ringing, Data Set Ready (DSR), asserted by modems to show their presence, Clear To Send (CTS), asserted by modems if they can receive data, Data Terminal Ready (DTR), asserted by terminals to show their presence, Request To Send (RTS), asserted by terminals if they can receive data. The section RS232 Cabling describes these signals and how they are connected.

The above paragraph alluded to hardware flow control. Hardware flow control is a method that two connected devices use to tell each other electronically when to send or when not to send data. A modem in general drops (logical 0) its CTS line when it can no longer receive characters. It re-asserts it when it can receive again. A terminal does the same thing instead with the RTS signal. Another method of hardware flow control in practice is to perform the same procedure in the previous paragraph except that the DSR and DTR signal.

Note that hardware flow control requires the use of additional wires. The benefit to this however is crisp and reliable flow control. Another method of flow control used is known as software flow control. This method requires a simple 3 wire serial communication link, transmit data, receive data, and signal ground. If using this method, when a device can no longer receive, it will transmit a character that the two devices agreed on. This character is known as the XOFF character. This character is generally a hexadecimal 13. When a device can receive again it transmits an XON character that both devices agreed to. This character is generally a hexadecimal 11.

RS232

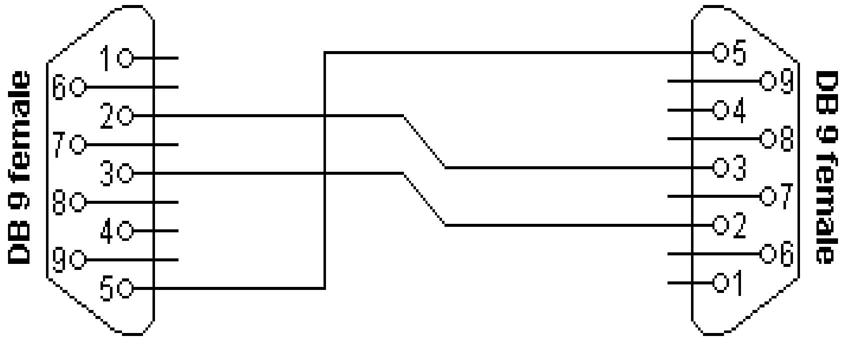
When we look at the connector pin out of the RS232 port, we see two pins which are certainly used for flow control. These two pins are **RTS**, request to send and **CTS**, clear to send. With **DTE/DCE** communication (i.e. a computer communicating with a modem device) **RTS** is an output on the **DTE** and input on the **DCE. CTS** are the answering signal coming from the **DCE.**

Before sending a character, the **DTE** asks permission by setting its **RTS** output. No information will be sent until the **DCE** grants permission by using the **CTS** line.

If the **DCE** cannot handle new requests, the **CTS** signal will go low. A simple but useful mechanism allows flow control in one direction. The assumption is that the **DTE** can always handle incoming information faster than the **DCE** can send it. In the past, this was true. Modem speeds of 300 baud were common and 1200 baud was seen as a high speed connection.

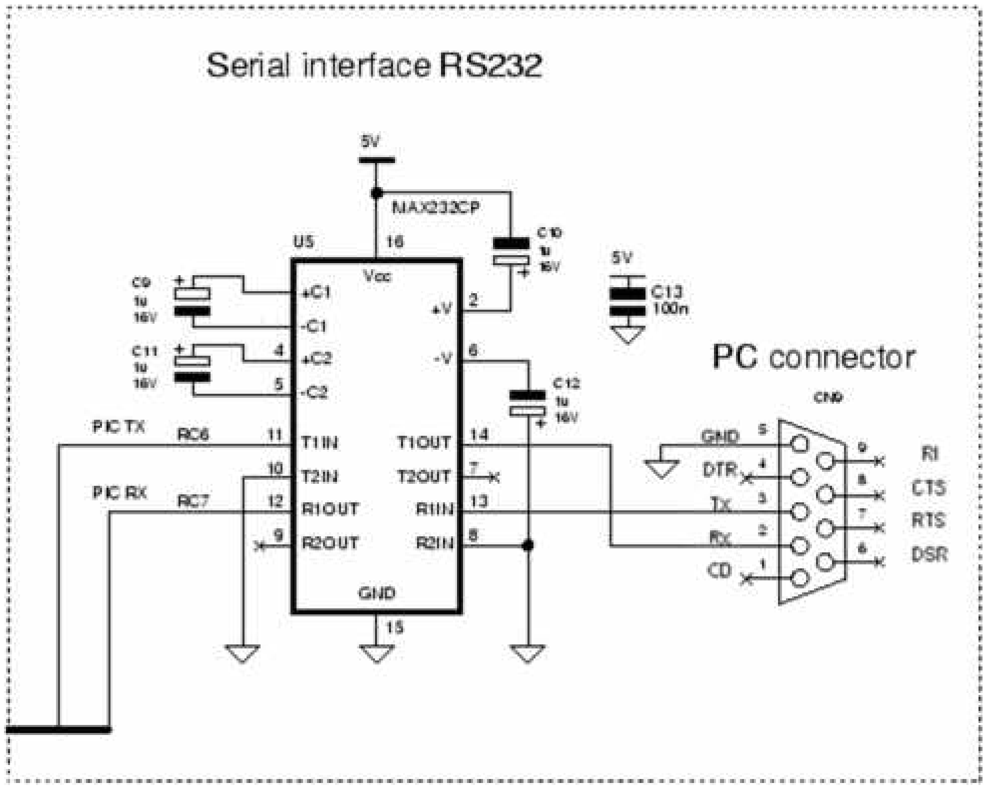
For further control of the information flow, both devices have the ability to signal their status to the other side. For this purpose, the **DTR** data terminal ready and **DSR** data set ready signals are present. The **DTE** uses the **DTR** signal to signal that it is ready to accept information, whereas the **DCE** uses the **DSR** signal for the same purpose. Using these signals involves not a small protocol of requesting and answering as with the **RTS/CTS** handshaking. These signals are in one direction only.

The last flow control signal present in **DTE/DCE** communication is the **CD** carrier detect. It is not used directly for flow control, but mainly an indication of the ability of the modem device to communicate with its counter part. This signal indicates the existence of a communication link between two modem devices.



Connector 1	Connector 2	Function
2	3	Rx ← TX
3	2	TX → Rx
5	5	Signal ground

Fig.4.1. Simple null modem without handshaking



COMPATIBILITY ISSUES

If you read about null modems, this three wire null modem cable is often talked about. Yes, it is simple but can we use it in all circumstances? There is a problem, if either of the two devices checks the **DSR** or **CD** inputs. These signals normally define the ability of the other side to communicate. As they are not connected, their signal level will never go high. This might cause a problem.

The same holds for the **RTS/CTS** handshaking sequence. If the software on both sides is well structured, the **RTS** output is set high and then a waiting cycle is started until a ready signal is received on the **CTS** line. This causes the software to hang because no physical connection is present to either **CTS** line to make this possible. The only type of communication which is allowed on such a null modem line is data-only traffic on the cross connected **Rx/TX** lines.

This does however not mean that this null modem cable is useless. Communication links like present in the Norton Commander program can use this null modem cable. This null modem cable can also be used when communicating with devices which do not have modem control signals like electronic measuring equipment etc.

As you can imagine, with this simple null modem cable no hardware flow control can be implemented. The only way to perform flow control is with software flow control using the **XOFF** and **XON** characters.

2.2.5 BRAINWAVE SENSOR

The TGAM is Neuro Sky's primary brainwave sensor ASIC module designed for mass market applications. The TGAM processes and outputs EEG frequency spectrums, EEG signal quality, raw EEG, and three Neuro Sky eSense meters attention; meditation; and eyeblinks. With simple dry electrodes, this module is excellent for use in toys, video games, and wellness devices because of its low power consumption, which is suitable for portable battery-driven applications.

TGAM

ThinkGear ASIC Module • Directly connects to dry electrode (as opposed to conventional medical wet sensors) • One EEG channel with three contacts: EEG; REF; and GND • Improper fit detected through "Poor Signal Quality" warning from ASIC to reset if off the head for four consecutive seconds, or if it is receiving a poor signal for seven consecutive seconds • Advanced filtering technology with high noise immunity • Low power consumption suitable for portable battery-driven applications • Max power consumption 15mA @ 3.3 V • Raw EEG data output at 512 bits per second

Measures • Raw brainwave signal • Processing and output of EEG power spectrums (Alpha, Beta, etc.) • Processing and output of Neuro Sky proprietary eSense meter for Attention, Meditation, and other future meters • EEG/ECG signal quality analysis (can be used to detect poor contact and whether the device is off the head) • Eyeblink detection Physical • Dimensions (Max) 2.79cm x 1.52cm x 0.25cm • Weight (Max) 130 mg

UART (serial) standard output interface

• 1200, 9600, 57600 output baud rate

• 8bits

• Parity: none

• Stop Bit: 1

Configurable Options AC Noise Filter

• 50 Hz

• 60 Hz

SPECIFICATION

• 512 bits per second sampling frequency • 3-100Hz frequency range • ESD Protection: 4kV Contact Discharge; 8kV Air Discharge • Max Power Consumption: 15mA @ 3.3V • Operating voltage 2.97 ~3.63V

To be used with:

Electrodes • Maximum surface area of ~150mm2 (but less surface area is optimal) • Ag/AgCl, Stainless Steel, Gold, or/and Silver (both solid and plated material works) • EEG electrode located above the left or right eye on the forehead • Ground and reference electrodes located behind the ear or at the earlobe • Have enough pressure to prevent movement, with a minimum of 0.8 PSI Electrodes • Length of less than 12 inches, the longer the higher the susceptibility to noise • Shielding (not necessary for the ground) • Thinner than AWG28.

Output

- Raw-Brainwaves
- Processing and output of EEG power spectrums (Alpha, Beta, etc.)
- Processing and output of Neuro Sky proprietary eSense meter for Attention, Meditation, and other future meters
- EEG/ECG signal quality analysis (can be used to detect poor contact and whether the device is off the head)

2.2.6 DC MOTORS

A DC motor in simple words is a device that converts direct current(electrical energy) into mechanical energy. It’s of vital importance for the industry today, and is equally important for engineers to look into the **working principle of DC motor** Working principle.

The DC or **direct** current **motor** works on the principal, when a current carrying conductor is placed in a magnetic field, it experiences a torque and has a tendency to move. This is known as motoring action. If the direction of current in the wire is reversed, the direction of rotation also reverses. When magnetic field and electric field interact they produce a mechanical force, and based on that the working principle of **dc motor** established. The direction of rotation of a this motor is given by Fleming’s left hand rule, which states that if the index finger, middle finger and thumb of your left hand are extended mutually perpendicular to each other and if the index finger represents the direction of magnetic field, middle finger indicates the direction of current, then the thumb represents the direction in which force is experienced by the shaft of the **dc motor**.

DC motors are widely used in robotics because of their small size and high energy output. They are excellent for powering the drive wheels of a mobile robot as well as powering other mechanical assemblies.

Ratings and Specifications

Several characteristics are important in selecting a DC motor. The first two are its input ratings that specify the electrical characteristics of the motor.

Operating Voltage

If batteries are the source of power for the motor, low operating voltages are desirable because fewer cells are needed to obtain the specified voltage. However, the electronics to drive motors are typically more efficient at higher voltages. Typical DC motors may operate on as few as 1.5 Volts or up to 100 Volts or more. Roboticists often use motors that operate on 6, 12, or 24 volts because most robots are battery powered, and batteries are typically available with these values.

Operating Current

The ideal motor would produce a great deal of power while requiring a minimum of current. However, the current rating (in conjunction with the voltage rating) is usually a good indication of the power output capacity of a motor. The power input (current times voltage) is a good indicator of the mechanical power output. Also, a given motor draws more current as it delivers more output torque. Thus current ratings are often given when the motor is stalled. At this point it is drawing the maximum amount of current and applying maximum torque. A low voltage (e.g., 12 Volt or less) DC motor may draw from 100 mA to several amperes at stall, depending on its design.

The next three ratings describe the motor's output characteristics:

Speed

Usually this is specified as the speed in rotations per minute (RPM) of the motor when it is unloaded, or running freely, at its specified operating voltage. Typical DC motors run at speeds from one to twenty thousand RPM. Motor speed can be measured easily by mounting a disk or LEGO pulley wheel with one hole on the motor, and using a slotted optical switch and oscilloscope to measure the time between the switch openings.

Torque

The torque of a motor is the rotary force produced on its output shaft. When a motor is stalled it is producing the maximum amount of torque that it can produce. Hence the torque rating is usually taken when the motor has stalled and is called the stall torque. The motor torque is measured in ounce-inches (in the English system) or Newton-meters (metric). The torque of small electric motors is often given in milli-Newton-meters (mN-m) or 1/1000 of a N-m. A rating of one ounce-inch means that the motor is exerting a tangential force of one ounce at a radius of one inch from the center of its shaft. Torque ratings may vary from less than one ounce-inch to several dozen ounce-inches for large motors.

Power

The power of a motor is the product of its speed and torque. The power output is greatest at about half way between the unloaded speed (maximum speed, no torque) and the stalled state (maximum torque, no speed). The output power in watts is about (torque) x (rpm).

DC motors

Are inexpensive, small, and powerful motors that are widely used. Gear-train reductions are typically needed to reduce the speed and increase the torque output of the motor.

Types of dc motors

There are several types of motors commonly used in robotic and related applications.

DC motors

Are inexpensive, small, and powerful motors that are widely used. Gear-train reductions are typically needed to reduce the speed and increase the torque output of the motor.

Brushed DC motor

Brushed DC motor, or simply a "DC motor" is a classical example of electrical motor. As discussed before, a motor has a rotor and a stator with one of them being a permanent magnet. In a brushed DC motor, the rotor has permanent magnet and the stator has electromagnets. Since the motor needs a way to detect the rotor's orientation, it uses brushes as a commutator which is a piece of rotor touching the shaft. When the rotor rotates (in turn the brush rotates), it detects the change in orientation and flips the current. DC motors are available in different sizes and at different speeds. Although DC motors run at enough speeds, they are generally useless in robots as they produce the slightest torque. DC motors have only two wires running into them; one for ground and the other for power.

Stepper motors

A stepper motor (or step motor) is a brushless DC electric motor that divides a full rotation into a number of equal steps. Themotor's position can then be commanded to move and hold at one of these steps without any feedback sensor (an open-loop controller), as long as the motor is carefully sized to the application.

Also called actuators, do not rotate continuously, but turn in fixed increments, and resist a change in their fixed positions.

They require special driving circuits to apply the correct sequence of currents to their multiple coils. They are commonly used in robotics, particular in mechanisms that perform linear positioning, such as floppy and hard disk drive head motors and X-Y tables.

Servo motors

A servomotor is a rotary actuator that allows for precise control of angular position, velocity and acceleration. It consists of a suitable motor coupled to a sensor for position feedback. It also requires a relatively sophisticated controller, often a dedicated module designed specifically for use with servomotors.

Are used for angular positioning, such as in radio control airplanes to control the position of wing flaps, or in RC cars to turn the wheels. The output shaft of a servo does not rotate freely as do the shafts of DC motors, but rather is made to seek a particular angular position under electronic control. In effect, a servo motor is a combination of a DC motor, a shaft position sensor, and a feedback circuit. A servo motor also usually includes a built-in gear-train and is capable of delivering high torques directly. No servo motors are included in the 1999 ELEC 201 kit.

Linear DC motor

Not likely to be used in standard mobile robots, a linear DC motor is a normal DC motor with its stator spread out. To be more specific, a brushed DC motor has a rotor spinning inside a stator; in a classical linear DC motor, the stator is unwrapped and laid out in the form of a track made of flat coils. The rotor rolls over the stator in a straight line.

2.3 SOFTWARE DESCRIPTION

2.3.1 Embedded C

The C for microcontrollers and the standard C syntax and semantics are slightly different. The former is aimed at the general purpose programming paradigm whereas the latter is for a specific target microcontroller such as 8051 or PIC. The underlying fact is that everything will be ultimately mapped into the microcontroller machine code. If a certain feature such as indirect access to I/O registers is inhibited in the target microcontroller, the compiler will also restrict the same at higher level. Similarly some C operators which are meant for general purpose computing are also not available with the C for microcontrollers. Even theoperators and constructs which may lead to memory inefficiency are not available in C programming meant for microcontrollers

Be aware that the target code should fit in the limited on-chip memory of the processor. Even the I/O functions available in standard C such as printf() or scanf() are either not made available in C compilers for microcontrollers or advised not to use them. These functions eat up lot of memory space and are not time-efficient owing to the dragging of supporting functions like floating point routines and lot of delimiters. Another striking difference in case of embedded systems programs is that they do nothave the umbrella or support of the operating system. The programmer has to be accustomed with the absence of system calls which makes life easy in traditional C.

2.3.2 MATLAB

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modelling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including graphical user interface building

MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. This allows you to solve many technical computing problems, especially those with matrix and vector formulations, in a fraction of the time it would take to write a program in a scalar non interactive language such as C or FORTRAN.

The name MATLAB stands for matrix laboratory. MATLAB was originally written to provide easy access to matrix software developed by the LINPACK and EISPACK projects. Today, MATLAB uses software developed by the LAPACK and ARPACK projects, which together represent the state-of-the-art in software for matrix computation.

MATLAB has evolved over a period of years with input from many users. In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis.

MATLAB features a family of application-specific solutions called toolboxes. Very important to most users of MATLAB, toolboxes allow you to learn and apply specialized technology. Toolboxes are comprehensive collections of MATLAB functions (M-files) that extend the MATLAB environment to solve particular classes of problems. Areas in which toolboxes are available include signal processing, control systems, neural networks, fuzzy logic, wavelets, simulation, and many others.

The MATLAB System

The MATLAB system consists of five main parts:

Development Environment. This is the set of tools and facilities that help you use MATLAB functions and files. Many of these tools are graphical user interfaces. It includes the MATLAB desktop and Command Window, a command history, and browsers for viewing help, the workspace, files, and the search path.

The MATLAB Mathematical Function Library. This is a vast collection of computational algorithms ranging from elementary functions like sum, sine, cosine, and complex arithmetic, to more sophisticated functions like matrix inverse, matrix eigenvalues, Bessel functions, and fast Fourier transforms.

The MATLAB Language. This is a high-level matrix/array language with control flow statements, functions, data structures, input/output, and object-oriented programming features. It allows both "programming in the small" to rapidly create quick and dirty throw-away programs, and "programming in the large" to create complete large and complex application programs.

Handle Graphics®. This is the MATLAB graphics system. It includes high-level commands for two-dimensional and three-dimensional data visualization, image processing, animation, and presentation graphics. It also includes low-level commands that allow you to fully customize the appearance of graphics as well as to build complete graphical user interfaces on your MATLAB applications.

The MATLAB Application Program Interface (API). This is a library that allows you to write C and FORTRAN programs that interact with MATLAB. It include facilities for calling routines from MATLAB (dynamic linking), calling MATLAB as a computational engine, and for reading and writing MAT-files.

DEVELOPMENT ENVIRONMENT

Introduction

This chapter provides a brief introduction to starting and quitting MATLAB, and the tools and functions that help you to work with MATLAB variables and files. For more information about the topics covered here, see the corresponding topics under Development Environment in the MATLAB documentation, which is available online as well as in print.

Starting and Quitting MATLAB

Starting MATLAB

On a Microsoft Windows platform, to start MATLAB, double-click the MATLAB shortcut icon on your Windows desktop.

On a UNIX platform, to start MATLAB, type matlab at the operating system prompt.

After starting MATLAB, the MATLAB desktop opens - see MATLAB Desktop.
You can change the directory in which MATLAB starts, define startup options including running a script upon start-up, and reduce start-up time in some situations.

Quitting MATLAB

To end your MATLAB session, select Exit MATLAB from the File menu in the desktop, or type quit in the Command Window. To execute specified functions each time MATLAB quits, such as saving the workspace, you can create and run a finish M script.

MATLAB Desktop

When you start MATLAB, the MATLAB desktop appears, containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB.

The first time MATLAB starts, the desktop appears as shown in the following illustration, although your Launch Pad may contain different entries.

You can change the way your desktop looks by opening, closing, moving, and resizing the tools in it. You can also move tools outside of the desktop or return them back inside the desktop (docking). All the desktop tools provide common features such as context menus and keyboard shortcuts.

You can specify certain characteristics for the desktop tools by selecting Preferences from the File menu. For example, you can specify the font characteristics for Command Window text. For more information, click the Help button in the Preferences dialog box.

Desktop Tools

This section provides an introduction to MATLAB's desktop tools. You can also use MATLAB functions to perform most of the features found in the desktop tools. The tools are:

- Current Directory Browser
- Workspace Browser
- Array Editor
- Editor/Debugger
- Command Window
- Command History
- Launch Pad
- Help Browser

Command Window

Use the Command Window to enter variables and run functions and M-files.

Command History

Lines you enter in the Command Window are logged in the Command History window. In the Command History, you can view previously used functions, and copy and execute selected lines. To save the input and output from a MATLAB session to a file, use the diary function.

Running External Programs

You can run external programs from the MATLAB Command Window. The exclamation point character! is a shell escape and indicates that the rest of the input line is a command to the operating system. This is useful for invoking utilities or running other programs without quitting MATLAB. On Linux, for example,!emacsmagik.m invokes an editor called emacs for a file named magik.m. When you quit the external program, the operating system returns control to MATLAB.

Launch Pad

MATLAB's Launch Pad provides easy access to tools, demos, and documentation.

Help Browser

Use the Help browser to search and view documentation for all your Math Works products. The Help browser is a Web browser integrated into the MATLAB desktop that displays HTML documents.

To open the Help browser, click the help button in the toolbar, or type help browser in the Command Window. The Help browser consists of two panes, the Help Navigator, which you use to find information, and the display pane, where you view the information.

Help Navigator

Use to Help Navigator to find information. It includes:

Product filter - Set the filter to show documentation only for the products you specify.

Contents tab - View the titles and tables of contents of documentation for your products.

Index tab - Find specific index entries (selected keywords) in the MathWorks documentation for your products.

Search tab - Look for a specific phrase in the documentation. To get help for a specific function, set the Search type to Function Name.

Favourites tab - View a list of documents you previously designated as favorites.

Display Pane

After finding documentation using the Help Navigator, view it in the display pane. While viewing the documentation, you can:

Browse to other pages - Use the arrows at the tops and bottoms of the pages, or use the back and forward buttons in the toolbar.

Bookmark pages - Click the Add to Favorites button in the toolbar.

Print pages - Click the print button in the toolbar.

Find a term in the page - Type a term in the Find in page field in the toolbar and click Go.

Other features available in the display pane are: copying information, evaluating a selection, and viewing Web pages.

Current Directory Browser

MATLAB file operations use the current directory and the search path as reference points. Any file you want to run must either be in the current directory or on the search path.

Search Path

To determine how to execute functions you call, MATLAB uses a search path to find M-files and other MATLAB-related files, which are organized in directories on your file system. Any file you want to run in MATLAB must reside in the current directory or in a directory that is on the search path. By default, the files supplied with MATLAB and Math Works toolboxes are included in the search path.

Workspace Browser

The MATLAB workspace consists of the set of variables (named arrays) built up during a MATLAB session and stored in memory. You add variables to the workspace by using functions, running M-files, and loading saved workspaces.

To view the workspace and information about each variable, use the Workspace browser, or use the functions who and whos.

To delete variables from the workspace, select the variable and select Delete from the Edit menu. Alternatively, use the clear function.

Array Editor

Double-click on a variable in the Workspace browser to see it in the Array Editor. Use the Array Editor to view and edit a visual representation of one- or two-dimensional numeric arrays, strings, and cell arrays of strings that are in the workspace.

Editor/Debugger

Use the Editor/Debugger to create and debug M-files, which are programs you write to runMATLAB functions. The Editor/Debugger provides a graphical user interface for basic textediting, as well as for M-file debugging.

You can use any text editor to create M-files, such as Emacs, and can use preferences (accessible from the desktop File menu) to specify that editor as the default. If you use another editor, you can still use the MATLAB Editor/Debugger for debugging, or you can use debugging functions, such as dbstop, which sets a breakpoint.

MANIPULATING MATRICES

Entering Matrices

The best way for you to get started with MATLAB is to learn how to handle matrices. Start MATLAB and follow along with each example.

You can enter matrices into MATLAB in several different ways:

- Enter an explicit list of elements.
- Load matrices from external data files.
- Generate matrices using built-in functions.
- Create matrices with your own functions in M-files.

Start by entering Dürer's matrix as a list of its elements. You have only to follow a few basic conventions:

- Separate the elements of a row with blanks or commas.
- Use a semicolon, ; , to indicate the end of each row.
- Surround the entire list of elements with square brackets, [ ].

To enter Dürer's matrix, simply type in the Command Window

```
A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

MATLAB displays the matrix you just entered.

```
A =
   16    3    2   13
    5   10   11    8
    9    6    7    12
    4   15   14    1
```

This exactly matches the numbers in the engraving. Once you have entered the matrix, it is automatically remembered in the MATLAB workspace. You can refer to it simply as A.

Expressions

Like most other programming languages, MATLAB provides mathematical expressions, but unlike most programming languages, these expressions involve entire matrices. The building blocks of expressions are:

Variables

MATLAB does not require any type declarations or dimension statements. When MATLAB encounters a new variable name, it automatically creates the variable and allocates the appropriate amount of storage. If the variable already exists, MATLAB changes its contents and, if necessary, allocates new storage. For example,

```
num_students = 25
```

Creates a 1-by-1 matrix named num_students and stores the value 25 in its single element.

Variable names consist of a letter, followed by any number of letters, digits, or underscores. MATLAB uses only the first 31 characters of a variable name. MATLAB is case sensitive; it distinguishes between uppercase and lowercase letters. A and a are not the same variable. To view the matrix assigned to any variable, simply enter the variable name.

Numbers

MATLAB uses conventional decimal notation, with an optional decimal point and leading plus or minus sign, for numbers. Scientific notation uses the letter e to specify a power-of-ten scale factor. Imaginary numbers use either i or j as a suffix. Some examples of legal numbers are

```
3            -99           0.0001
9.6397238    1.60210e-20   6.02252e23
1i           -3.14159j     3e5i
```

All numbers are stored internally using the long format specified by the IEEE floating-point standard. Floating-point numbers have a finite precision of roughly 16 significant decimal digits and a finite range of roughly 10-308 to 10+308.

Operators

Expressions use familiar arithmetic operators and precedence rules.

+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Left division (described in "Matrices and Linear Algebra" in Using MATLAB)
^	Power
'	Complex conjugate transpose
( )	Specify evaluation order

Functions

MATLAB provides a large number of standard elementary mathematical functions, including abs, sqrt, exp, and sin. Taking the square root or logarithm of a negative number is not an error; the appropriate complex result is produced automatically. MATLAB also provides many more advanced mathematical functions, including Bessel and gamma functions. Most of these functions accept complex arguments. For a list of the elementary mathematical functions, type

Help elfun

For a list of more advanced mathematical and matrix functions,

type

helpspecfun

helpelmat

Some of the functions, like sqrt and sin, are built-in. They are part of the MATLAB core so they are very efficient, but the computational details are not readily accessible. Other functions, like gamma and sinh, are implemented in M-files. You can see the code and even modify it if you want. Several special functions provide values of useful constants.

GUI

A graphical user interface (GUI) is a user interface built with graphical objects, such as buttons, text fields, sliders, and menus. In general, these objects already have meanings to most computer users. For example, when you move a slider, value changes; when you press an OK button, your settings are applied and the dialog box is dismissed. Of course, to leverage this built-in familiarity, you must be consistent in how you use the various GUI-building components.

Applications that provide GUIs are generally easier to learn and use since the person using the application does not need to know what commands are available or how they work. The action that results from a particular user action can be made clear by the design of the interface.

The sections that follow describe how to create GUIs with MATLAB. This includes laying out the components, programming them to do specific things in response to user actions, and saving and launching the GUI; in other words, the mechanics of creating GUIs. This documentation does not attempt to cover the "art" of good user interface design, which is an entire field unto itself. Topics covered in this section include:

Creating GUIs with GUIDE

MATLAB implements GUIs as figure windows containing various styles of uicontrol objects. You must program each object to perform the intended action when activated by the user of the GUI. In addition, you must be able to save and launch your GUI. All of these tasks are simplified by GUIDE, MATLAB's graphical user interface development environment.

GUI Development Environment

The process of implementing a GUI involves two basic tasks:

- Laying out the GUI components
- Programming the GUI components

GUIDE primarily is a set of layout tools. However, GUIDE also generates an M-file that contains code to handle the initialization and launching of the GUI. This M-file provides a framework for the implementation of the call backs - the functions that execute when users activate components in the GUI.

Features of the GUIDE-Generated Application M-File

GUIDE simplifies the creation of GUI applications by automatically generating an M-file framework directly from your layout. You can then use this framework to code your application M-file. This approach provides a number of advantages:
The M-file contains code to implement a number of useful features (see Configuring Application Options for information on these features). The M-file adopts an effective approach to managing object handles and executing callback routines (see Creating and Storing the Object Handle Structure for more information). The M-files provides a way to manage global data (see Managing GUI Data for more information).

Beginning the Implementation Process

To begin implementing your GUI, proceed to the following sections:

Getting Started with GUIDE - the basics of using GUIDE

Selecting GUIDE Application Options - set both FIG-file and M-file options.

Using the Layout Editor - begin laying out the GUI.

Understanding the Application M-File - discussion of programming techniques used in the application M-file.

Application Examples - a collection of examples that illustrate techniques
Which are useful for implementing GUIs.

Command-Line Accessibility

When MATLAB creates a graph, the figure and axes are included in the list of children of their respective parents and their handles are available through commands such as find obj, set, and get. If you issue another plotting command, the output is directed to the current figure and axes.

GUIs are also created in figure windows. Generally, you do not want GUI figures to be available as targets for graphics output, since issuing a plotting command could direct the output to the GUI figure, resulting in the graph appearing in the middle of the GUI.

In contrast, if you create a GUI that contains an axes and you want commands entered in the command window to display in this axes, you should enable command-line access.

User Interface Controls

The Layout Editor component palette contains the user interface controls that you can use in your GUI. These components are MATLAB uicontrol objects and are programmable via their Call back properties. This section provides information on these components.

Figure

Figures are the windows that contain the GUI you design with the Layout Editor. See the description of figure properties for information on what figure characteristics you can control.

2.3.3 KEIL IDE

Keil Software is the leading vendor for 8/16-bit development tools (ranked at first position in the 2004 Embedded Market Study of the Embedded Systems and EE Times magazine). Keil Software is represented world-wide in more than 40 countries. Since the market introduction in 1988, the Keil C51 Compiler is the de facto industry standard and supports more than 500 current 8051 device variants. Now, Keil Software offers development tools for ARM.

Keil Software makes C compilers, macro assemblers, real-time kernels, debuggers, simulators, integrated environments, and evaluation boards for the 8051, 251, ARM, and XC16x/C16x/ST10 microcontroller families.

Keil Software is pleased to announce simulation support for the ATMel AT91 ARM family of microcontrollers. The Keil µVision Debugger simulates the complete ARM instruction-set as well as the on-chip peripherals for each device in the AT91 ARM/Thumb microcontroller family. The integrated simulator provides complete peripheral simulation. Other new features in the µVision Debugger include:

- An integrated Software Logic Analyzer that measures I/O signals as well as program variables and helps developers create complex signal processing algorithms.

- An Execution Profiler that measures time spent in each function, source line, and assembler instruction. Now developers can find exactly where programs spend the most time.

"Using nothing more than the provided simulation support and debug scripts, developers can create a high-fidelity simulation of their actual target hardware and environment. No extra hardware or test equipment is required. The Logic Analyzer and Execution Profiler will help developers when it comes time to develop and tune signaling algorithms." said Jon Ward, President of Keil Software USA, Inc.

2.3.4 FLASH PROGRAMMER

FLASH PROGRAMMERis a software that is used to dump the hex file into the pic controller.

- Straightforward and intuitive user interface
- Five simple steps to erasing and programming a device and setting any options desired
- Programs Intel Hex Files
- Automatic verifying after programming
- Fills unused Flash to increase firmware security
- Ability to automatically program checksums.
- Using the supplied checksum calculation routine your firmware can easily verify the integrity of a Flash block, ensuring no unauthorized or corrupted code can ever be executed
- Program security bits
- Check which Flash blocks are blank or in use with the ability to easily erase all blocks in use
- Read the device signature
- Read any section of Flash and save as an Intel Hex File

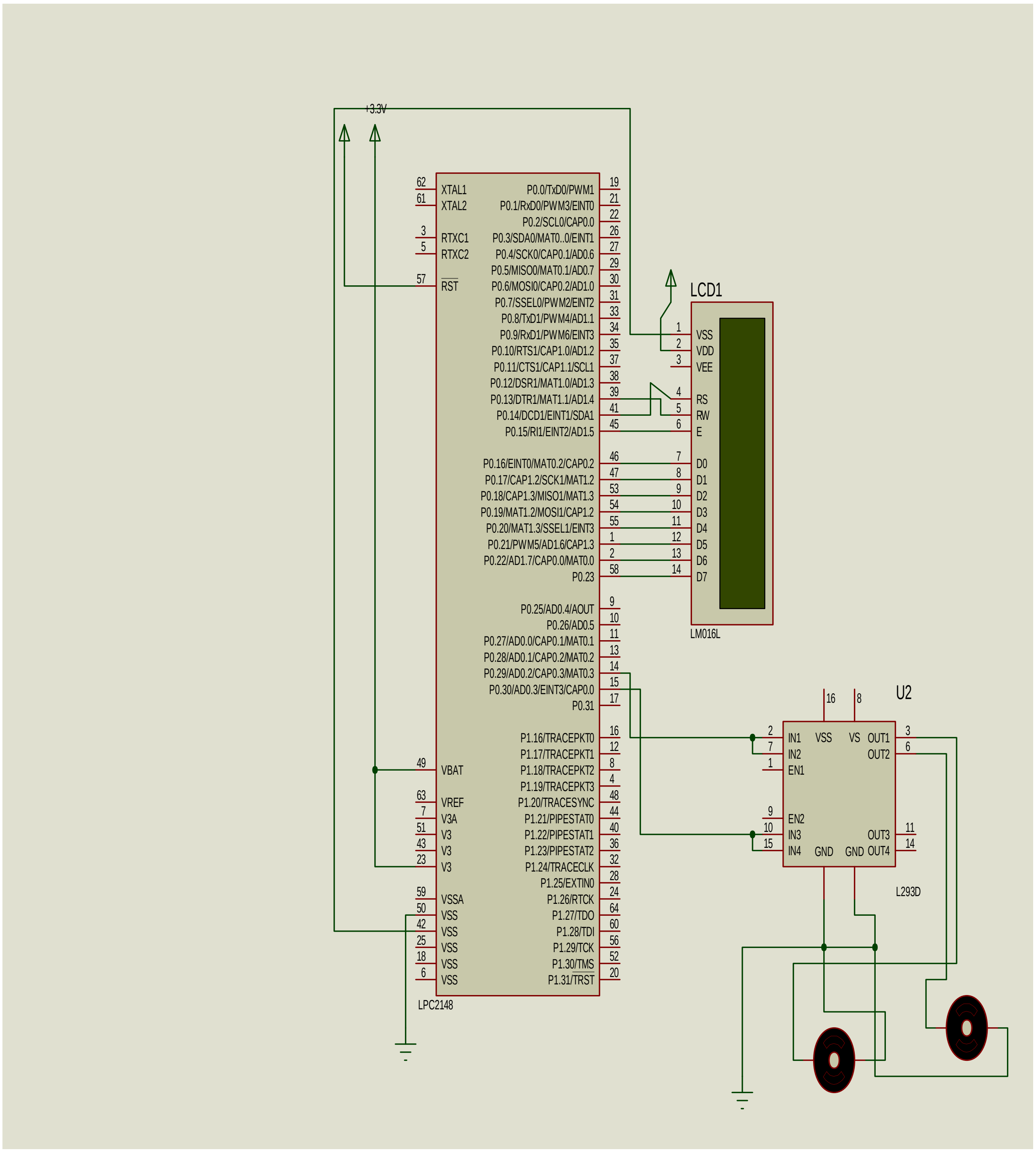
2.3.5OrCAD

OrCAD is a proprietary software tool suite used primarily for electronic design automation. The software is used mainly to create electronic prints for manufacturing of printed circuit boards, by electronic design engineers and electronic technicians to manufacture electronic schematics and diagrams, and for their simulation. The name OrCAD is a portmanteau, reflecting the software's origins: Oregon + CAD.

The OrCAD product line is fully owned by Cadence Design Systems. The latest iteration has the ability to maintain a database of available integrated circuits. This database may be updated by the user by downloading packages from component manufacturers, such as Analog Devices or Texas Instruments.

ORCAD really consists of tools. Capture is used for design entry in schematic form. You will probably be already familiar with looking at circuits in this form from working with other tools in your university courses. Layout is a tool for designing the physical layout of components and circuits on a PCB. During the design process, you will move back and forth between these two tools. The design flow diagram is given below:

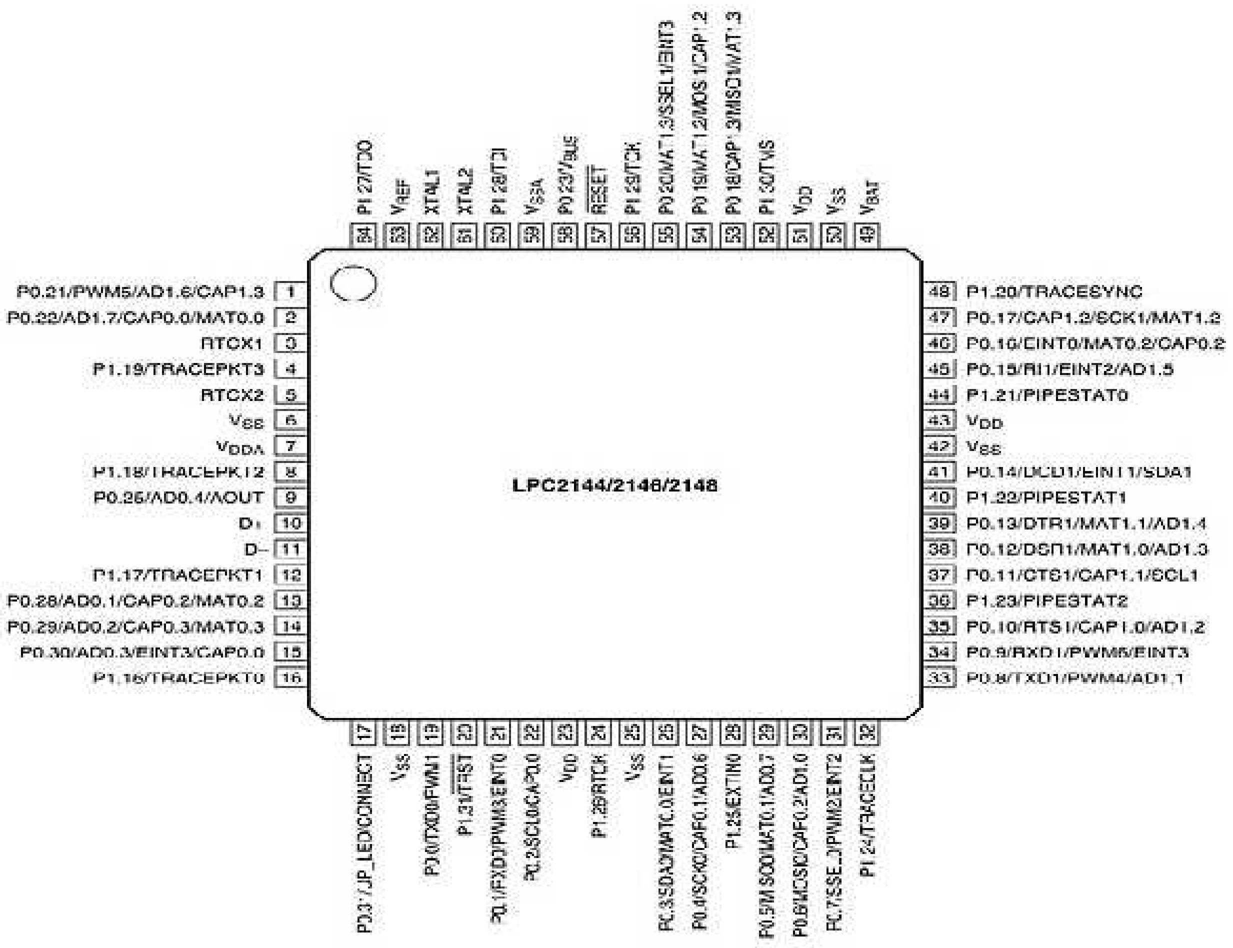
3.1 CIRCUIT DIAGRAM



3.2 CIRCUIT DESCRIPTION

3.2.1 ARM MICROCONTROLLER

LPC2148 PIN DIAGRAM



The Advantages of THUMB

THUMB instructions operate with the standard ARM register configuration, allowing excellent interoperability between ARM and THUMB states. Each 16-bit THUMB instruction has a corresponding 32-bit ARM instruction with the same effect on the processor model.

The major advantage of a 32-bit (ARM) architecture over a 16-bit architecture is its ability to manipulate 32-bit integers with single instructions, and to address a large address space efficiently. When processing 32-bit data, a 16-bit architecture will take at least two instructions to perform the same task as a single ARM instruction.

Memory map concepts and operating modes

The basic concept on the LPC2141/2/4/6/8 is that each memory area has a "natural" location in the memory map. This is the address range for which code residing in that area is written. The bulk of each memory space remains permanently fixed in the same location, eliminating the need to have portions of the code designed to run in different address ranges.

Because of the location of the interrupt vectors on the ARM7 processor (at addresses 0x0000 0000 through 0x0000 001C,) a small portion of the Boot Block and SRAM spaces need to be re-mapped in order to allow alternative uses of interrupts in the different operating modes Re-mapping of the interrupts is accomplished via the Memory Mapping Control feature.

Memory format

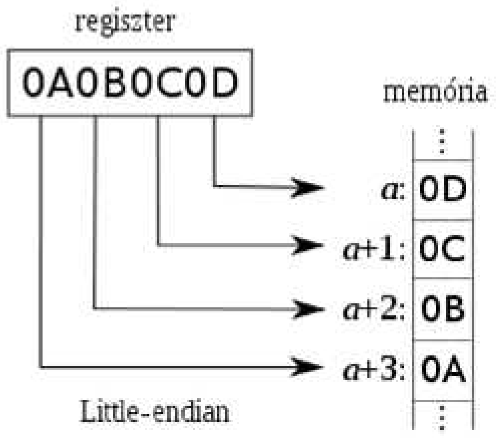
The processor views memory as a linear collection of bytes numbered in ascending order from zero. For example, bytes 0-3 in memory hold the first stored word, and bytes 4-7 hold the second stored word.There are two types of format

Big Endian format

In byte-invariant big-endian format, the processor stores the most significant byte of a word at the lowest-numbered byte, and the least significant byte at the highest-numbered byte. Therefore, byte 0 of the memory system connects to data.

Little Endian Format

In little-endian format, the lowest-numbered byte in a word is the least significant byte of the word and the highest-numbered byte is the most significant. Therefore, byte 0 of the memory system connects to data lines.



GPIO

General Purpose Input/Output (GPIO) is an interface available on some devices. A microprocessor, microcontroller or interface device may have one or more GPIO connections to interface with external devices and peripherals. These can act as input, to read digital signals from other parts of a circuit, or output, to control or signal to other devices. GPIOs are often arranged into groups, typically of 8 pins - a GPIO port - that usually have individual GPIOs configurable either as input or outputs. In some cases, GPIOs may be configurable to produce CPU interrupts and be able to use Direct Memory Access to move large quantities of data efficiently into or out of the device.

GPIO peripherals vary quite widely. In some cases, they are very simple, a group of pins that can be switched as a group to either input or output. In others, each pin can be setup flexibly to accept or source different logic voltages, with configurable drive strengths and pull up/downs. The input and output voltages are typically, though not universally limited to the supply voltage of the device with the GPIOs on, and may be damaged by greater voltages.

The GPIO peripheral provides dedicated general-purpose pins that can be configured as either inputs or outputs.When configured as an output, you can write to an internal register to control the state driven on the output pin. When configured as an input, you can detect the state of the input by reading the state of an internal register.

Purpose of the Peripheral

Most system on a chip (SoC) devices require some general-purpose input/output (GPIO) functionality in order to interact with other components in the system using low-speed interface pins. The control and use of the GPIO capability on this device is grouped together in the GPIO peripheral and is described in the following sections.

Interrupt latency

It is the amount of time between the assertion of an interrupt signal and the start of the associated interrupt service routine. Factors that affect interrupt latency include the length of time that interrupts are disabled during normal program execution, processor speed, and preemption of the processor by higher priority interrupts.

Interrupt service routine

A small piece of software executed in response to a particular interrupt. Abbreviated ISR.

Interrupt type

A unique number associated with each interrupt. The interrupt type is typically the processor's index into the interrupt vector.

Interrupt vector

The address of an interrupt service routine.This term is sometimes used incorrectly to refer to either the interrupt type or the address of the interrupt vector.

External Interrupt

External interrupts are a mechanism for I/O devices that communicate infrequently with the CPU to get the attention of the CPU. Rather than have the CPU constantly check to see if the I/O device needs attention (this is polling), the device interrupts the CPU.

There's a protocol between device and CPU that allows the device to indicate what kind of service it wants from the CPU. Usually, this is done using an interrupt number. The CPU then locates an interrupt handler based on this number, runs the code for the handler, and tells the device it's done.

Dealing with I/O devices can slow down the CPU a lot because devices are often as slow as hard drives, in response time. However, such interrupts are infrequent, so the CPU only has to manage the interrupts when needed.

At this point, you should know what an external interrupt is, and be able to summarize the protocol between the CPU and the interrupting device.

Vector interrupts

This is the fastest system. The onus is placed on the requesting device to request the interrupt, and identify itself. The identity could be a branching address for the desired interrupt-handling routine.

If the device just supplies an identification number, this can be used in conjunction with a lookup table to determine the address of the required service routine. Response time is best when the device requesting service also supplies a branching address.

Priority Interrupt Controller Chips (PIC's) are hardware chips designed to make the task of a device presenting its own address to the CPU simple. The PIC also assesses the priority of the devices connected to it. Modern PIC's can also be programmed to prevent the generation of interrupts which are lower than a desired level.

The decoded location is connected to the output of a priority encoder. The input of the priority encoder is connected to each device. When a device requests service, the priority encoder presents a special code combination (unique for each device) to the decoded memory location. The port thus holds the value or address associated with the highest device requesting service.

The priority encoder arranges all devices in a list, devices given a lower priority are serviced when no other higher priority devices need servicing. This simplifies the software required to determine the device, resulting in an increase in speed.

ADC Converter

ARM7 LPC2148 has 14 channel 10 bit ADC. It give the digital outputs with 10bit resolution.The ADC channels are classified into two types ADC0 and ADC1. ADC0 has six channels (ADC0.0 – ADC0.5) and ADC1 has 8 channels (ADC1.0 to 1.7).

The ADC control register (ADCR) and ADC Data Register(ADDR) and ADGDR (ADC Global Data Register) are used to develop the code in ARM7 2148.in the ADGDR the digital output is stored.

3.2.2 LCD

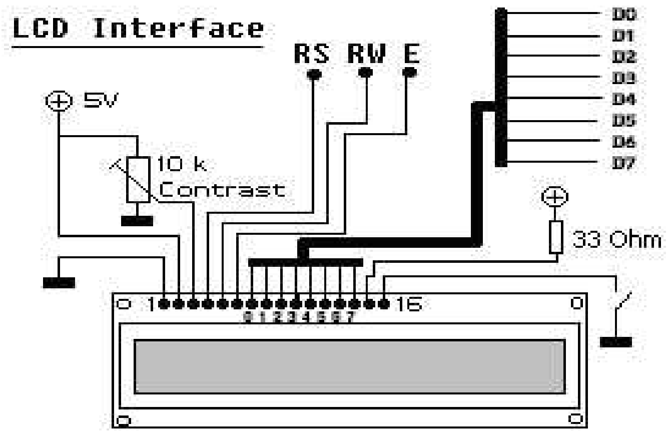


Fig 2.3 LCD Connections

Table LCD PIN DETAILS

Pin No	Name	Function	USE
1	Vss	Ground	
2	Vdd	+ve Supply	5v Volts Regulated DC
3	Vee	Contrast	This is used to set the contrast1
4	RS	Register Set	Register select signal 0:Instruction register (when writing) Busy flag & address counter (When reading) 1:Data register (when writing & reading)
5	R/W	Read / Write	Read/write select signal “0” for writing , “1” for reading
6	E	Enable	Operation (data read/write) enable signal
7	D0	Data Bit 0	
8	D1	Data Bit 1	
9	D2	Data Bit 2	

10	D3	Data Bit 3	
11	D4	Data Bit 4	
12	D5	Data Bit 5	
13	D6	Data Bit 6	
14	D7	Data Bit 7	
15	A	+4.2 for Back light	Positive supply for back light if available
16	K	Power supply Back light (0V)	

4.1.2 Steps to Interface LCD with Microcontroller

STEP 1: Identify:

Determine what you want LCD are available in many flavors which are specified as follows 16x1 , 16x2 , 20x2 in the format AxB where A is the number of columns (chatters) and B is the number of Rows (lines) An LCD might also be Back lit .

STEP 2: Connect: Most of the LCD's follow the standard Hitachi Pin out which is simply...

STEP 3: Interface:

Now connect pins RS ,RW ,E ,D0 - D7 to pins on the micro controller Lets suppose I connect Data bus on port A and the RS , RW , E on port B . (you can save pins by using LCD in Nibble Mode (4 data pins) and permanently grounding the RW line (always in write mode) . Now well see how to go from simple switching it on to graphics on the LCD.

Both the liquid crystal material and the alignment layer material contain ionic compounds. If an electric field of one particular polarity is applied for a long period of time, this ionic material is attracted to the surfaces and degrades the device performance. This is avoided either by applying an alternating current or by reversing the polarity of the electric field as the device is addressed (the response of the liquid crystal layer is identical, regardless of the polarity of the applied field).

When a large number of pixels are needed in a display, it is not technically possible to drive each directly since then each pixel would require independent electrodes. Instead, the display is multiplexed. In a multiplexed display, electrodes on one side of the display are grouped and wired together (typically in columns), and each group gets its own voltage source. On the other side, the electrodes are also grouped (typically in rows), with each group getting a voltage sink. The groups are designed so each pixel has a unique, unshared combination of source and sink. The electronics, or the software driving the electronics then turns on sinks in sequence, and drives sources for the pixels of each sink.

4. FEATURES

4.1 Features of ARM LPC2148

1. 16/32-bit ARM7TDMI-S microcontroller in a tiny LQFP64 package.
2. 8 to 40 KB of on-chip static RAM and 32 to 512 KB of on-chip flash program memory 128 bit wide interface/accelerator enables high speed 60 MHz operation.
3. In-System/In-Application Programming (ISP/IAP) via on-chip boot-loader software.
4. Embedded ICE RT and Embedded Trace interfaces offer real-time debugging with the on-chip Real Monitor software and high speed tracing of instruction execution.
5. USB 2.0 Full Speed compliant Device Controller with 2 KB of endpoint RAM. In addition, the LPC2146/8 provides 8 KB of on-chip RAM accessible to USB by DMA.
6. One or two (LPC2141/2 vs. LPC2144/6/8) 10-bit A/D converters provide a total of 6/14 analog inputs, with conversion times as low as 2.44 μs per channel.
7. Single 10-bit D/A converter provide variable analog output.
8. Two 32-bit timers/external event counters (with four capture and four compare channels each), PWM unit (six outputs) and watchdog.
9. Low power real-time clock with independent power and dedicated 32 kHz clock input
10. Multiple serial interfaces including two UARTs (16C550), two Fast I2C-bus (400 Kbit/s), SPI and SSP with buffering and variable data length capabilities.
11. Vectored interrupt controller with configurable priorities and vector addresses.
12. Up to nine edge or level sensitive external interrupt pins available.
13. 60 MHz maximum CPU clock available from programmable on-chip PLL with settling time of 100 μs.
14. On-chip integrated oscillator operates with an external crystal in range from 1 MHz to 30 MHz and with an external oscillator up to 50 MHz
15. Power saving modes include Idle and Power-down.
16. Individual enable/disable of peripheral functions as well as peripheral clock scaling for additional power optimization.
17. Processor wake-up from Power-down mode via external interrupt, USB, Brown-Out Detect (BOD) or Real-Time Clock (RTC).

18. Single power supply chip with Power-On Reset (POR) and BOD circuits:– CPU operating voltage range of 3.0 V to 3.6 V (3.3 V ± 10 %) with 5 V tolerant I/O pads.

the digital outputs with 10bit resolution.The ADC channels are classified into two types ADC0 and ADC1. ADC0 has six channels (ADC0.0 – ADC0.5) and ADC1 has 8 channels (ADC1.0 to 1.7).

Additional features

1. Power-down mode.
2. Measurement range 0 V to VREF (typically 3 V; not to exceed VDDA voltage level).
3. 10 bit conversion time ≥ 2.44 µs.
4. Burst conversion mode for single or multiple inputs.
5. Optional conversion on transition on input pin or Timer Match signal.
6. Global Start command for both converters is also available in LPC214x series
7. UART : ARM7 2148 have 2 UART channels UART0 and UART1.P0.0 and P0.1 are used for UART0 Tx and Rx pins. And P0.8 and P0.9 is used for UART1.

4.2 FEATURES OF BRAIN WAVE SENSOR

1. Uses the TGAM1 module
2. Automatic wireless pairing
3. TRIPLE AAA Battery
4. 8-hours battery run time
5. Bluetooth v2.1 Class 2 (10 meters range).
6. Static Headset ID (headsets have a unique ID for pairing purposes)
7. MATLAB, Android and iOS support
8. UART Baud rate: 57,600 Baud

PROGRAM

```
function PSEMB301
%run this function to connect and plot raw EEG data
%make sure to change portnum1 to the appropriate COM port
clear all
close all
data_BLINK = zeros(1,256);    %preallocate buffer
data_ATTENTION = zeros(1,256);
data_MEDITATION = zeros(1,256);
   X= zeros(1,256);
portnum1 = 6;   %COM Port BRAINWAVE  #
comPortName1 = sprintf('\\\\.\\COM%d', portnum1);
% Baud rate for use with TG_Connect() and TG_SetBaudrate().
%TG_BAUD_57600 =      57600;
TG_BAUD_115200  =   115200;
% Data format for use with TG_Connect() and TG_SetDataFormat().
TG_STREAM_PACKETS =     0;
% Data type that can be requested from TG_GetValue().
TG_DATA_BATTERY            =    0 ;
TG_DATA_POOR_SIGNAL        =    1 ;
TG_DATA_ATTENTION          =    2 ;
TG_DATA_MEDITATION         =    3 ;
TG_DATA_RAW                =    4 ;
TG_DATA_DELTA              =    5 ;
TG_DATA_THETA              =    6 ;
TG_DATA_ALPHA1             =    7 ;
TG_DATA_ALPHA2             =    8 ;
TG_DATA_BETA1              =    9 ;
TG_DATA_BETA2              =    10 ;
TG_DATA_GAMMA1             =    11 ;
TG_DATA_GAMMA2             =    12 ;
```

```
TG_DATA_BLINK_STRENGTH     =    37 ;
TG_DATA_READYZONE          =    38 ;
%load thinkgear dll
loadlibrary('Thinkgear.dll');
fprintf('Thinkgear.dll loaded\n');
%get dll version
%dllVersion = calllib('Thinkgear', 'TG_GetDriverVersion');
% fprintf('ThinkGear DLL version: %d\n', dllVersion );
%%
% Get a connection ID handle to ThinkGear
connectionId1 = calllib('Thinkgear', 'TG_GetNewConnectionId');
if ( connectionId1 < 0 )
    error( sprintf( 'ERROR: TG_GetNewConnectionId() returned %d.\n', connectionId1 ) );
end;
% Set/open stream (raw bytes) log file for connection
errCode = calllib('Thinkgear', 'TG_SetStreamLog', connectionId1, 'streamLog.txt' );
if( errCode < 0 )
    error( sprintf( 'ERROR: TG_SetStreamLog() returned %d.\n', errCode ) );
end;
% Set/open data (ThinkGear values) log file for connection
errCode = calllib('Thinkgear', 'TG_SetDataLog', connectionId1, 'dataLog.txt' );
if( errCode < 0 )
    error( sprintf( 'ERROR: TG_SetDataLog() returned %d.\n', errCode ) );
end;
% Attempt to connect the connection ID handle to serial port "COM3"
errCode=calllib('Thinkgear','TG_Connect',connectionId1,comPortName1,TG_BAUD_115200,T
G_STREAM_PACKETS );
if ( errCode < 0 )
    error( sprintf( 'ERROR: TG_Connect() returned %d.\n', errCode ) );
   else disp('conected!!');
end
```

```
% fprintf( 'Connect
if(calllib('Thinkgear','TG_EnableBlinkDetection',connectionId1,1)==0)
   disp('blinkdetectenabled');
end
%record data
serialOne=serial('COM1', 'BaudRate', 9600);%CONTROLLER CABLE
j = 0;
i = 0;
k = 0;
l = 0;
Blink=0;
Drive_mode = 0;
X = 0:1:255;
while (i < 100)   %loop for 20 seconds
%while(0)
   if (calllib('Thinkgear','TG_ReadPackets',connectionId1,1) == 1)   %if a packet was read...
      if
(calllib('Thinkgear','TG_GetValueStatus',connectionId1,TG_DATA_BLINK_STRENGTH  )  ~=
0)   %if RAW has been updated
            j = j + 1;
            data_BLINK
(j)=calllib('Thinkgear','TG_GetValue',connectionId1,TG_DATA_BLINK_STRENGTH );
            disp('BLINK = ');
            disp(data_BLINK (j));
            if(data_BLINK(j) > 40 )
 %               if(data_BLINK(j) < 95)
                 Blink = Blink+1;
%               end
            end
```

```
                if(Blink == 3)
                  Blink=0;
 %                  if(Drive_mode ==1)
%                       Drive_mode =0;
%                   else
                  Drive_mode =1;
%                   end
                end
               if(Drive_mode == 1)
%                    if(Blink==2)
%                        Blink=0;
%                        fopen(serialOne);
%                        fwrite(serialOne,'Q');
%                        fclose(serialOne);
%                    end
                 if(data_BLINK (j) > 70)
    %                   if(Drive_mode == 1)
                         fopen(serialOne);
                         fwrite(serialOne,'A'); %OPEN
                         fclose(serialOne);
    %                   end
                  end
                  if(data_BLINK (j) < 60)
   %                    if(Drive_mode == 1)
                         fopen(serialOne);
                         fwrite(serialOne,'A');
                         fclose(serialOne);
   %                    end
                  end
               end
         end
```

```
%    end
%   if (calllib('Thinkgear','TG_ReadPackets',connectionId1,1) == 1)   %if a packet was read...
        if(Drive_mode == 1)
        if  (calllib('Thinkgear','TG_GetValueStatus',connectionId1,TG_DATA_ATTENTION  )  ~=
0)   %if RAW has been updated
%           l = l + 1;
            k = k + 1;
            i = i + 1;
 %           X(i) = i;
            data_ATTENTION(k)                                                              =
calllib('Thinkgear','TG_GetValue',connectionId1,TG_DATA_ATTENTION  );
            disp('ATTENTION = ');
            disp(data_ATTENTION(k));
%           plot(data_ATTENTION)
%           axis([0 100 0 100])
%           drawnow;
           Blink=0;
%          if (calllib('Thinkgear','TG_GetValueStatus',connectionId1,TG_DATA_MEDITATION )
~= 0)   %if RAW has been updated
%           l = l + 1;
%                                                                   data_MEDITATION(l)        =
calllib('Thinkgear','TG_GetValue',connectionId1,TG_DATA_MEDITATION  );
%           disp('MEDITATION = ');
%           disp(data_MEDITATION(l));
%           if(Drive_mode == 1)
             if(data_ATTENTION(k) > 50)
                 fopen(serialOne);
                 fwrite(serialOne,'B');              %CLOSE
                 fclose(serialOne);
             %else if(data_ATTENTION(k) < 40)
             %    fopen(serialOne);
```

```
                  %  fwrite(serialOne,'D');
                    %fclose(serialOne);
                    %end
                end
 %             end
          plot(X,data_ATTENTION,'-r',X,data_BLINK,'-*k');
            axis([0 100 0 200])
            drawnow;
%         end
        end
        end
    end
   end
%disconnect
calllib('Thinkgear', 'TG_FreeConnection', connectionId1 );
```

CONCLUSION

Our project has been completed successfully by the co-operation of batch members. by doing this project, we have obtained valuable knowledge and we had also met some difficulties which would be overcome by sharing the idea with each, which makes our project successful one. our project "**BRAIN MACHINE INTERFACE FOR WRIST MOVEMENT USING ROBOTIC ARM"**is very useful in future life

BIBLIOGRAPHY

[1] A. Massimo, “In Memoriam Pierre Gloor (1923–2003): an appreciation”, Epilepsia, 45(7), 882, July 2004.

[2] A. M. Grass, and F. A. Gibbs, “A Fourier transform of the electroencephalogram”, J. Neurophysiol., 1, 521–526, 1938.

[3] S.Sanei and J. Chambers, “EEG Signal Processing,” John Wiley & Sons,Ltd, 2007, pp 4-13.

[4] G. Pfurtscheller, and R. Rupp, “EEG-based neuroprosthesis control: a step towards clinical practice”, Neuroscience letters, vol.382, no.1,pp.169–174, 2005.

[5] G. Pfurtscheller, G.R. Muller, J. Pfurtscheller, H.J. Gerner, and R. Rupp,“ ‘thought’–control of functional electrical stimulation to restore hand grasp in a patient with tetraplegia”, Neuroscience letters, vol.351, no.1,pp.33-36, 2003.

[6] T. Carlson, L. Tonin, S. Perdikis, R. Leeb, and J. R. Mill´an, “A Hybrid BCI for Enhanced Control of a Telepresence Robot”, Proceedings 35th Annual International Conference of the IEEE EMBS, Osaka, Japan, July 2013.

[7] X. Jiang, T. Cao, F. Wan, P. U. Mak, P. Mak, and M. I. Vai, “Implementation of SSVEP based BCI with Emotiv EPOC,” Proceeding IEEE International Conference on Virtual Environments,Human-Computer Interfaces and Measurement Systems, Tianjin, China, pp. 34-37, July 2012.

[8] P. Sharma, “Automatic detection of non-convulsive seizures,” M. Techdissertation, Dept. Electronics Engineering, Aligarh Muslim Univ.,Aligarh, India, 2012.

[9] Thasneem Fathima, M. Bedeeuzzaman, Omar Farooq and Yusuf U Khan,“Wavelet Based Features for Epileptic Seizure Detection”, MES Journal of Technology and Management, pp. 108-112, May 2011.