Chapter 1

INTRODUCTION

1.1 Introduction

In today's world, communication plays a vital role in each one's life. Vocal communication helps in speeding up information transfer and understanding the information in much better way. People who have voice are considered lucky since this precious God given gift is bestowed on them. What about those who are not so lucky to be blessed with a voice? How would this particular person communicate with the practical world? What would be the difficulties that they must be facing during communication?

Consider a scenario in which a person who cannot speak is waiting for bus at the bus stop and wants to ask about the bus timings. How would he communicate in sign language with another person who does not understand sign language? This sparked off the idea behind our project: to design a portable language translator that would convert this sign language into an understandable language (English). In this project, we have a glove and a portable display which will be worn by the handicapped person who cannot speak. On this glove, we mount flex sensors that sense the bends in the fingers of that person. Proportional to the amount of bend, the resistance of the flex sensor increases.

The flex sensors are connected to the microcontroller ADC input port. This change in resistance causes a change in voltage across it. The analog voltage is then converted into digital form by the ADC present in the microcontroller. Here the digital values are stored for future pattern matching. When a sign is made, pattern matching is done and if the probability of match is high, it is taken as that character.

This is then displayed on the portable LCD screen. Since LCD is portable it means, it is light in weight and can be carried anywhere. It can be used as a badge also. Our project is dedicated to help humanity and those who are not endowed with the power of speech. The portable feature of the sign translator enables it to be used anywhere. This project serves as an interface between sign language and normal spoken English language and makes the life of these unfortunate people a lot better and simpler!

1.2 Purpose of the project

The application aims to serve as a sign language translator, by translating the American Standard Sign Language to plain text (English) which is displayed on a portable LCD. The product trains itself for use, when used for the first time. Then for successive usage, the unit responds to the learned finger patterns and prints the character corresponding to the sign done. This project serves as an interface between two different worlds i.e. people who use sign language and normal spoken English language and makes the life of these unfortunate people a lot better and simpler!

1.3 Problem Definition

- The user needs a handy portable device which would convert sign language to text and help him/her communicate with a normal person efficiently.
- This unit has to be flexible enough to adjust to different people's hand sizes and finger bends.
- The unit must be portable and should be able to be used on the go.

1.3.1 Existing System

The existing product in the market is unfeasible one. This system is mainly equipped with a camera and computer. Here it uses the "matlab" technology for detection of the expression which is used to symbolize a word or character. It is based on American Sign Language (ASL) as well as British Sign language (BSL). Since the system is equipped with a camera and computer, it becomes quite tedious and difficult to carry the whole system around. Hence it is not portable. Since it uses the "matlab" technology, the processing time required is quite high. Hence it is inefficient to use.

1.3.2 Proposed System

This system is designed to help people who cannot speak, to communicate with normal people who can. The product aims at accurate conversion of sign language to text to know what the user wants to say. This will be displayed on a portable LCD screen that will be worn by the user. Hence the other person reads this text and responds correctly to it. In addition, this unit must be flexible enough to adjust to many finger bends and hence must learn the finger bends of the user.

1.4 Scope of the project

This project aims at helping humankind; especially those who cannot speak. This portable translator makes it a handy device to be used on the go. Using this device enables clear communication between people and helps the person who doesn't know sign language to understand its meaning and reciprocate in a better way.

The unit is designed to learn the various bends of the fingers through the flex sensors, and store them in a small database. When the user wears the glove for the first time, he actually makes the 'signs' and the unit learns the various contours and bend angles. This enables the unit to be more flexible and adjust to different people. This device can also be used as an aid to enhance the learning of sign language for students and elders alike. Those who do not know the sign language can practice on this unit and check and learn various signs.

1.5 Report Organization

The current introductory section provides an overall picture of what has been completed till date and the brief introduction about each chapter.

Chapter 1: Introduction

This section focuses on the purpose and the scope of project 'Portable Sign Language Translator'. It also highlights the limitations of the existing system with regards to proposed system.

Chapter 2: Literature Survey

This section focuses on the research carried out to understand the problem definition and the solution domain that is proposed.

Chapter 3: Software Requirement Specification

This section provides us the information about specific requirements of the proposed system.

Chapter 4: System Design

This section describes the design aspects of the product, the development model followed and the architecture used etc.

Chapter 5: Conclusion

This section deals with the completion of the modules till date.

Chapter 2

LITERATURE SURVEY

2. 1 Literature Survey

To get an in depth idea about project numerous datasheets were studied concerning the hardware necessary for the project. Apart from these various sites were visited which provided with some valuable information about the field. Various e-books on the subject also came in handy for learning how to use the AVR Studio 4 for embedded C coding. In this section we summarize some of the datasheets, sites and e-books that influenced the design and flow of the project.

The material also proved to be useful in the selection of design strategy to implement the various modules involved in this application. In addition to this we also visited schools like Xavier Academy- School for the Differently Abled situated at Old Goa, to find out more on the sign language and to get a firsthand experience of how the sign language is taught in the schools, what languages are followed, the syllabus and methods of teaching etc.

This proved very informative and invaluable as our target users were those who are actually students in the school! The information was sought first hand from the Principal of the school! We also got a chance to interact with the students of the school. This gave us a better understanding of what to design and how to go about designing it. The datasheets we read gave an in-depth knowledge of the hardware surrounding our project. The e-book written by Sepehr Naimi (BIHE) on how to use AVR Studio was very informative and helped us in learning a lot about embedded C and most of the commands and functions of the AVR microcontroller.

2.2 American Sign Language

American Sign Language (ASL) is a complete, complex language that employs signs made with the hands. It is the first language of many deaf North Americans, and one of several communication options available to deaf people. ASL is said to be the fourth most commonly used language in the United States. No one form of sign language is universal. For example, British Sign Language (BSL) differs notably from ASL. Different sign languages are used in different countries or regions.

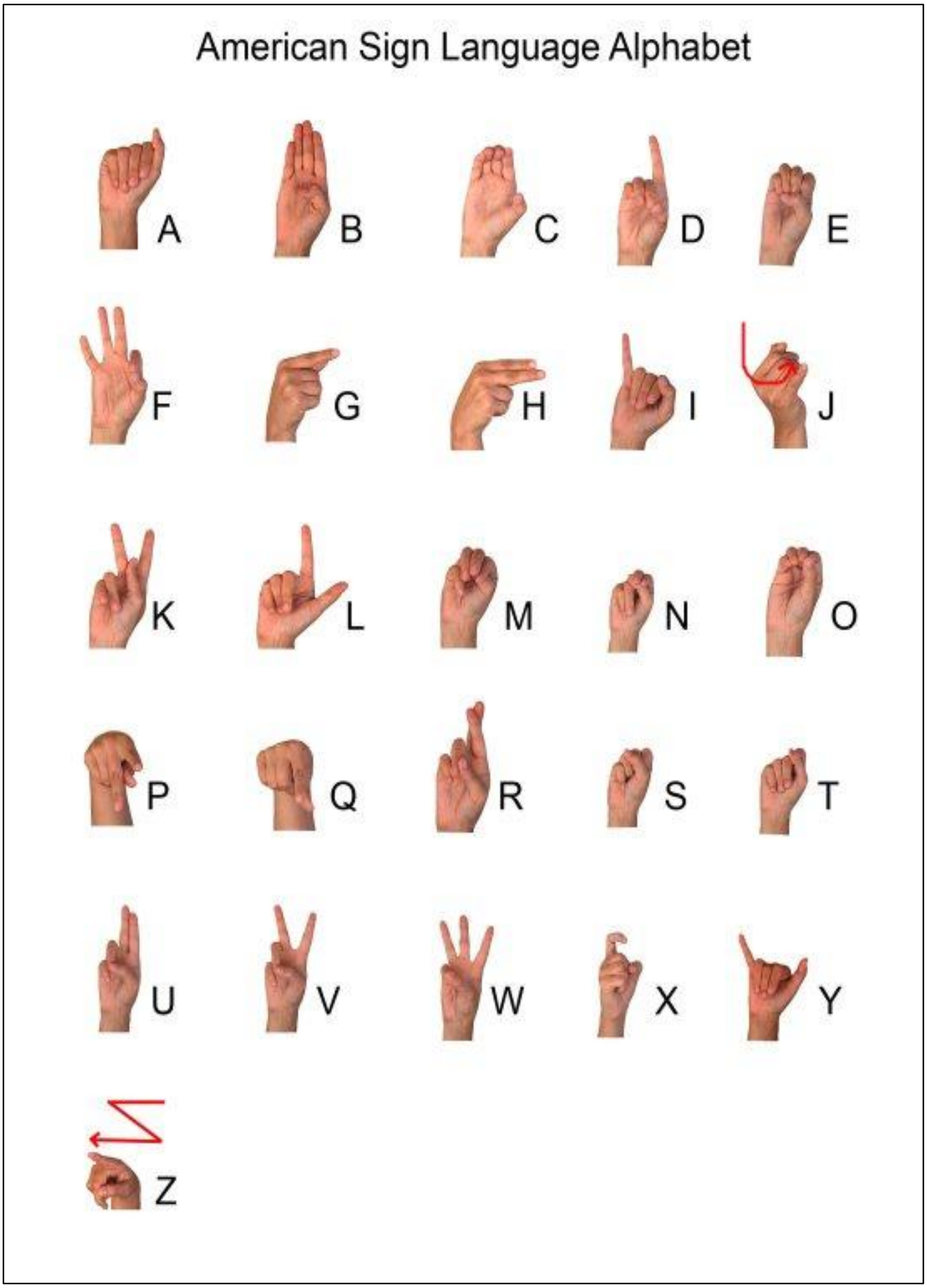


Fig 2.1: American Sign Language Alphabet

The exact beginnings of ASL are not clear. Many people believe that ASL came mostly from French Sign Language (FSL). It was in that year that a French teacher named Laurent Clerc, brought to the United States by Thomas Gallaudet, founded the first school for the deaf in Hartford, Connecticut. Clerc began teaching FSL to Americans, though many of his students were already fluent in their own forms of local, natural sign language. Today’s ASL likely contains some of this early American signing. Which language had more to do with the formation of modern ASL is difficult to prove. Modern ASL and FSL share some elements, including a substantial amount of vocabulary. However, they are not mutually comprehensible.

In spoken language, the different sounds created by words and tones of voice (intonation) are the most important devices used to communicate. Sign language is based on the idea that sight is the most useful tool a deaf person has to communicate and receive information. Thus, ASL uses hand shape, position, and movement. Like any other language, fluency in ASL happens only after a long period of study and practice.

Even though ASL is used in America, it is a language completely separate from English. It contains all the fundamental features a language needs to function on its own. It has its own rules for grammar, punctuation, and sentence order. ASL evolves as its users do, and it also allows for regional usage and jargon. Every language expresses its features differently; ASL is no exception. Just as with other languages, specific ways of expressing ideas in ASL vary as much as ASL users themselves do. ASL users may choose from synonyms to express common words. ASL also changes regionally, just as certain English words are spoken differently in different parts of the country. Ethnicity, age, and gender are a few more factors that affect ASL usage and contribute to its variety.

Parents are often the source of a child’s early acquisition of language. A deaf child who is born to deaf parents who already use ASL will begin to acquire ASL as naturally as a hearing child picks up spoken language from hearing parents. However, language is acquired differently by a deaf child with hearing parents who have no prior experience with ASL. Some hearing parents choose to introduce sign language to their deaf children. Hearing parents who choose to learn sign language often learn it along with their child. Nine out of every ten children who are born deaf are born to parents who hear.

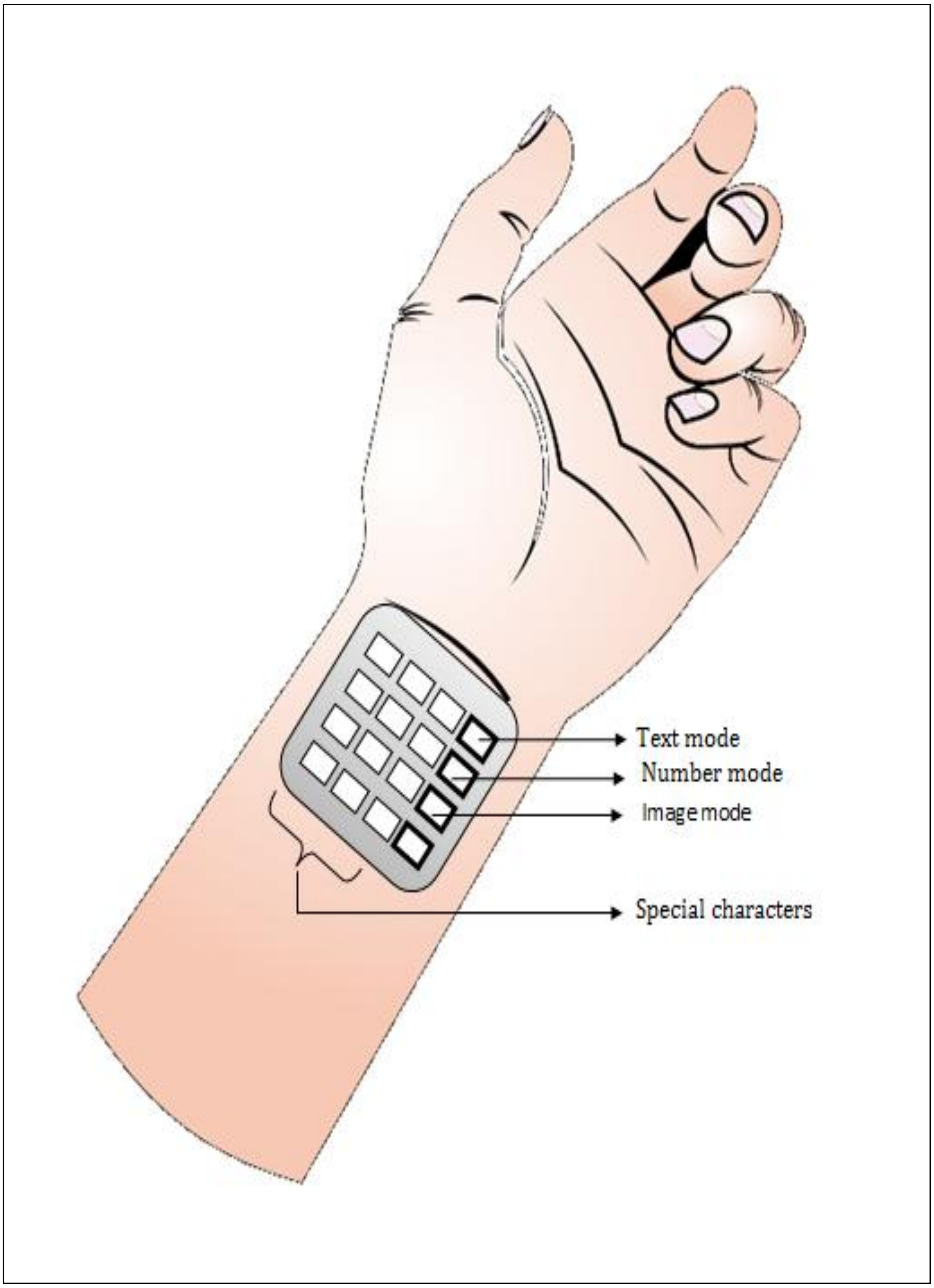


Fig 2.2: Glove Model

Parents should introduce deaf children to language as early as possible. The earlier any child is exposed to and begins to acquire language, the better that child’s communication skills will become. Research suggests that the first six months are the most crucial to a child’s development of language skills. All newborns should be screened for deafness or hearing loss before they leave the hospital or within the first month of life. Very early discovery of a child’s hearing loss or deafness provides parents with an opportunity to learn about communication options. Parents can then start their child’s language learning process during this important stage of development.

Some studies focus on the age of ASL acquisition. Age is a critical issue for people who acquire ASL, whether it is a first or second language. For a person to become fully competent in any language, exposure must begin as early as possible, preferably before school age. Other studies compare the skills of native signers and non-native signers to determine differences in language processing ability. Native signers of ASL consistently display more accomplished sign language ability than non-native signers, again emphasizing the importance of early exposure and acquisition.

Other studies focus on different ASL processing skills. Users of ASL have shown ability to process visual mental images differently than hearing users of English. Though English speakers possess the skills needed to process visual imagery, ASL users demonstrate faster processing ability, suggesting that sign language enhances certain processing functions of the human brain.

You will notice two arrows in this image--one for the letter J and one for the letter Z. For the letter J, you will make the hand shape for the letter I, then trace a J in the air (the way *you* would see it). For the letter Z, you will make a number 1 hand shape and trace the letter Z into the air (the way *you* would see it). C has a hand shape that forms the letter C with your hand. I don't like to turn my C's so that the side of my hand is facing the person I am talking to when I sign them. That can be hard on the wrist after a while.

The letter F can be signed with the three fingers apart (like they are in the image) or together. I prefer apart because it's not too forceful on your hands. Letters M, N, and T are very similar. For M your thumb is under your first three fingers, for N your thumb is under your first two fingers, and for T your thumb is under only your first finger. P has the same hand shape as the letter K, just with your middle finger pointing to the floor. Q has a similar hand shape to the letter G (with your thumb and pointer finger pointed straight) but with your thumb and pointer finger pointing to the floor.

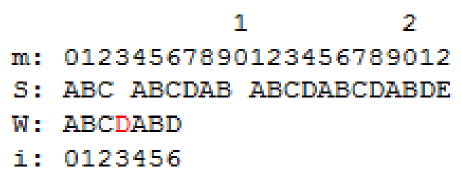
The concept regarding maximum usage of hand area was cleared. This was necessary to provide faster accessibility and easiness to the user during its use. This is prior since user should not find any difficulty in switching from one character to the other. So to do so and get the conceptual idea on track, a rough system model is shown in figure. The model provides a pad with set of buttons on to it. This setup will be installed on the inner side of the hand which will allow user to use it frequently. The provisions for special symbols that are used on regular bases are made available.

2.3 Knuth Morris Pratt (KMP) Algorithm

The Knuth–Morris–Pratt string searching algorithm (or KMP algorithm) searches for occurrences of a "word" W within a main "text string" S by employing the observation that when a mismatch occurs, the word itself embodies sufficient information to determine where the next match could begin, thus bypassing re-examination of previously matched characters. The algorithm was conceived by Donald Knuth and Vaughan Pratt and independently by James H. Morris in 1977, but the three published it jointly.

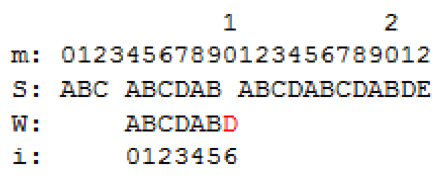
Worked example of the search algorithm

To illustrate the algorithm's details, we work through a (relatively artificial) run of the algorithm. At any given time, the algorithm is in a state determined by two integers, m and i, which denote respectively the position within S which is the beginning of a prospective *match* for W, and the *index* in W denoting the character currently under consideration. This is depicted, at the start of the run, like



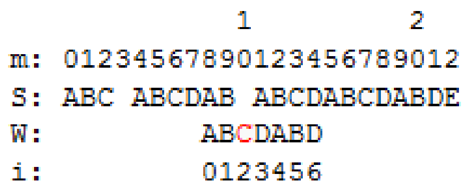
```
             1         2
m: 01234567890123456789012
S: ABC ABCDAB ABCDABCDABDE
W: ABCDABD
i: 0123456
```

We proceed by comparing successive characters of W to "parallel" characters of S, moving from one to the next if they match. However, in the fourth step, we get S[3] is a space and W[3] = 'D', a mismatch. Rather than beginning to search again at S[1], we note that no 'A' occurs between positions 0 and 3 in S except at 0; hence, having checked all those characters previously, we know there is no chance of finding the beginning of a match if we check them again. Therefore we move on to the next character, setting m = 4 and i = 0. (m will first become 3 since m + i - T[i] = 0 + 3 - 0 = 3 and then become 4 since T[0] = -1)



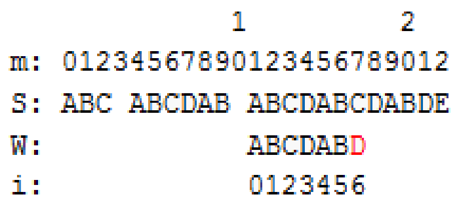
```
              1         2
m: 01234567890123456789012
S: ABC ABCDAB ABCDABCDABDE
W:      ABCDABD
i:      0123456
```

We quickly obtain a nearly complete match "ABCDAB" when, at W[6] (S[10]), we again have a discrepancy. However, just prior to the end of the current partial match, we passed an "AB" which could be the beginning of a new match, so we must take this into consideration. As we already know that these characters match the two characters prior to the current position, we need not check them again; we simply reset m = 8, i = 2 and continue matching the current character. Thus, not only do we omit previously matched characters of S, but also previously matched characters of W.



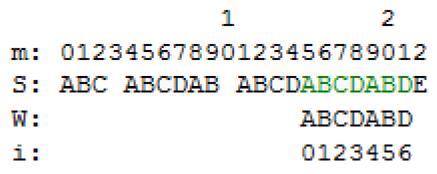
```
              1         2
m: 01234567890123456789012
S: ABC ABCDAB ABCDABCDABDE
W:          ABCDABD
i:          0123456
```

This search fails immediately, however, as the pattern still does not contain a space, so as in the first trial, we return to the beginning of W and begin searching at the next character of S: m = 11, reset i = 0.



```
              1         2
m: 01234567890123456789012
S: ABC ABCDAB ABCDABCDABDE
W:             ABCDABD
i:             0123456
```

Once again we immediately hit upon a match "ABCDAB" but the next character, 'C', does not match the final character 'D' of the word W. Reasoning as before, we set m = 15, to start at the two-character string "AB" leading up to the current position, set i = 2, and continue matching from the current position.



```
             1         2
m: 01234567890123456789012
S: ABC ABCDAB ABCDABCDABDE
W:                ABCDABD
i:                0123456
```

This time we are able to complete the match, whose first character is S[15].

Chapter 3

SOFTWARE REQUIREMENT SPECIFICATION

3.1 Introduction

3.1.1 Purpose

Our project is dedicated to help humanity and those who are not endowed with the power of speech. The portable feature of the sign translator enables it to be used anywhere. This project serves as an interface between sign language and normal spoken English language and makes the life of these unfortunate people a lot better and simpler. With machine learning, the unit can be trained by the user himself to respond to the user's finger bends!

3.1.2 Document Conventions

This SRS is pretty straightforward, divided up into modules detailing an overall description, the external interface requirements, system features, and other nonfunctional requirements. As this is the final draft, any future modifications of this document would involve adapting the product to changing systems and uses. We hope to have the product evolve to changing times as to ensure continued use and success.

The team members have prepared the overall information in this document to the best of their ability. Once read, it is evident that each section is important to the overall SRS and significant to the project in its own right. As the SRS is divided into number of modules it will be easier for the user to get in depth knowledge at each and every step.

The different modules are as follows:

- Designing of glove and hardware interfacing of flex sensors with microcontroller
- Designing of pattern matching embedded software for the microcontroller
- Designing of data packets, encoding scheme and transmitter module
- Designing of data decoding scheme and receiver module
- Designing of graphics and text functions
- Designing of embedded display driver software and hardware interfacing

3.1.3 Intended Audience and Reading Suggestions

This document is intended for any student, staff who is interested in studying how to use this portable sign language translator as a teaching aid. This project is also suited for people who cannot speak and need a handy source to interpret sign language to communicate with people who can speak.

The SRS goes into detail about what our unit does, how the control passes from one step to other and so on. The information provided here describes certain features, goals, user requirements and information about the unit. The aim of this project is to make an easy-to-use, portable sign language translator for those who cannot speak, that can be used on the go!

3.1.4 Product Scope

This project aims at helping humankind; especially those who cannot speak. This portable translator makes it a handy device to be used on the go. Using this device enables clear communication between people and helps the person who doesn’t know sign language to understand its meaning and reciprocate in a better way.

The unit is designed to learn the various bends of the fingers through the flex sensors, and store them in a small database. When the user wears the glove for the first time, he actually makes the ‘signs’ and the unit learns the various contours and bend angles. This enables the unit to be more flexible and adjust to different people. This device can also be used as an aid to enhance the learning of sign language for students and elders alike. Those who do not know the sign language can practice on this unit and check and learn various signs.

3.2 Overall Description

3.2.1 Product Perspective

This project designed in embedded C for the AVR microcontroller, is used to convert sign language to text, for those who cannot speak. The product consists of a glove mounted with flex sensors and a portable graphic LCD display for displaying text. When the sign corresponding to the ASL (American Standard Sign Language), is made for the first time, the microcontroller trains itself by storing the values corresponding to the user (each user will have different hand size and bend angles of the fingers). As the user makes different signs, pattern matching is done simultaneously. If the probability of match with a particular character is high, the output character is displayed on the LCD.

3.2.2 Product Features

The Portable Sign Language Translator is used for intelligent conversion of sign language (ie A.S.L.) into plain text that is displayed on the screen. By this, a person who has no knowledge of sign language can communicate with a person who can't speak.

System Feature 1:Self-Learning

i) Embedded database creation:

C is standardized language. The C programming language is perhaps the most popular for programming embedded systems. It also allows the most precise control of input and output. Here we have used the EEPROM memory as a database or 'hard disk' for storing the sampled values at real-time. The user wears the glove and makes signs corresponding to a letter. As the signs are made, the digital values after conversion are mapped in the EEPROM in a 2-dimentional fashion corresponding to each character.

ii) Pattern Matching Algorithms

Pattern Matching is the act of checking some sequence of tokens for the presence of the constituents of some pattern. In our case of pattern recognition, the match usually need not has to be exact, it can be partial also.

System Feature 2: Easy, Fast and Efficient translation

i) Description

Portable sign language translator provides an ease in the flow of communication and behaves as an interface between sign language and Standard English language. Here user (who cannot speak) just have to wear a glove and provide input to the device by simply doing signs. Sampling for digital conversion is done at a frequency of 125 KHz. This is fast enough to cope with the speed of the person making the sign.

ii) Response

The portable display device is used by the user (who cannot speak) and can be worn as a badge or wear it around the neck as an identity card. This provides the user with an easy interface to communicate. The algorithm works by pattern matching a real-time stored sample with an actual real-time sample taken at that instant. This makes the conversion more accurate and efficient. Also, we implement the 'multiple sampling technique', wherein many samples are taken and pattern matched for a 'probabilistic pattern match' with the stored sample.

Other features are:

- Character displayed vividly on the display.
- With machine learning, the user can train the unit to adapt to his/her own hand.
- Faster and better character recognition by using multiple sampling.
- Portability: can be used anywhere
- Provision for other language support
- Provision for image support
- Battery operated and safe to use

3.2.3 User Classes and Characteristics

Some of the users identified for this system will include:

- **Teachers**
  This product can be used as a teaching aid by teachers who teach sign language in schools.
- **Students**
  Since this product converts sign language to text, it can be used by students who are just learning sign language, to check if they are doing the signs correctly.
- **General people who cannot speak**
  This product can be used as a sign language to text converter for those who cannot speak. Knowing the sign language and wanting to communicate with a normal person, this product acts as an interface between them. This project provides the user with the

means to communicate efficiently. This project converts sign language into text in Real time. As the user does the sign, characters are printed on the LCD screen. The user has to wear a glove and train the unit first, suiting to his/hand. Then the unit can be simply used to translate the signs into characters.

3.2.4 User Documentation

There will be a brief set of instructions for how to train the unit and a user manual in general. With all of this information available to the user, there should be no difficulty in using this unit or troubleshooting it if any problems arise.

3.2.5 Assumptions and Dependencies

The project is mainly divided into six modules:

1) Designing of glove and hardware interfacing of flex sensors with microcontroller
2) Designing of pattern matching embedded software for the microcontroller
3) Designing of data packets, encoding scheme and transmitter module
4) Designing of data decoding scheme and receiver module
5) Designing of graphics and text functions
6) Designing of embedded display driver software and hardware interfacing

Each module has functionality and is dependent on each other.

3.3 Specification Requirement

3.3.1 Hardware requirement

This project is designed in embedded C for the AVR series of microcontrollers with a minimum of 4 ports. Here we use the ATmega 32 and the ATmega 328. The ATmega 32 is used as the display driver interface for the graphic LCD and the ATmega 328 is used as the interface for reading the inputs form the flex sensors. The ADC input of the microcontroller ATmega 328 is used to read the analog values form the flex sensors and convert them into digital values. The graphic LCD driver and embedded graphic library is compatible with any KS0108B based (128 x 64 pixels) LCD driver and has support for various graphic functions like rectangle, lines, circle etc. it also has functions like text and bitmap image display. This library is specially designed for AVR microcontrollers. The various hardware components that will be used and perform respective functions in the system are described below.

3.3.1.1 Flex Sensors

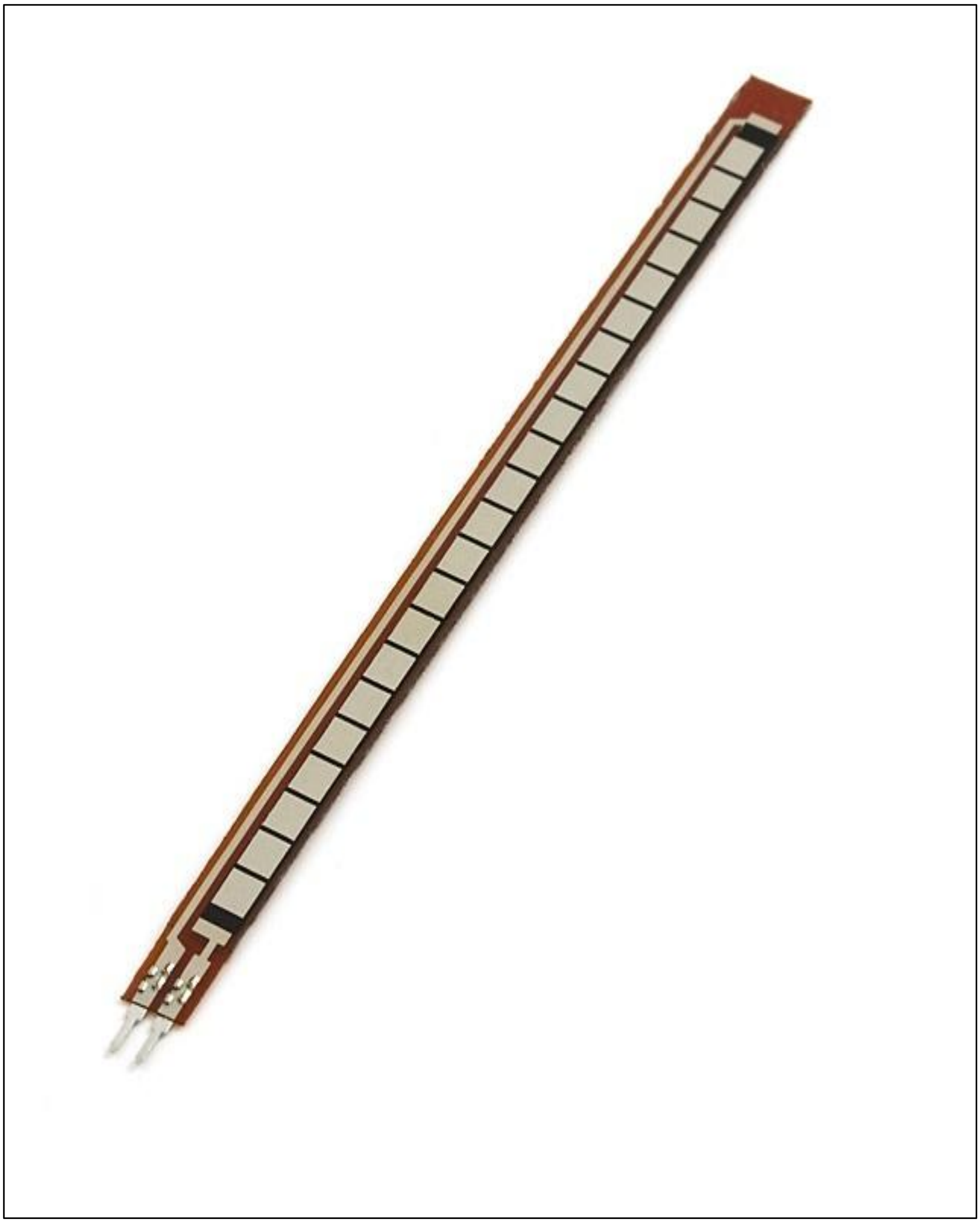


Fig 3.1: Flex sensor

Flex sensors are sensors that change in resistance depending on the amount of bends in the sensors. They convert the change in bend to electrical resistance. More the bend, more the resistance value. They are usually in the form of thin strip from 1”-5” long that vary in resistance. They can be made unidirectional or bidirectional. They can be of various sizes and range lies between 1Ω-20kΩ, 20kΩ-50kΩ and 50kΩ-200kΩ.

Flex sensors are analog resistors. They work as variable analog voltage dividers. Inside the flex sensors are the carbon resistive elements within a thin flexible substrate. More carbon means less resistance. When the substrate is bend the sensor produces a resistance output relative to the bend radius. Smaller the radius, higher the resistance value.

Features:

- Angle displacement measurement.
- Bends and flexes physically with motion device.
- Simple construction.
- Low profile.

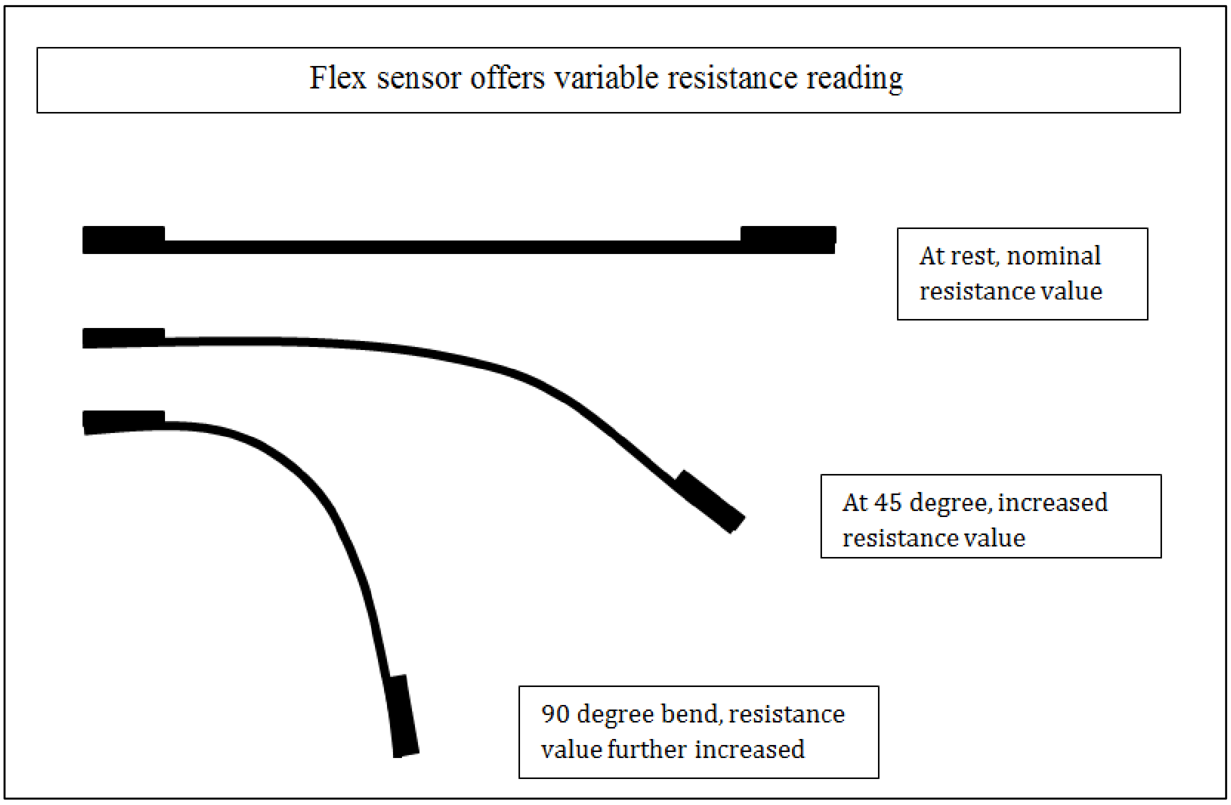

Flex sensor offers variable resistance reading
At rest, nominal resistance value
At 45 degree, increased resistance value
90 degree bend, resistance value further increased

Fig 3.2: Mechanism of flex sensor

Mechanical specification:

- Life cycle:> 1 million
- Height:<0.43mm(0.017")
- Temperature range: -35 to +80 degrees

Electrical specification:

- Flat resistance: 25kΩ
- Resistance tolerance: -30% to +30%
- Bend resistance range: 45kΩ to 125kΩ (depending on bend radius)
- Power rating: 0.50 watts continuous. 1watt peak

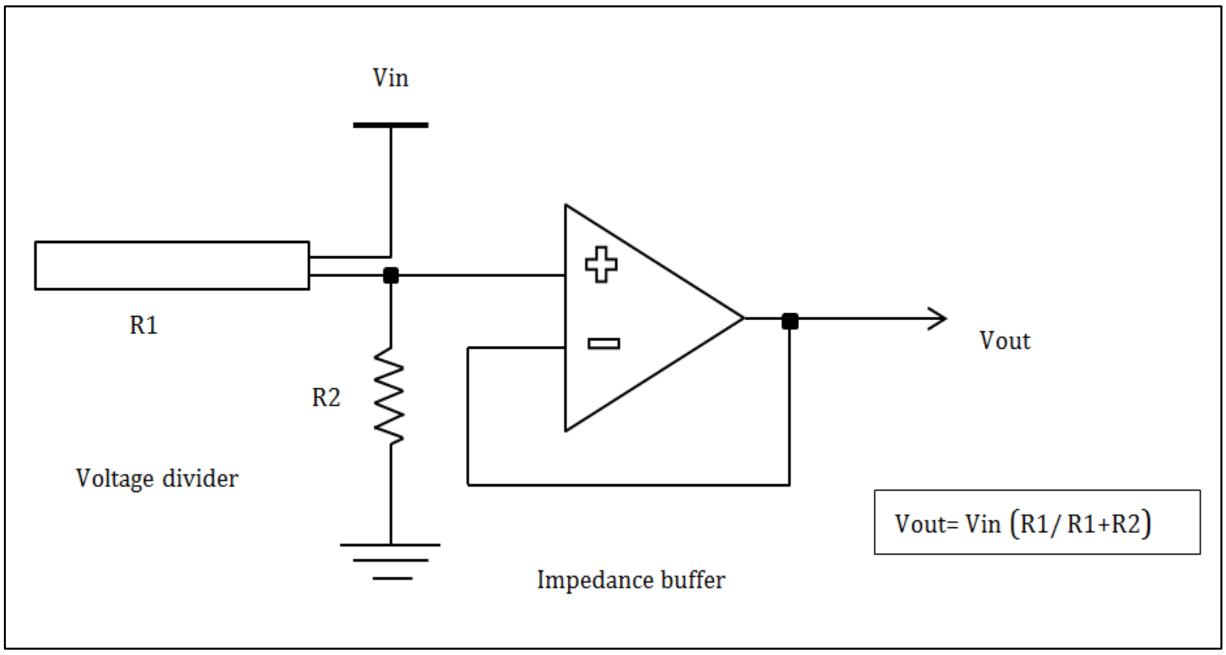


Fig 3.3 Circuit for flex sensor

3.3.1.2 Microcontroller Atmega 32

Features

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

• Advanced RISC Architecture

– 131 Powerful Instructions – Most Single-clock Cycle Execution

– 32 × 8 General Purpose Working Registers

– Fully Static Operation

– Up to 16 MIPS Throughput at 16MHz

– On-chip 2-cycle Multiplier

• High Endurance Non-volatile Memory segments

– 32Kbytes of In-System Self-programmable Flash program memory

– 1024Bytes EEPROM

– 2Kbytes Internal SRAM

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

– Data retention: 20 years at 85°C/100 years at 25°C(1)

– Optional Boot Code Section with Independent Lock Bits

In-System Programming by On-chip Boot Program

True Read-While-Write Operation

– Programming Lock for Software Security

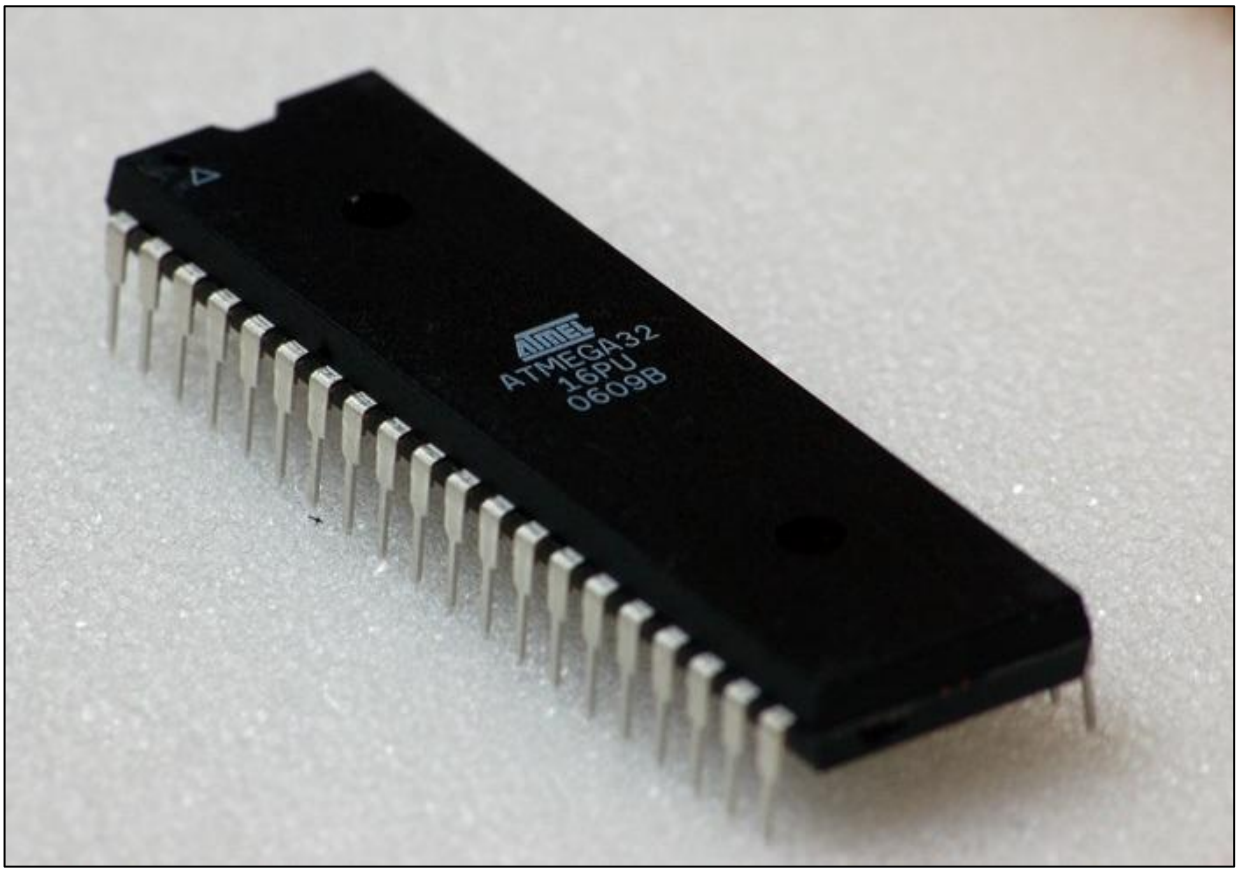


Fig 3.4 Microcontroller Atmega 32

• JTAG (IEEE std. 1149.1 Compliant) Interface
  – Boundary-scan Capabilities According to the JTAG Standard
  – Extensive On-chip Debug Support
  – Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface

• Peripheral Features
  – Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  – One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture

• Mode
  – Real Time Counter with Separate Oscillator
  – Four PWM Channels
  – 8-channel, 10-bit ADC
    8 Single-ended Channels
    7 Differential Channels in TQFP Package Only
    2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  – Byte-oriented Two-wire Serial Interface
  – Programmable Serial USART
  – Master/Slave SPI Serial Interface
  – Programmable Watchdog Timer with Separate On-chip Oscillator
  – On-chip Analog Comparator

• Special Microcontroller Features
  – Power-on Reset and Programmable Brown-out Detection
  – Internal Calibrated RC Oscillator
  – External and Internal Interrupt Sources
  – Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby

• I/O and Packages
  – 32 Programmable I/O Lines
  – 40-pin

• Operating Voltages
  - 2.7V - 5.5V for ATmega32L
  - 4.5V - 5.5V for ATmega32

• Speed Grades
  - 0 - 8MHz for ATmega32L
  - 0 - 16MHz for ATmega32

• Power Consumption at 1MHz, 3V, 25°C
  - Active: 1.1mA
  - Idle Mode: 0.35mA
  - Power-down Mode: < 1Ma

The Atmel®AVR®AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega32 provides the following features: 32Kbytes of In-System Programmable Flash Program memory with Read-While-Write capabilities, 1024bytes EEPROM, 2Kbyte SRAM, 32 general purpose I/O lines, 32 general purpose working registers, a JTAG interface for Boundary scan, On-chip Debugging support and programming, three flexible Timer/Counters with compare modes, Internal and External Interrupts, a serial programmable USART, a byte oriented Two-wire Serial Interface, an 8-channel, 10-bit ADC with optional differential input stage with programmable gain (TQFP package only), a programmable Watchdog Timer with Internal Oscillator, an SPI serial port, and six software selectable power saving modes.

The Idle mode stops the CPU while allowing the USART, Two-wire interface, A/D Converter, SRAM, Timer/Counters, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next External Interrupt or Hardware Reset. In Power-save mode, the

Asynchronous Timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping.

The ADC Noise Reduction mode stops the CPU and all I/O modules except Asynchronous Timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low-power consumption. In Extended Standby mode, both the main Oscillator and the Asynchronous Timer continue to run.

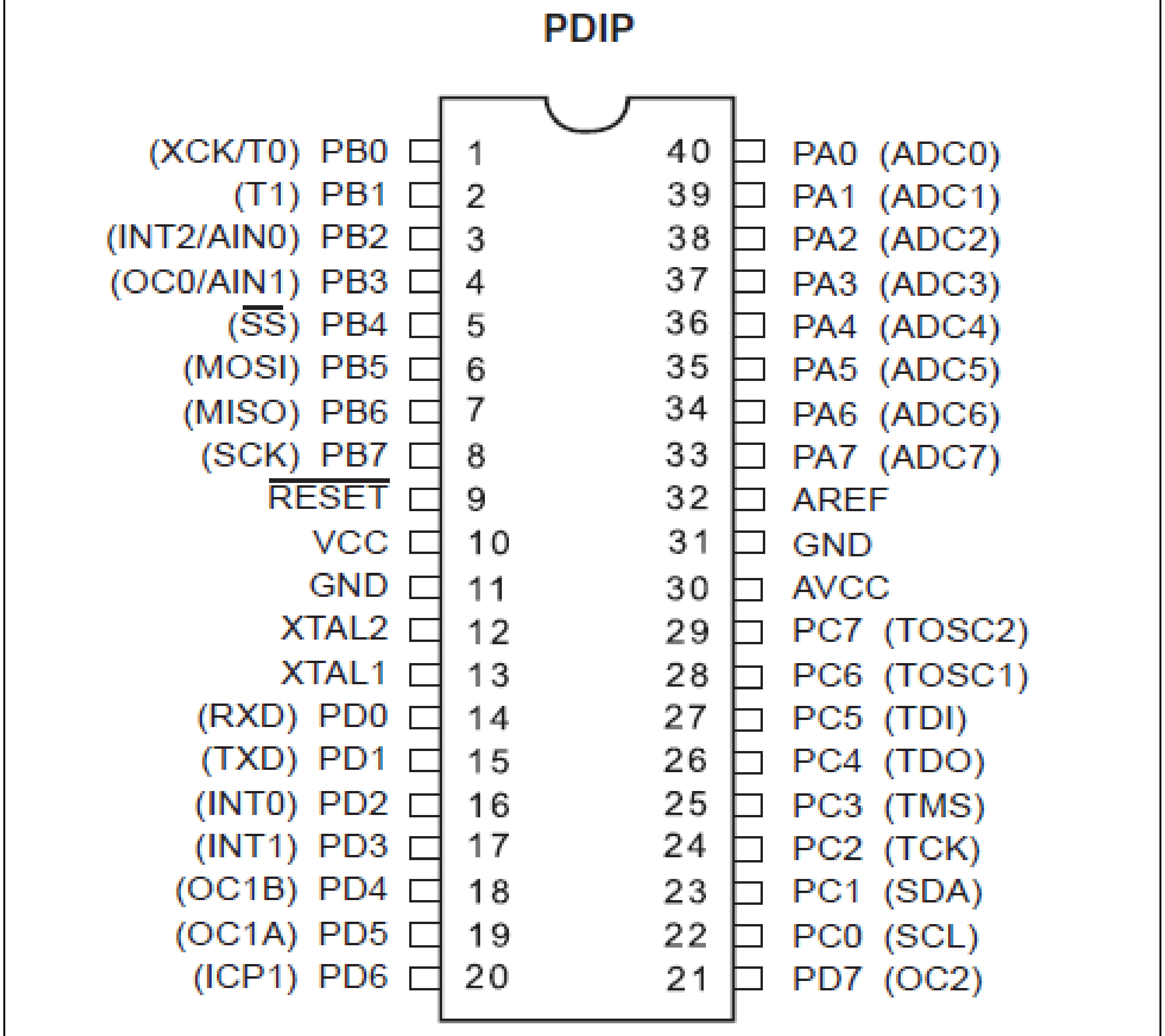


Fig 3.5: Pin outs of Microcontroller Atmega 32

The device is manufactured using Atmel's high density nonvolatile memory technology. The On chip ISP Flash allows the program memory to be reprogrammed in-system through an SPI serial interface, by a conventional nonvolatile memory programmer, or by an On-chip Boot program running on the AVR core. The boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation.

By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega32 is a powerful microcontroller that provides a highly-flexible and cost-effective solution to many embedded control applications. The Atmel AVR ATmega32 is supported with a full suite of program and system development tools including: C compilers, macro assemblers, program debugger/simulators, in-circuit emulators, and evaluation kits.

3.3.1.4 Microcontroller Atmega 328

Features

• High Performance, Low Power AVR® 8-Bit Microcontroller

• Advanced RISC Architecture

– 131 Powerful Instructions – Most Single Clock Cycle Execution

– 32 x 8 General Purpose Working Registers

– Fully Static Operation

– Up to 20 MIPS Throughput at 20 MHz

– On-chip 2-cycle Multiplier

• I/O and Packages

– 23 Programmable I/O Lines

– 28-pin

• High Endurance Non-volatile Memory Segments

– 32K Bytes of In-System Self-Programmable Flash program memory

– 1K Bytes EEPROM

– 2K Bytes Internal SRAM

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

– Data retention: 20 years at 85°C/100 years at 25°C(1)

– Optional Boot Code Section with Independent Lock Bits

In-System Programming by On-chip Boot Program

True Read-While-Write Operation

– Programming Lock for Software Security

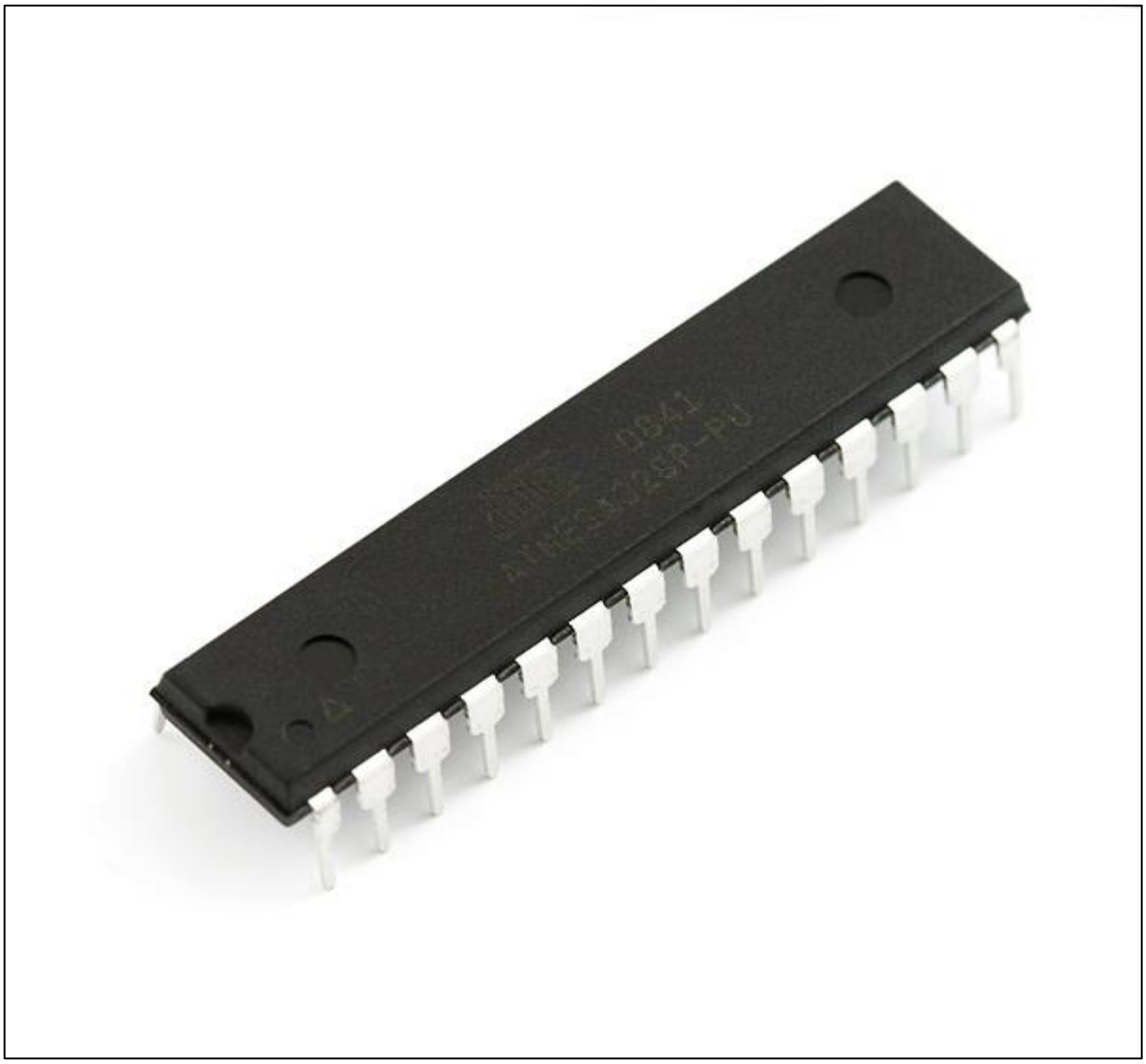


Fig 3.6: Microcontroller Atmega 328

• Peripheral Features

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

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

– Real Time Counter with Separate Oscillator

– Six PWM Channels

– 8-channel 10-bit ADC in TQFP and QFN/MLF package
  Temperature Measurement

– 6-channel 10-bit ADC in PDIP Package
  Temperature Measurement

– Programmable Serial USART

– Master/Slave SPI Serial Interface.

– Byte-oriented 2-wire Serial Interface (Philips I2C compatible)

– Programmable Watchdog Timer with Separate On-chip Oscillator

– On-chip Analog Comparator

– Interrupt and Wake-up on Pin Change

• Temperature Range:

– -40°C to 85°C

• Special Microcontroller Features

– Power-on Reset and Programmable Brown-out Detection

– Internal Calibrated Oscillator

– External and Internal Interrupt Sources

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

• Operating Voltage:

– 1.8 – 5.5V

• Speed Grade:

– 0 – 4 MHz@1.8 – 5.5V, 0 – 10 MHz@2.7 – 5.5.V, 0 – 20 MHz @ 4.5 – 5.5V

• Power Consumption at 1 MHz, 1.8V, 25°C

– Active Mode: 0.2 mA

– Power-down Mode: 0.1 μA

– Power-save Mode: 0.75 μA (Including 32 kHz RTC)

The AVR core combines a rich instruction set with 32 general purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The Atmega328 provides the following features: 32K bytes of In-System Programmable Flash with Read-While-Write capabilities, 1K bytes EEPROM, 2K bytes SRAM, 23 general purpose I/O lines, 32 general purpose working registers, three flexible Timer/Counters with compare modes, internal and external interrupts, a serial programmable USART, a byte-oriented 2-wire Serial Interface, an SPI serial port, a 6-channel 10-bit ADC (8 channels in TQFP and QFN/MLF packages), a programmable Watchdog Timer with internal Oscillator, and five software selectable power saving modes.

The Idle mode stops the CPU while allowing the SRAM, Timer/Counters, USART, 2-wire Serial Interface, SPI port, and interrupt system to continue functioning. The Power-down mode saves the register contents but freezes the Oscillator, disabling all other chip functions until the next interrupt or hardware reset. In Power-save mode, the asynchronous timer continues to run, allowing the user to maintain a timer base while the rest of the device is sleeping.

The ADC Noise Reduction mode stops the CPU and all I/O modules except asynchronous timer and ADC, to minimize switching noise during ADC conversions. In Standby mode, the crystal/resonator Oscillator is running while the rest of the device is sleeping. This allows very fast start-up combined with low power consumption. The device is manufactured using Atmel's high density non-volatile memory technology.

The On-chip ISP Flash allows the program memory to be reprogrammed In-System through an SPI serial interface, by a conventional non-volatile memory programmer, or by an On-chip Boot program running on the AVR core. The Boot program can use any interface to download the application program in the Application Flash memory. Software in the Boot Flash section will continue to run while the Application Flash section is updated, providing true Read-While-Write operation.

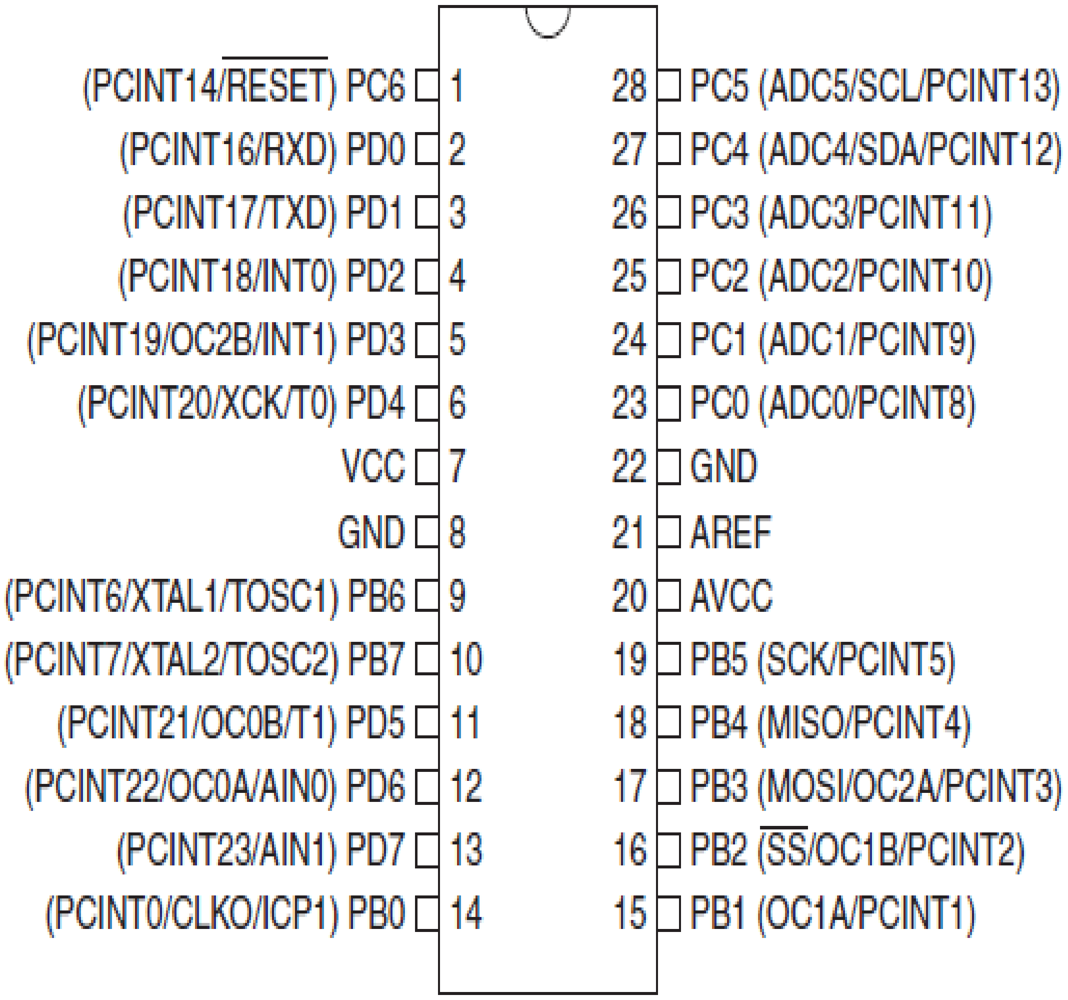


Fig 3.7: Pin outs of microcontroller Atmega 328

By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega328 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications. The ATmega328 AVR is supported with a full suite of program and system development tools including: C Compilers, Macro Assemblers, Program Debugger/Simulators, In-Circuit Emulators, and Evaluation kits.

3.3.1.4 Analog to digital conversion (ADC)

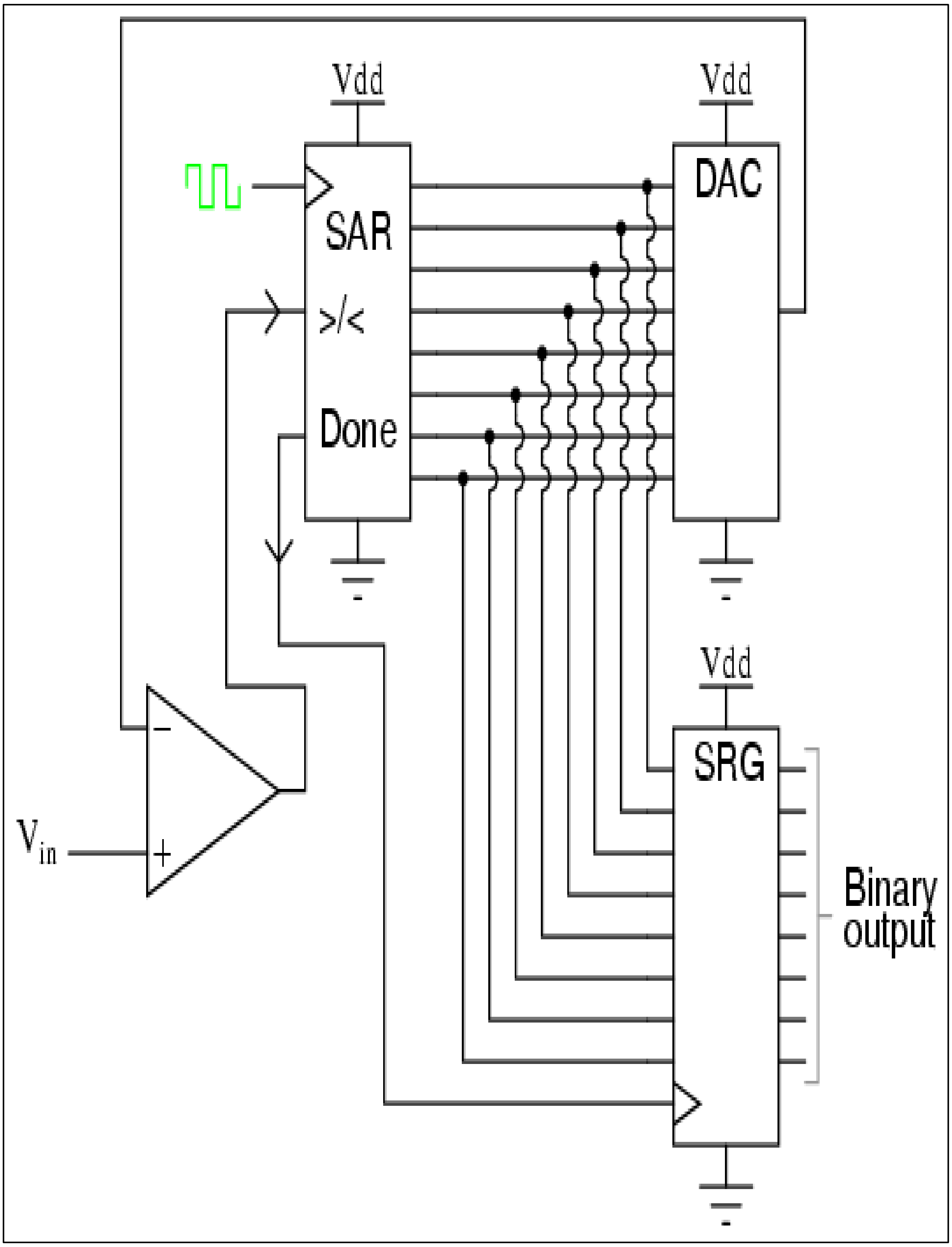


Fig 3.8: Successive approximation ADC

Successive approximation ADC

Features

• 10-bit Resolution
• 0.5 LSB Integral Non-linearity
• ±2 LSB Absolute Accuracy
• 13 µs - 260 µs Conversion Time
• Up to 15 kSPS at Maximum Resolution
• 8 Multiplexed Single Ended Input Channels
• 7 Differential Input Channels
• 2 Differential Input Channels with Optional Gain of 10x and 200x
• Optional Left adjustment for ADC Result Readout
• 0 - VCC ADC Input Voltage Range
• Selectable 2.56V ADC Reference Voltage
• Free Running or Single Conversion Mode
• ADC Start Conversion by Auto Triggering on Interrupt Sources
• Interrupt on ADC Conversion Complete
• Sleep Mode Noise Canceler

The ATmega32 features a 10-bit successive approximation ADC. The ADC is connected to an 8-channel Analog Multiplexer which allows 8 single-ended voltage inputs constructed from the pins of Port A. The single-ended voltage inputs refer to 0V (GND). The device also supports 16 differential voltage input combinations. Two of the differential inputs (ADC1, ADC0 and ADC3, ADC2) are equipped with a programmable gain stage, providing amplification steps of 0 dB (1x), 20 dB (10x), or 46 dB (200x) on the differential input voltage before the A/D conversion. Seven differential analog input channels share a common negative terminal (ADC1), while any other ADC input can be selected as the positive input terminal. If 1x or 10x gain is used, 8-bit resolution can be expected. If 200 x gain is used, 7-bit resolution can be expected.The ADC contains a Sample and Hold circuit which ensures that the input voltage to the ADC is held at a constant level during conversion.

A block diagram of the ADC is shown in Figure 98. The ADC has a separate analog supply voltage pin, AVCC. AVCC must not differ more than ±0.3V from VCC. Internal reference voltages of nominally 2.56V or AVCC are provided On-chip. The voltage reference may be externally decoupled at the AREF pin by a capacitor for better noise performance.

Operation

The ADC converts an analog input voltage to a 10-bit digital value through successive approximation. The minimum value represents GND and the maximum value represents the voltage on the AREF pin minus 1 LSB. Optionally, AVCC or an internal 2.56V reference voltage may be connected to the AREF pin by writing to the REFSn bits in the ADMUX Register. The internal voltage reference may thus be decoupled by an external capacitor at the AREF pin to improve noise immunity. The analog input channel and differential gain are selected by writing to the MUX bits in ADMUX. Any of the ADC input pins, as well as GND and a fixed band gap voltage reference, can be selected as single ended inputs to the ADC. A selection of ADC input pins can be selected as positive and negative inputs to the differential gain amplifier. If differential channels are selected, the differential gain stage amplifies the voltage difference between the selected input channel pair by the selected gain factor.

This amplified value then becomes the analog input to the ADC. If single ended channels are used, the gain amplifier is bypassed altogether.The ADC is enabled by setting the ADC Enable bit, ADEN in ADCSRA. Voltage reference and input channel selections will not go into effect until ADEN is set. The ADC does not consume power when ADEN is cleared, so it is recommended to switch off the ADC before entering power saving sleep modes. The ADC generates a 10-bit result which is presented in the ADC Data Registers, ADCH and ADCL. By default, the result is presented right adjusted, but can optionally be presented left adjusted by setting the ADLAR bit in ADMUX. If the result is left adjusted and no more than 8-bit precision is required, it is sufficient to read ADCH. Otherwise, ADCL must be read first, then ADCH, to ensure that the content of the Data Registers belongs to the same conversion. Once ADCL is read, ADC access to Data Registers is blocked. This means that if ADCL has been read, and a conversion completes before ADCH is read, neither register is updated and the result from the conversion is lost. When ADCH is read, ADC access to the ADCH and ADCL Registers is re-enabled.The ADC has its own interrupt which can be triggered when a conversion completes. When ADC access to the Data Registers is prohibited between reading of ADCH and ADCL, the interrupt will trigger even if the result is lost.

3.3.1.5 Serial Peripheral Interface (SPI)

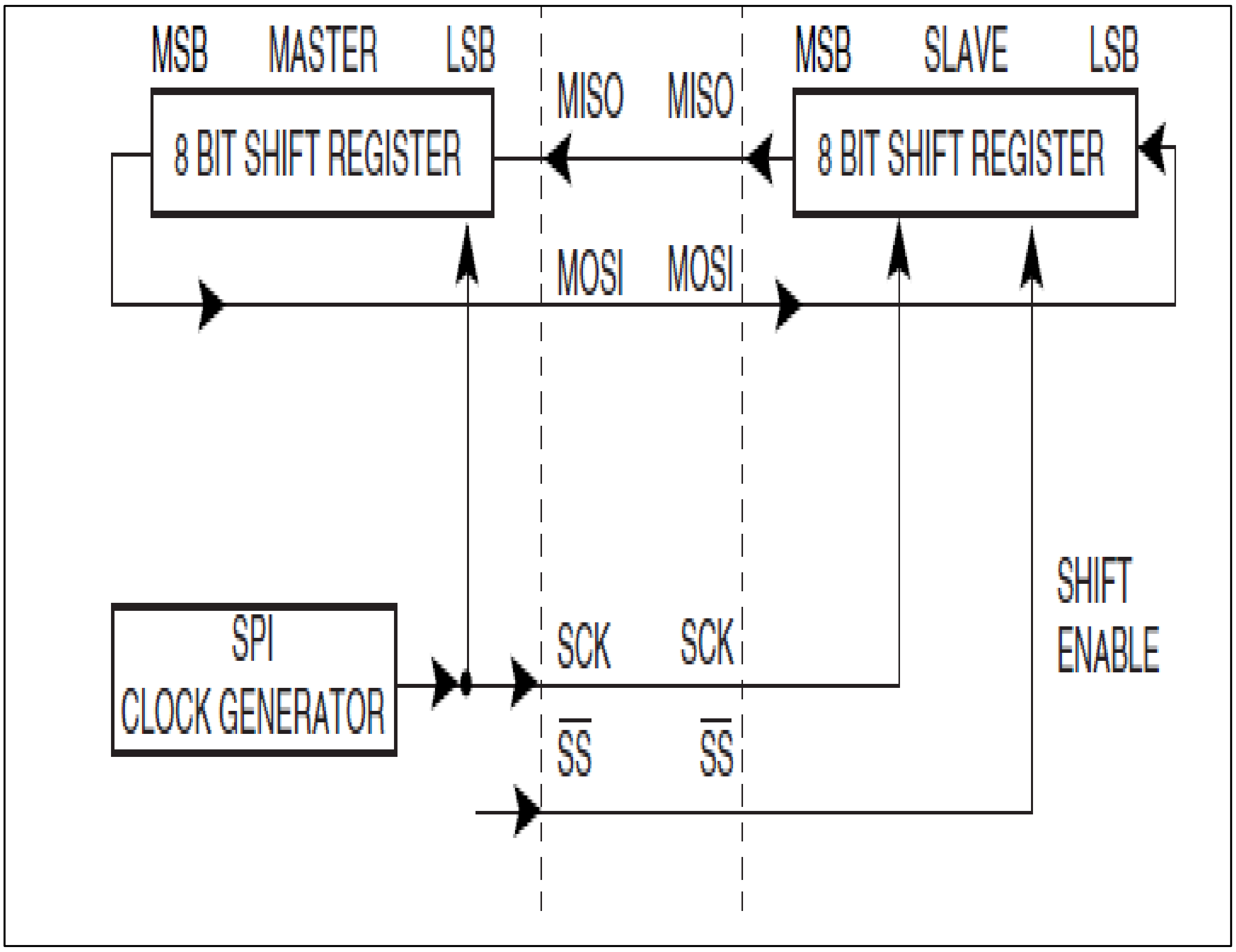


Fig 3.9: Serial Peripheral Interface (SPI)

The Serial Peripheral Interface (SPI) allows high-speed synchronous data transfer between the ATmega32 and peripheral devices or between several AVR devices. The ATmega32 SPI includes the following features:

- Full-duplex, Three-wire Synchronous Data Transfer
- Master or Slave Operation
- LSB First or MSB First Data Transfer
- Seven Programmable Bit Rates
- End of Transmission Interrupt Flag
- Write Collision Flag Protection
- Wake-up from Idle Mode
- Double Speed (CK/2) Master SPI Mode

The interconnection between Master and Slave CPUs with SPI is shown in Figure 66. The system consists of two Shift Registers, and a Master clock generator. The SPI Master initiates the communication cycle when pulling low the Slave Select SS pin of the desired Slave. Master and Slave prepare the data to be sent in their respective Shift Registers, and the Master generates the required clock pulses on the SCK line to interchange data. Data is always shifted from Master to Slave on the Master Out – Slave In, MOSI, line, and from Slave to Master on the Master In– Slave Out, MISO, line

. After each data packet, the Master will synchronize the Slave by pulling high the Slave Select, SS, line. When configured as a Master, the SPI interface has no automatic control of the SS line. This must be handled by user software before communication can start. When this is done, writing byte to the SPI Data Register starts the SPI clock generator, and the hardware shifts the eight bits into the Slave. After shifting one byte, the SPI clock generator stops, setting the end of Transmission Flag (SPIF). If the SPI Interrupt Enable bit (SPIE) in the SPCR Register is set, an interrupt is requested. The Master may continue to shift the next byte by writing it into SPDR, or signal the end of packet by pulling high the Slave Select, SS line. The last incoming byte will be kept in the Buffer Register for later use.

When configured as a Slave, the SPI interface will remain sleeping with MISO tri-stated as long as the SS pin is driven high. In this state, software may update the contents of the SPI Data Register, SPDR, but the data will not be shifted out by incoming clock pulses on the SCK pin until the SS pin is driven low. As one byte has been completely shifted, the end of Transmission Flag, SPIF is set. If the SPI Interrupt Enable bit, SPIE, in the SPCR Register is set, an interrupt is requested. The Slave may continue to place new data to be sent into SPDR before reading the incoming data. The last incoming byte will be kept in the Buffer Register for later use. The system is single buffered in the transmit direction and double buffered in the receive direction. This means that bytes to be transmitted cannot be written to the SPI Data Register before the entire shift cycle is completed.

When receiving data, however, a received character must be read from the SPI Data Register before the next character has been completely shifted in. Otherwise, the first byte is lost. In SPI Slave mode, the control logic will sample the incoming signal of the SCK pin. To ensure correct sampling of the clock signal, the minimum low and high periods should be: Low periods: longer than 2 CPU clock cycles. High periods: longer than 2 CPU clock cycles. Slave Mode When the SPI is configured as a Slave, the Slave Select (SS) pin is always input. When SS is held low, the SPI is activated, and MISO becomes an output if configured so by the user. All other pins are inputs.

When SS is driven high, all pins are inputs except MISO which can be user configured as an output, and the SPI is passive, which means that it will not receive incoming data. Note that the SPI logic will be reset once the SS pin is driven high. The SS pin is useful for packet/byte synchronization to keep the slave bit counter synchronous with the master clock generator. When the SS pin is driven high, the SPI Slave will immediately reset the send and receive logic, and drop any partially received data in the Shift Register.

Master Mode When the SPI is configured as a Master (MSTR in SPCR is set), the user can determine the direction of the SS pin. If SS is configured as an output, the pin is a general output pin which does not affect the SPI system. Typically, the pin will be driving the SS pin of the SPI Slave. If SS is configured as an input, it must be held high to ensure Master SPI operation. If the SS pin is driven low by peripheral circuitry when the SPI is configured as a Master with the SS pin defined as an input, the SPI system interprets this as another master selecting the SPI as a slave and starting to send data to it.

To avoid bus contention, the SPI system takes the following actions:

1. The MSTR bit in SPCR is cleared and the SPI system becomes a slave. As a result of the SPI becoming a slave, the MOSI and SCK pins become inputs.

2. The SPIF Flag in SPSR is set, and if the SPI interrupt is enabled, and the I-bit in SREG is set, the interrupt routine will be executed.

Thus, when interrupt-driven SPI transmission is used in master mode, and there exists a possibility that SS is driven low, the interrupt should always check that the MSTR bit is still set. If the MSTR bit has been cleared by a slave select, it must be set by the user to re-enable SPI master Mode.

3.3.1.6 Transmitter & Receiver module (CC2500)

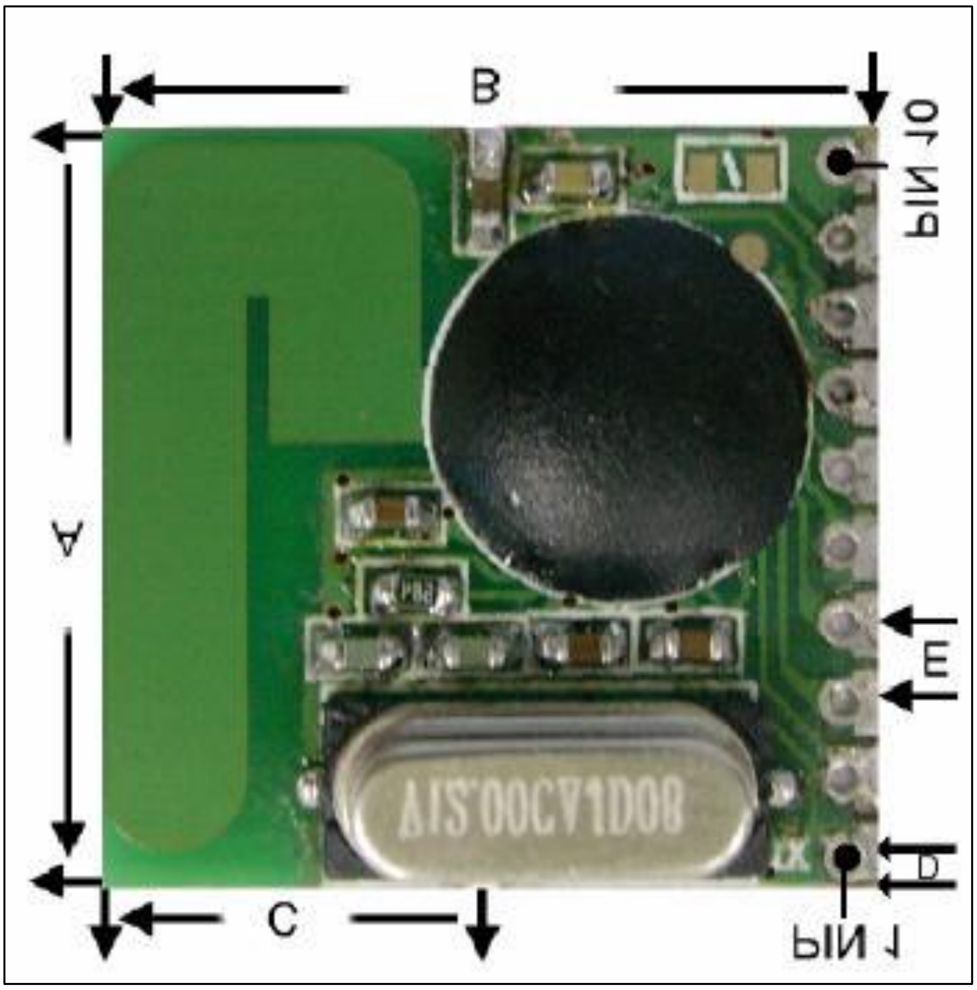


Fig 3.10: CC2500 Transceiver RF module

The CC2500 is a 2.4GHz multi-channel transceiver RF module. It operates with a 2.4 GHz carrier and the data is modulated with FSK (Frequency Shift Keying) modulation. It’s designed specifically for high speed data reception and transmission.

Applications

- Toys
- Wireless Mouse
- Remote Control
- Home and factory automation
- Wireless security and access control
- Battery Powered wireless devices

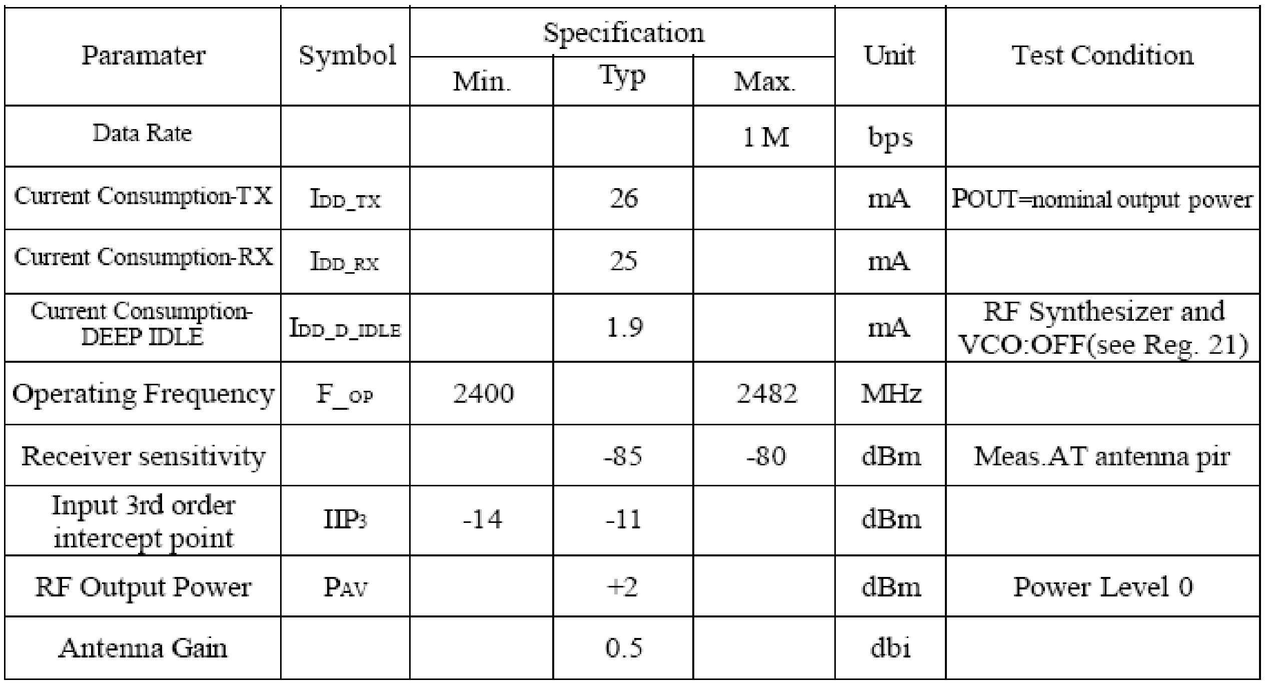
- Wireless Voice / headsets / Skype earphone and VOIP

Absolute Maximum Ratings

Rating	Value	Units
VDD Supply Voltage	2.5 to 3.7	V
Operating temperature	-40 to +85	℃
Input RF Level	+10	dBm
Frequency Hopping	81	channels

Table 3.1 Absolute Maximum Ratings

Characteristics



Paramater	Symbol	Specification			Unit	Test Condition
		Min.	Typ	Max.		
Data Rate				1 M	bps	
Current Consumption-TX	IDD_TX		26		mA	POUT=nominal output power
Current Consumption-RX	IDD_RX		25		mA	
Current Consumption-DEEP IDLE	IDD_D_IDLE		1.9		mA	RF Synthesizer and VCO:OFF(see Reg. 21)
Operating Frequency	F_OP	2400		2482	MHz	
Receiver sensitivity			-85	-80	dBm	Meas.AT antenna pir
Input 3rd order intercept point	IIP3	-14	-11		dBm	
RF Output Power	PAV		+2		dBm	Power Level 0
Antenna Gain			0.5		dbi	

Table 3.2 General characteristics

Mechanical Characteristics

CC2500 (H) ：IC → Chip Mechanical Characteristics

CC2500 (W) ：IC → Package

	PIN	Description
PIN 1	+3.3V	
PIN 2	SPI_MISO	SPI data out
PIN 3	RESET_n	Reset input, active low
PIN 4	SPI_CLK	SPI data in
PIN 5	SPI_MOSI	SPI data in
PIN 6	SPI_SS	SPI slave select input
PIN 7	FIFO_FLAG	FIFO full/empty
PIN 8	PKT_FLAG	Packet TX/RX flag
PIN 9	BRCLK	clk. out.(optional)
PIN10	GND	

Table 3.3 Mechanical Characteristics

3.3.1 .7 128 x 64 Dots Graphic LCD (JHD12864E)

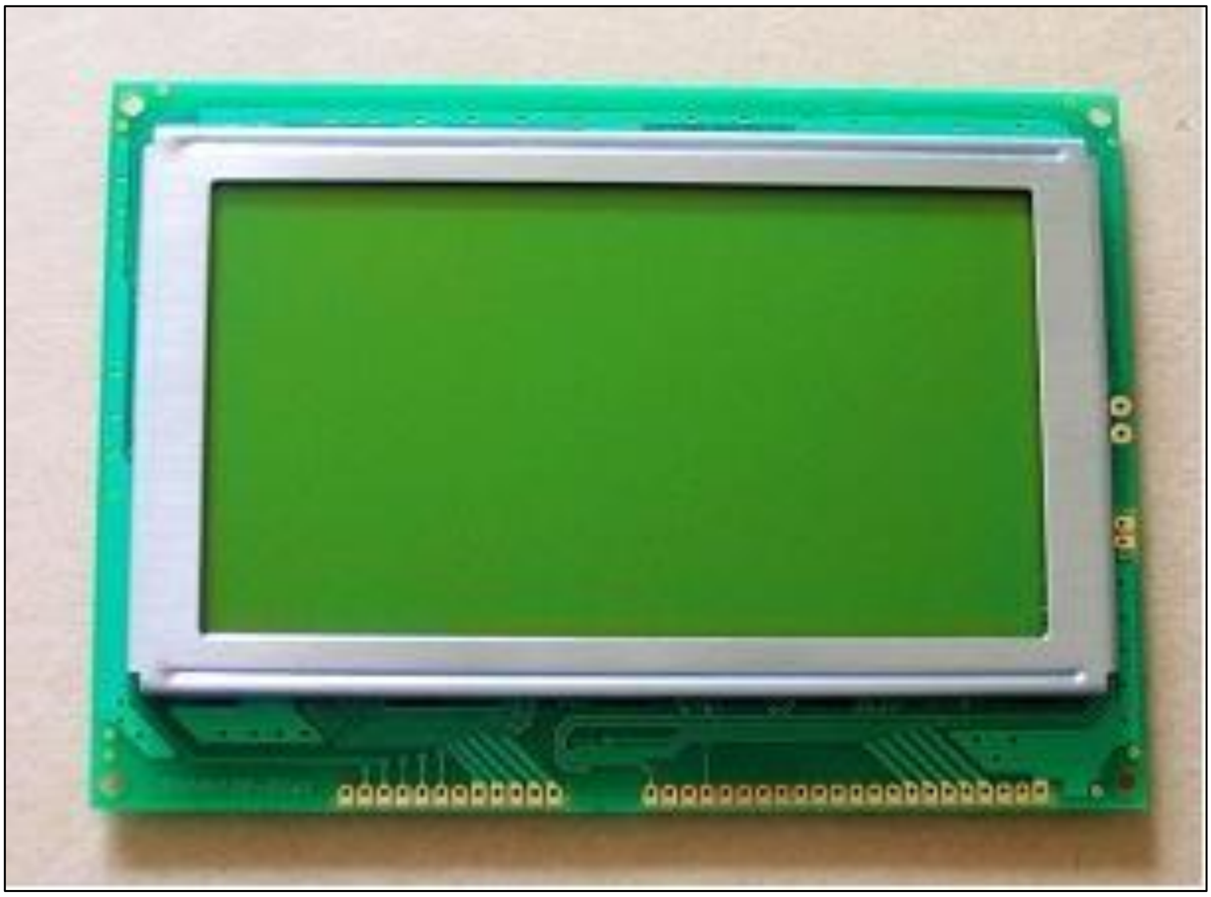


Fig 3.11: 128 x 64 Dots Graphic LCD

Features

• Built-in controller (KS0108 - KS0108 or Equivalent)

• + 5V power supply

• 1/64 duty cycle

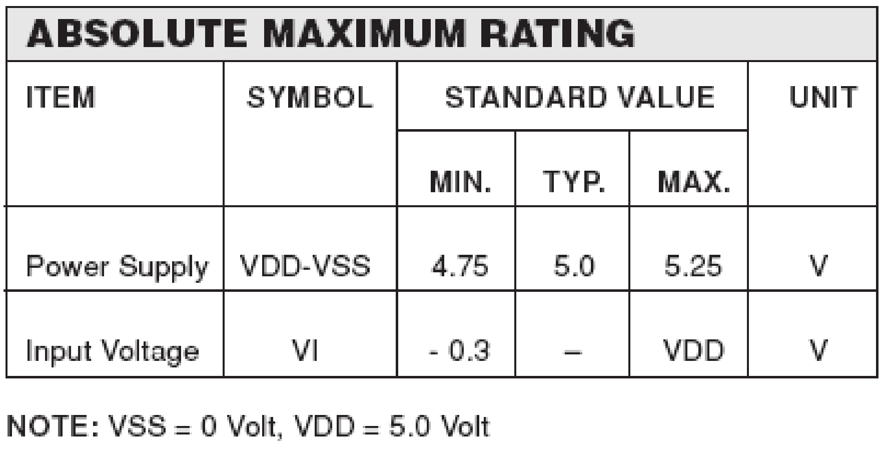
• N.U. option

Mechanical Data

MECHANICAL DATA		
ITEM	**STANDARD VALUE**	**UNIT**
Module Dimension	80.0 x 70.0	mm
Viewing Area	72.0 x 40.0	mm
Dot Size	0.48 x 0.48	mm
Dot Pitch	0.52 x 0.52	mm

Table 3.4 Mechanical Data

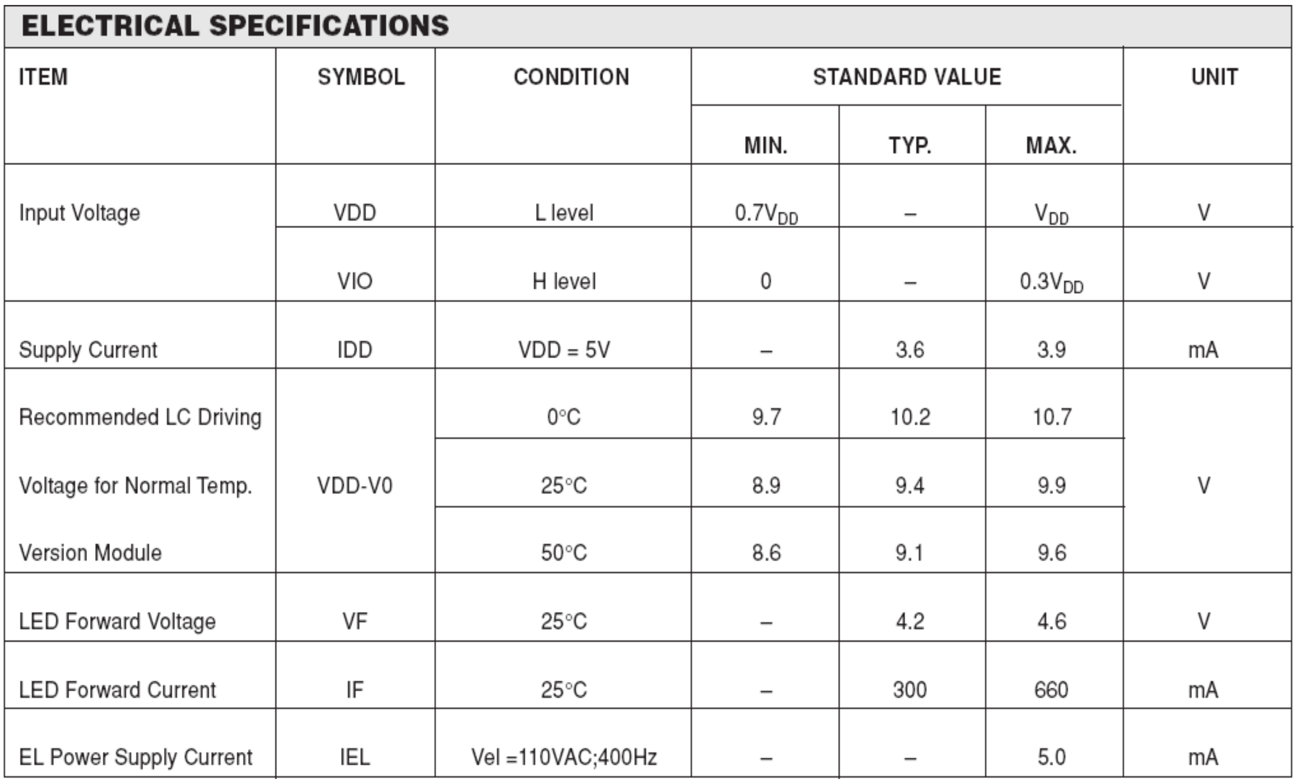
Absolute Maximum Ratings



ABSOLUTE MAXIMUM RATING					
ITEM	SYMBOL	STANDARD VALUE			UNIT
		MIN.	TYP.	MAX.	
Power Supply	VDD-VSS	4.75	5.0	5.25	V
Input Voltage	VI	- 0.3	–	VDD	V

NOTE: VSS = 0 Volt, VDD = 5.0 Volt

Table 3.5 Absolute Maximum Ratings



ELECTRICAL SPECIFICATIONS

ITEM	SYMBOL	CONDITION	STANDARD VALUE			UNIT
			MIN.	TYP.	MAX.	
Input Voltage	VDD	L level	$0.7V_{DD}$	–	V_{DD}	V
	VIO	H level	0	–	$0.3V_{DD}$	V
Supply Current	IDD	VDD = 5V	–	3.6	3.9	mA
Recommended LC Driving Voltage for Normal Temp. Version Module	VDD-V0	0°C	9.7	10.2	10.7	V
		25°C	8.9	9.4	9.9	
		50°C	8.6	9.1	9.6	
LED Forward Voltage	VF	25°C	–	4.2	4.6	V
LED Forward Current	IF	25°C	–	300	660	mA
EL Power Supply Current	IEL	Vel =110VAC;400Hz	–	–	5.0	mA

Table 3.6 Electrical specification

In order to place any information on the screen, it is important to understand how the bits control what is on the screen. There are 8192 pixels on a 128 X 64 pixel screen and each is pixels is control by a series of instructions. The Y address refers to which line the pixels should be written to and the X page sets the column to which they will be written to. There are three basic steps that must be done in order to determine where the pixels will go. First, the Y address must be set. The Y address actually has a counter so it need only be set once and then every time there is a data write it will be incremented to the next line. This allows the driver to scan through the lines and display the proper data. The next step that must be completed is setting the X page. This determines which column of the screen will now be written to. The third step is issuing a data write command. Whatever data bits are high will be darkened on the Y address line in the X page. The driver scans through the pages, using the internal Y address counter to its advantage and resetting the X page (and eventually the CS) as it scans through the lines. One very important thing to note is that the X and Y axes are reversed from what they usually would be. Every 128 x 64 graphical LCD that we found was arranged in this way, even if it was not controlled by the KS0108 so we suspect it is the convention.

PIN NUMBER	SYMBOL	FUNCTION
1	Vss	GND
2	Vdd	Power Supply (+ 5V)
3	Vo	Contrast Adjustment
4	D/L	Data/Instruction
5	R/W	Data Read/Write
6	E	H → L Enable Signal
7	DB0	Data Bus Line
8	DB1	Data Bus Line
9	DB2	Data Bus Line
10	DB3	Data Bus Line
11	DB4	Data Bus Line
12	DB5	Data Bus Line
13	DB6	Data Bus Line
14	DB7	Data Bus Line
15	CS1	Chip Select for IC1
16	CS2	Chip Select for IC2
17	RST	Reset
18	Vee	Negative Voltage Output
19	A	Power Supply for LED (4.2V)
20	K	Power Supply for LED (0V)

Table 3.7 Pin functional description

To reiterate, to select which bit on the screen will be effected:

- Set Y address which chooses one of the 64 vertical rows in the half of the screen that is currently selected by CS1 or CS2.
- Set the X page which selects which of the 8 horizontal stripes the 8 data bits will be placed in.
- Write data to the 8 bits that were just selected using a function such as write data in this driver.

The process of selecting where on the screen the bit is going to go is just a narrowing process. CS1 and CS2 narrow it down to half the screen. Y address narrows it down to a 1 bit wide stripe of height 64 in that half of the screen. X page picks an 8 bit chunk of this 64 bit stripe and write data writes 8 bits to that chunk. This is the essence of putting bits on the screen.

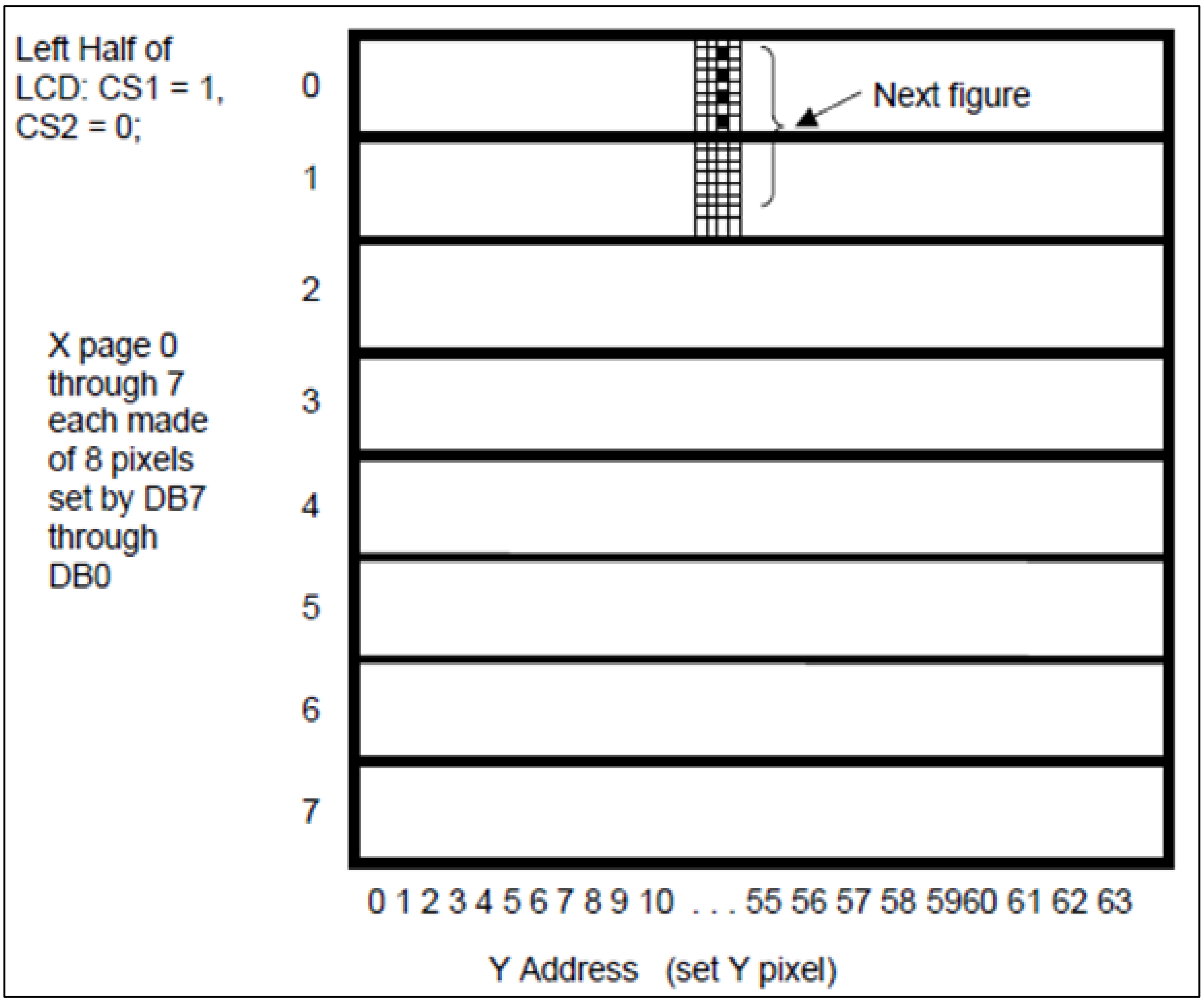

Left Half of LCD: CS1 = 1, CS2 = 0;
Next figure
X page 0 through 7 each made of 8 pixels set by DB7 through DB0
0
1
2
3
4
5
6
7
0 1 2 3 4 5 6 7 8 9 10 . . . 55 56 57 58 5960 61 62 63
Y Address (set Y pixel)

Fig 3.12 Plotting of pixel on LCD screen

3.3.1 .8 Keypad 4x4

Many applications require large number of keys connected to a computing system. Example includes a PC keyboard, Cell Phone keypad and Calculators. If we connect a single key to MCU, we just connect it directly to i/o line. But we cannot connect, say 10 or 100 keys directly MCUs I/O. Because:-

- It will take up precious I/O line.
- MCU to Keypad interface will contain lots of wires.

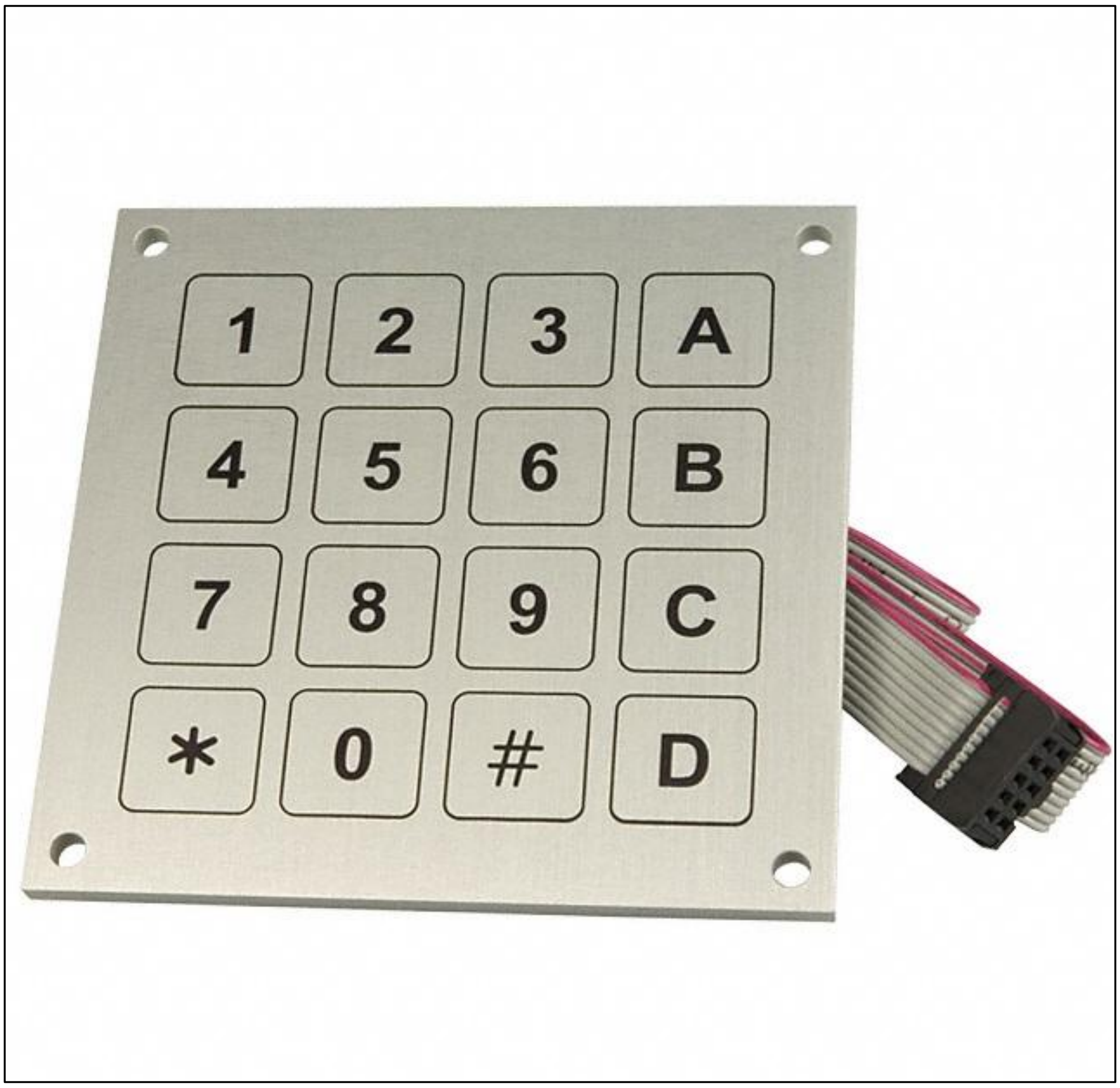


Fig 3.13: Keypad 4x4

We want to avoid all these troubles so we use some clever technique. The technique is called multiplexed matrix keypad. In this technique keys are connected in a matrix (row/column) style as shown below.

The rows R0 to R3 are connected to Input lines of Microcontroller. The I/O pins where they are connected are made Input. This is done by setting the proper DDR Register in AVR. The columns C0 to C3 are also connected to MCUs I/O line. These are kept at High Impedance State (AKA input), in high z state (z= impedance) state these pins are neither HIGH nor LOW they are in TRISTATE. And in their PORT value we set them all as low, so as soon as we change their DDR bit to 1 they become output with value LOW. One by One we make each Column LOW (from high Z state) and read state of R0 to R3.

As you can see in the image above C0 is made LOW while all other Columns are in HIGH Z State. We can read the Value of R0 to R3 to get their pressed status. If they are high the button is NOT pressed. As we have enabled internal pull-ups on them, these pull-ups keep their value high when they are floating (that means NOT connected to anything). But when a key is pressed it is connected to LOW line from the column thus making it LOW.

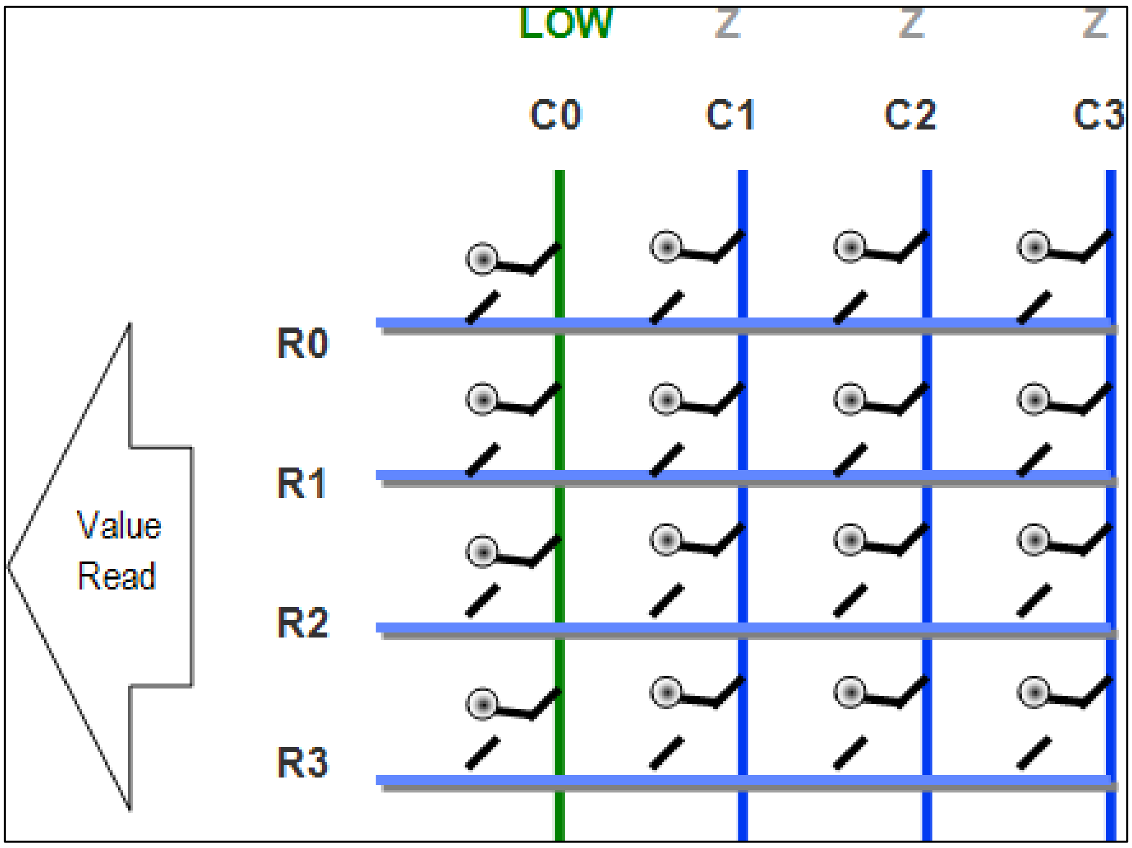


Fig 3.14 Matrix Keypad Basic Connection Column 0 Selected

After that we make the C0 High Z again and make C1 LOW. And read R0 to R3 again. This gives us status of the second column of keys. Similarly we scan all columns.

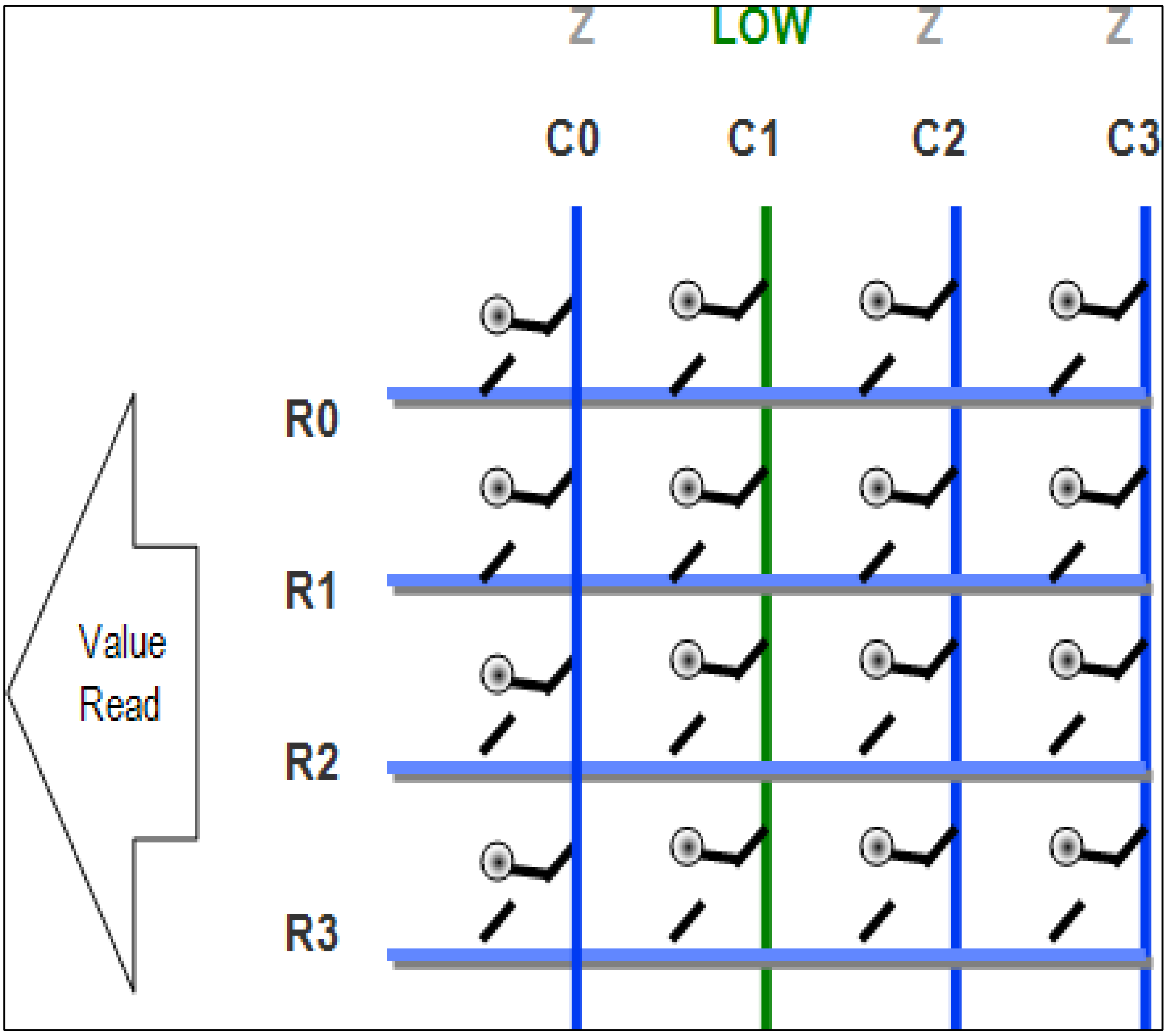


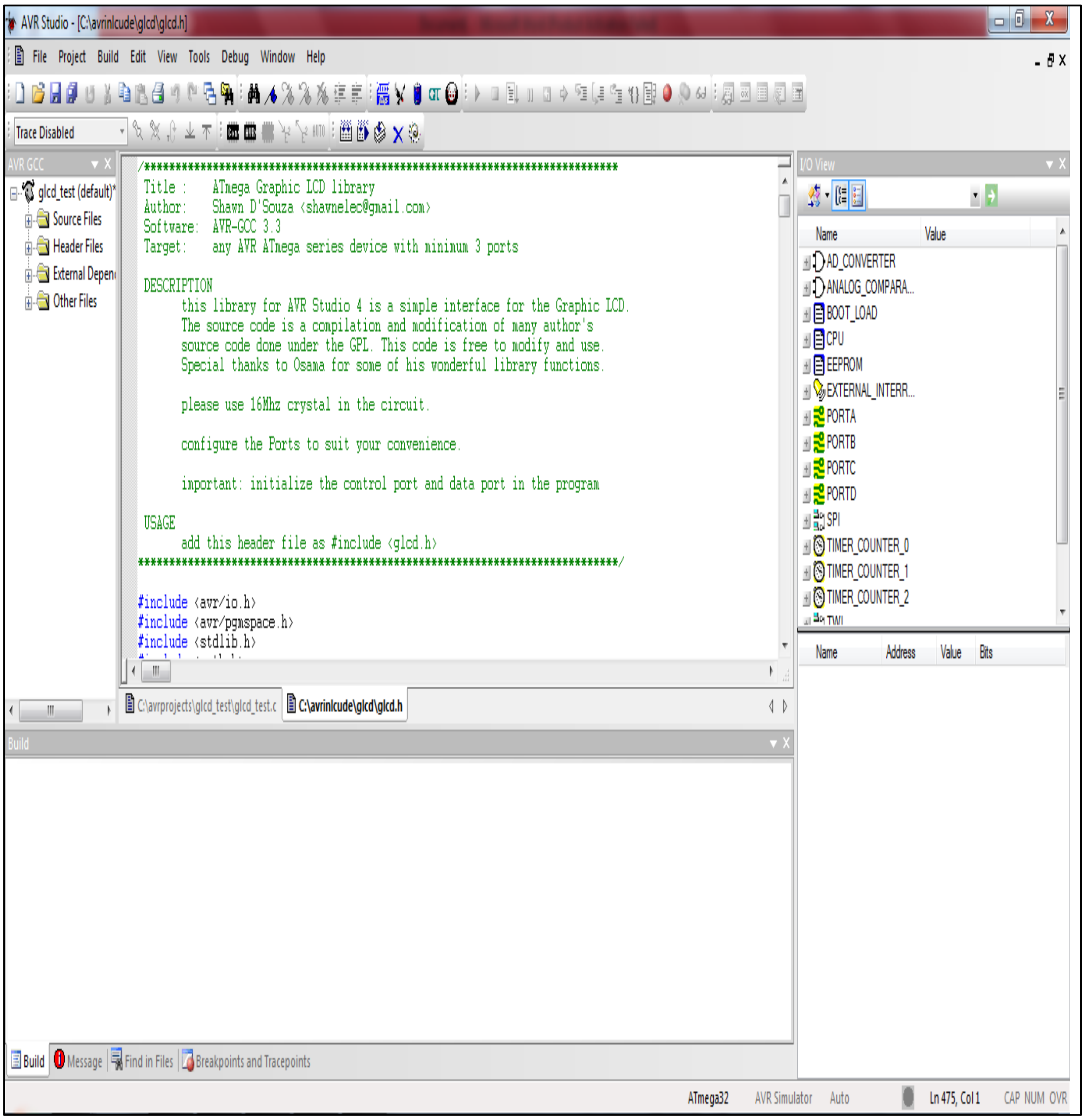
Fig 3.15 Matrix Keypad Basic Connection Column 1 Selected

3.3.2 Software requirement

3.3.2.1 AVR Studio 4

AVR Studio 4 is free software provided by the manufacturers of the AVR microcontrollers, to enable users to program in a user friendly environment for the AVR microcontroller. It features a fully functional GUI with the register view and tools for editing the code. In AVR Studio 4, we can write code in C, Assembly and Pascal. It also features a debugger to check the code for its correctness. The debugger enables the designer to check his/her code step by step.

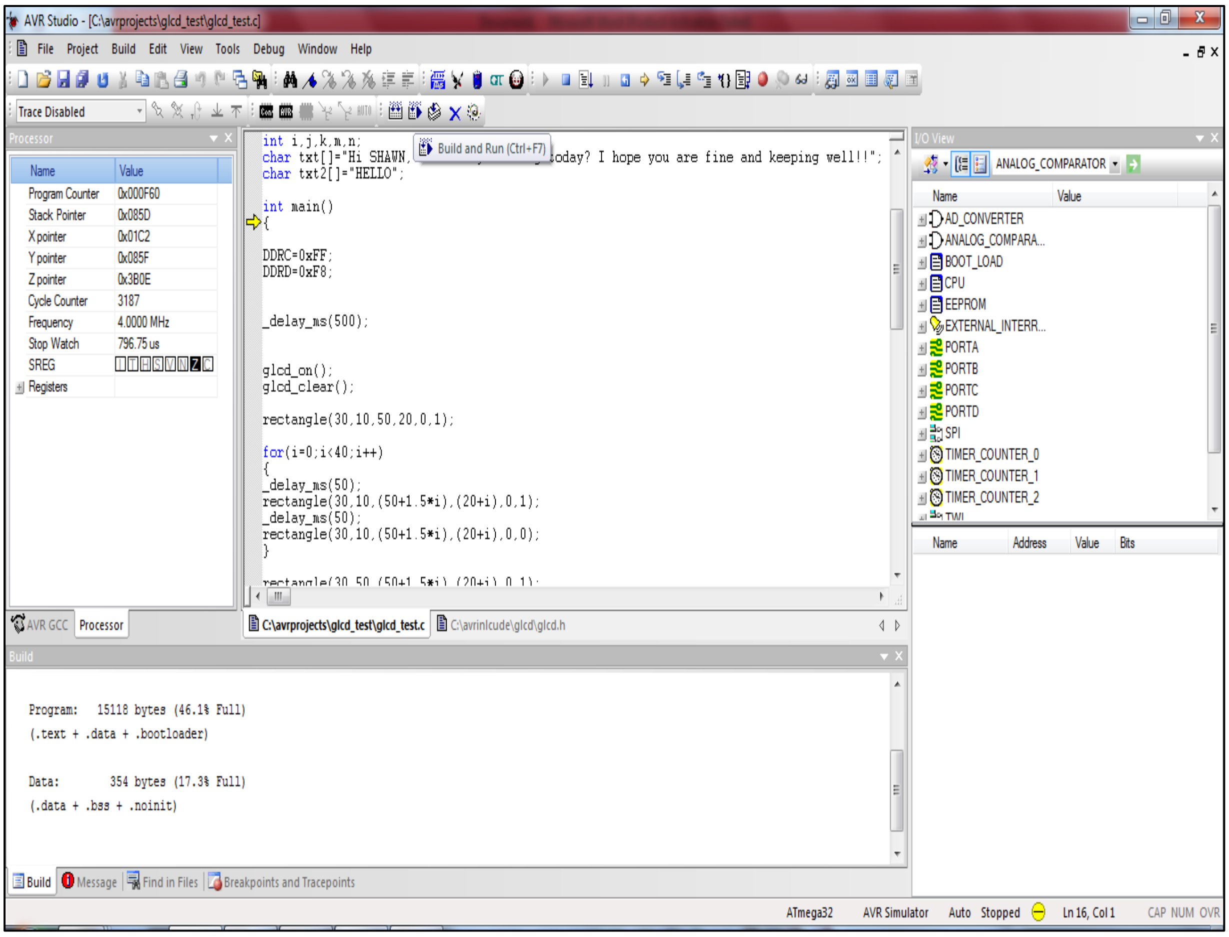
In AVR Studio 4, the user can write his/her own header files to be included in the source program code. The user can then compile and run his code. The AVR simulator shows the register status in a graphical view. When the code is built and run, a hex code is generated for the same. This hex code is stored in a folder called ‘default’. The hex file is then burnt into a microcontroller using a specific burner for the microcontroller.



```
/*****************************************************************************
 Title :    ATmega Graphic LCD library
 Author:    Shawn D'Souza <shawnelec@gmail.com>
 Software:  AVR-GCC 3.3
 Target:    any AVR ATmega series device with minimum 3 ports

 DESCRIPTION
       this library for AVR Studio 4 is a simple interface for the Graphic LCD.
       The source code is a compilation and modification of many author's
       source code done under the GPL. This code is free to modify and use.
       Special thanks to Osama for some of his wonderful library functions.

       please use 16Mhz crystal in the circuit.

       configure the Ports to suit your convenience.

       important: initialize the control port and data port in the program

 USAGE
       add this header file as #include <glcd.h>
*****************************************************************************/

#include <avr/io.h>
#include <avr/pgmspace.h>
#include <stdlib.h>
```

Fig 3.16: Snapshot of AVR Studio 4

AVR Studio 4 features a wide choice of microcontrollers and can support almost all the 8-bit microcontroller range offered by Atmel. This tool serves as a wonderful GUI to code in and to debug your code.



Name	Value
Program Counter	0x000F60
Stack Pointer	0x085D
X pointer	0x01C2
Y pointer	0x085F
Z pointer	0x3B0E
Cycle Counter	3187
Frequency	4.0000 MHz
Stop Watch	796.75 us
SREG	ITHSVNZC
Registers	

```c
int i,j,k,m,n;
char txt[]="Hi SHAWN, [illegible]today? I hope you are fine and keeping well!!";
char txt2[]="HELLO";

int main()
{

DDRC=0xFF;
DDRD=0xF8;


_delay_ms(500);


glcd_on();
glcd_clear();

rectangle(30,10,50,20,0,1);

for(i=0;i<40;i++)
{
_delay_ms(50);
rectangle(30,10,(50+1.5*i),(20+i),0,1);
_delay_ms(50);
rectangle(30,10,(50+1.5*i),(20+i),0,0);
}

rectangle(30,50,(50+1.5*i),(20+i),0,1);
```

```
Program:   15118 bytes (46.1% Full)
(.text + .data + .bootloader)

Data:        354 bytes (17.3% Full)
(.data + .bss + .noinit)
```

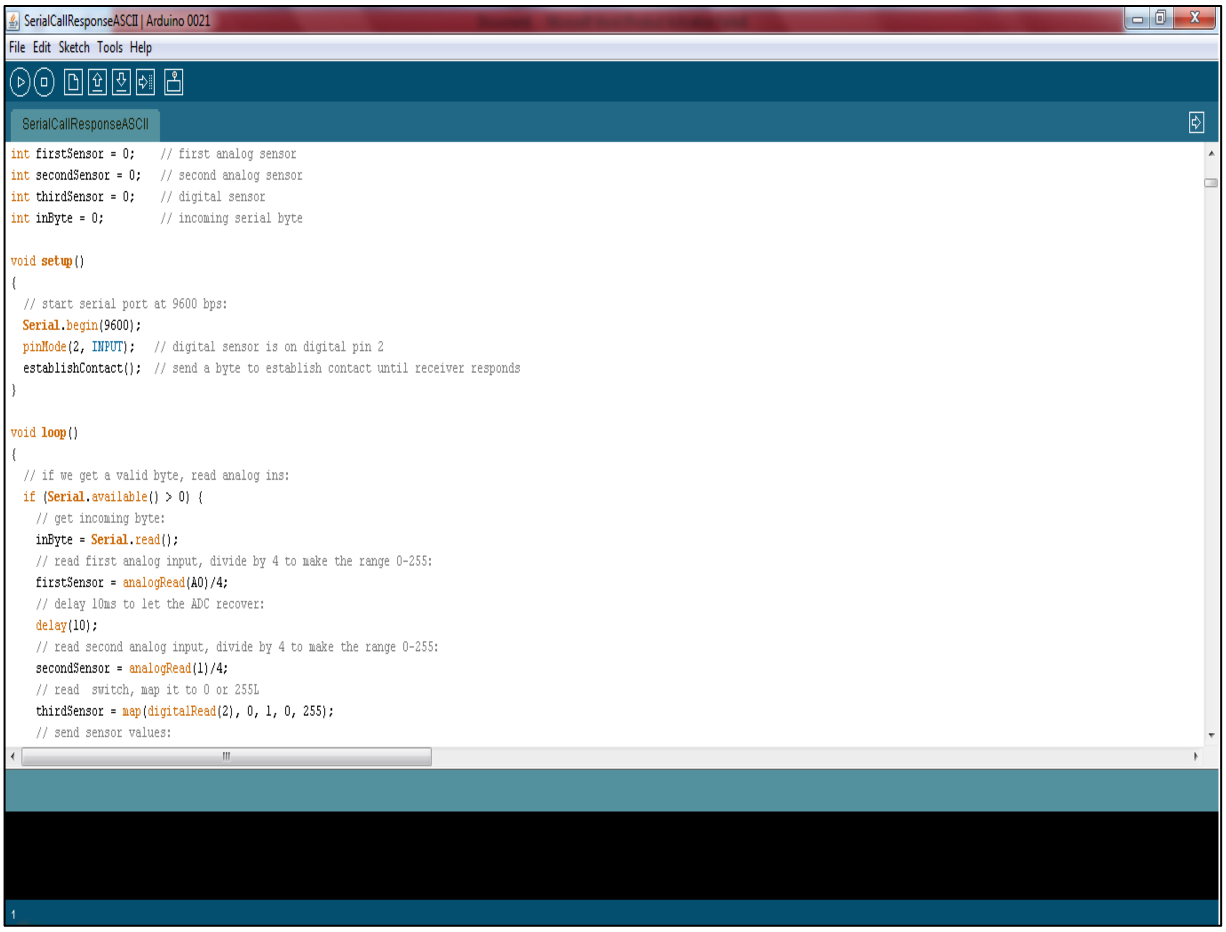
Name	Address	Value	Bits

Fig 3.17: Building and Running code in AVR Studio 4

3.3.2.2 Arduino

Arduino is an open-source single-board microcontroller, descendant of the open-source Wiring platform, designed to make the process of using electronics in multidisciplinary projects more accessible. The hardware consists of a simple open hardware design for the Arduino board with an Atmel AVR processor and on-board I/O support. The software consists of a standard programming language compiler and the boot loader that runs on the board.

Arduino hardware is programmed using a Wiring-based language (syntax + libraries), similar to C++ with some simplifications and modifications, and a Processing-based IDE. Currently shipping versions can be purchased pre-assembled; hardware design information is available for those who would like to assemble an Arduino by hand. Additionally, variations of the Italian-made Arduino—with varying levels of compatibility—have been released by third parties; some of them are programmed using the arduino software.

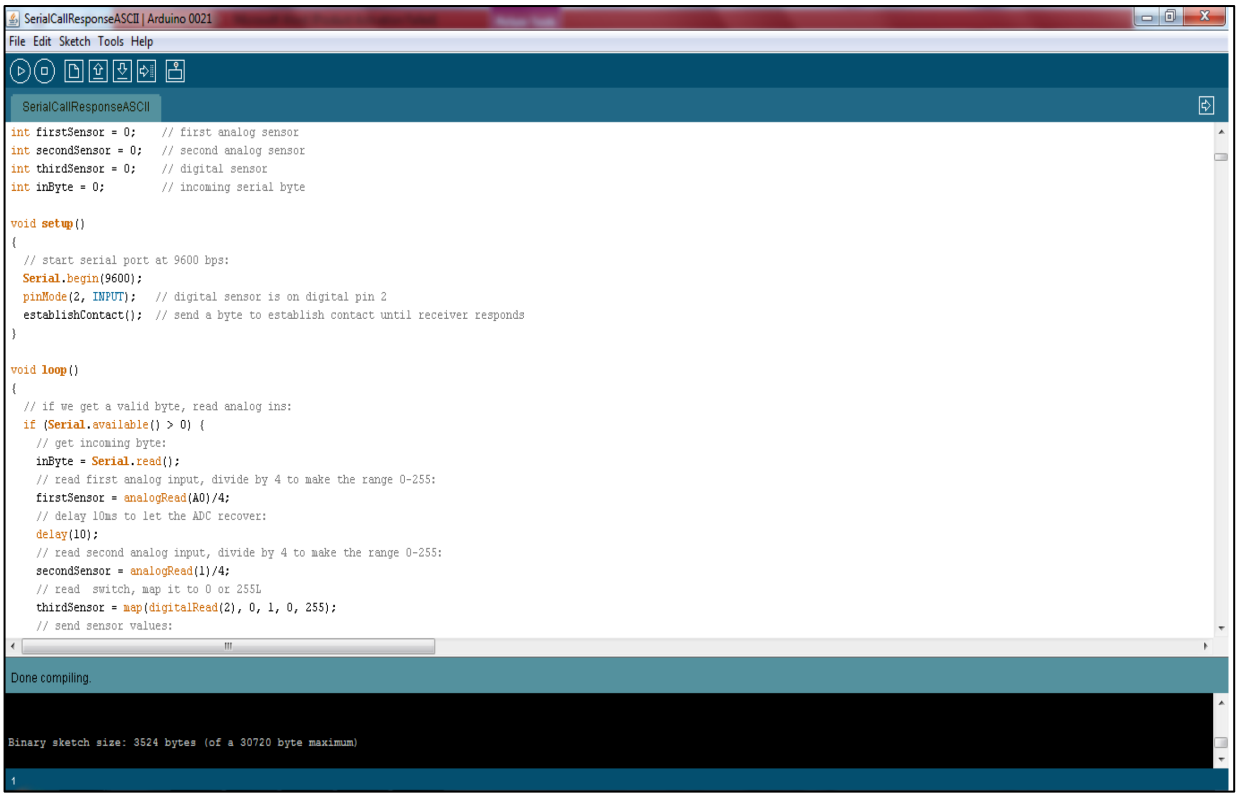

SerialCallResponseASCII | Arduino 0021
File Edit Sketch Tools Help
SerialCallResponseASCII
1

```
int firstSensor = 0;    // first analog sensor
int secondSensor = 0;   // second analog sensor
int thirdSensor = 0;    // digital sensor
int inByte = 0;         // incoming serial byte

void setup()
{
  // start serial port at 9600 bps:
  Serial.begin(9600);
  pinMode(2, INPUT);   // digital sensor is on digital pin 2
  establishContact();  // send a byte to establish contact until receiver responds
}

void loop()
{
  // if we get a valid byte, read analog ins:
  if (Serial.available() > 0) {
    // get incoming byte:
    inByte = Serial.read();
    // read first analog input, divide by 4 to make the range 0-255:
    firstSensor = analogRead(A0)/4;
    // delay 10ms to let the ADC recover:
    delay(10);
    // read second analog input, divide by 4 to make the range 0-255:
    secondSensor = analogRead(1)/4;
    // read  switch, map it to 0 or 255L
    thirdSensor = map(digitalRead(2), 0, 1, 0, 255);
    // send sensor values:
```

Fig 3.18: Snapshot of Arduino

An Arduino board consists of an 8-bit Atmel AVR microcontroller with complementary components to facilitate programming and incorporation into other circuits. An important aspect of the Arduino is the standard way that connectors are exposed, allowing the CPU board to be connected to a variety of interchangeable add-on modules (known as shields). Official Arduinos have used the megaAVR series of chips, specifically the ATmega8, Atmega168, Atmega328, Atmega1280, and Atmega2560. A handful of other processors have been used by Arduino compatibles.

Most boards include a 5 volt linear regulator and a 16 MHz crystal oscillator (or ceramic resonator in some variants), although some designs such as the LilyPad run at 8 MHz and dispense with the onboard voltage regulator due to specific form-factor restrictions. An Arduino's microcontroller is also pre-programmed with a boot loader that simplifies uploading of programs to the on-chip flash memory, compared with other devices that typically need an external chip programmer.


SerialCallResponseASCII | Arduino 0021
File Edit Sketch Tools Help
SerialCallResponseASCII

```
int firstSensor = 0;    // first analog sensor
int secondSensor = 0;   // second analog sensor
int thirdSensor = 0;    // digital sensor
int inByte = 0;         // incoming serial byte

void setup()
{
  // start serial port at 9600 bps:
  Serial.begin(9600);
  pinMode(2, INPUT);   // digital sensor is on digital pin 2
  establishContact();  // send a byte to establish contact until receiver responds
}

void loop()
{
  // if we get a valid byte, read analog ins:
  if (Serial.available() > 0) {
    // get incoming byte:
    inByte = Serial.read();
    // read first analog input, divide by 4 to make the range 0-255:
    firstSensor = analogRead(A0)/4;
    // delay 10ms to let the ADC recover:
    delay(10);
    // read second analog input, divide by 4 to make the range 0-255:
    secondSensor = analogRead(1)/4;
    // read  switch, map it to 0 or 255L
    thirdSensor = map(digitalRead(2), 0, 1, 0, 255);
    // send sensor values:
```

```
Done compiling.

Binary sketch size: 3524 bytes (of a 30720 byte maximum)
```

Fig 3.19: Compiling code in Arduino

At a conceptual level, when using the Arduino software stack, all boards are programmed over an RS-232 serial connection, but the way this is implemented varies by hardware version. Serial Arduino boards contain a simple inverter circuit to convert between RS-232-level and TTL-level signals. Current Arduino boards are programmed via USB, implemented using USB-to-serial adapter chips such as the FTDI FT232. Some variants, such as the Arduino Mini and the unofficial Boarduino, use a detachable USB-to-serial adapter board or cable, Bluetooth or other methods. (When used with traditional microcontroller tools instead of the Arduino IDE, standard AVR ISP programming is used.)

The Arduino board exposes most of the microcontroller’s I/O pins for use by other circuits. The Diecimila, now superseded by the Duemilanove, for example, provides 14 digital I/O pins, six of which can produce PWM signals, and six analog inputs. These pins are on the top of the board, via female 0.1 inch headers. Several plug-in application “shields” are also commercially available.

The Arduino IDE is a cross-platform application written in Java, and is derived from the IDE for the Processing programming language and the *Wiring* project. It is designed to introduce programming to artists and other newcomers unfamiliar with software development. It includes a code editor with features such as syntax highlighting, brace matching, and automatic indentation, and is also capable of compiling and uploading programs to the board with a single click. There is typically no need to edit make files or run programs on the command line.

The Arduino IDE comes with a C/C++ library called “Wiring” (from the project of the same name), which makes many common input/output operations much easier. Arduino programs are written in C/C++, although users only need define two functions to make a runnable program:

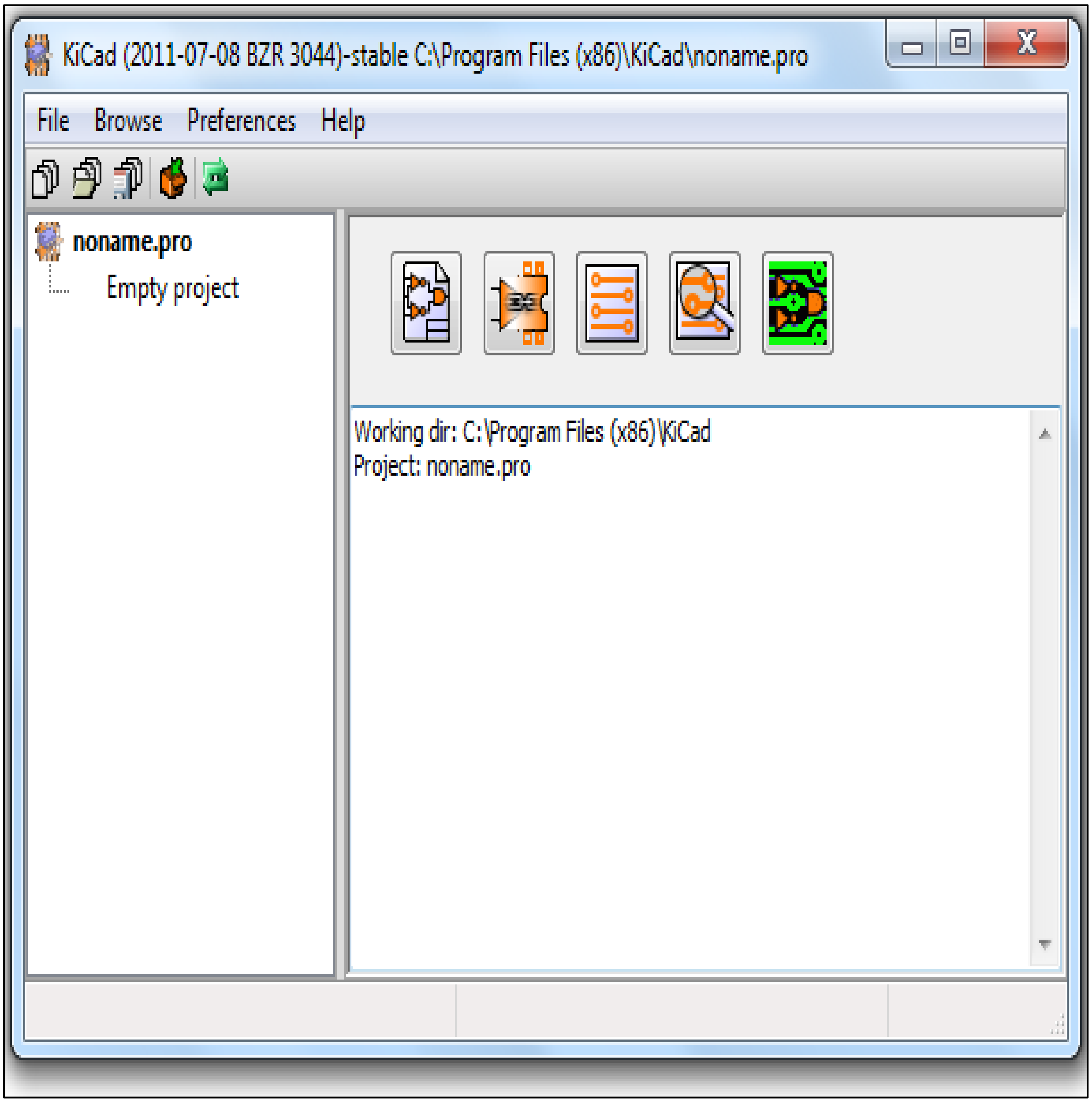
- setup() – a function run once at the start of a program that can initialize settings
- loop() – a function called repeatedly until the board powers off

A typical first program for a microcontroller simply blinks a LED (light-emitting diode) on and off. In the Arduino environment, the user might write a program like this:

```
#define LED_PIN 13
void setup () {
   pinMode (LED_PIN, OUTPUT);     // enable pin 13 for digital output
}
void loop () {
   digitalWrite (LED_PIN, HIGH);  // turn on the LED
   delay (1000);                  // wait one second (1000 milliseconds)
   digitalWrite (LED_PIN, LOW);   // turn off the LED
   delay (1000);                  // wait one second
}
```

For the above code to work correctly, the positive side of the LED must be connected to pin 13 and the negative side of the LED must be connected to ground. The above code would not be seen by a standard C++ compiler as a valid program, so when the user clicks the “Upload to I/O board” button in the IDE, a copy of the code is written to a temporary file with an extra include header at the top and a very simple main() function at the bottom, to make it a valid C++ program.

3.3.2.3 KiCAD


KiCad (2011-07-08 BZR 3044)-stable C:\Program Files (x86)\KiCad\noname.pro
File
Browse
Preferences
Help
noname.pro
Empty project

Working dir: C:\Program Files (x86)\KiCad
Project: noname.pro

Fig 3.20: KiCad Startup Screen

KiCad is an open source software suite for Electronic Design Automation (EDA). It facilitates the design of schematics for electronic circuits and their conversion to PCBs (printed circuit board) design. KiCad was developed by Jean-Pierre Charras, and features an integrated environment for schematic capture and PCB layout design. Tools exist within the package to create a bill of materials, artwork and Gerber files, and 3D views of the PCB and its components.

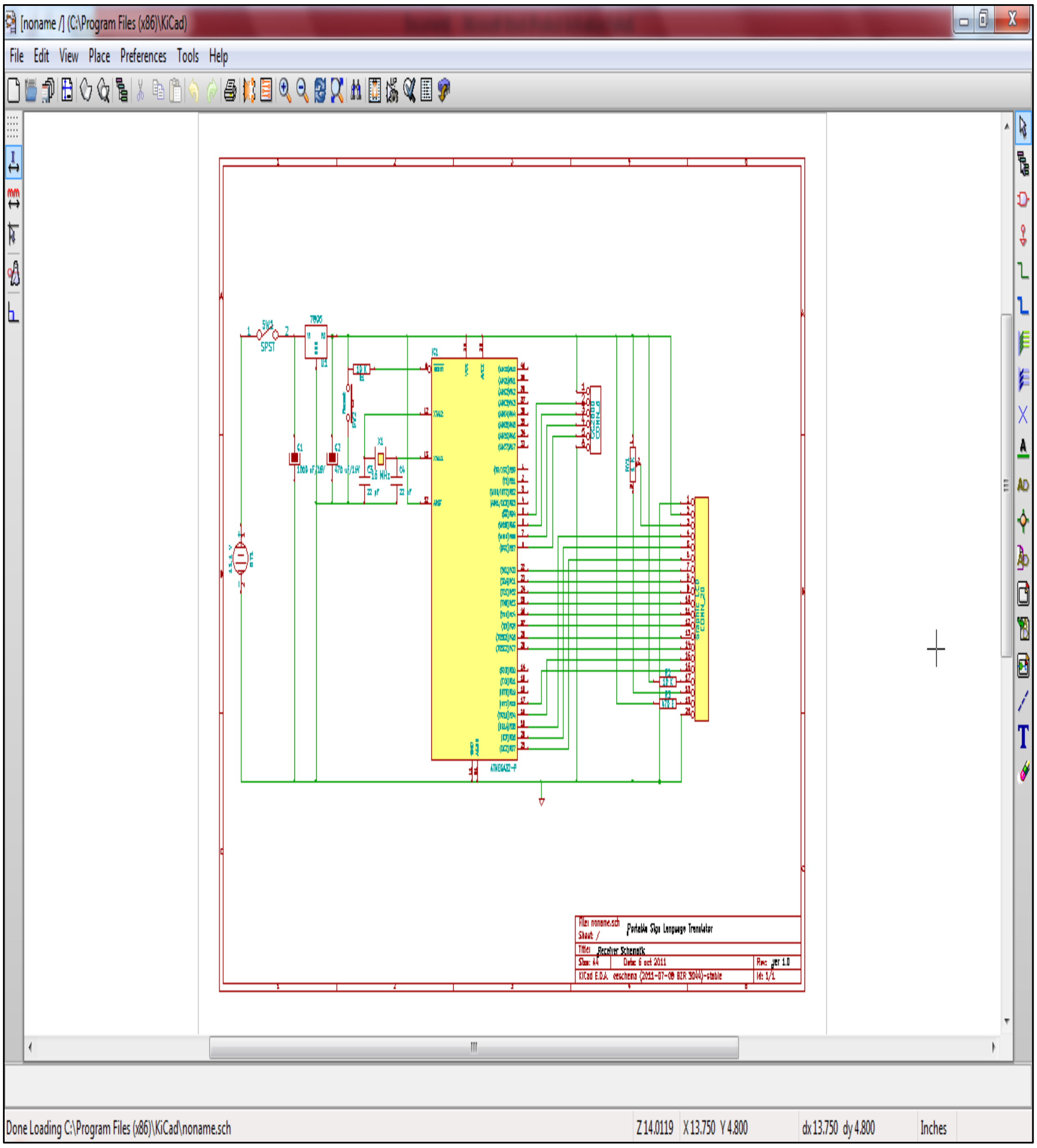
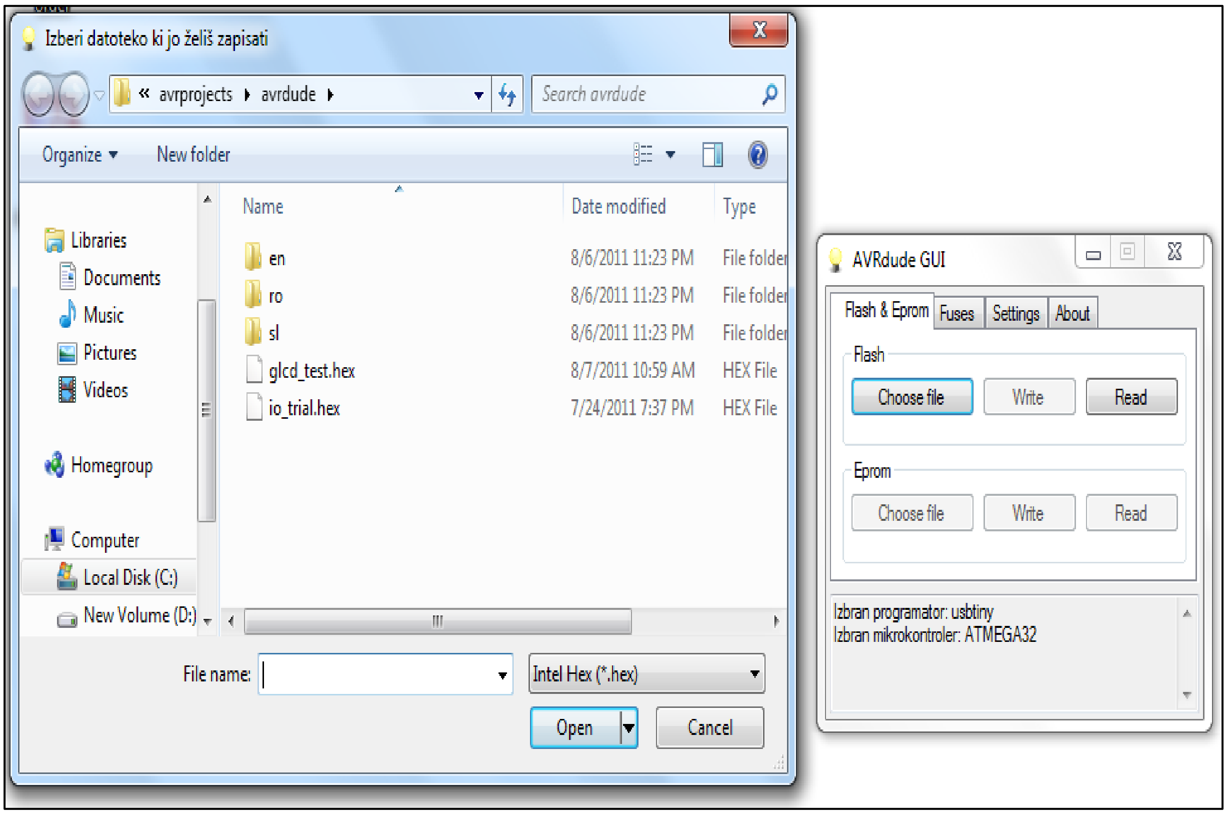

[noname /] (C:\Program Files (x86)\KiCad)
File Edit View Place Preferences Tools Help
Portable Sign Language Translator
Receiver Schematic
Done Loading C:\Program Files (x86)\KiCad\noname.sch
Z 14.0119 X 13.750 Y 4.800 dx 13.750 dy 4.800 Inches

Fig 3.21 Circuit Design and Checking in KiCad

KiCad uses an integrated environment for all of the stages of the design process: Schematic Capture, PCB layout, Gerber file generation/visualization and library editing. KiCad is cross-platform, written with WxWidgets to run on FreeBSD, Linux, Microsoft Windows and Mac OS X. Many component libraries are available, and users can add custom components. The custom components can be available on a per-project basis, or installed for use in any project. There are also tools to help with importing components from other EDA applications, for instance Eagle. Configuration files are in well documented plain text. This fact helps with interfacing to CVS's or SVN and with making automated component generation scripts.

Multiple languages are supported, such as English, Catalan, Czech, German, Greek, Spanish, Finnish, French, Hungarian, Italian, Japanese, Korean, Dutch, Polish, Portuguese, Russian, Slovene, Swedish, and Chinese. The 3D PCB viewing function is implemented with the Wings3D subdivision modeler. KiCad has a built-in basic autorouter. Alternatively, the freeware java-based FreeRouting can be used.

3.3.2.4 AVRdude GUI

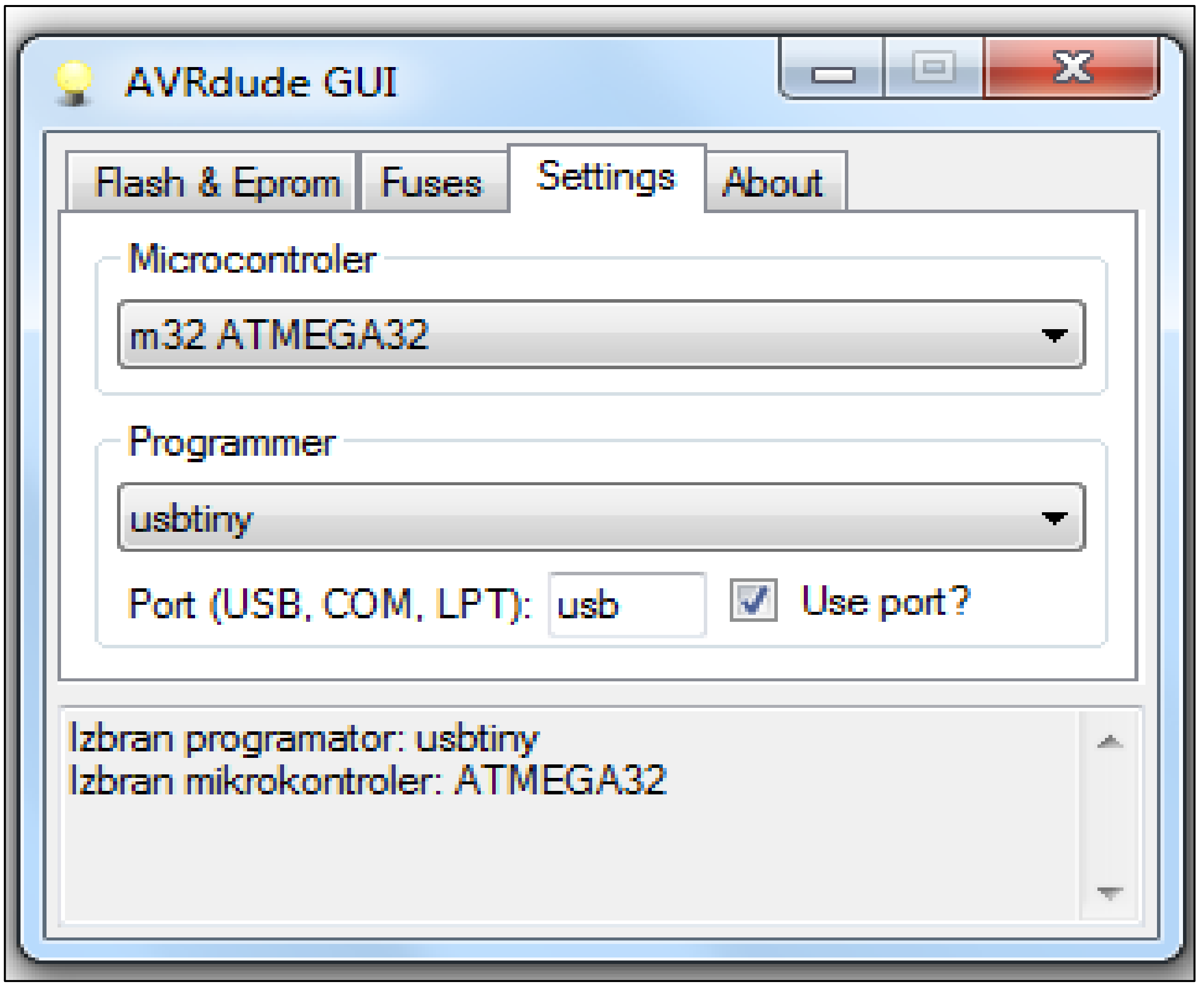


Name	Date modified	Type
en	8/6/2011 11:23 PM	File folder
ro	8/6/2011 11:23 PM	File folder
sl	8/6/2011 11:23 PM	File folder
glcd_test.hex	8/7/2011 10:59 AM	HEX File
io_trial.hex	7/24/2011 7:37 PM	HEX File

```
Izbran programator: usbtiny
Izbran mikrokontroler: ATMEGA32
```

Fig 3.22: Choosing the microcontroller and programmer

AVRdude GUI is a Graphical User Interface to burn hex code into the flash memory of microcontrollers. If features a very user-friendly interface to set the Lfuse and Hfuse, select the programmer type and make, select the microcontroller etc. Using this interface, we can also burn data into the EEPROM of the selected microcontroller.



Microcontroler: m32 ATMEGA32

Programmer: usbtiny

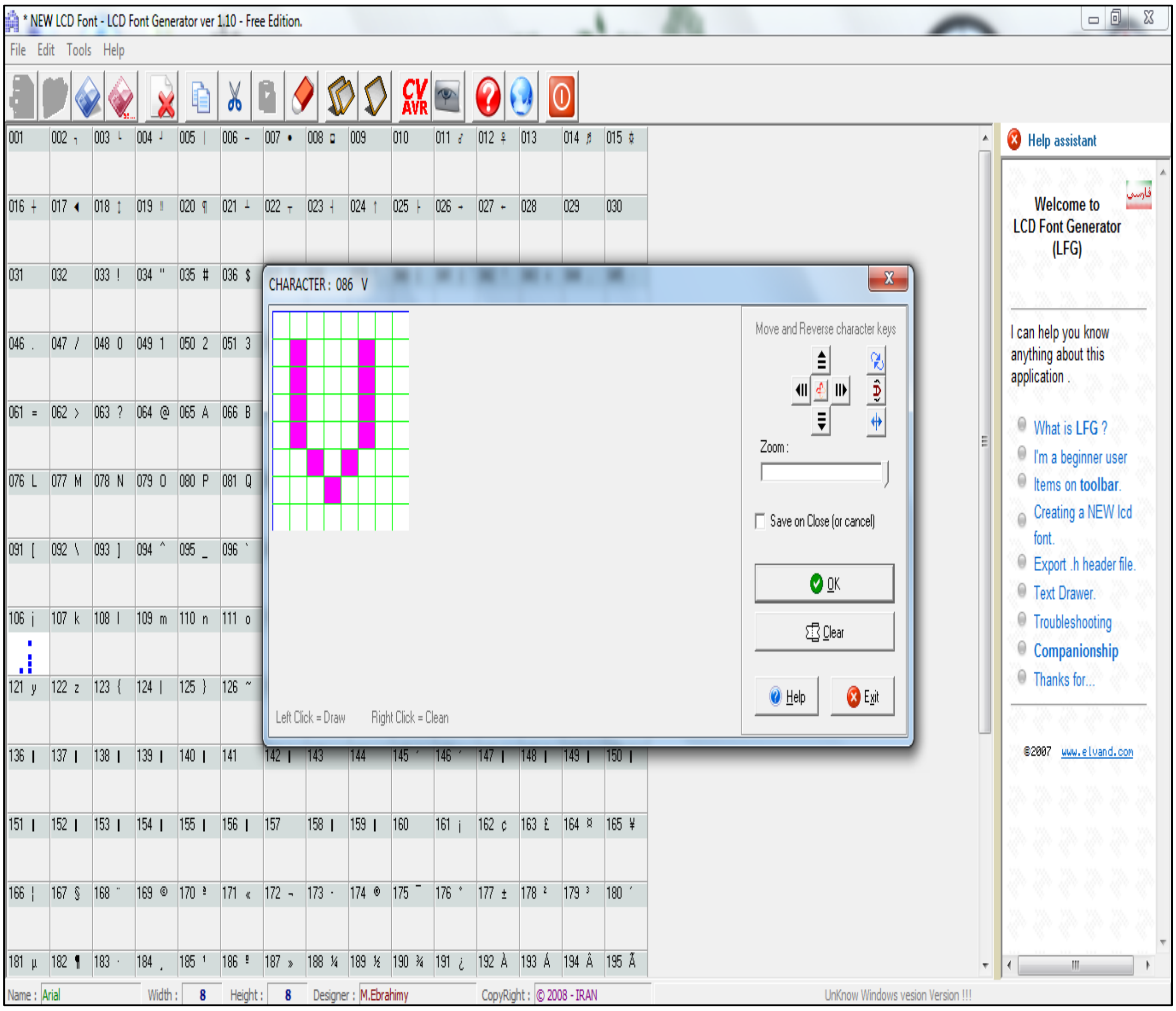
Port (USB, COM, LPT): usb [✓ Use port?]

```
Izbran programator: usbtiny
Izbran mikrokontroler: ATMEGA32
```

Fig 3.23 Applying settings

3.3.2.5 LCD font Generator

LCD Font Generator is useful software to make character or symbol fonts for a Graphical LCD Device, used with microcontrollers .When you make a powerful project, using graphic LCD, here's a question: how can make a beauty and nice font for it, simple and fast? With LFG, you can make beautiful fonts easily, using wizards and then export it to a '.h' (C compilers header file). This header file contains all data for your font in the form of a font array of bits. This can be added to the project with a '#include<filename.h>'. The font array can be then called from the header file, in the main program. Corresponding to the ASCII value of the character, the font array is indexed and the bit values are sent via the dataport to the LCD. This font file generated is used for C compiles such as CodeVision , WinAvr , and similar C Compilers.



Welcome to LCD Font Generator (LFG)

I can help you know anything about this application .

Name : Arial Width : 8 Height : 8 Designer : M.Ebrahimy CopyRight : © 2008 - IRAN

Fig 3.24: LCD font Generator

Features:

● Compatible with all graphic LCDs such as (AGM1232G,HDG12864,Nokia7110,PG128128) and all size (such as 122x32 , 64x128 , 128x128 , 240x64 , 96x65).

●Support best font characters size between 1.16 pixels per with or height.

●Secure and protect font file copyright, designer and name to save author law.

●Very small font files size.

● Rotate and reverse characters.

● Drawing any text on LCD and change its size, zoom, sensitive and position.

● Generating font map as .h header file, compatible with C Compilers such as Code Vision, WinAvr, AVR Studio 4, etc.

3.3.2.5 Bitmap2lcd basic

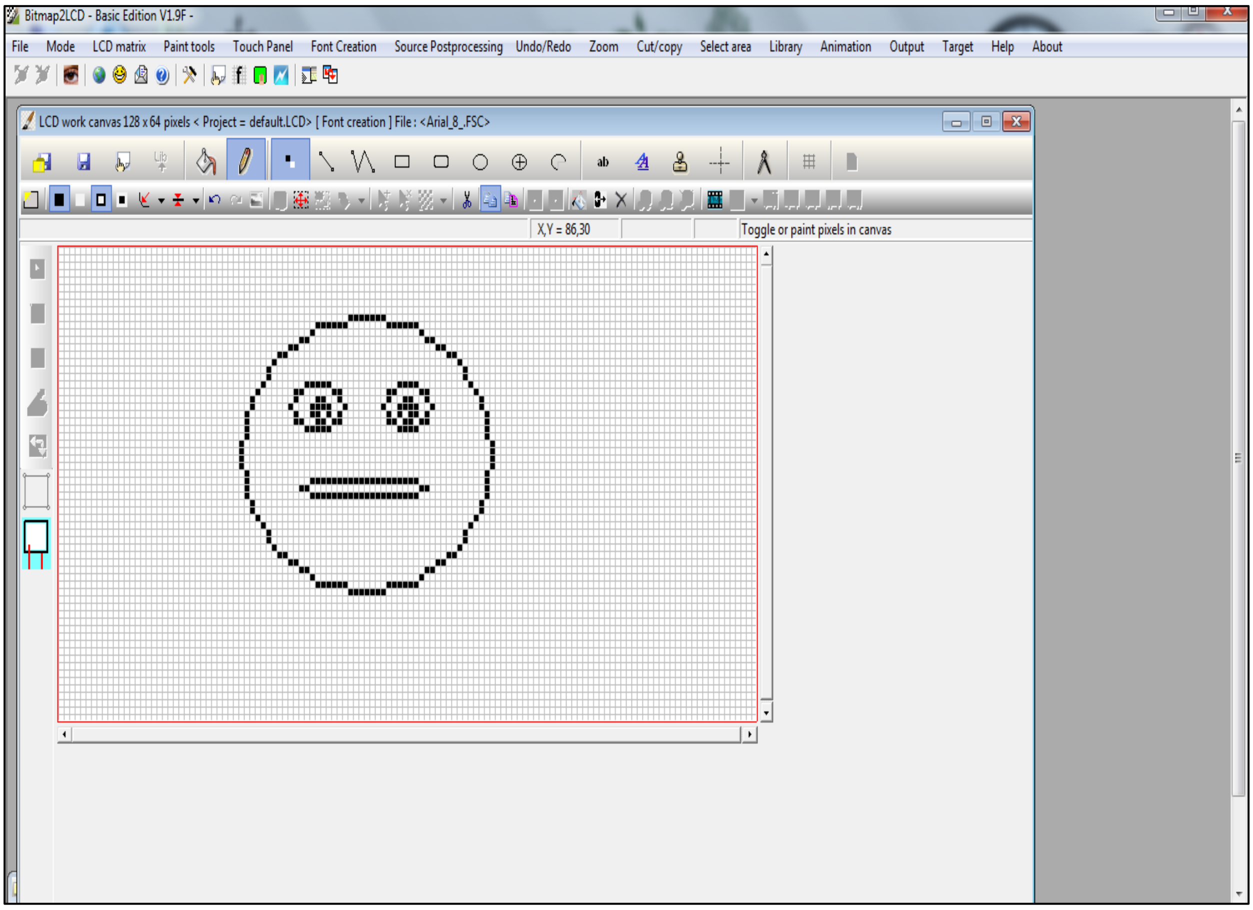

Bitmap2LCD - Basic Edition V1.9F -
File Mode LCD matrix Paint tools Touch Panel Font Creation Source Postprocessing Undo/Redo Zoom Cut/copy Select area Library Animation Output Target Help About
LCD work canvas 128 x 64 pixels < Project = default.LCD> [Font creation] File : <Arial_8_FSC>
X,Y = 86,30
Toggle or paint pixels in canvas

Fig 3.25: Bitmap2lcd basic

This is a user-friendly software that is used to draw images for the Graphic LCD. It is a feature rich package that combines drawing of simple bitmap images to animation schemes. This software comes with an inbuilt image to bitmap array converter. Once the image is exported, it is stored in the specified location as a ‘.h’ file. This ‘.h’ file will have to be included in the C project. The image is put to the LCD when the 8-bit data is read from the bitmap array and accordingly plotted to the screen.

Chapter 4

DESIGN

4.1 System design

4.1.1 Block diagram

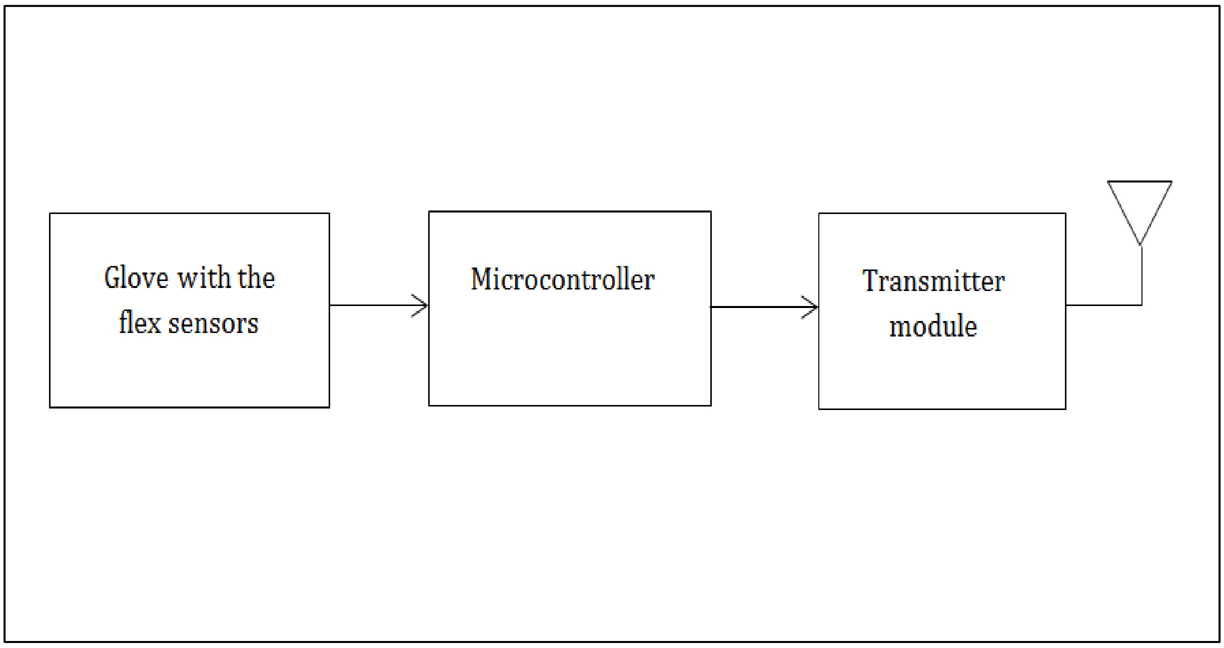


Fig 4.1: Block diagram at transmitter side

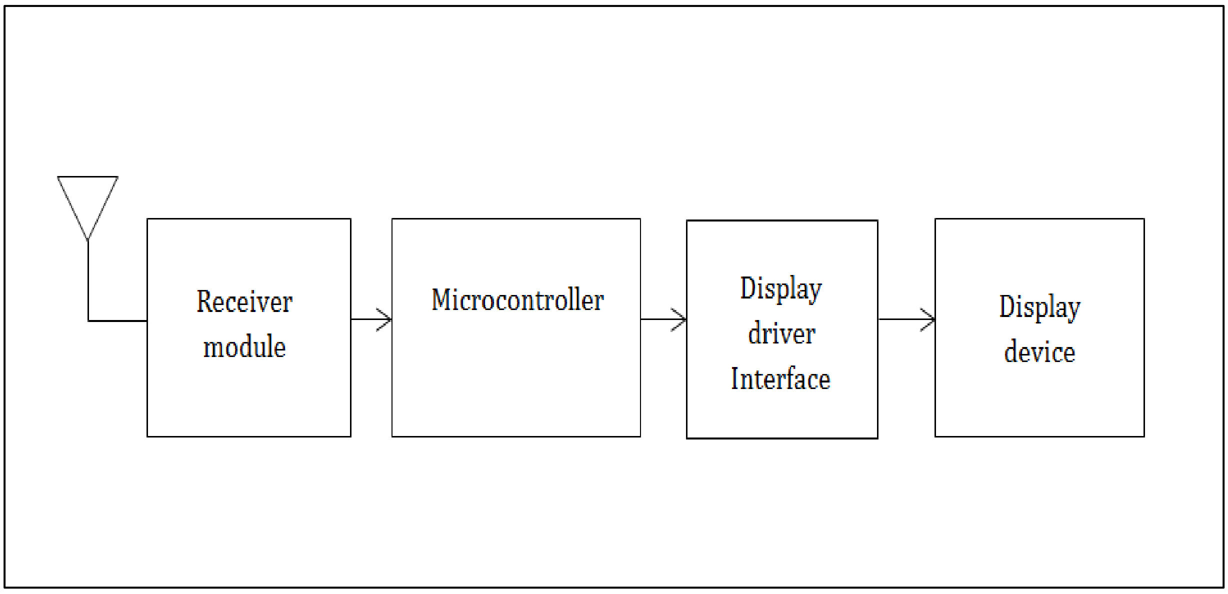


Fig 4.2: Block diagram at receiver side

4.1.2 Project flow diagram

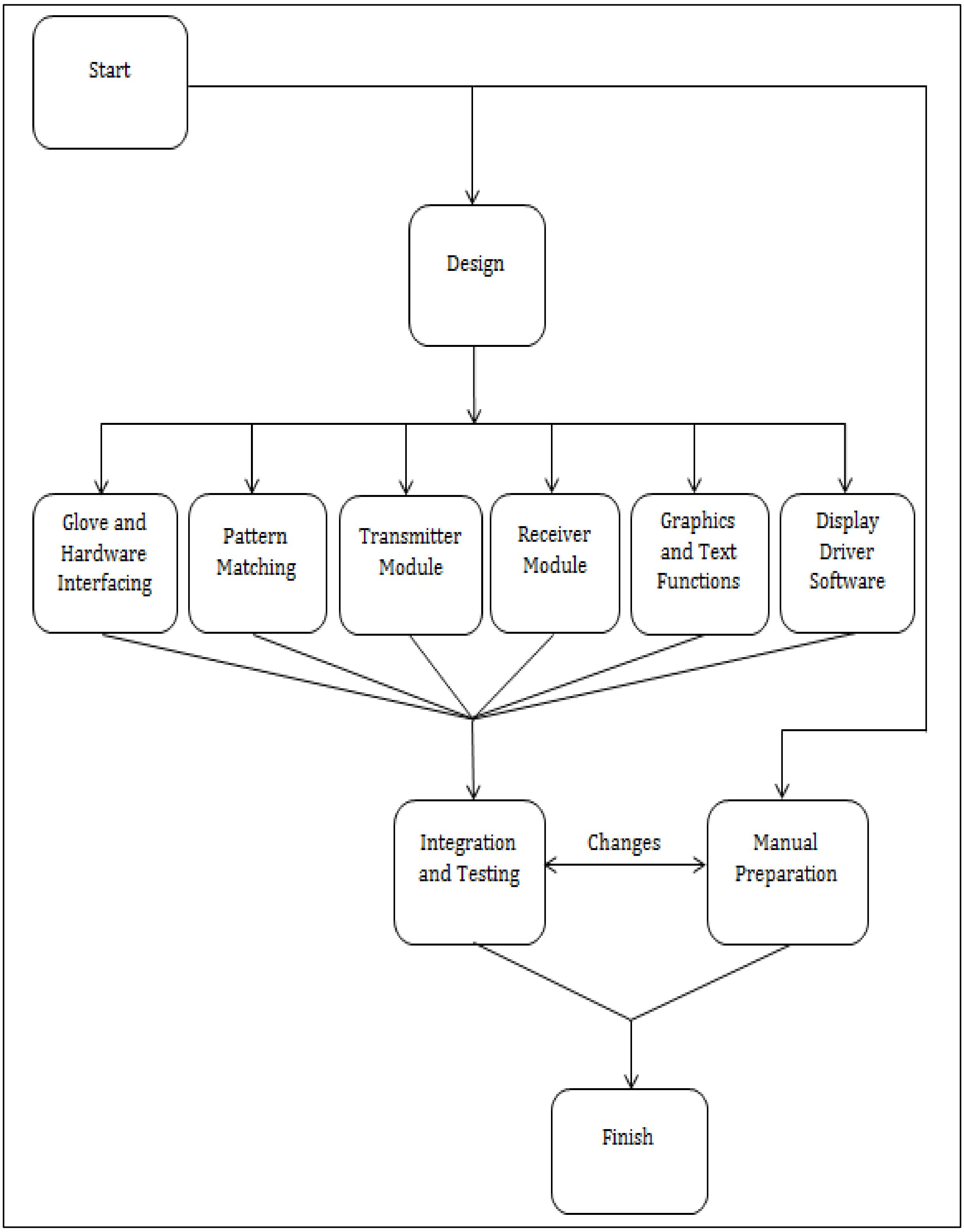


Fig 4.3: Project flow diagram

4.1.3 Waterfall Model

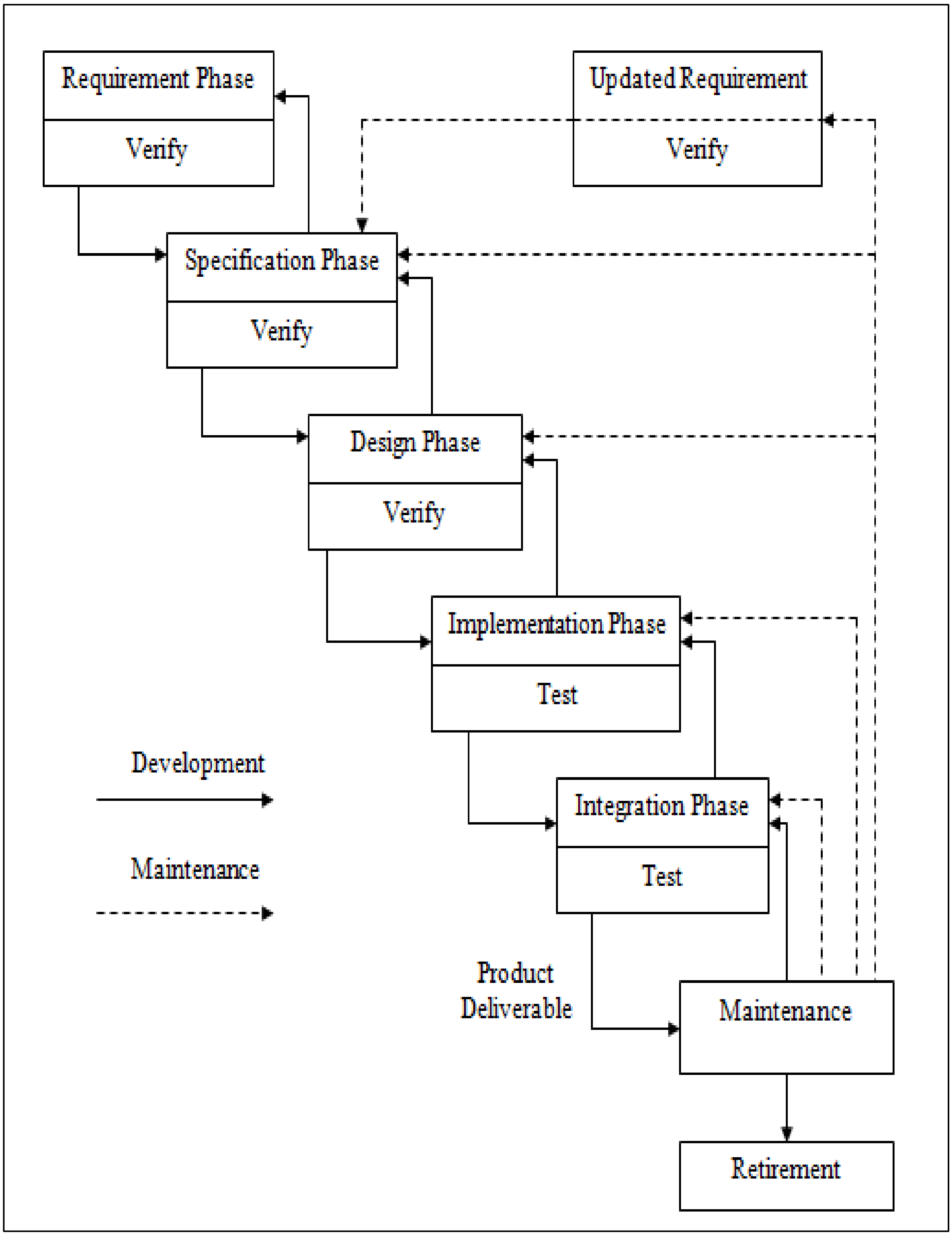


Fig4.4: Waterfall Model

“**The Waterfall Model**” is used as the lifecycle model for the project. In “The Waterfall” approach, the whole process of software development is divided into separate process phases. All the phases of this model are cascaded to each other so that second phase is started as and when defined set of goals are achieved for first phase and it is signed off, so the name “Waterfall Model”.

The stages of “The Waterfall Model” are:

Requirement Phase

All possible requirements of the system to be developed are captured in this phase. Requirements are set of functionalities and constraints that the end user expects from the system. The requirements are gathered from the end-user by consultation, these requirements are analyzed for their validity and the possibility of incorporating the requirements in the system to be developed is also studied. Finally, a Requirement Specification document is created which serves the purpose of guideline for the next phase of the model.

System & Software Design

Before a starting for actual coding, it is highly important to understand what we are going to create and what it should look like? The requirement from first phase are studied in this phase and is prepared. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

Implementation & Unit Testing

On receiving system design documents, the work is divided in modules/units and actual coding is started. The system is first developed in small programs called units, integrated in the next phase. Each unit is developed and tested for its functionality; this is referred to as Unit Testing. Unit testing mainly verifies if the modules/units meet their specifications.

Integration & System Testing

As above, the system is first divided in units which are developed and tested for their functionalities. These units are integrated into a complete system during Integration phase and tested if all modules/units coordinate between each other and the system as a whole behaves as per the specifications. After successfully testing the software, it is delivered to the customer.

Operations & Maintenance

This phase of "The Waterfall Model" is virtually never ending phase (Very long). Generally, problems with the system developed (which are not found during the development life cycle) come up after its practical use starts, so the issues related to the system are solved after deployment of the system. Not all the problems come in picture directly but they arise time to time and needs to be solved; hence this process is referred as Maintenance.

4.2 Software and Hardware Design

4.2.1 Use case diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

The building blocks of use case diagrams are:

- Use cases: A use case describes a sequence of actions that provide something of measurable value to an actor and is drawn as a horizontal ellipse.
- Actors: An actor is a person, organization, or external system that plays a role in one or more interactions with the system.
- System boundary boxes: A rectangle is drawn around the use cases, called the system boundary box, to indicate the scope of system. Anything within the box represents functionality that is in scope and anything outside the box is not.

The various Use cases are:

- **Sensor Analog to Digital Conversion:** This represents the process of analog to digital conversion. I.e. conversion of the voltage drop across the flex sensors (analog voltage) to digital values (1’s and 0’s). Before taking multiple samples, the unit checks if the ADC value falls in the preset range.

- **Data Processing:** This involves pattern learning, pattern matching, multiple sampling and data packing. **Pattern learning** involves the plotting of the sampled values to the EEPROM. These values are used as reference values for the subsequent pattern matches. **Pattern matching** involves the matching of the real-time patterns with those stored in the EEPROM. **Multiple sampling** involves taking multiple samples in real-time and storing them temporarily in the RAM in order to pattern match the values against the values stored in the EEPROM. **Data packing** finally is the packing of the character that was pattern matched during the pattern matching phase.

- **Character matching:** This occurs is at the receiver's end. This consists of data unpacking, font pattern recognition and character display. **Data unpacking** is the process of extracting the data (character) from the data packet received. This process also includes the checking of the checksum of the data packet to see whether the data received is the same as the transmitted one. **Font pattern recognition** involves the matching the bitmap character (8 contiguous locations of 8 bits) in the font array with the character received in order to display the character on the LCD. **Character display** is the actual character display on the LCD. For this we need a driver library with interfacing commands for the LCD.

The various actors are:

- **Master microcontroller:** This is the microcontroller that is present in the glove transmitter unit. It is responsible for learning the bends of the user's fingers, pattern matching and sending the character to the display unit.

- **Slave microcontroller:** This is the microcontroller that is used as a driver for the LCD device. It displays the characters on the screen, sends the interfacing commands, and matches the character ASCII value with the font file values in order to display the character.

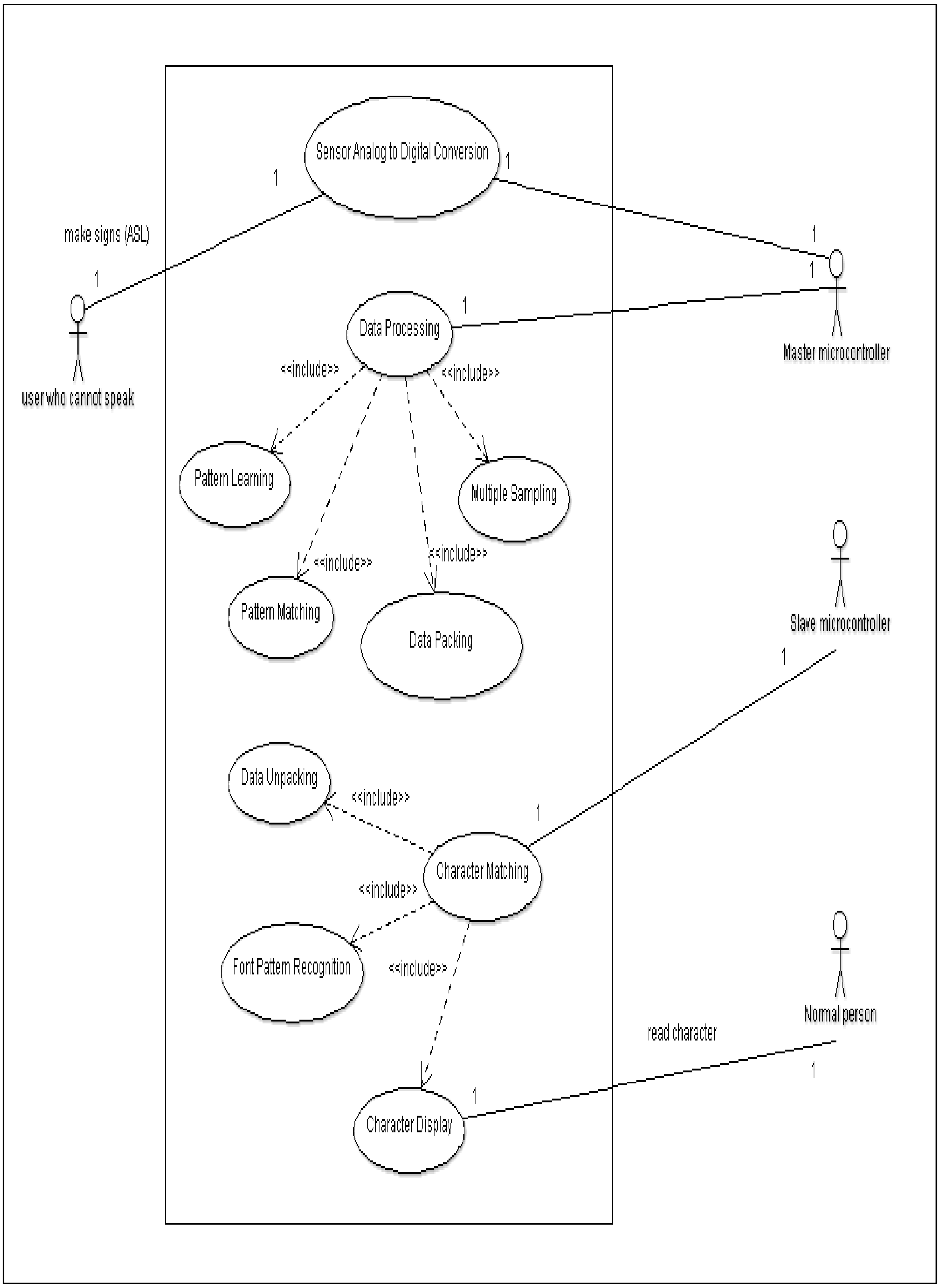


Fig 4.5: Use case diagram

4.2.2 Context level DFD

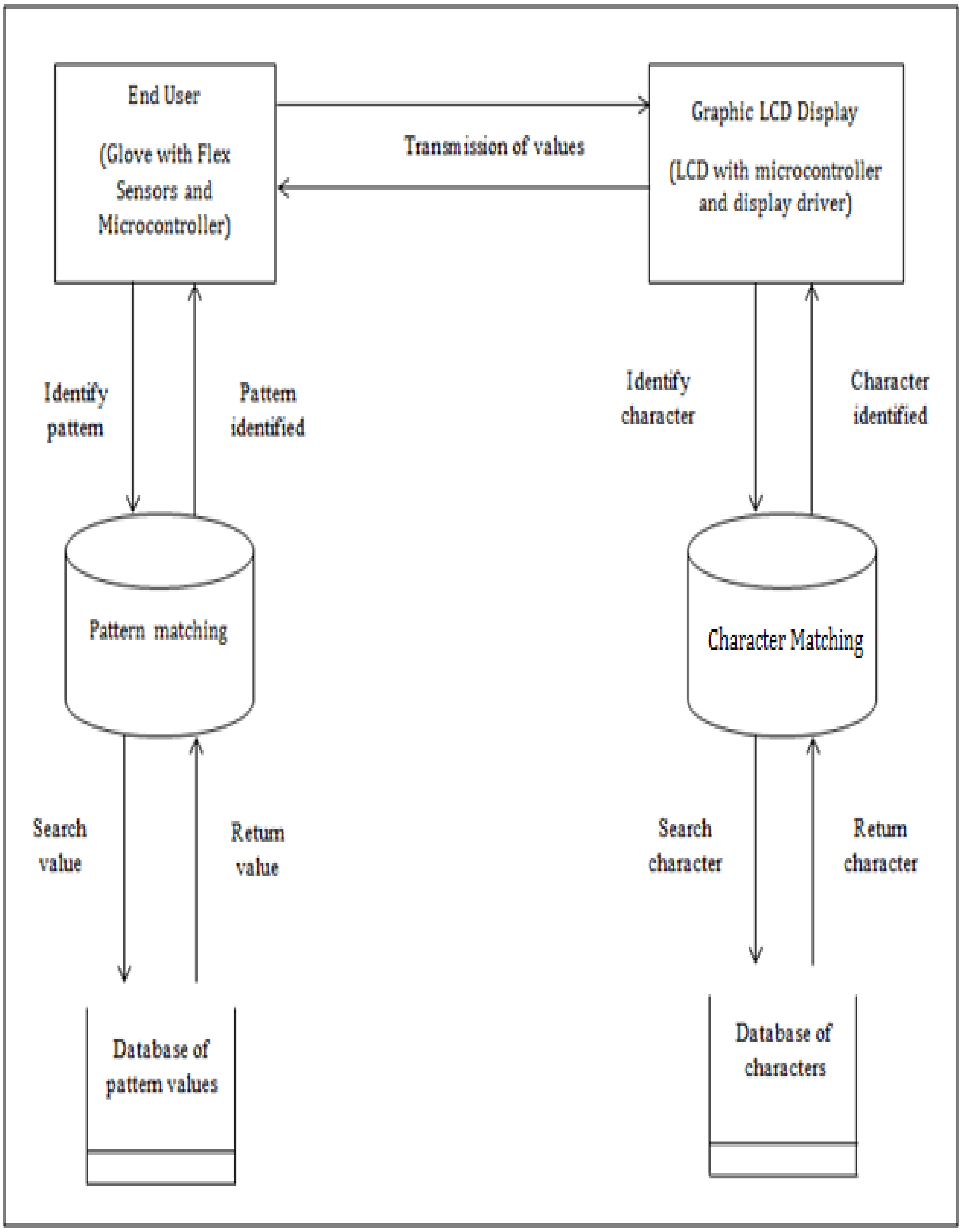


Fig 4.6: Context level DFD

4.2.3 Flow chart of transmitter

The system works on the real time samples taken and pattern matched with the already stored samples. Multiple samples are taken and the probability of an approximate match is calculated. Taking the probability of multiple samples reduces error and increases the probability of success of a match. For example the probability of finding a person wearing a red shirt in a sample space of 1000 is higher than the probability of finding a red shirt in a sample space of just 100. When the user first wears the glove, the unit does not know the bend values of his/her fingers as nothing is stored in the EEPROM. Hence the unit first checks if there are values stored in the EEPROM before proceeding with pattern matching. If samples are already stored in the EEPROM, the EEPROM write flag is set. Hence the unit can proceed with the pattern matching. If the flag is reset (i.e. 0) then the unit asks the user to store the samples corresponding to each letter before proceeding.

For real-time pattern matching, the unit checks if the ADC sampled values are within a preset range. If these values lie within that range, the unit then starts to take multiple samples of the voltage drop values. The ADC has a sampling frequency of around 125 KHz. This frequency is enough to give us around 100 samples in about a millisecond. We have to wait till sampling is complete and hence a small delay is introduced. These ADC real-time samples are temporarily stored on the RAM. Next the KMP algorithm is executed and the values are read from the EEPROM and matched against the values stored in the RAM. If there is a 80% or more probability of a character match, then that character is returned as the matched character. For reading the values from the EEPROM, a special page table is created which contains the character address and the pointer to the physical EEPROM address location. A ± 10 difference between the patterns matched is tolerated. This is done so as to compensate for the errors in sampling and physical sensor defects.

The KMP algorithm then returns the character, if there is a match and if no match is found it repeats the sampling process again to check for the rest of the samples. This character is packed along with its checksum and a flag byte. This is implemented in as a data structure consisting of flag (char), the data (int) and checksum (int). Before transmission, the data is buffered. This is done because at some point of time, the sampling may be faster than the data transmission or the character may be returned earlier than expected or the data transmitted may be slow. To deal with all these conditions, a buffer is inserted which ensures that data is transmitted with a smoothly. The data is then transmitted via the SPI port.

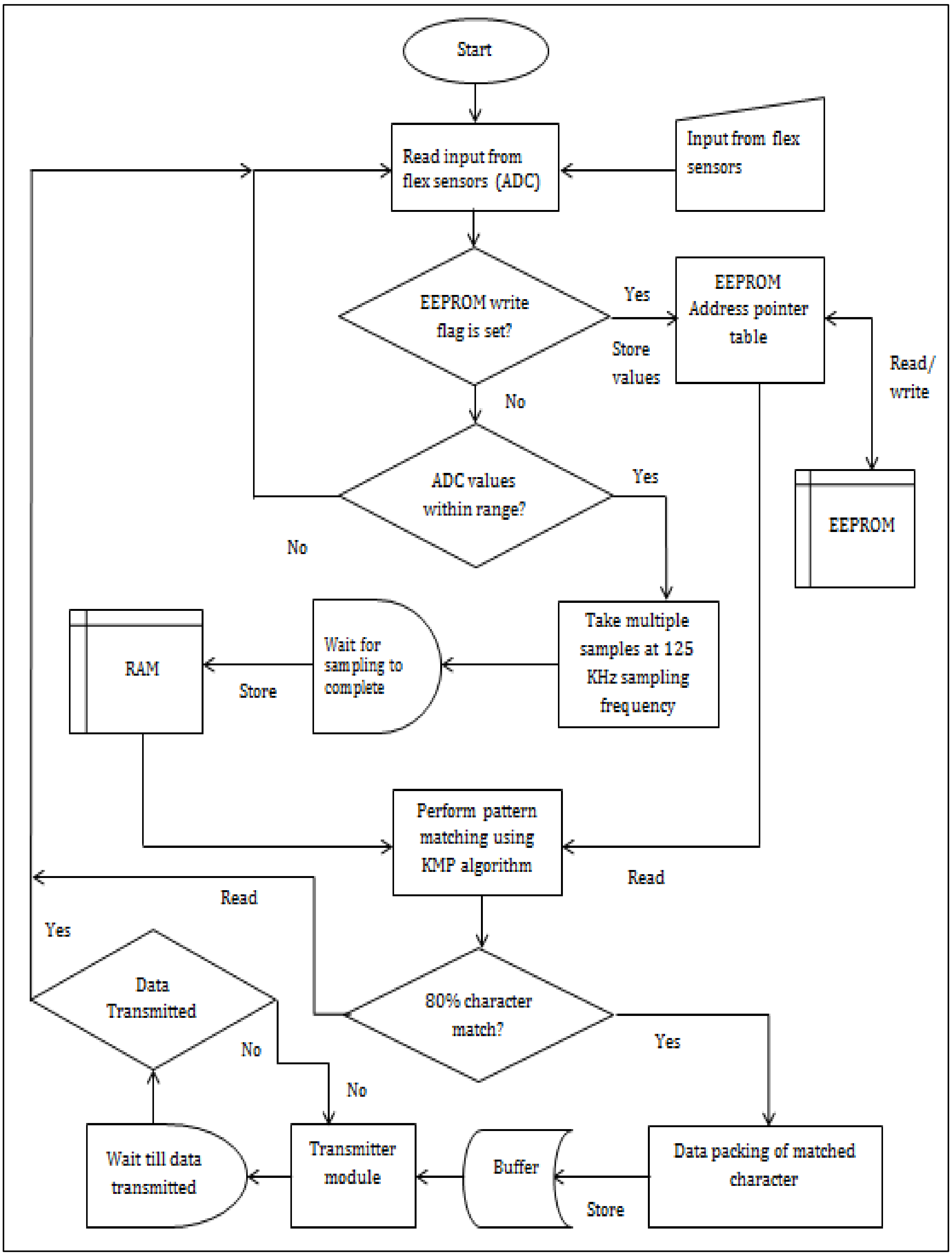

Start
Read input from flex sensors (ADC)
Input from flex sensors
EEPROM write flag is set?
Yes
Store values
EEPROM Address pointer table
Read/ write
EEPROM
No
ADC values within range?
Yes
No
Take multiple samples at 125 KHz sampling frequency
Wait for sampling to complete
Store
RAM
Perform pattern matching using KMP algorithm
Read
Read
Yes
Data Transmitted
No
80% character match?
Yes
No
Wait till data transmitted
Transmitter module
Buffer
Store
Data packing of matched character

Fig 4.7: Flow chart of transmitter

4.2.4 Flow chart of Receiver

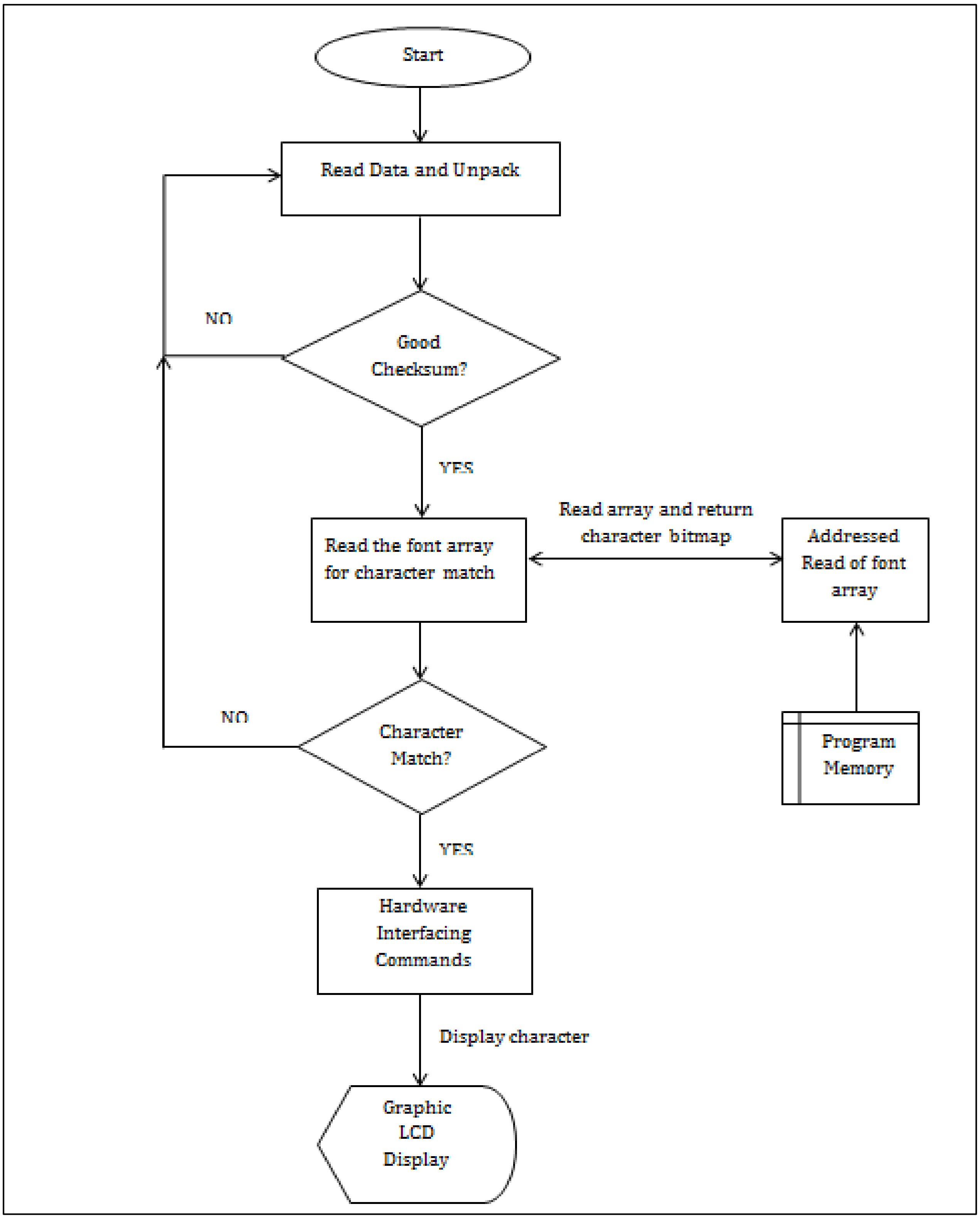

Start
Read Data and Unpack
NO
Good Checksum?
YES
Read the font array for character match
Read array and return character bitmap
Addressed Read of font array
Program Memory
NO
Character Match?
YES
Hardware Interfacing Commands
Display character
Graphic LCD Display

Fig 4.8: Flow chart of Receiver

At the receiver’s end, the data is read via the SPI port. The read data from the register is buffered. The data is then unpacked and the packed is checked for a valid flag and checksum. If the checksum is good then the data is returned. Each character (data) is sent to the display driver software. There is a special font array bitmap file that contains the representations (in bits) of the character on the graphic LCD. A character match of the ASCII value against the font array index is done. The font array is arranged so that the ASCII value of each character corresponds to the respective bit value in that index location of the array. Since the graphic LCD is divided into 8 x 8 pixels for each character, the font array is read for 8 consecutive locations, with each location containing 8 bits data. (1bit is allocated to each pixel). Once the character bitmap is returned, the data is sent to the Graphic LCD data port for display. The data is placed on the port, subsequently making the EN pin high in order to read it and low while changing data on the port.

4.2.5 Transmitter Schematic

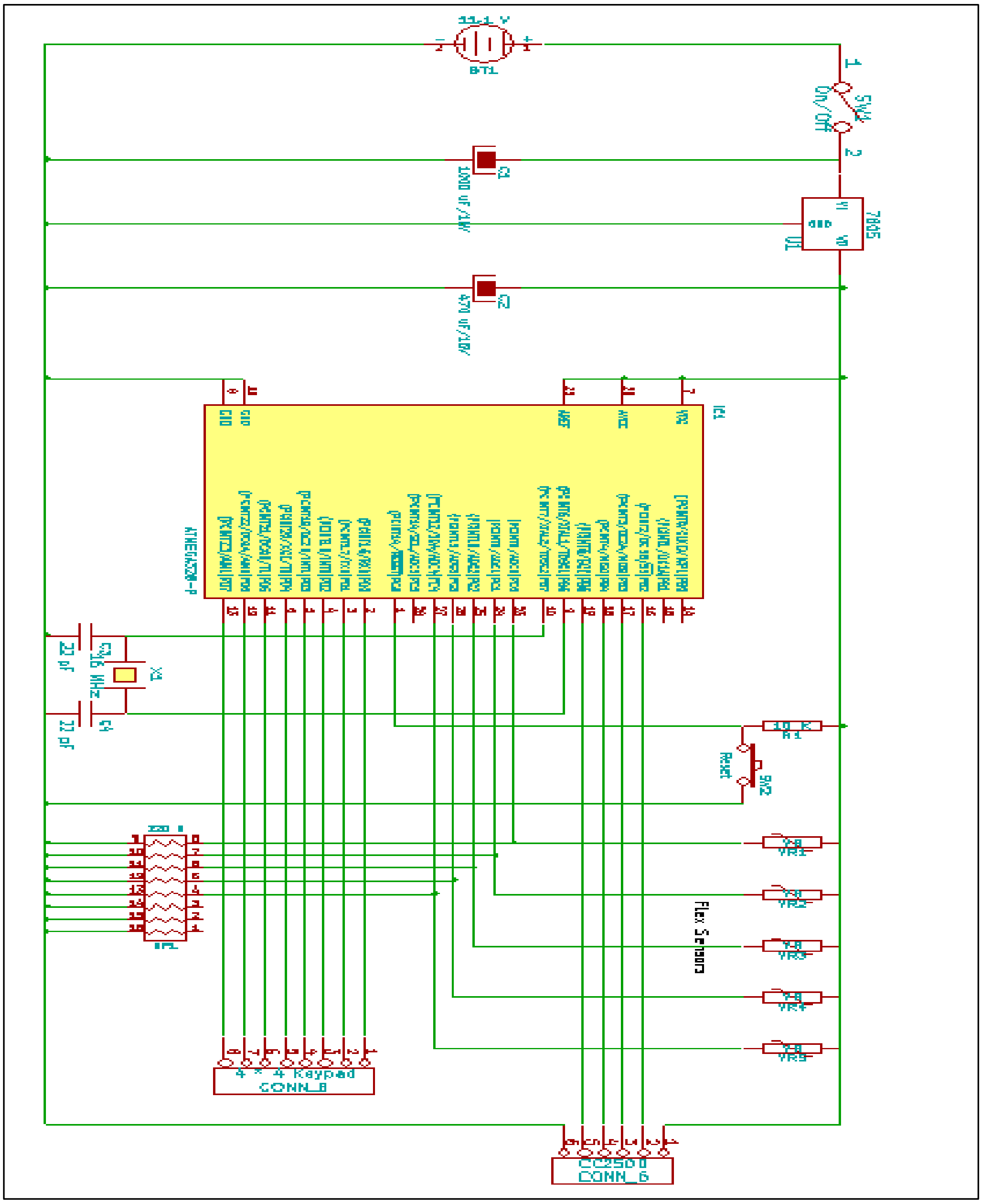


Fig 4.9: Transmitter Schematic

4.2.6 Receiver Schematic

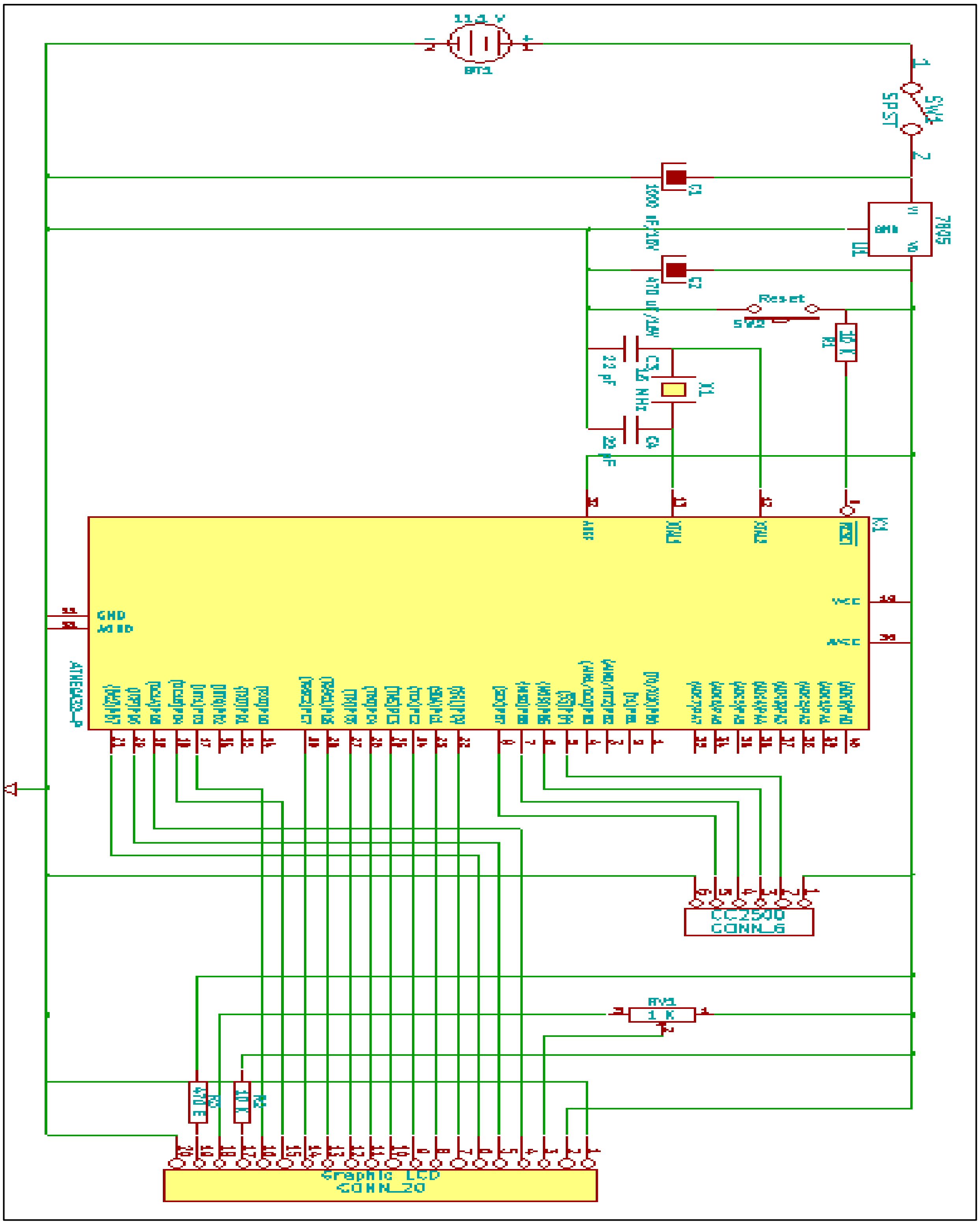


Fig 4.10: Receiver Schematic

Chapter 5

CONCLUSION

5.1 CONCLUSION

The various functionalities to be included in the Portable Sign Language Translator were studied based on the user specification and the literature survey conducted. In addition, the following work was also carried out.

- ✓ In the literature survey, a study on the ASL (American Standard Language) was done. Different pattern matching algorithms that can be used in our project were studied.
- ✓ For the hardware, we studied different microcontrollers and their features. The components and software required to build up the project were found out. Based on these findings, six modules were designed.
- ✓ The SRS (Software Requirement Specification) was documented and project design was carried out.
- ✓ The circuit diagram was drawn after study of the datasheets of the respective components.
- ✓ The embedded display driver software and interfacing of the graphic LCD with the microcontroller was implemented.

5.2 Further Work

Further work deals with the implementation and testing of the project modules. This involves:

- ➢ The construction of the circuit board and the layout of the components on the board.
- ➢ Module-wise software coding.
- ➢ Integrating and testing these modules
- ➢ Testing the project as a whole

REFERENCES:

Datasheets (pdf version):

- flex sensor 4.5”
- Atmega 32 microcontroller
- Atmega 328 microcontroller
- KS0108B Graphic LCD driver
- LCD-128G064E Graphic LCD datasheet
- CC2500

E-books:

- Avr Studio Software Tutorial by Sepehr Naimi
- Tuxgraphics AVR C-programming tutorial (http://tuxgraphics.org/electronics)
- C Programming for Microcontrollers- Quick Start Guide: The Next Generation by Joe Pardue

Websites:

- www.arduino.co.cc
- http://www.buzzle.com/articles/waterfall-model-phases.html
- http://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm
- http://www.cmi.ac.in/~kshitij/talks/kmp-talk/kmp.pdf
- http://www.ics.uci.edu/~eppstein/161/960227.html
- http://extremeelectronics.co.in/avr-tutorials/4x3-matrix-keypad-interface-avr-tutorial
- http://extremeelectronics.co.in/avr-tutorials/interfacing-graphical-lcd-with-avr-mcu-%E2%80%93-part-i
- http://extremeelectronics.co.in/avr-tutorials/interfacing-graphical-lcd-with-avr-mcu-%E2%80%93-part-ii
- http://extremeelectronics.co.in/avr-tutorials/interfacing-graphical-lcd-with-avr-mcu-%E2%80%93-part-iii

ACRONYMS

ADC	Analog to Digital Convertor
ALU	Arithmetic Logic Unit
ASL	American Sign Language
BSL	British Sign Language
CISC	Complex Instruction Set Computing
CPU	Central Processing Unit
DFD	Data Flow Diagram
EEPROM	Electrically Erasable Programmable Read Only Memory
GND	Ground
GUI	Graphics User Interface
KMP	Knuth Morris Pratt
LCD	Liquid Crystal Display
LED	Light Emitting Diode
LSB	Least Significant Bit
MSB	Most Significant Bit
RISC	Reduced Instruction Set Computing
SPI	Serial Peripheral Interface
SRS	Software Requirement Specification
SRAM	Static RAM
USART	Universal Synchronous Asynchronous Receiver Transmitter
UART	Universal Asynchronous Receiver Transmitter