PROJECT REPORT

On

Development of Three Dimensional Display Based on lighting scheme using Microcontroller

This Project submitted to the department of Electrical and Electronic Engineering to fulfill the requirements of the degree of Bachelor of Science in Electrical and Electronic Engineering.

SUBMITTED BY

NAME:	ID:
MD. JEWEL MONDAL	073800057
MD. RUHUL AMIN	081800056
S. M. EMRAN	091800024
K.M. SHAFKAT JANIL	081800088
MD. RAKIBUL HASAN	091800049

Supervised by

ABU SHAFIN MOHAMMED MAHDEE JAMEEL

(Lecturer, Faculty of Engineering & Technology)

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

EASTERN UNIVERSITY, BANGLADESH

AUGUST, 2012

DECLERATION

We do here by solemnly declare that, the work presented in this project report has been carried by us and has been previously submitted to any other university for an academic certificate/diploma or degree.

Signatures of the Candidates:

……………………

MD. JEWEL MONDAL

ID#073800057

……………………….

MD. RUHUL AMIN

ID#081800056

……………………......

K.M.SHAFKAT JANIL

ID#081800088

………………..

S. M. EMRAN

ID#091800024

……………………..

MD. RAKIBUL HASAN

ID#091800049

APPROVAL

The project report design and implementation of development of three dimensional display (3D) based on lighting scheme using Microcontroller has submitted to the following members of the board of examiners of the faculty of Engineering & Technology in partial fulfillment of the requirement for the Bachelor of degree of Science in Electrical & Electronics Engineering by following students and has been accepted as satisfactory.

1. MD. JEWEL MONDAL	**ID#073800057**
2. MD. RUHUL AMIN	**ID#081800056**
3. K.M. SHAFKAT JANIL	**ID#081800088**
4. S. M. EMRAN	**ID#091800024**
5. MD. RAKIBUL ISLAM	**ID#091800049**

Supervisor	**External**	**Chairperson**
………………………………..	……………………..	…………………………..
ABU SHAFIN MOHAMMAD MAHDEE JAMIL MIRZA GOLAM RABBANI	**MD. ASHIF IQBAL SIKDAR**	**PROF. DR.**
Sr. Lecturer	**Lecturer**	**Chairperson**
Faculty of E & T	**Faculty of E & T**	**Faculty of E & T**
Eastern University, Bangladesh	**Eastern University, Bangladesh**	**Eastern University, Bangladesh**

DEDICATION

Dedicated to our beloved parents and our honorable teachers.

ACKNOWLEDGEMENT

At first, we are grateful to the Almighty Allah, who helped us to complete this project papers work successfully.

We would like to thank the following people for their valuable assistance during the year:

It is a real pleasure to express our deepest appreciation sincere gratitude and gratefulness to our project supervisor **Abu Shafin Mohammad Mahdee Jameel**, Lecturer, Faculty of E&T, Eastern University, Bangladesh. For his supervision and on his ongoing assistance throughout the year. His willingness to be a mentor, to engage in problem solving and to provide feedback was greatly appreciated and valued for his assistance in designing PFI in an efficient and positive manner.

Our friends and family, thank you for your support and assistance.

At last, we would like to remember our almighty Allah for whom we can get education.

ABSTRACT

Now a days we cannot imagine world without technology. By developing technology, microcontroller is very important part for the performance of any type programming through electronics,

In the project LEDs present many advantages over incandescent light sources including lower energy consumption, longer life time, improved robustness, smaller size, faster switching easy installation application, friendly environment, and greater reliability. We designed our LED light scheme with advanced technology which can replace directly the conventional bulbs such as incandescent bulbs, lamppost, sodium as a rod light,

In this project we made work three dimensional display with micro-controller based. We are supplied additional computer power supply for the 3D display. We used program based microcontroller technology in this project. So we used here programmable device AT mega 16.

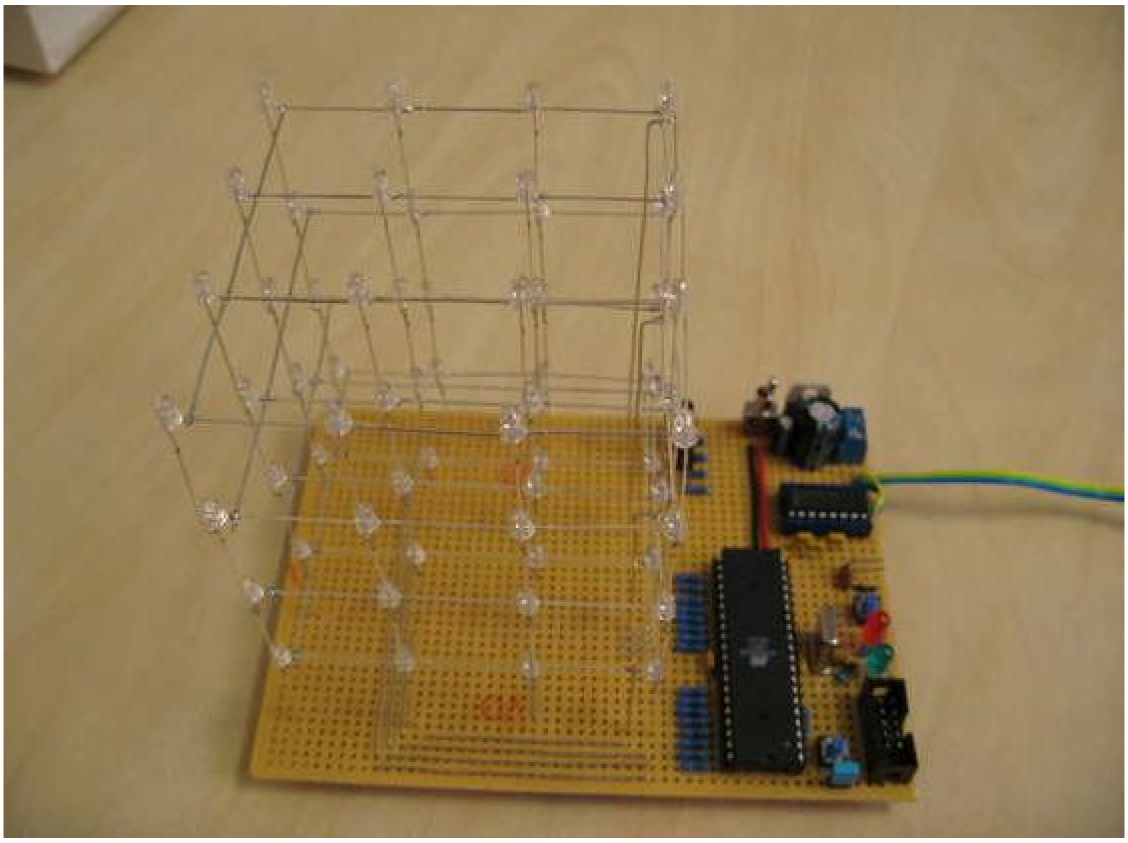
We used here 64 LEDs for the design a cube and we also used here capacitor, resistor, and extra power supply. Light will be bright and dark as programming code.

INDEX

TABLE OF CONTENTS: **PAGE NO.**

CONTENTS

1.1 Led Cube 4x4x4



Project
Started: 12- 08-2011
Finished: 06-08-2012

1.2 Introduction

This project is based on microcontroller for the lightning scheme of led cube from the Website www.instructables.com it was designed by “CHR” I’ve changed this design to fit my purpose and materials while still being able to keep the same.

1.3 Goal

--To know microcontroller and it’s application.

--Create a similar led cube to the one from intractable.

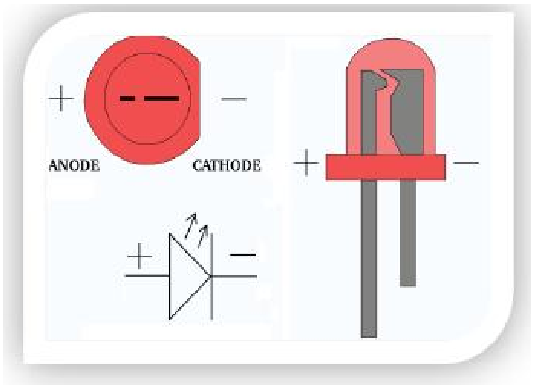
--Program different behavior.

--Make it look as nice as possible.

1.4 Hardware

Led's:

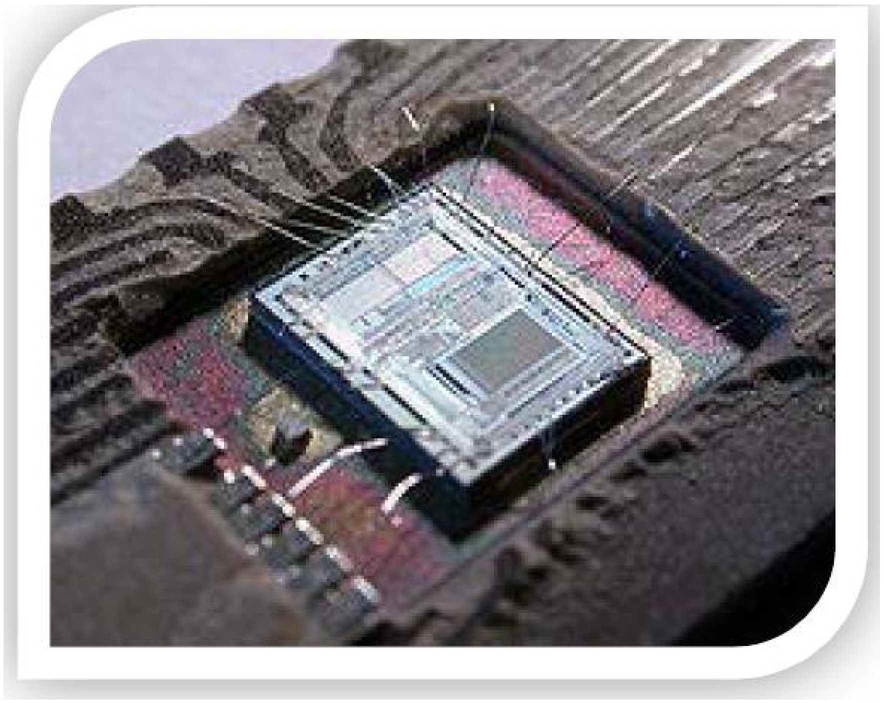
Light emitting diodes are diodes which produce light when current passes through in the right direction.



Just like any diode the other direction is completely blocked. Led's are nowadays commonly used as light sources because they are smaller, very efficient and have
a longer lifetime.

1.5 Microcontrollers:

Microcontrollers are small computers in a small package. They contain several things such as a processor, ram and programmable memory and most last but not least inputs and outputs. Microcontrollers are used in most automatically controlled devices e.g. microwaves, cellphones and alarms. They are very convenient to use for products produced in huge quantities because they can be programmed very quickly. And they are very small compared to previous



techniques in which processor ram and programmable memory are separated. These computers in a chip are nowadays also used in mp3 players in which another function is very useful, this function is the sleep function, and microcontrollers are able to wake up immediately on the touch of a button.

Microcontroller can be termed as a single on chip computer which includes number of peripherals like RAM, EEPROM, Timers etc., required to perform some predefined task.

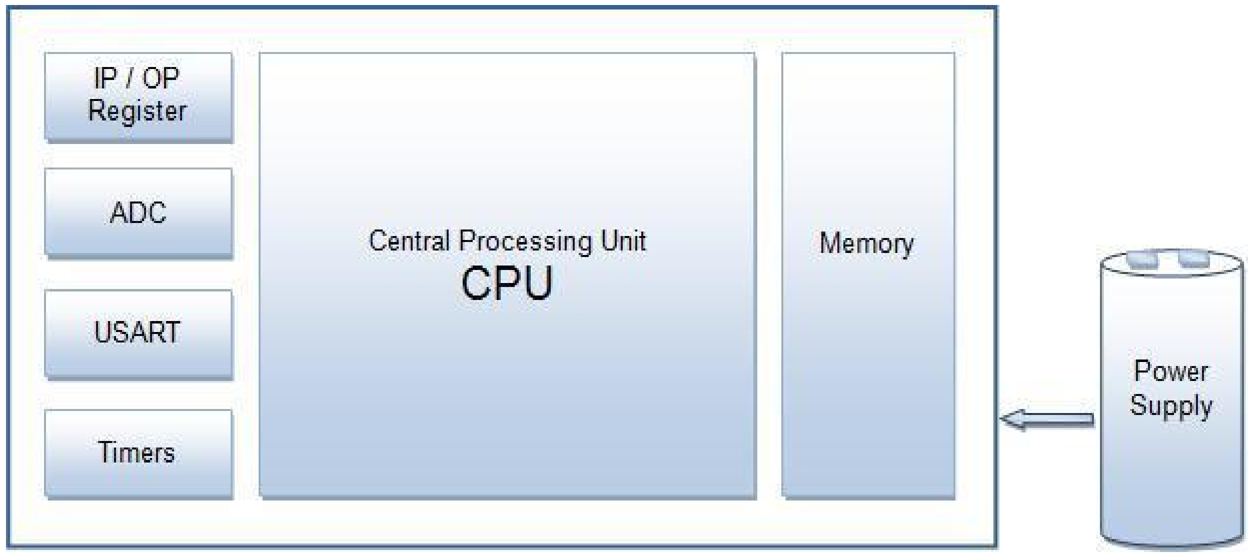


Figure: : Architecture of AVR Microcontroller

Does this mean that the microcontroller is another name for a computer…?
The answer is NO!

The computer on one hand is designed to perform all the general purpose tasks on a single machine like you can use a computer to run a software to perform calculations or you can use a computer to store some multimedia file or to access internet through the browser, whereas the microcontrollers are meant to perform only the specific tasks, for e.g., switching the AC off automatically when room temperature drops to a certain defined limit and again turning it ON when temperature rises above the defined limit.

There are number of popular families of microcontrollers which are used in different applications as per their capability and feasibility to perform the desired task, most common of these are 8051, **AVR** and PIC microcontrollers.

1.6 Difference between Microprocessor and Microcontroller:

A microcontroller differs from a microprocessor in many ways. The first and most important difference is its functionality. In order that the microprocessor may be used, other components such as memory must be added to it. Even though the microprocessors are considered to be powerful computing machines, their weak point is that they are not adjusted to communicating to peripheral equipment. Simply, In order to communicate with peripheral environment, the microprocessor must use specialized circuits added as
External chips, in short microprocessors are the pure heart of the computers. This is how it was in the beginning and remains the same today.

On the other hand, the microcontroller is designed to be all of that in one. No other specialized external components are needed for its application because all necessary circuits which otherwise belong to peripherals are already built into it. It saves the time and space needed to design a device.

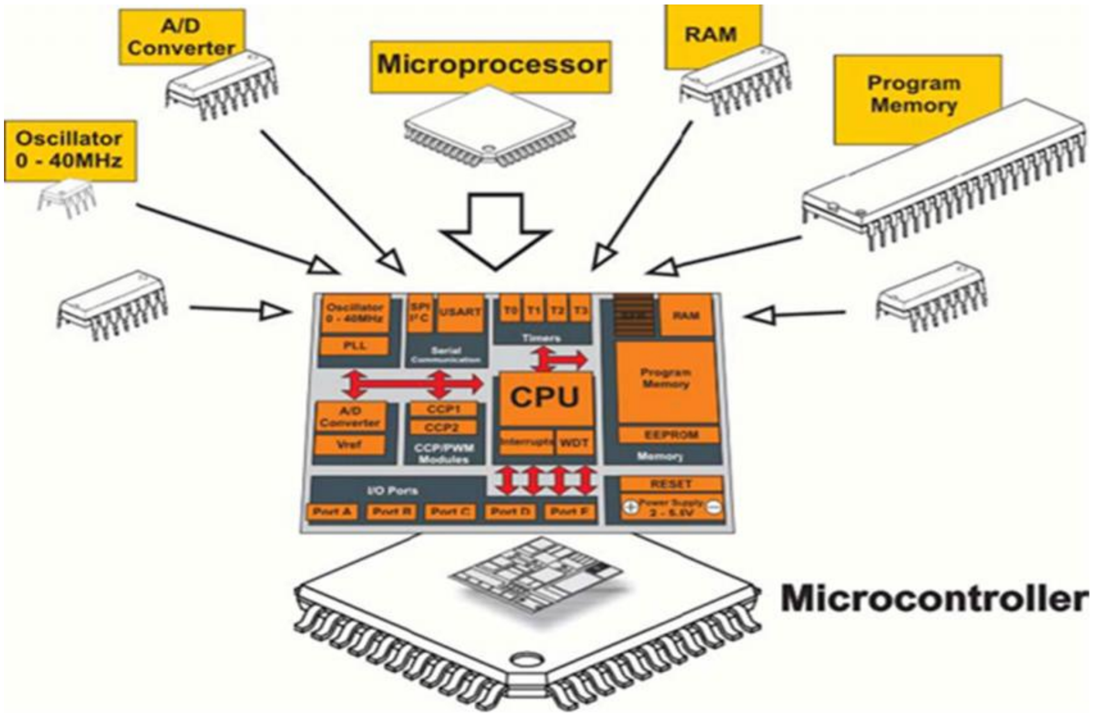


Figure: Different Between Microcontroller versus Microprocessor

Microprocessor = cpu
Microcontroller = cpu + peripherals + memory
Peripherals = ports + clock + timers + uarts + adc converters +lcd drivers + dac + other stuff.
Memory = eeprom + sram + eprom + flash
A microcontroller has a combination of all this stuff.
A microprocessor is just of cpu.

1.7 Memory Unit:

Memory is part of the microcontroller used for data storage. The easiest way to explain it is to compare it with a filing cabinet with many drawers. Suppose, the drawers are clearly marked so that it is easy to access any of them. It is easy enough to find out the contents of the drawer by reading the label on the front of the drawer.

Each memory address corresponds to one memory location. The content of any location becomes known by its addressing. Memory can either be written to or read from. There are several types of memory within the microcontroller.

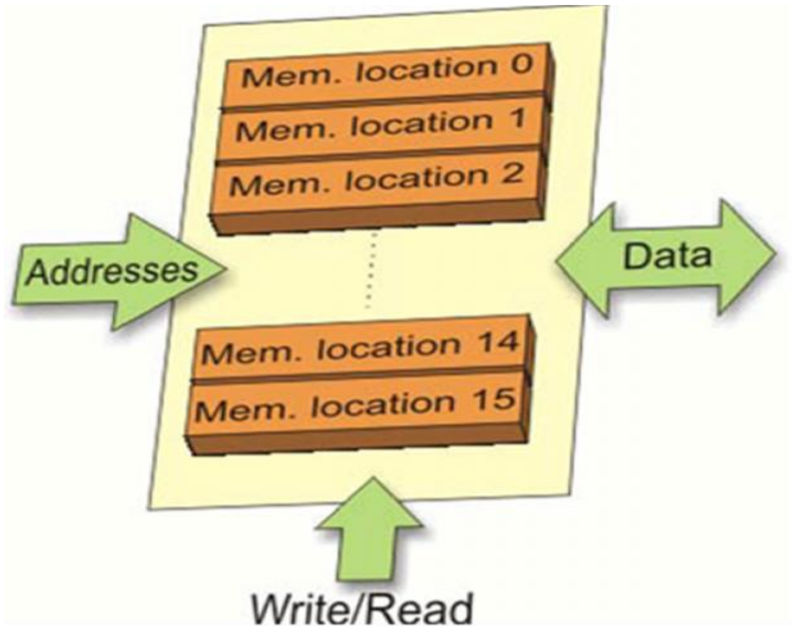


Figure: Memory Location

ROM (Read Only Memory) is used to permanently save the program being executed. The size of a program that can be written depends on the size of this memory. Today's microcontrollers commonly use 8-bit addressing, which means that they are able to address up to 64 Kb of memory, i.e. 65535 locations. As a novice, your program will rarely exceed the limit of several hundred instructions. There are several types of ROM. Exceed the limit of several hundred instructions. There are several types of ROM.

1.8 Interrupt:

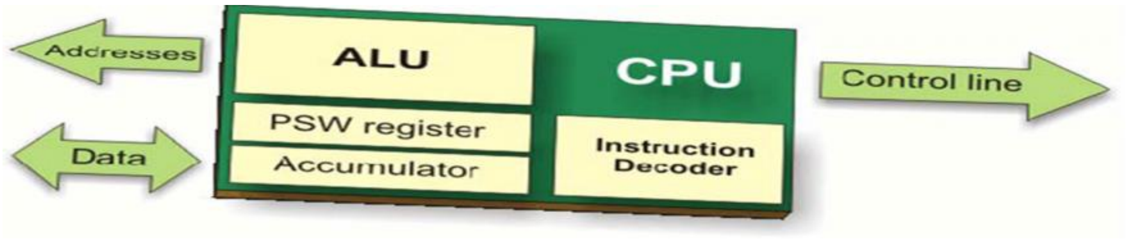


Figure : Central Processing Unit (CPU)

The most programs use interrupts in regular program execution. The purpose of the microcontroller is mainly to react on changes in its surrounding. In other words, when some event takes place, the microcontroller does something... For example, when you push a button on a remote controller, the microcontroller will register it and respond to the order by changing a channel, turn the volume up or down etc. If the microcontroller spent most of its time endlessly a few buttons for hours or days... It would not be practical. The microcontroller has learnt during its evolution a trick. Instead of checking each pin or bit constantly, the microcontroller delegates the "wait issue" to the "specialist" which will react only when something attention worthy happens. The signal which informs the central processor about such an event is called an INTERRUPT

1.9 CPU (Central Processor Unit):

As its name suggests, this is a unit which monitors and controls all processes inside the microcontroller. It consists of several smaller subunits, of which the most important are:

- Instruction Decoder is a part of the electronics which recognizes program instructions and runs other circuits on the basis of that. The "instruction set" which is different for each microcontroller family expresses the abilities of this circuit.

- Arithmetical Logical Unit (ALU) performs all mathematical and logical operations upon data.

- Accumulator is a SFR closely related to the operation of the ALU. It is a kind of working desk used for storing all data upon which some operation should be performed (addition, shift/move etc.). It also stores the results ready for use in further processing. One of the SFRs, called a Status Register (PSW), is closely related to the accumulator. It shows at any given moment the "status" of a number stored in the accumulator (number is greater or less than zero etc.).

1.10 Power Supply Unit:

There are two things worth attention concerning the microcontroller power supply circuit. Brown-out is a potentially dangerous state which occurs at the moment the microcontroller is being turned off or in situations when power supply voltage drops to the limit due to electric noise. As the microcontroller consists of several circuits which have different operating voltage levels, this state can cause its out-of-control performance. In order to prevent it, the microcontroller usually has built-in circuit for brown out reset. This circuit immediately resets the whole electronics when the voltage level drops below the limit.

Reset pin is usually marked as MCLR (Master Clear Reset) and serves for external reset of the microcontroller by applying logic zero (0) or one (1), depending on type of the microcontroller. In case the brown out circuit is not built in, a simple external circuit for brown out reset can be connected to this pin.

1.11 Timer/Counter:

The microcontroller oscillator uses quartz crystal for its operation. Even though it is not the simplest solution, there are many reasons to use it. Namely, the frequency of such oscillator is precisely defined and very stable; the pulses it generates are always of the same which makes them ideal for time measurement. Such oscillators are used in quartz watches. If it is necessary to measure time between two events, it is sufficient to count pulses coming from this oscillator. That is exactly what the timer does. Most programs use these miniature electronic "stopwatches".
These are commonly 8- or 8-bit SFRs and their content is automatically incremented by each coming pulse. Once a register is completely loaded - an interrupt is generated! If the timer registers use an internal quartz oscillator for their operation then it is possible to measure time between two events (if the register value is T1 at the moment measurement has started, and T2 at the moment it has finished, then the elapsed time is equal to the result of subtraction T2-T1). If the registers use pulses coming from external source then such a timer is turned into a counter**.**

This is only a simple explanation of the operation itself.

1.12 Counter:

If a timer is supplying pulses into the microcontroller input pin then it turns into a counter. Clearly, it is the same electronic circuit. The only difference is that in this case pulses to be counted come through the ports and their duration (width) is mostly not defined. This is why they cannot be used for time measurement, but can be used to measure anything else: products on an assembly line, number of axis rotation, passengers etc. (depending on sensor in use).

1.13 A/D Converter:

External signals are usually fundamentally different from those the microcontroller understands (Ones and Zeros), so that they have to be converted in order for the microcontroller to understand them. An analogue to digital converter is an electronic circuit which converts continuous signals to discrete digital numbers. This module is therefore used to convert some analogue value into binary number and forwards it to the CPU for further processing. In other words, this module is used for input pin voltage measurement (analogue value). The result of measurement is a number (digital value) used and processed later in the program.

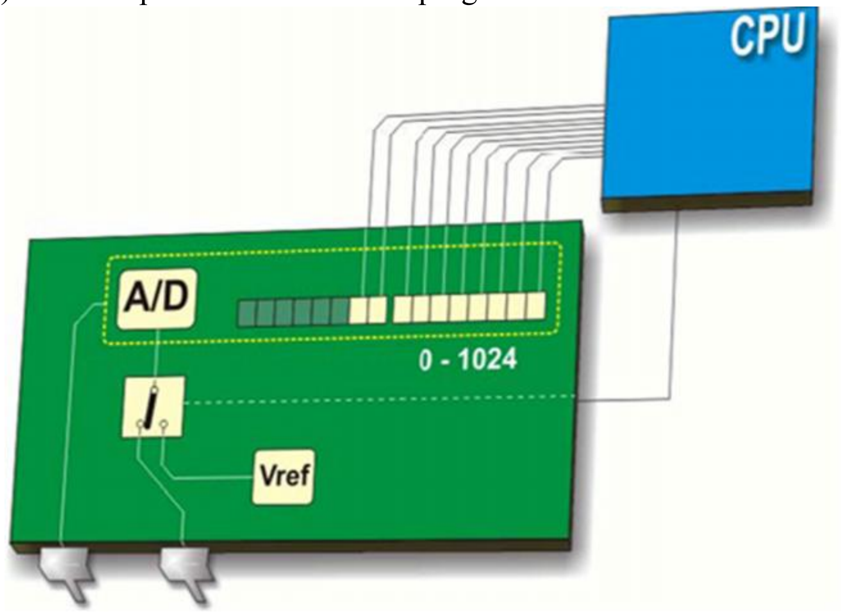


Figure-1.10: Connection Analogue to Digital Converter and CPU

1.14 RISC (Reduced Instruction Set Computer):

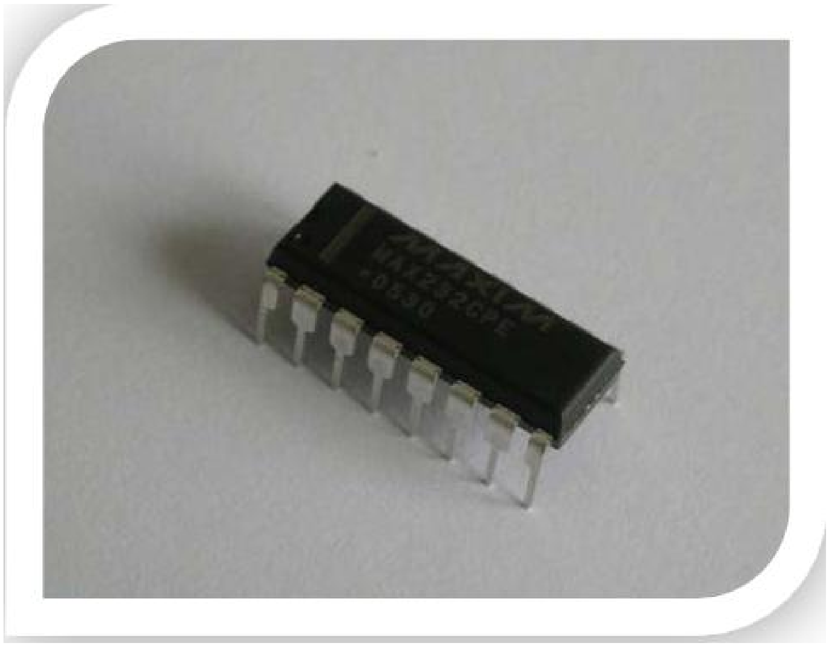
In this case, the microcontroller recognizes and executes only basic operations (addition, subtraction, copying etc.). All other more complicated operations are performed by combining these (for example, multiplication is performed by performing successive addition). The constrains are obvious (try by using only a few words, to explain to someone how to reach the airport in some other city). However, there are also some great advantages. First of all, this language is easy to learn.

Besides, the microcontroller is very fast so that it is not possible to see all the arithmetic "acrobatics" it performs. The user can only see the final result of all those operations. At last, it is not so difficult to explain where the airport is if you use the right words. For example: left, right, kilometers etc.

1.15 CISC (Complex Instruction Set Computer):

CISC is the opposite of RISC! Microcontrollers designed to recognize more than 200 different instructions can do much and are very fast. However, one needs to understand how to take all that such a rich language offers, which is not at all easy.

1.16 MAX232:



The MAX232 chip is a chip which functions as an interpreter between the computer and the AVR. It converts the signals from an RS-232 serial port (computer) to the RX, TX, and signals (AVR).

1.17 USBasp:

The USBasp is another interpreter though this one works over the computers USB port. It is easier than the MAX232 because this is a ready to use solution, whereas the MAX232 still needs to be connected to the rest of the board. The USBasp also has a status led which is very convenient

1.18 Wiring:

Well wires, these are cables that let through electricity. This can be as a power supply or as a matter of signals. This cable in the picture is the 10 pins connecter used to connect the USBasp with the 10 pins connecter on the protoboard.

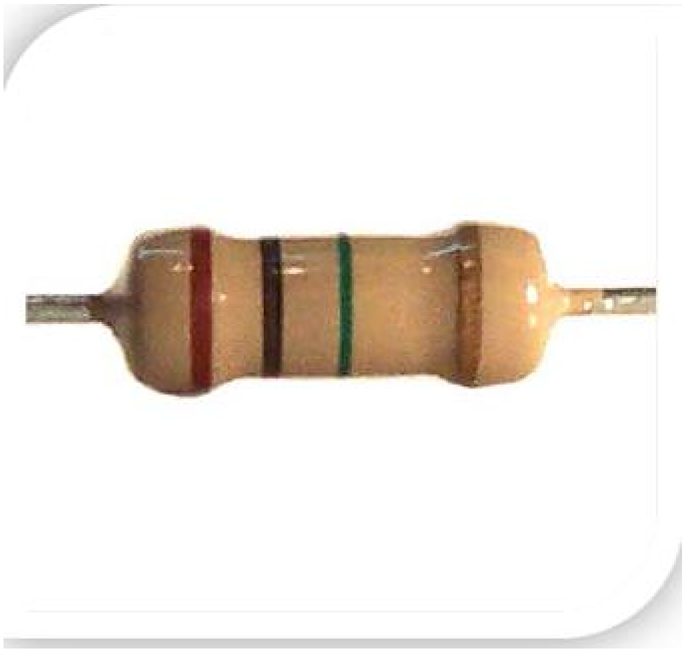
1.19 Capacitors:

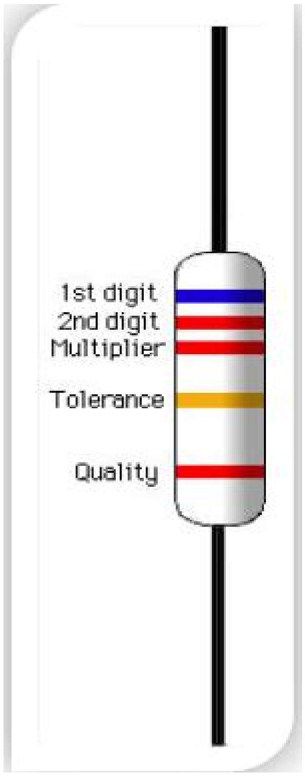
Capacitors are used to for many different things. But in this project they are mainly used as stabilizers for example between a ground and 5v wire. Otherwise the current might blow the microcontroller.

1.20 Resistors:

A resistor is a component that 'resists' the flow of electricity. The flow of electricity is called current. Every resistor has a certain value telling how much it resists the flow.

This resistance is called ohm, it's most commonly shown with the omega Ω.





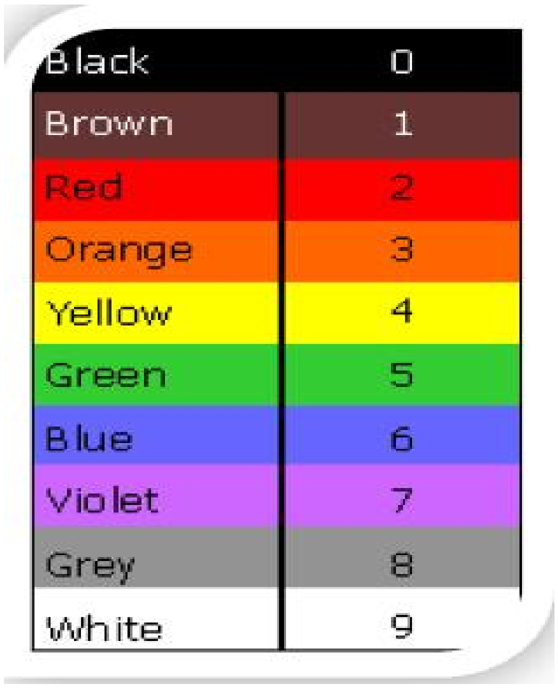
These resistors are needed on the breadboard to prevent a led from getting to much current. This would kill the LED. It's the same case with the Infrared LED.

Resistors are color coded. One gold, silver or blank ring indicating the tolerance the resistor can differ from the given value. And 3 rings which
Together make the value in ohm. The fifth ring isn't used to often.

Each color is a number varying from 1 to 9. See the table on the right.
For example when a resistor is in the above case blue, red, red it means the first digit is "6" the second digit is "2" and it must be multiplied by "2 x 10" or a hundred which would make it 6200 Ω.

Once such a resistor is made it will not change its value. The only thing which could happen is it gets burned when its limit is overridden by supplying a high enough voltage killing the entire thing. When this happens the resistor won't do anything anymore and can

be thrown away.
For example when a resistor is in the above case blue, red, red it means the first digit is "6" the second digit is "2" and it must be multiplied by "2 x 10" or a hundred which

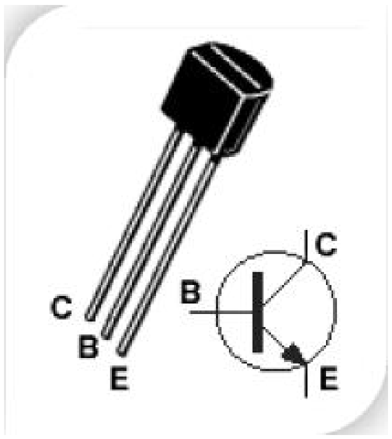


Black	0
Brown	1
Red	2
Orange	3
Yellow	4
Green	5
Blue	6
Violet	7
Grey	8
White	9

would make it 6200 Ω.

Once such a resistor is made it will not change its value. The only thing which could happen is it gets burned when its limit is overridden by supplying a high enough voltage killing the entire thing.

1.21 Transistor:



In this project the transistor is used merely as a switch to allow more current to pass through the circuit bypassing the AVR which cannot take this much current.

1.22 Crystal oscillator:

Crystals are used together with   microcontrollers to make them run faster. The standard clock speed (cycles per second) of the Atmega 16 series is 1 MHz but with our crystal oscillator it can run at 14.7456 MHz which makes it look much smoother.

A Crystal oscillator is an electronic circuit that uses mechanical resonance of a vibrating crystal to create an electrical signal with one exact frequency.

2.1 AVR Microcontroller:

AVR was developed in the year 1996 by Atmel Corporation. The architecture of AVR was developed by Alf-Egil Bogen and Vegard Wollan. AVR derives its name from its developers and stands for Alf-Egil Bogen Vegard Wollan RISC microcontroller, also known as Advanced Virtual RISC. The AT90S8515 was the first microcontroller which was based on AVR architecture however the first microcontroller to hit the commercial market was AT90S1200 in the year 1997.

2.2 AVR microcontrollers Categories:

AVR microcontrollers are available in three categories:

1. **TinyAVR** – Less memory, small size, suitable only for simpler applications
2. **MegaAVR** – These are the most popular ones having good amount of memory (upto 256 KB), higher number of inbuilt peripherals and suitable for moderate to complex applications.
3. **XmegaAVR** – Used commercially for complex applications, which require large program memory and high speed.

The following table compares the above mentioned AVR series of microcontrollers:

Series Name	Pins	Flash Memory	Special Feature
TinyAVR	6-32	0.5-8 KB	Small in size
MegaAVR	28-100	4-256KB	Extended peripherals
XmegaAVR	44-100	8-384KB	DMA , Event System included

2.3 What’s special about AVR?

They are fast: AVR microcontroller executes most of the instructions in single execution cycle. AVRs are about 4 times faster than PICs, they consume less power and can be operated in different power saving modes. Let’s do the comparison between the three most commonly used families of microcontrollers.

Descriptions	8051	PIC	AVR
SPEED	Slow	Moderate	Fast
MEMORY	Small	Large	Large
ARCHITECTURE	CISC	RISC	RISC
ADC	Not Present	Inbuilt	Inbuilt

AVR is an 8-bit microcontroller belonging to the family of Reduced Instruction Set Computer (**RISC**). In RISC architecture the instruction set of the computer are not only fewer in number but also simpler and faster in operation.

The other type of categorization is CISC (Complex Instruction Set Computers). We will explore more on this when we will learn about the architecture of AVR microcontrollers in following section.

Let's see what this entire means. What is 8-bit? This means that the microcontroller is capable of transmitting and receiving 8-bit data. The input/output registers available are of 8-bits. The AVR families controllers have register based architecture which means that both the operands for an operation are stored in a register and the result of the operation is also stored in a register. Following figure shows a simple example performing OR operation between two input registers and storing the value in Output Register.

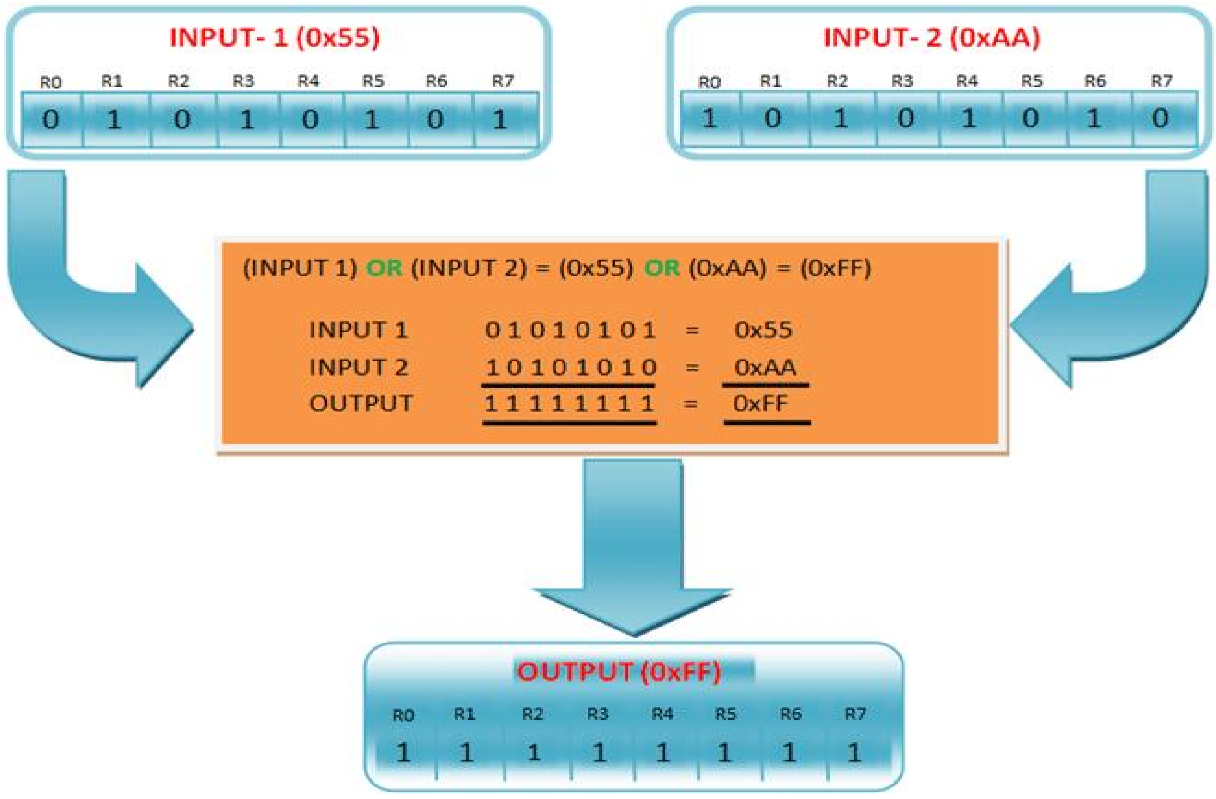


Figure: OR operation between two input registers and storing in Output Register

The CPU takes values from two input registers INPUT-1 and INPUT-2, performs the logical operation and stores the value into the OUTPUT register. All this happens in 1 execution cycle.

2.4 Features of Atmega16:

In our journey with the AVR we were working on microcontroller, which is a 40-pin IC and belongs to the mega AVR category of AVR family. Some of the features of Atmega8 are:

- 8KB of Flash memory
- 1KB of SRAM
- 512 Bytes of EEPROM
- Available in 40-Pin DIP
- 8-Channel 10-bit ADC
- Two 8-bit Timers/Counters
- One 8-bit Timer/Counter
- 4 PWM Channels
- In System Programmer (ISP)
- Serial USART
- SPI Interface
- Digital to Analog Comparator.

2.5 Architecture of AVR:

The AVR microcontrollers are based on the advanced RISC architecture and consist of 32 x 8-bit general purpose working registers. Within one single clock cycle, AVR can take inputs from two general purpose registers and put them to ALU for carrying out the requested operation, and transfer back the result to an arbitrary register.

The ALU can perform arithmetic as well as logical operations over the inputs from the register or between the register and a constant. Single register operations like taking a complement can also be executed in ALU. We can see that AVR does not have any register like accumulator as in 8051 family of microcontrollers; the operations can be performed between any of the registers and can be stored in either of them.

AVR follows Harvard Architecture format in which the processor is equipped with separate memories and buses for Program and the Data information. Here while an instruction is being executed, the next instruction is pre-fetched from the program memory.

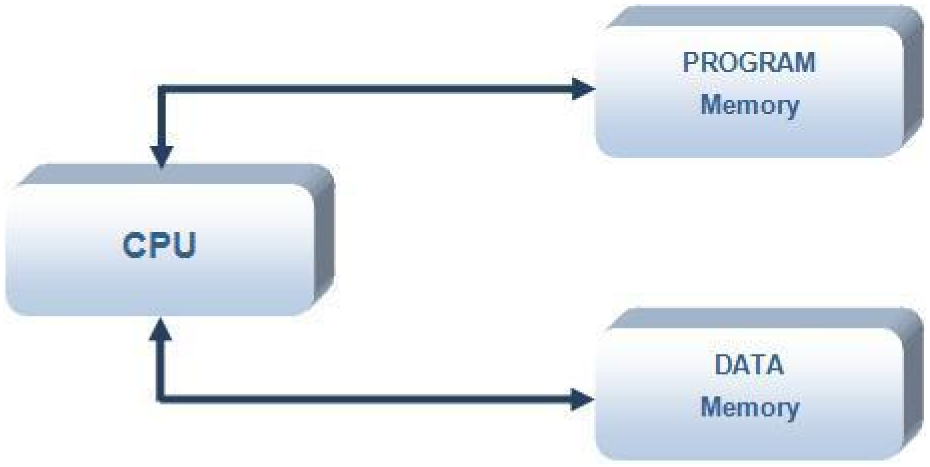


Figure-2.2: AVR Architecture Format

Since AVR can perform single cycle execution, it means that AVR can execute 1 million instructions per second if cycle frequency is 1MHz. The higher is the operating frequency of the controller, the higher will be its processing speed. We need to optimize the power consumption with processing speed and hence need to select the operating frequency accordingly.

2.6Atmel At mega 16 AVR:

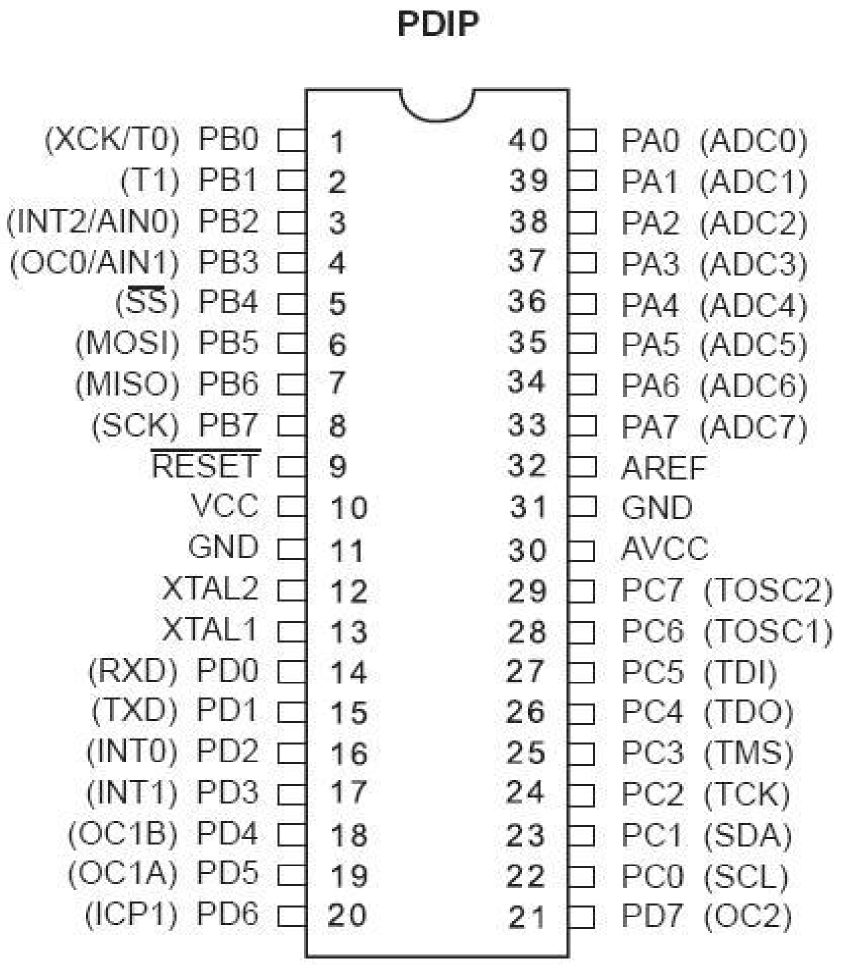
The ATmega16 microcontroller used in this lab is a 40-pin wide DIP (Dual In Line) package chip. This chip was selected because it is robust, and the DIP package interfaces with proto supplies like solderless bread boards and solder-type perf-boards. This same microcontroller is available in a surface mount package, about the size of a dime. Surface mount devices are more useful for circuit boards built for mass production. Figure 1 below shows the 'pin-out' diagram of the ATmega16. This diagram is very useful, because it tells you where power and ground should be connected, which pins tie to which functional hardware, etc. ATmega16 Pin-out diagram. Notice that some of the pins have alternate functions (shown in parentheses).

Throughout the semester, you will need to know things about the ATmega16 (or other components) That are not covered in the lab instructions. Therefore it is important that you become familiar with Documentation available from various sources. Your first task is to locate the ATmega16 Manual, and save it for yourself for future reference. It can be found in a pdf format on Atmel's

Website (www.atmel.com), AVR Freaks (http://www.avrfreaks.net/), or by searching the web. From the manual, you can find information about the ATmega16's features and how to use them. The At mega series has become very popular as it was one of the first to use on-chip flash memory which could be rewritten infinite amount of times whereas other microcontrollers used EEPROM (One time programmable ROM). It kept being popular in homebuilt projects which involve microcontrollers. This is mainly because it's still a cheap series of microcontrollers with lots of functions, but it's also because they're relatively easy to program.

I chose this exact microprocessor merely because I could stick to the previously made program. (at the time being I wasn't sure I was going to be able to rewrite the program, or even design a completely new one

2.7 Pin Diagram of AT mega 16 Microcontroller:



This is the Pin Diagram of of AT mega 16 Microcontroller

3.1 Working Procedure:

How to control 64 LEDs without using 64 individual wires Multiplexing!
Running a wire to the anode of each led would obviously be impractical, and would look really bad. One way to get around this, is to split the cube into 4 layers of 16x16 LEDs.

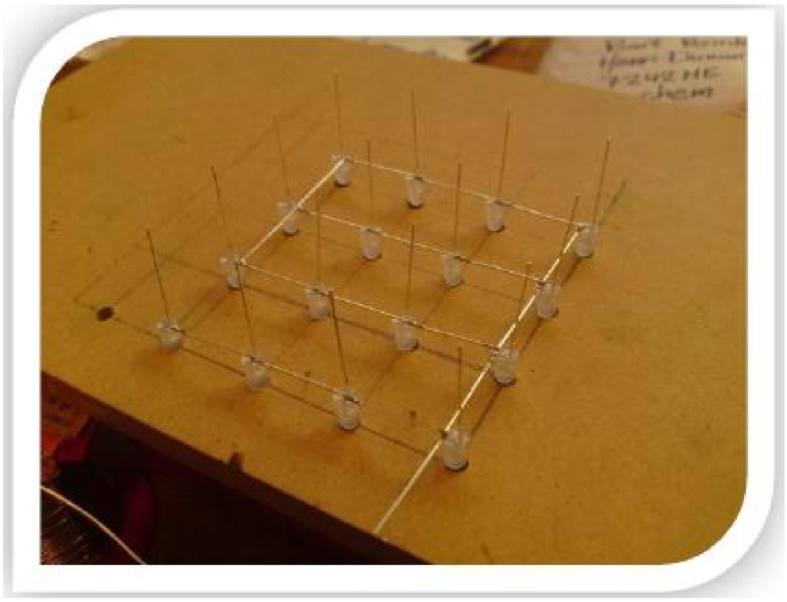
All the LEDs aligned in a vertical column share a common anode (+).
All the LEDs on a horizontal layer share a common cathode (-).

Now if i want to light up the LED in the upper left corner in the back (0, 0, 3), I just supply GND (-) to the upper layer, and VCC (+) to the column in the left corner. If I only want to light up one led at a time, or only light up more than one layer at the same time. This works fine.

However, if I also want to light up the bottom right corner in the front (3, 3, 0), I run into problems. When I supply GND to the lower layer and VCC to the front left column, I also light up the upper right led in the front (3,3,3), and the lower left LED in the back (0,0,0). This ghosting effect is impossible to work around without adding 64 individual wires.

The way to work around it is to only light up one layer at a time, but do it so fast that the eye doesn't recognize that only one layer is lit at any time. This relies on a phenomenon called Persistence of vision. Each layer is a 4x4 (16) image. If we flash 4 16 led images one at a time, really fast, we get a 4x4x4 3d image

3.2 Assembly of the led's:



Take a piece of wood and make holes of the size of your led's, in my case 5mm place the holes at pins length of each other. Do this very accurately! If you have one hole in the wrong place you'll have it wrong in four different layers.

Put your led's one by one in the holes bend one leg so it touches the leg of the other led and solder them together. Do this for every led till you've got all four levels finished.

When you've got them all it should look something like this. Then you'll have to solder the different layers together finishing the cube itself.

Now that we've done the easy things were going to focus on the real electronics behind this cube. I'm not going to describe how I've connected each and every wire because that would be boring. What you're trying to do is connect every wire to the correct pin. The correct way is shown on the next page.

Important is that before you start this you should make sure your desk is clean and tidy! Otherwise you'll soon lose track.

The circuit shown on the next page was made by me. I did this as this uses the physical connection of the AVR whereas
The other circuit showed the schematic connections.

3.3 Wiring is done:

What to do now? Well that’s easy you’ve probably discovered that your led cube is not yet lighting up. Well that’s because you’ll need a 5v power supply. This could be anything, I’ve used a cell phone charger as this was available to me.

You take cut of the connecter and strip the wire for about 2 cm’s. Depending on the brand of charger you’ll find several wires. Most of the time you’ll gonna want to use the black and red one. Being the
Black ground and the red one the 5V. Connect these two to the corresponding place on the protoboard and now it should be finished.

Mind you there is always the troubleshooting. In most cases you will have connected one wire the wrong way. Or not at all go through all the wires step by step and part by part if the cube won’t react when you try to reach it in the next step.

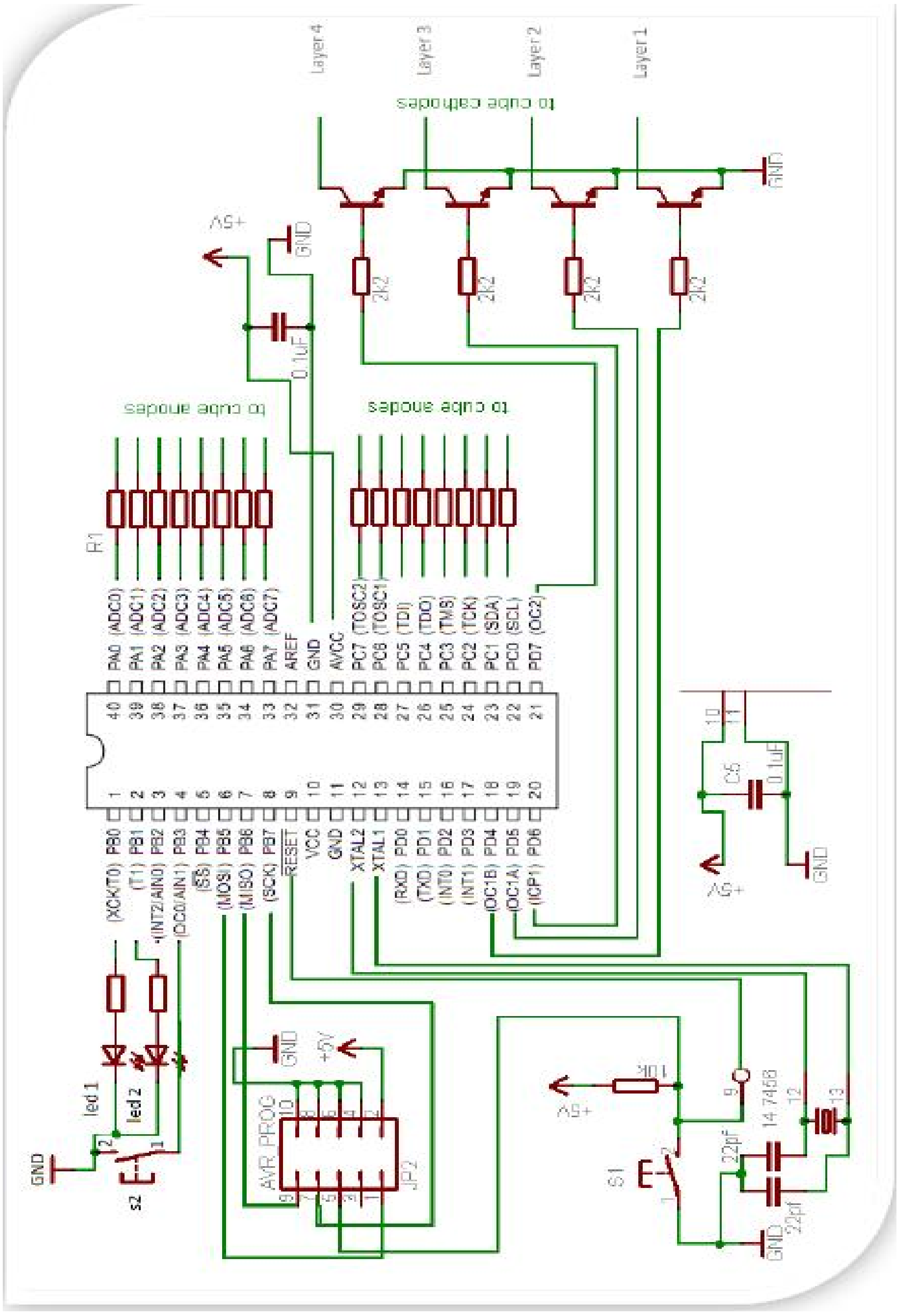
3.4 Chief selection:

The fast task is to select the chip. Here for our work we select the chip is ATmega8 and we also determined the clock 16.000000 MHz Figure 01, below, shows the new AVR project window within BASCOM AVR

3.5 Enabling Analogue to Digital Conversion:

Enabling the ADC is needed for use in the control functions of the device. Using the ADC is also made simpler by using BASCOMAVR. Enabling ADC on the MCU is achieved by ticking the ADC Enabled selection in BASCOMAVR, and then selects the use 8 bit option and then selects the voltage reference as the voltage across the AREF pin. BASCOMAVR also gives an option for the ADC speed, and in this case it was chosen to be the fastest available, that being 1000.000 kHz.

4.1Circuit Diagram:

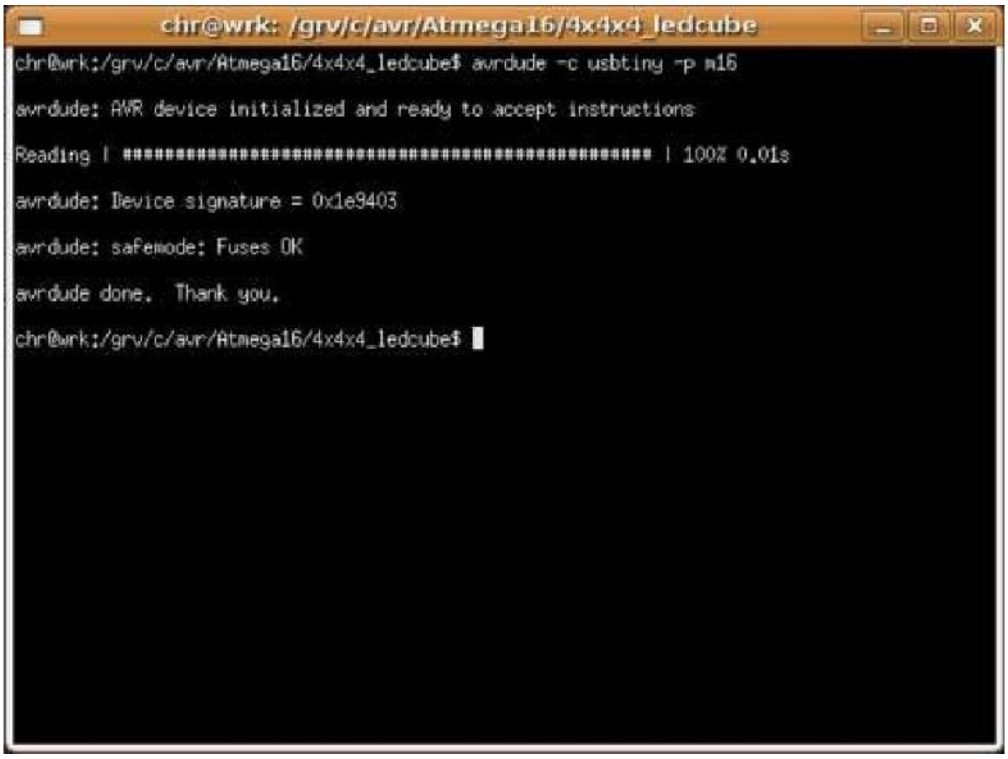


4.2 Programming:

Programming takes up lots of time. You'll need to think of exactly how you want the program to run. And you'll need to tell (write down) every step you want it to do. This can be done in multiple languages though "C" is the easiest for our purpose.

I've not yet learned to master this language and therefor it was impossible to design my own program. I was however able to edit the current program more to my liking. I'll soon go on and will make my own patterns. It will need some time tough as there is a current problem programming through USBasp with a 64-bit operating system.

To flash the program to the led cube you'll need a program that allows you to write data to the flash memory of the atmega16 on the led cube. This can be done by using AVRdude. AVRdude is a command line tool which takes some time to get used to.



```
chr@wrk:/grv/c/avr/Atmega16/4x4x4_ledcube$ avrdude -c usbtiny -p m16

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e9403

avrdude: safemode: Fuses OK

avrdude done.  Thank you.

chr@wrk:/grv/c/avr/Atmega16/4x4x4_ledcube$
```

This chapter discusses how the software for this thesis was implemented into the device. The process undertaken to reach a software solution which meets the specifications.

4.3 Program Development:

The Firmware for the controller was developed with the aid of AVR Studio, which is a program that allows the user to write and compile BASIC programs for the Atmel AVR range of micro-controllers. It also allows for in-system programming thus making an ideal software development tool.

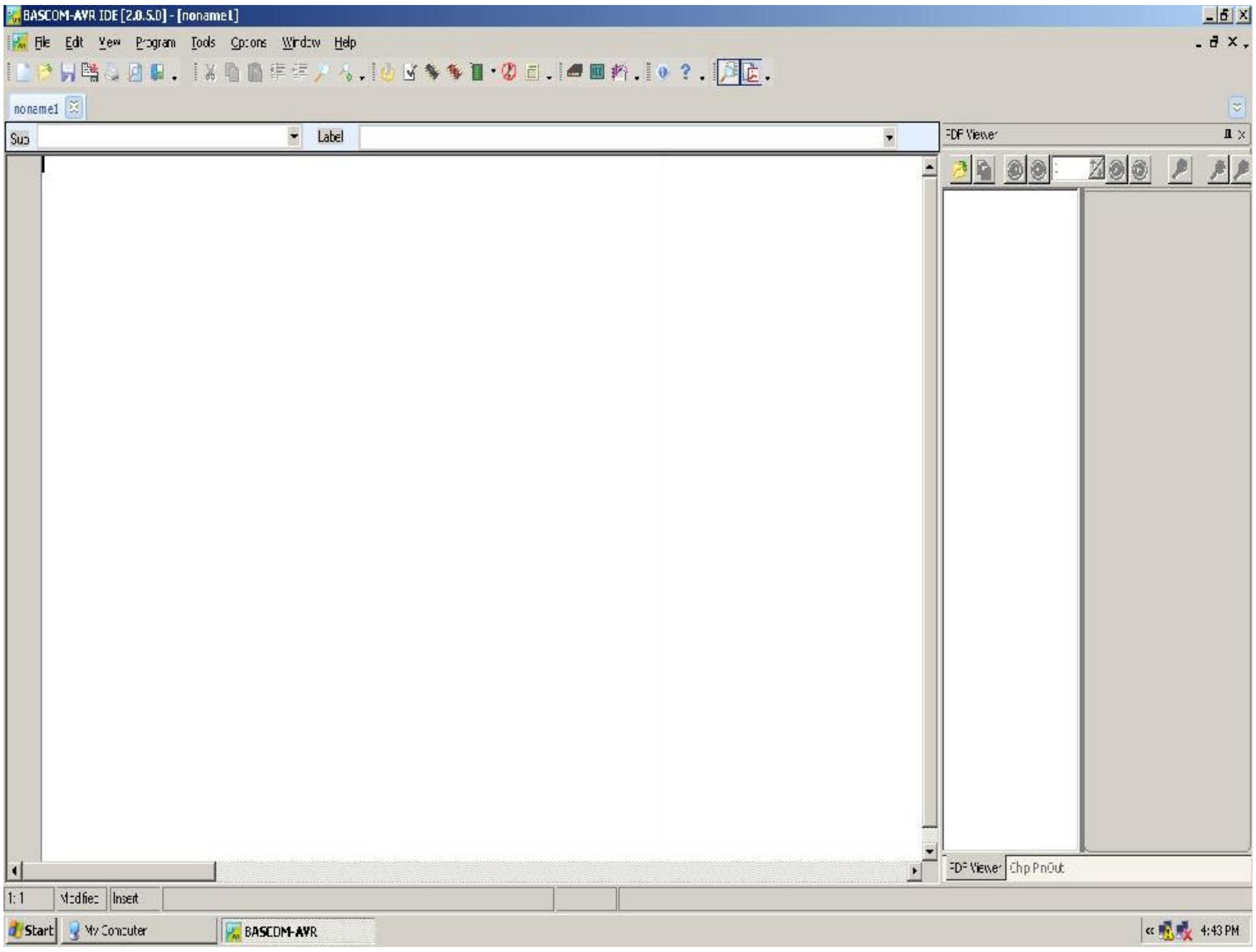
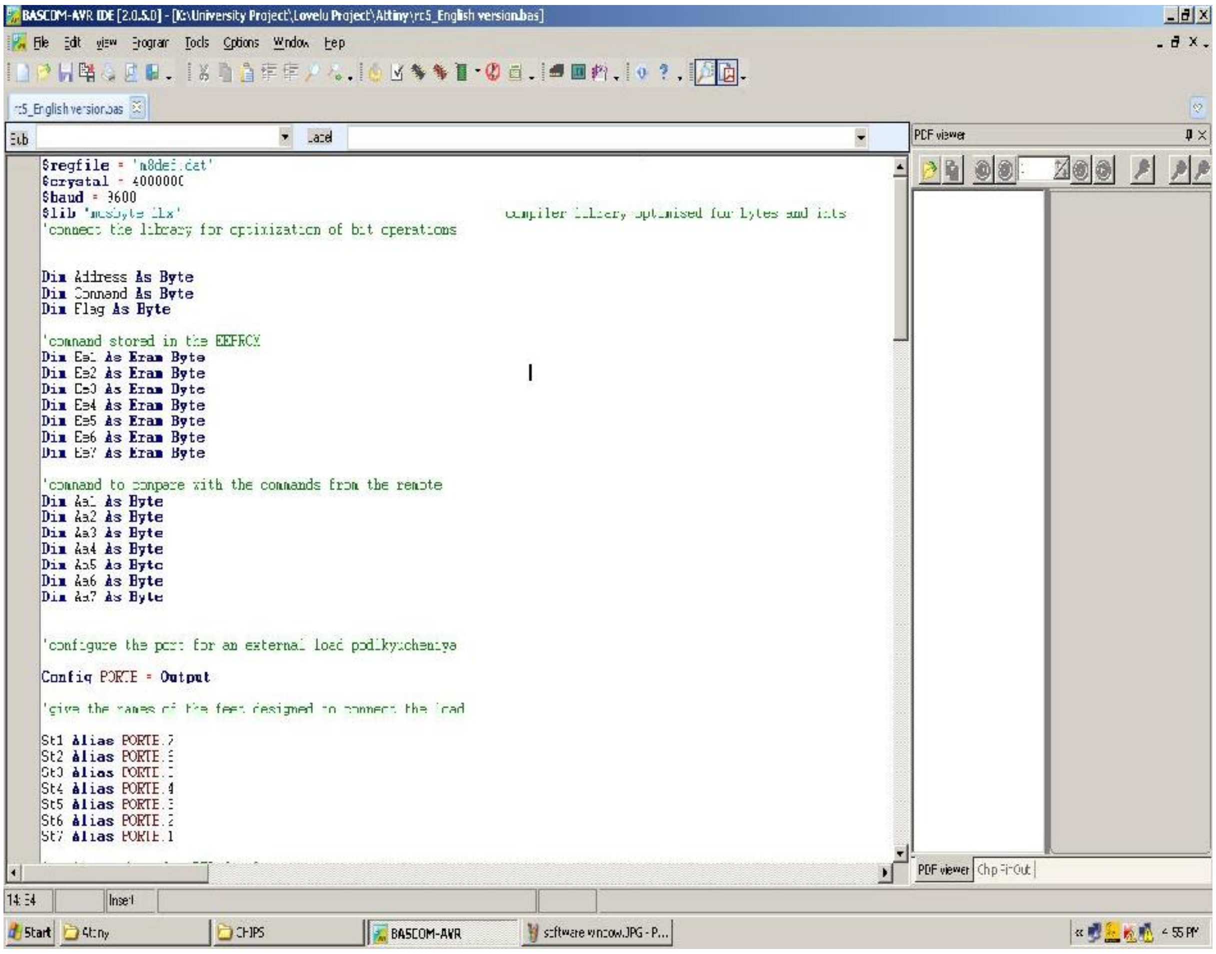


Figure: AVR Studio microcontroller development system.


BASCOM-AVR IDE [2.0.5.0] - [K:\University Project\Lovelu Project\Attiny\rc5_English version.bas]
File Edit View Program Tools Options Window Help
rc5_English version.bas
Sub
Label
PDF viewer
PDF viewer Chip PinOut
14:34
Insert
Start Attiny CHIPS BASCOM-AVR software window.JPG - P...

```
$regfile = 'm8def.dat'
$crystal = 4000000
$baud = 9600
$lib 'mcsbyte.lbx'                                  'compiler library optimised for bytes and ints
'connect the library for optimization of bit operations


Dim Address As Byte
Dim Command As Byte
Dim Flag As Byte

'command stored in the EEPROM
Dim Ee1 As Eram Byte
Dim Ee2 As Eram Byte
Dim Ee3 As Eram Byte
Dim Ee4 As Eram Byte
Dim Ee5 As Eram Byte
Dim Ee6 As Eram Byte
Dim Ee7 As Eram Byte

'command to compare with the commands from the remote
Dim Aa1 As Byte
Dim Aa2 As Byte
Dim Aa3 As Byte
Dim Aa4 As Byte
Dim Aa5 As Byte
Dim Aa6 As Byte
Dim Aa7 As Byte


'configure the port for an external load podklyuchenlya

Config PORTE = Output

'give the names of the feet designed to connect the load

St1 Alias PORTE.7
St2 Alias PORTE.6
St3 Alias PORTE.5
St4 Alias PORTE.4
St5 Alias PORTE.3
St6 Alias PORTE.2
St7 Alias PORTE.1
```

Figure: AVR Studio Window Showing Chip selection Options

4.4 Output Three Dimensional Display:

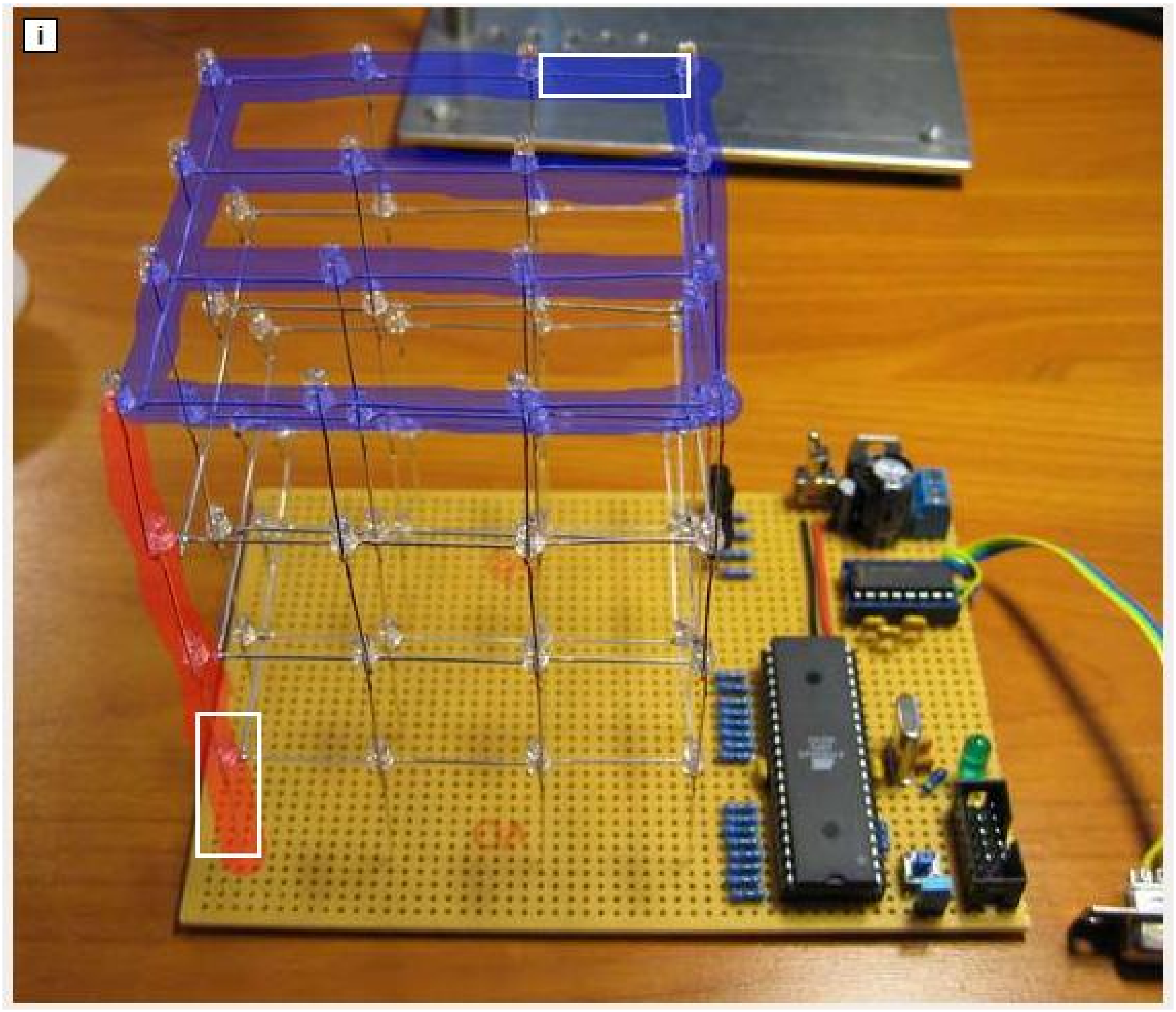


Figure: This is the Output Three Dimensional Display.

4.5 Final Code:

```
Chip Type: ATmega 16

Program Type: Application

AVR Clock frequency: 8.000000 MHz

Memory model: 0

External RAM size: 0

Data Stack size: 8.38KB

******************************************************************************/

// ###############################################

//

// 4x4x4 LED Cube project

// By Mahadee Jamil

#include <avr/io.h>

#include <avr/interrupt.h>

#include <avr/pgmspace.h>


// Define USART stuff

// CPU speed and baud rate:

#define FOSC 14745600

#define BAUD 9600

// Are used to calculate the correct USART timings

#define MYUBRR (((((FOSC * 10) / (16L * BAUD)) + 5) / 10) - 1)



// define masks used for status LEDs and input buttons.

#define LED_GREEN 0x01
```

```
#define LED_RED 0x02
#define BUTTON 0x08
// define port used for status and input.
#define LED_PORT PORTB
#define BUTTON_PORT PORTB


// define masks for the layer select.
#define LAYER1 0x80
#define LAYER2 0x40
#define LAYER3 0x20
#define LAYER4 0x10
#define LAYERS 0xf0 // All layers
#define LAYERS_R 0x0f // the inverse of the above.
#define LAYER_PORT PORTD

// define LED grid ports
// each of the grid ports are connected to two rows of leds.
// the upper 4 bits is one row, the lower 4 bits are one row.
#define GRID1 PORTC
#define GRID2 PORTA


void ioinit (void); // initiate IO on the AVR
void bootmsg (void); // blink some leds to indicate boot or reboot
void delay_ms (uint16_t x); // delay function used throughout the program
void led_red(unsigned char state); // led on or off
void led_green(unsigned char state);
void launch_effect (int effect); // effect program launcher


// *** Cube buffer ***
// The 3D image displayed on the cube is buffered in a 2d array 'cube'.
// The 1st dimension in this array is the Z axis of the cube.
// The 2nd dimension of the array is the Y axis.
// Each byte is a stripe of leds running along the X axis at the given
// Z and Y coordinates.
// Only the 4 lower bits are used, since the cube is only 4x4x4.
// This buffer design was chosen to have code compatability with a 8x8x8 cube.
// "volatile" makes the variables reachable from within the interrupt functions
volatile unsigned char cube[4][4];


// We sometimes want to draw into a temporary buffer so we can modify it
// before writing it to the cube buffer.
// e.g. invert, flip, reverse the cube..
volatile unsigned char tmpcube[4][4];


// What layer the interrupt routine is currently showing.
volatile unsigned char current_layer;


// Low level geometric functions
#include "draw.c"


// Static animation data
#include "frames.c"
```

```
// Fancy animations to run on the cube
#include "effect.c"



int main (void)
{
        // Initiate IO ports and peripheral devices.
        ioinit();

        // Indicate that the device has just booted.
        bootmsg();

        int x;
        int i;
        int z;

        // Set the layer to start drawing at
        current_layer = 0x00;

        // Enable interrupts to start drawing the cube buffer.
        // When interrupts are enabled, ISR(TIMER2_COMP_vect)
        // will run on timed intervalls.
        sei();

        // Main program loop.
        while (1)
        {
            for (i=0;i<13;i++)
            {
                launch_effect(i);
            }
            // Comment the loop above and uncomment the line below
            // if you want the effects in random order (produced some bugs.. )
            //launch_effect(rand()%13);
        }

}

// Launches one of those fancy effects.
void launch_effect (int effect)
{
        switch (effect)
        {
            // Lights all the layers one by one
            case 0:
                loadbar(1000);
                break;

            // A pixel bouncing randomly around
            case 1:
                // blink
                boingboing(150,500,0x03,0x01);
                break;
```

```
// Randomly fill the cube
// Randomly empty the cube
case 2:
        fill(0x00);
        random_filler(100,1,500,1);
        random_filler(100,1,500,0);
        break;

// Send voxels randomly back and forth the Z axis
case 3:
        sendvoxels_rand_z(150,500,2000);
        break;

// Spinning spiral
case 4:
        effect_spiral(1,75,1000);
        break;

// A coordinate bounces randomly around the cube
// For every position the status of that voxel is toggled.
case 5:
        // toggle
        boingboing(150,500,0x03,0x02);
        break;

// Random raindrops
```

```
case 6:
        effect_rain(20,5000,3000,500);
        break;

// A snake randomly bounce around the cube.
case 7:
        // snake
        boingboing(150,500,0x03,0x03);
        break;

// Spinning plane
case 8:
        effect_spinning_plane(1,50,1000);
        break;

// set x number of random voxels, delay, unset them.
// x increases from 1 to 20 and back to 1.
case 9:
        random_2();
        break;

// Set all 64 voxels in a random order.
// Unset all 64 voxels in a random order.
case 10:
        random_filler2(200,1);
        delay_ms(2000);
        random_filler2(200,0);
```

```
                delay_ms(1000);
                break;

            // bounce a plane up and down all the directions.
            case 11:
                flyplane("z",1,1000);
                delay_ms(2000);
                flyplane("y",1,1000);
                delay_ms(2000);
                flyplane("x",1,1000);
                delay_ms(2000);
                flyplane("z",0,1000);
                delay_ms(2000);
                flyplane("y",0,1000);
                delay_ms(2000);
                flyplane("x",0,1000);
                delay_ms(2000);
                break;

            // Fade in and out at low framerate
            case 12:
                blinky2();
                break;
        }
}

// ** Diagnostic led functions **
// Set or unset the red LED
void led_red(unsigned char state)
{
        if (state == 0x00)
        {
            LED_PORT &= ~LED_RED;
        } else
        {
            LED_PORT |= LED_RED;
        }
}

// Set or unset the green LED
void led_green(unsigned char state)
{
        if (state == 0x00)
        {
            LED_PORT &= ~LED_GREEN;
        } else
        {
            LED_PORT |= LED_GREEN;
        }
}


// Cube buffer draw interrupt routine
```

```
ISR(TIMER2_COMP_vect)
{

        // AND the reverse bitmask onto the layer port.
        // This disables all the layers. rendering all the leds off.
        // We don't want to see the cube updating.
        LAYER_PORT &= LAYERS_R;

        // Take the current 2D image at the current layer along the Z axis
        // and place it on the LED grid.
        GRID1 = (0x0f & cube[current_layer][0]) | (0xf0 & (cube[current_layer][1] << 4));
        GRID2 = (0x0f & cube[current_layer][2]) | (0xf0 & (cube[current_layer][3] << 4));

        // Enable the apropriate layer
        LAYER_PORT |= (0x01 << (7 - current_layer));

        // The cube only has 4 layers (0,1,2,3)
        // If we are at layer 3 now, we want to go back to layer 0.
        if (current_layer++ == 3)
            current_layer = 0;
}

void ioinit (void)
{
        // ### Initiate I/O

        // Data Direction Registers
        // Bit set to 1 means it works as an output
        // Bit set to 1 means it is an input
        DDRA = 0xff;        // Inner cube byte
        DDRB = 0xf7;        // ISP and 0-1: led. 3: button
        DDRC = 0xff;        // Outer cube byte
        DDRD = 0xff;        // Layer select

        // Set all ports OFF, and enable pull up resistors where needed.
        PORTA = 0x00;
        PORTC = 0x00;
        PORTB = 0x08; // Enable pull up button.
        PORTD = 0x00;

        // ### Initiate timers and USART

        // Frame buffer interrupt
        TCNT2 = 0x00;       // initial counter value = 0;
        TIMSK |= (1 << OCIE2); // Enable CTC interrupt

        // Every 1024th cpu cycle, a counter is incremented.
        // Every time that counter reaches 15, it is reset to 0,
        // and the interrupt routine is executed.
        // 14745600/1024/15 = 960 times per second
        // There are 4 layers to update..
        // 14745600/1024/15/4 = 240 FPS
        // == flicker free :)
        OCR2 = 15;                      // interrupt at counter = 15
```

```
        TCCR2 = 0x05;                   // prescaler = 1024
        TCCR2 |= (1 << WGM01); // Clear Timer on Compare Match (CTC) mode
        // Initiate RS232
        // USART Baud rate: 9600
        UBRRH = MYUBRR >> 8;
        UBRRL = MYUBRR;
        // UCSR0C - USART control register
        // bit 7-6      sync/ascyn 00 = async,  01 = sync
        // bit 5-4      parity 00 = disabled
        // bit 3        stop bits 0 = 1 bit  1 = 2 bits
        // bit 2-1      frame length 11 = 8
        // bit 0        clock polarity = 0
        UCSRC  = 0b10000110;
        // Enable RS232, tx and rx
        UCSRB = (1<<RXEN)|(1<<TXEN);
        UDR = 0x00; // send an empty byte to indicate powerup.

}



// Blink the status LEDs a little to indicate that the device has just booted.
// This is usefull to see if an error is making the device reboot when not supposed to.
// And it looks cool.
void bootmsg (void)
{
        int i;
LED_PORT |= LED_GREEN;
```

```
        for (i = 0; i < 2; i++)
        {
            // Blinky
            delay_ms(1000);
            LED_PORT &= ~LED_GREEN;
            LED_PORT |= LED_RED;
            // Blink
            delay_ms(1000);
            LED_PORT &= ~LED_RED;
            LED_PORT |= LED_GREEN;
        }
        delay_ms(1000);
        LED_PORT &= ~LED_GREEN;
}
// Delay function used in graphical effects.
void delay_ms(uint16_t x)
{
  uint8_t y, z;
  for ( ; x > 0 ; x--){
    for ( y = 0 ; y < 90 ; y++){
      for ( z = 0 ; z < 6 ; z++){
        asm volatile ("nop");
      }
    }
  }
}
```

5.1Conclusion:

I’ve created a led cube which works and does exactly what the one on intractable net does. I think it looks pretty too. I’m quite happy with it even though I’ve not been able to program different behavior yet. I hope to be able to do this later.

5.2 Future Developments and Improvements:

There are a number of ways in which this device could be improved in any future development.

In future we want to add humidity sensor to measure the humidity at room.

Humidity sensor needed to control the humidity at room.

Temperature sensor needed to control the temperature.

More efficient software would also benefit the device and the means of upgrading the software on the MCU has been catered for our project.

Using Triac instead relay that can be provided low power consume and given the more efficiency. Reducing the power losses in the system would increase the efficiency of the device .In future we use 4*3 keypad for password based door lock protection.

One last major improvement would be to use as many surface mounted components as possible, particularly using a surface mounted MCU and resistors and as many other components that could be replaced with surface mounted equivalents. Future we will design a better & compact PCB and good quality circuit by using surface mounted component.

5.3 Discussion:

Even though I’m completely happy about the whole project I reckon there are things I could’ve done better.

Like:

- I could’ve let someone recheck the way I soldered the lets together, this would have saved me multiple hour of work and troubleshooting.
- I should have used multiple colors of wires making troubleshooting easier.
- I should have bought the USBasp directly as the serial ports are really outdated and are therefore not supported anymore.
- I should have made the report directly after the project or even while working on the project.

Reference:

http://www.instructables.com/id/LED-Cube-4x4x4/

http://www.dickbest.nl/webshop/index.php

http://www.topled.nl/

http://www.circuitsonline.net/schakelingen/112/computer-en-microcontroller/avr-programmer.html

http://www.zegeniestudios.net/ldc/