Report on

**Industrial Training 2015**

Submitted to

Chairperson

Department of Electrical, Electronics and Communication Engineering

Submitted by: - Abhinav Ghai

Roll No.: – 13ECU002

Branch: – ECE

Section: - 5A (E1)

**Acknowledgements**

The opportunity of training at daKine Technologies Pvt. Ltd was great chance to learn and develop my professional skills. I am extremely overwhelmed to be a part of it. I am also grateful for having a chance to meet wonderful professionals who led me through this training and guided me for future.

I express my thanks to Mr. Vineet Kumar Maheshwari, Director of DKT for taking part in useful decision & giving necessary advices and guidance and arranged all facilities to make life easier. I choose this moment to acknowledge his contribution gratefully.

I give my deepest gratitude to my father Dr. Punit Ghai and mother Ms. Meenu Ghai for their blessings and moral support.

I regard this opportunity as a big milestone in my professional and career development. I will make great efforts to implement gained skills and knowledge in the best possible way.

Hope to continue cooperation with all of you in the future.

Abhinav Ghai

**Table of Contents**

1. Abstract
2. Introduction
3. Learning Linux
4. Command: make
5. Understanding Variables
6. Object Oriented Design
7. Markdown
8. Learning Microcontroller
9. Learning Atmel ATmega 328
10. Learning Photo sensor
11. Learning Shift Register
12. Introduction to Arduino Uno
13. Learning Procurement
14. Industrial Visit
15. Project
    1. Implementation
    2. Initial Logic
    3. Implementation on Hardware
    4. Working on PCB, Hard Wired

* Circuit for ATmega 328
* Circuit for Traffic Sensor
* Circuit for LCD
* Circuit for Traffic LEDs
* Circuit for Power
* Extracts of Code and Explanation

**Abstract**

6 weeks of training at **DKT** enabled exposure to mix of software and hardware practical knowledge. During this period, there were a couple of visits to Industries to see use of electronics and software. Electronics and software are interdependent disciplines and together serve several industries, via control systems, data sensing, data recording, and analysis. Electronics enables the capturing, access to the real world, while software is much bigger pie as it explores various possibilities of application of data and controls on the same hardware.

Electronics advances in low powered devices, smaller footprints and revolution in addressability is enabling to dumb physical devices like watches, shoes as intelligent devices. Software is able to reach and collect data from remotest possible items that human being may be using. This is generating Exabyte of data and hence throwing lot of information which is being analyzed through advanced statistical techniques using software to make predictions and optimizations in operations.

Knowledge in **C-Language** and **digital electronics** was mostly used in the training and further strengthened while trying to implement new things. Logical reasoning and conceptualizing / doing things on paper is key to design and solving problems using software and hardware.

A project of **Traffic Lights** was used as a means to get exposed to all the ground level work for creating systems. We started from experimenting with individual components, integrating them as single system and evolving software to deal with complex situations. It was interesting to see, how things can become complex from seemingly a simple solutions.

Overall, world of software and hardware, when fused together to open new world of possibilities is quite exciting. For me it is just a beginning.

# Introduction

da Kine Technologies Pvt. Ltd. (DKTPL) started in 2010. It started with an idea to create a product in education which can help connect tutors to remote students. Later it ventured into various other projects and products in the domain of automation - ERP, CRM and System Solutions. Embedded is one of the important aspects to connect to the data sensing and control components of the solutions offered.

I company in this firm as a trainee and worked in the development and testing of the project of **Traffic Lights.**

**Learnings**

1. **Learning LINUX**

**Basic Terminal Commands**

1. **pwd** - pwd stands for "print working directory”. The *pwd* command will allow you to know in which directory you're located.
2. **ls** -The *ls* command shows ('*list*') the files in current directory.
3. **cd** -The *cd* command allows changing directories.  
   * To navigate into the root directory, use "**cd /**"
   * To navigate to your home directory, use "**cd**" or "**cd ~**"
   * To navigate up one directory level, use "**cd ..**"
   * To navigate to the previous directory (or back), use "**cd -**"
4. **rm** - Removes or deletes a file in directory.
5. **rmdir** - The *rmdir* command deletes an empty directory.  
   -To delete a directory and all of its contents recursively, use **rm -r** instead.
6. **mkdir** - The *mkdir* command allows to create directories.

**Compilation of C Programs**

gcc is the "GNU" C Compiler.  
The standard way to compile the program is with the command  
**gcc programsourcecode.c -o executablefilename -I.**

### 

### 

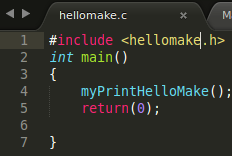
### Command: make

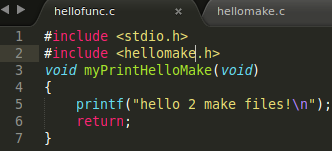
Without a Makefile, the typical approach to the test/modify/debug cycle is to use the up arrow in a terminal to go back to your last compile command so you don't have to type it each time, especially once you've added a few more .c files to the mix. Unfortunately, this approach to compilation has two downfalls.

* First, if you lose the compile command or switch computers you have to retype it from scratch, which is inefficient at best.
* Second, if you are only making changes to one .c file, recompiling all of them every time is also time-consuming and inefficient.

The purpose of the make utility is to determine automatically which pieces of a large program need to be recompiled, and issue the commands to recompile them.  
To prepare to use make, you must write a file called the Makefile that describes the relationships among files in your program, and the states the commands for updating each file. In a program, typically the executable file is updated from object files, which are in turn made by compiling source files.

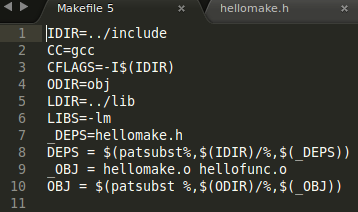
(**Test Code)**







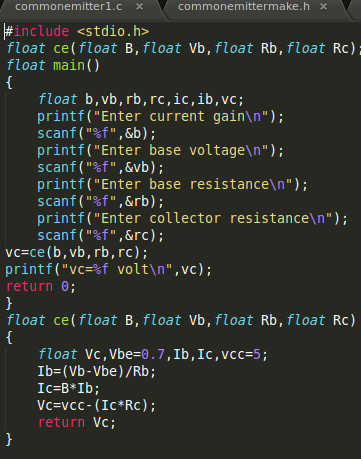
### (Makefile)



1. **Understanding Variables**

**Global Variables**-These are the variables which are declared outside all the functions. These variables can be used in all the functions.

**Local Variables**-These are the variables which are declared inside a function. These can only be used in the function itself.



**In this code B, Vb ,Rb, Rc are local variables whereas b, vb etc. are global variables.**

**Static Variable**- These variables don't disappear when function is not active. When a function comes in action again the variables retain their value which they had last time.

**Volatile** - It is a qualifier that is applied to a variable when it is declared. It tells the compiler that the value of the variable may change at any time-without any action being taken by the code the compiler finds nearby.

### Object Oriented Design

#### Objects and Classes

**Objects** - These are the things with independent runtime existence. These are entities in object oriented systems.  
At run time objects interact with each other by sending messages.  
**Class** - Class is a collection of objects of same type.  
For example - there are fruits in a basket like banana, orange and apple. In this example fruit is a class and banana, orange and apple are its objects.

#### Inheritance

It is the property in oop in which sub classes inherit or acquire the properties of its main class.

#### Polymorphism

It means the ability to take more than on form.  
Consider a situation. There is a class names Area. It has three objects circle, square & triangle with functions area(circle), area(square) & area(triangle) respectively. But the operation area is different for all three figures. So polymorphism will be helpful in objects with different internal functions to show same external functions.

#### Operator Overloading

It is the process in which an operator is asked to do different task at different instances.  
For example - operation of addition adds two integers. But when this operator is used in strings it forms a new string by concatenation.

1. **Markdown**

### Markdown is a markup language with plain text formatting syntax designed so that it can be converted to HTML and many other formats using a tool by the same name. Markdown is often used to format readme files, for writing messages in online discussion forums, and to create rich text using a plain text editor.

### Basic Commands

### To make italics and bold –

### To make a word/words italic underscore ( \_ ) is used before and after that word.

### For example - \_hello\_ will become italic.

### To make a word/words bold (\*\*) are used before and after that word.

### For example - \*\*hello\*\* will become bold.

### \_\*\*hello\*\*\_ will become both italic as well as bold.

### If we want whole line to be italic or bold then these instructions will be used at the start and end of the line.

### Headers-

### To make headers place ( # ) before that phrase. Place the same number of hash marks as the size of the header.

### For example

### (Input)

### (Output)

### Links –

### To create links wrap the link text in square brackets ([ ]) and then wrap the link in parenthesis.

### For example – To visit google.com you will write [text]([www.google.com](http://www.google.com)).

### Line Break –

### To start a new line, just insert two spaces after end of the line.

### List-

### For an ordered list text would be typed like following in the editor-

### 

### For an unordered list text would be typed like following in the editor –

### (Input)

### (Output)

### Learning Microcontroller

### Basic Terminology

1. **EPROM** - Erasable Programmable Read Only Memory.  
   It is erased by exposing to strong UV light source.  
   It is Non Volatile.  
   It was used in microcontrollers like Intel 8048 and C versions of PIC Microcontrollers.
2. **EEPROM** - Electrically Erasable Programmable ROM.  
   It is used to store small amount of data.  
   While EPROM is removed for erasing and programming, EEPROM can be erased and programmed by applying **special programming signals.**  
   I is used in AVR for semi-permanent data storage.
3. **Flash Memory** - It is developed from EEPROM. It was first developed by Toshiba.  
   It is of two types - 1.NAND 2. NOR. It is used in memory cards, USB flash drives, SSDs.
4. **Microcontroller** - It’s a small computer on single integrated circuit.  
   It contains processor core, memory and input output peripherals.
5. **AVR** - It is single chip bit microcontroller.  
   It was developed by Atmel in 1996.  
   It uses **flash memory**.
6. **GPIO** - General Purpose Input/ Output.  
   GPIO in AVR is controlled by 3 **8-bit registers**.
   * DDRx - Data Direction Register. It configures pins as i/p or o/p.
   * Portx - Output Port Register. It sets output value on pins configured.
   * PINx - It is input register. It is used for pin toggling. It is used to read input signal.
   * x is **Pin Identifier.**
7. **MPU** - Memory Protection Unit.  
   Memory protection is a way to prevent process from accessing memory that hasn't been allocated to it.  
   It prevents a bug or malware within a process from affecting other processes.
8. **Watchdog Timer (WDT)** - It is used to detect and recover from computer malfunction.  
   During normal operation, the computer regularly restarts the watchdog timer to prevent from elapsing or timing out. If due to a hardware fault or program error, the computer fails to restart the watchdog, the timer will elapse and generate a timeout signal. It generates timeout signal to initiate corrective measures.
9. **SPI** - Serial Peripheral Interface.  
   It is used for short communication primarily in embedded system. Communication is in full duplex mode.
10. **CAN** - Control Area Network Bus.  
    It allows communicating without host computer.
11. **I2S** - Integrated Interchip Sound.  
    It is used for connecting digital audio devices together.  
    PCM audio data is communicated between ICs.
12. **Time Stamp** - Sequence of characters or encoded information identifying when an event occurred is called time stamp.
13. **Input Capture** - Method of dealing with input.  
    Embedded system using input capture will record a timestamp in memory when a input signal is received.
14. **Output Compare** - Capability to trigger an output at specific time, based on a timestamp in memory, is called output compare.
15. **UART**- Universal Asynchronous Receiver and Transmitter.
16. **USART** – Universal Synchronous Asynchronous Receiver and Transmitter**.**

**USART is same as UART but it has added capability to act synchronously which allows for a higher baud rate.  In UART the transmitter and receiver needs to be set to common bit rate which is its major disadvantage.**

**In USART separate clock is provided.**

1. **I2C** - Inter-Integrated Circuit.  
   These are used for attaching lower speed peripherals to computers.
2. **Interrupt** – It is a condition which tells microprocessor to temporarily work on another task and later come the previous task.

**For example –** Many computers use interrupt driven input/output. When user click mouse button or give command by keyboard then CPU pause that work and execute the command given by the user and then it returns to the current program.

**Interrupts are of two types – External interrupt and internal interrupt.**

### Internal interrupt is triggered by the software and external interrupt is caused by external hardware module.

### Learning Atmel ATmega 328

### Features

1. CPU - 8 bit AVR
2. External Interrupts - 24
3. SPI - 2
4. I2C - 1
5. UART - 1
6. ADC Channels - 8
7. Analog Comparators - 1
8. Timers - 3
9. Output Compare Channels - 6
10. Input Capture Channels - 1
11. PWM channels - 6
12. Watchdog
13. Flash (Kbytes) – 32
14. Max. Operating Frequency – 20 MHz
15. Pin count – 28
16. EEPROM – 1024 bytes
17. Operating Voltage – 1.8V – 5.5V

### AVR Instruction SET

## Operational Instructions

By these instructions we can perform various operations like addition, multiplication and subtraction.  
Various instructions for addition are:-

1. **ADC**- It adds two registers with carry flag and put value in the destination register.
2. **ADD**- It adds two registers and put final value in the destination register.
3. **ADIW**- It adds immediate value to register pair and places the result in register pair. d={24,26,28,30} k=0-63;

Various instructions for multiplication are:-

1. **MUL**- It multiplies two unsigned numbers contained in Rd and Rr. R1:R0=Rd\*Rr. Where R1 is contains MSB and R0 contains LSB.
2. **MULS**-It multiplies two signed numbers in Rd and Rr.
3. **MULSU**- Multiply signed with unsigned. In this Rd is signed and Rr is unsigned number. d=16-23 r=16-23

Various instructions for subtraction are:-

1. **SBC**
2. **SBCI**
3. **SBIW**
4. **SUB**
5. **SUBI**  
   ***NOTE***-In all these instructions the all are subtracted from number present in Rd and then the resultant numbers are stored in Rd.

## Logical Instructions

These are the instructions through with logical operations are performed like OR, AND etc. Some of them are:-

1. **AND** – Performs logical AND b/w contents of Rd and Rr and resultant is stored in Rd.
2. **ANDI**- Logical and with immediate. It performs AND b/w contents of Rd and Constant K. d=16-23 K=0-255
3. **OR and ORI** – Performs logical OR between Rd and Rr and Rd and constant K respectively. For ORI d=16-23 K=0-255.
4. **EOR** – Exclusive OR.

## SET Instructions

1. **SEC** – Set carry Flag. C<-1. After this command we can use ADC or SBC or SBCI.
2. **SEH** – Sets Half Carry flag. H<-1.
3. **SEI** – Sets global interrupt. I<-1.  
   **SEN, SES, SET, SEV, SEZ** are other set commands in which Negative flag, signed flag , T flag, overflow flag respectively are set.  
   ***NOTE***-These flags are there in SREG.
4. **SBI** – Sets bit in I/O register.
5. **SBR** – Sets bits in Register. Rd<-Rd v K.

## Skip Instructions

1. **SBIC** – Skip if bit in I/O register is cleared. In this instruction it tests that single bit of I/O register. If that is found 0 then the next statement will be skipped.
2. **SBIS** - Skip if bit in I/O register is set. In this instruction it tests that single bit of I/O register. If that is found 1 then the next statement will be skipped.
3. **SBRC** – Skip bit if register is cleared. If the bit in Rr is found to be zero then next statement is skipped.
4. **SBRS** - Skip bit if register is set. If the bit in Rr is found to be zone then next statement is skipped. b=0-7 r=0-31

## Clear Instructions

1. **CLC, CLH, CLI, CLN, CLT, CLV, CLZ** are some of the clear instructions which clear the respective flags in Status register.
2. **BCLR** – Bit clear from SREG. It clears a bit from SREG.  
   Difference between the clear commands used above and BCLR is that in this we specify the bit number of SREG which we want to get cleared. But in above instructions it is already given which we want to clear. It is hardcore instruction. And in BCLR we can change easily the flag we want to clear.
3. **CBI** – Clear bit in I/O register. It clears 32 lower registers. I/O(A , b)=0 A =0-31 b=0-7
4. **CBR** – It clears specified bit in register Rd. It is done by ANDing it with the complement of the constant K.

## Compare Instructions

1. **CP** – Compare. It compares contents of Rd and Rr.
2. **CPCI** – It compares Rd and Rr taking carry in previous stage into account.
3. **CPI** – Compare with Immediate. It compares with the constant K.

*NOTE*-*After all these statements conditional branch statement can be executed*

1. **CPSE** – Compare skip if Equal. If Rd and Rr are equal next statement is skipped.

## Conditional Branch Instructions

* These are the instructions which are executed when the condition specified is done and then these statements take those much steps further from that execution specified by the user.
* **BRBC, BRBS, BRCC, BRCS, BREQ, BRIE, BRLO, BRLT** are some of the conditional branch instructions.

## Store Instructions

1. **ST** – It stores indirect one byte from a register to data space. Data location space is provided by the X pointer register. It can be left unchanged, post incremented or pre decremented.
2. **STD** – It stores data using Y pointer register. It can be left unchanged, pre decremented or post incremented.
3. **STD** – It stores data using z indirect register.
4. **STS** – It stores data direct from Rr to data space.

## Load Instructions

1. **LD** – It loads indirect one byte from a register to data space. Data location space is provided by the X pointer register. It can be left unchanged, post incremented or pre decremented.
2. **LDD** – It loads data using Y pointer register. It can be left unchanged, pre decremented or post incremented.
3. **LDD** – It loads data using Z indirect register.
4. **LDS** – It Loads data direct from Rr to data space.

## Jump Instructions

1. **JMP** – It jumps to an address. Difference between Conditional branch and JMP instruction is that it is unconditional and it jumps directly to an address while in branch it takes steps further in a program specified by the user.
2. **IJMP** – It is the indirect jump to an address pointed by Z Pointer register in the register file.
3. **RJMP** – Relative Jump.

**I/O PORTS Registers**

Every port has three register bit.

* DDxn
* PORTxn
* PINxn

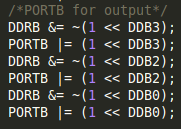
DDxn is accessed at DDRx I/O Address.  
PORTxn is accessed at PORTx I/O Address.  
PINxn is accessed at PINx I/O Address.

If DDx is 1 this implies that the pin is output pin and if DDx pin is 0 then it is input pin.  
When DDx is, if PORTx is 1 then pull up resistor is activated. It can be deactivated either when DDx is 1 or when PORTx is 0.

If PORTx is when DDx is 1 then the pin is driven high and if PORTx is 0 then the pin in driven low.

If PINx is 1, it toggles the value of PORTx irrespective of the value of DDx.





**Interrupts**

PCINT0, PCINT1, PCINT2 are some of the interrupt vectors with definition Pin change interrupt request0, 1, 2. Program address for PCINT0, PCINT1 and PCINT2 are 0x0006, 0x0008, 0x000A respectively.

**External Interrupts**

The External Interrupts are triggered by the INT0 and INT1 pins or any of the PCINT23-0 pins.

The pin change interrupt PCI2 will trigger if any enabled PCINT [23:16] pin toggles. The pin change interrupt PCI1 will trigger if any enabled PCINT [14:8] pin toggles. The pin change interrupt PCI0 will trigger if any enabled PCINT [7:0] pin toggles.

**Register Description**

* **PCICR** – Pin Change Interrupt Control Register



**Bit 7:3 -** These bits are unused bits

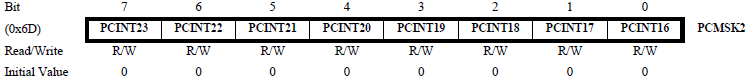
**Bit 2 – PCIE2:** Pin Change Interrupt Enable 2. When the PCIE2 bit is set (one) and the I-bit in

the Status Register (SREG) is set (one), pin change interrupt 2 is enabled. Any change on any enabled PCINT [23:16] pin will cause an interrupt.

**Bit 1 – PCIE1:** Pin Change Interrupt Enable 1. When the PCIE1 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 1 is enabled. Any change on any enabled PCINT [14:8] pin will cause an interrupt

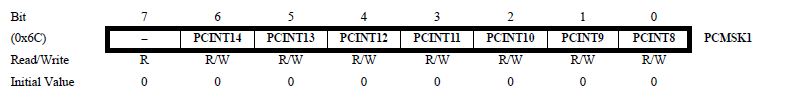
**Bit 0 – PCIE0:** Pin Change Interrupt Enable 0. When the PCIE0 bit is set (one) and the I-bit in the Status Register (SREG) is set (one), pin change interrupt 0 is enabled. Any change on any enabled PCINT [7:0] pin will cause an interrupt.

* **PCMSK2 –** Pin Change Mask 2



**Each PCINT [23:16]-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT [23:16] is set and the PCIE2 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin.**

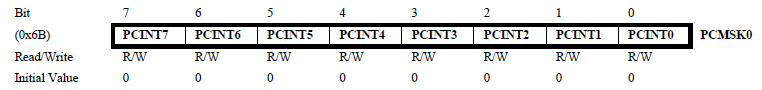
* **PCMSK1- Pin Change Mask 1**



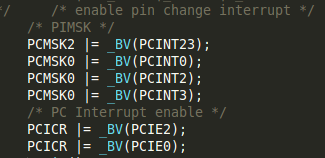
**Bit 7 –** Reserved

**Bit 6:0-** Each PCINT [14:8]-bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT [14:8] is set and the PCIE1 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin.

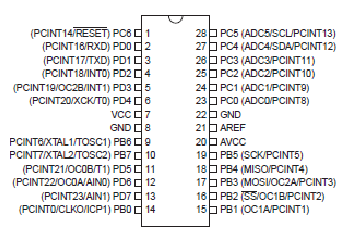
* **PCMSK0-** Pin Change Mask 0

****

Each PCINT [7:0] bit selects whether pin change interrupt is enabled on the corresponding I/O pin. If PCINT [7:0] is set and the PCIE0 bit in PCICR is set, pin change interrupt is enabled on the corresponding I/O pin.



**Pin Diagram**



### 

### Learning Photo sensor

### 

### This photo sensor circuit will work with 3-9 volts. It has an Op Amp LM 358 which works as comparator in this circuit. It has two IR photo diodes, one as transmitter and other as receiver.

### The IR photo diode (receiver) is used in a potential divider in a reverse biased mode. **A threshold voltage is set at the inverting terminal of the Op Amp using a potentiometer so you can set its sensitivity.**

### **When the IR light reflects from a lighter surface, say your hand comes close enough, the resistance of the photodiode would decrease and this in turn when exceeds the threshold voltage will make the output of the Op Amp go high and the LED will light up.**

### 

### Learning Shift Registers (74HC595 and 74HC164)

### Pin Diagram

### (74HC595)

### 

### (74HC164)

### 

### We know that microcontrollers have limited number of outputs. So what if we want to glow many LEDs and control them according to our wish? The solution for this problem is shift registers. A shift register has eight outputs. So if we connect the input of shift register with one of the outputs of microcontroller we can glow eight LEDs. The count for LEDs can be increased by increasing the number of shift registers.

### When SER pin (Serial input) pin is high the serial input for next pin gets shifted in. When SRCLK pin (serial clock) is high the values het shifted to right. When RCLK pin (Register clock) is high the O/P pins get updated with new data. The 9th pin of 74HC595 dumps registered value and this pin can become I/P for the next register if we want cascading in registers.

### For cascading in 74HC164 13th pin will become the input for next register.

### Introduction with Arduino UNO

### The first small project which I did was blinking of LED on the Arduino Uno board. Then I implemented various small programs on it like running photo sensor. In this we connected the output of sensor to the digital pin of Arduino Board. Now when we put our hand in front of the photodiode the serial monitor on IDE will be showing 1. Otherwise it will be showing 0.

### 

### 

### Then we used Arduino board for testing of our components like LCD, shift registers, Analog read using potentiometer.

### Learning Procurement

Following steps followed:

* Solution Design
* Prepare Bill Of Materials
* Vendor Selection - research through internet and other known sources
* In-Market research for costly items
* Procurement
* Components Entry / Management in ERP

### 

1. **Industrial Visit**

On June 3, 2015 we had an industrial visit at "Hindustan Pumps Pvt. Ltd.”, Karnal.

**About**

Hindustan Pumps Pvt. Ltd. was established in 1975. It achieved the certification of ISO 9001:1994 on 1999. Company has been manufacturing pump and motors of different types and is using a single brand name of 'Hindustan' since beginning. In the year 2003 company launched its second brand 'Uttam' as a marketing strategy for opening new market avenues. Company has acclaimed reputation of high standard performance with excellent after-sale-service.

### Purpose of Visit

### The main purpose of our visit was to see the **application of automation and control systems in the industry**.

### Products

Hindustan Pumps manufactures variety of pumps. Some of the products are:-

1. Bore well Submersible Pumps - These are used in agriculture, industries etc.
2. Monoblock Pumps - These find their application in Mines, Dam sites etc.
3. Induction Motors - These are for domestic purposes.
4. Sewerage Pumps - These are used in drainage and sewerage pumping systems.

**Processes in Pump Manufacture**

Various processes were seen during the manufacture of the pumps like drilling, cutting etc. These are as follows: -

1. Drilling - This process was done by VMC machines.
2. Cutting - This process was done in CNC machines. Manual Lathe machines were also there.



1. Winding - For bigger pumps winding was done manually as they require less number of turns which can be easily counted. But for small motors various machines were there for winding.
2. Assembling
3. Compressing
4. Balancing - Rods were fitted horizontally in balancing machine and through meter they were seeing at which angle the weight on the rod is more. At that point they drilled that rod to reduce the weight.



1. Casting and Molding
2. Painting
3. Drying - For fast drying of paint and preventing it from dust the pumps were put in oven.
4. Testing speed, torque.

**Process of Process Improvement**

Hindustan Pumps maintains information on the defects raised either during production or material supplied by the vendor. In the latter case the defect is raised back to the supplier and he is asked for the corrective action plans to be taken.

For the defects raised during production Hindustan Pumps maintain a register in which the person involved during production at that time raises the ticket for the defect. This register is checked daily by the person in-charge. A **Root Cause Analysis** is done by the whole team and the corrective measures are taken. There was a problem that was related to mismatch between in the specifications required and part produced. A root cause analysis was done and it was found that the root cause was machine setting.

In their endeavor to achieve best quality of output, they have put in place process automation / traceability. ERP system is being put in place to log such defects and analyzed for trend.

Now Hindustan Pumps is giving stress upon implementation of **Toyota Culture** in their company. In Toyota Culture the worker (producer) himself tries to identify defects and eliminates the fault in process. Their goal is achieve zero-defect manufacturing.

According to my research on **Toyota Culture,** there are two main things Toyota mainly focuses on, one is Continuous Improvement and the other is Respect for People.

Continuous improvement involves Challenge, Kaizen and Genchi Genbutsu.

**Challenge-** The Company form a long time vision. All of them face the challenge courageously and they work creatively to realize their dreams.

**Kaizen-** Literal meaning of kaizen is continuous improvement. In business sense it refers to activities that continually improve all functions and involve all employees from the CEO to the assembly line workers. The company improves business operations continuously and always try for innovation and evolution.

**Genchi Genbutsu-** It means go and see. It is part of the Toyota Way Philosophy: it encourages employees to go to the source and find the facts to make correct decisions build consensus and achieve goals.

Respect for people involves respect as well as teamwork.

**Respect-** They respect everyone and try to build mutual trust.

**Teamwork-** They stimulate personal and professional growth, share opportunities of development and maximize individual and team performance.

**Conclusion**

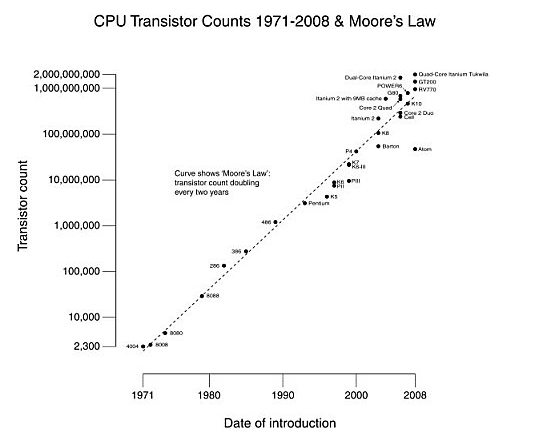
The part which is appreciable about this firm is that maximum work of production of pumps is done in the factory itself. There is no outsourcing. At the startup to reduce the cost various things like racks for stores, lifts and several transporting machines were made in the factory itself. This plant needs huge amount of cooperation, hard work and sincerity among the workers and we saw it at much greater extent than we expected, which is also appreciable.

1. **Introduction to VLSI**

I had a visit to firm **COVERIFY, Gurgaon** where the director of the company Mr. Punit Goyal introduced me to VLSI and shared some knowledge of Multicore Processors.

**Increase in number of Transistors**

There was a discussion on the increase in the number of transistors in an integrated circuit. Here he related Moore’s law, that the number of transistors per square inch on integrated circuits had doubled every year since the integrated circuit was invented.



 The more transistors that are put in a small area reduces the amount of space between each transistor reducing the resistance producing less heat and allowing the processor to calculate more instructions per cycle and it allows engineers to put more cores in a smaller space.

**VLSI -** Very Large Scale Integration. Many ICs had a limited function to perform. VLSI overcame it. It is the process of creating an IC by combining thousands of transistors into a single chip.

**HDL -** Hardware Descriptive Language. It is the language used to program the structure, design and operation of electronic circuits. It gives description of an electronic circuit that allows for analysis, simulation and testing of that circuit.

**VHDL-** VHSIC Hardware Description Language. It is a HDL which is used in design automation to describe the behavior and structure of system and circuit designs.

**Verilog** - It is a HDL used to model electronic systems. It is used for design and verification of digital circuits at RTL.

Verilog is derived from C programming language whereas VHDL has its roots in Ada programming language.  
Verilog is case sensitive whereas VHDL is not case sensitive.

CVC is a Verilog HDL compiled simulator. CVC has the ability to simulate in either interpreted or compiled mode. OSS CVC is licensed under a Perl style artistic open source license.

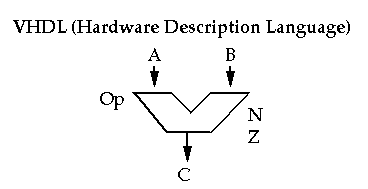
[**http://www.tachyon-da.com/**](http://www.tachyon-da.com/) **(Link for downloading CVC)**

**RTL** - Register Transfer Level. It is a design abstraction which models a synchronous digital circuit in terms of the flow of digital signals between hardware registers, and the logical operations performed on those signals.  
It focuses on describing how the signal will flow in the circuit. It is used in logic design phase on IC design cycle.  
Both VHDL and Verilog implement register-transfer-level abstractions.

**EAD or ECAD** - Electronic Design Automation. It is software tools for designing PCB and ICs.

**The VLSI Design Process**

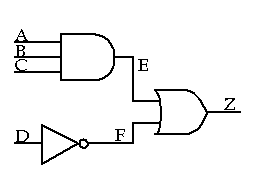
1. Decide specification- First specification of circuit is decided like at what clock speed it will work, what is its purpose, how much power will it consume etc.
2. Behavioral simulation- The functional behavior of the design (or a parameter such as power) is determined by applying a set of excitation vectors to a circuit model.



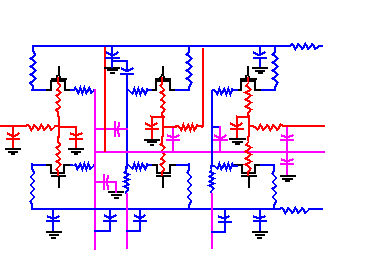
1. Register Transfer Level Design-



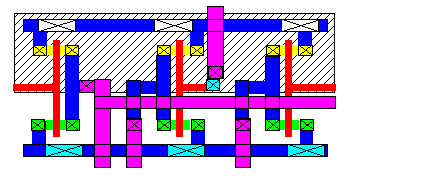
1. Logic Simulation- Simulation is modelling of a design, its function and performance. Logic simulation includes verification of logic which needs to be performed by the circuit.



1. Timing Simulation- time simulation includes the verification of clock speed of the circuit.



1. Physical Design- Finally physical design of the circuit is made ready and further testing of the circuit is done.



**Processors**

There was a discussion on processors, RISC and CISC instructions and multicore processing.

**RICS -** Reduced Instruction Set Computing.

* Simplified instructions are used in RISC.
* It provides high performance.
* It uses fewer microprocessor cycles per instruction.
* RISC has separate instructions of I/O and data processing.
* ARM, Atmel AVR uses RICS.

**CISC-** Complex instruction set computing.

* Complex instructions are used in CISC.
* Single instruction can execute low level operation such as load from memory set.
* Intel x86 processors uses CISC.

In RISC it is not amount of work that is reduced, it is pipelining that helps it prefetching instructions and execute them in parallel, therefore reducing number of clock cycles it take for a processor to execute.

**Multicore Processors-** It is a type of architecture where a single physical processor contains the core logic of two or more processors. In multicore they can run multiple instructions at same time which increases overall speed programs manageable to parallel computing.  
It consumes less energy and gives high performance.

**Multithreading** - It is the ability of a CPU or a single core in a multicore processor to execute multiple processes.  
It aims to increase utilization of a single core by using instruction level parallelism.

**PROJECT**

**TRAFFIC CONTROL SYSTEM**

**Problem Statement**: - To Implement Logic for Traffic at a Quad.



By- Abhinav Ghai

### To get the exposure of link of software with the hardware I worked on a project of Traffic Control System.

### Different peripherals which are used in the project are:-

### Shift registers 74HC164 and 74HC595

### LCD Module RG1602A

### ATmega 328 processor

### RS232 convertor

### MAX 232

### AVR Programmer

### LEDs

### Capacitors

### Photo sensors

### Resistances

### Crystal (16MHz)

### Implementation

### To implement this logic I initialized from very basic level and slowly with time increased its complexity to include various features in this console. There were a lot of difficulties faced in this process. The most challenging job was to integrate software with hardware.

### Initial Logic

### There are four poles having four lights Red, Yellow, Green, and Green Left each. Green left indicating the side only to passengers who want to go to left side can move rest will stay.

### One pole is facing one side of the quad as depicted in the diagram below.

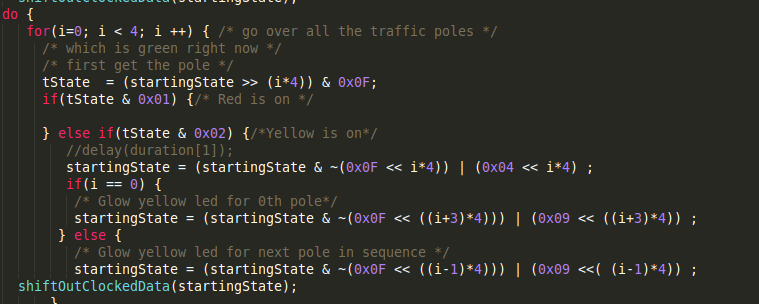
### Initially on side A Left Green light is on, Red is on for sides B and C and Green is on for side D. After the duration for these lights is over pole having Left Green light ON will change to Yellow light and rest will change to Red. In this case pole on side A will change to Yellow. After the duration is over the pole having Yellow light will change to Green and pole next to that side will change to Left Green. In this case light on side A will be green and on B will be Left Green. This process will continue in this logic.

### 

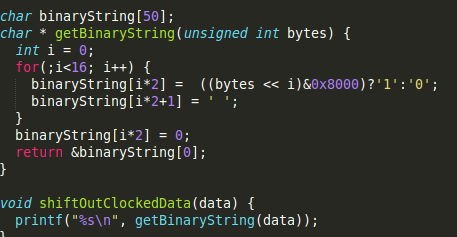
### Extract of the C Code for this logic is below:-

### 

Problem in this code was that for shifting out for the LEDs through the shift registers only two bytes are required as there are total 16 LEDs and in this code four bytes are use, one for each pole. Keeping the logic same I changes the code and its extract is below:-

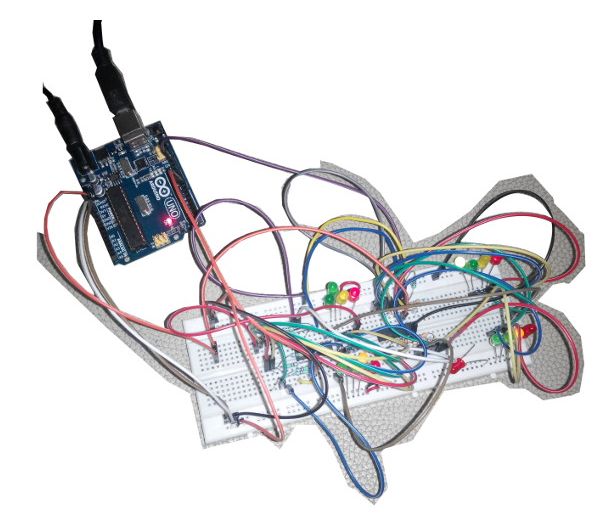


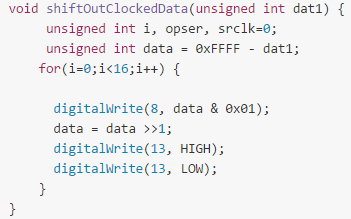
After this there was a need of program for shifting out these bytes. In hardware this data will be shifted by shift registers due to which LEDs will be glowing.



**Implementation On Hardware**

It was decided to make circuit on breadboard and use Arduino Uno. We shifted our program stepwise in Arduino’s IDE (Integrated Development Environment). The photograph of circuit and extracts of the code are below:-





**Troubleshooting**

While working on Arduino based projects, there came several instances, where it took lot of time to make things work. Some of the instances are worth capturing here, as they often repeat and knowing them beforehand can save some precious minutes of working on hardware.

* **Random behavior of the Port pins on the peripheral ICs**

While working on breadboard, too much engrossed in the pin connections between Arduino, Resistances and LEDs, VSS and VDD connections were forgotten. This caused this spurious behavior. It took some time, to arrive at the conclusion of missing supply and correct. Other doubts that came in mind were: power supply was apparently loaded; IC 74164 has gone bad, we are not using right sequence of instructions on controller.

* **Bits mapped on software don’t match with the physical placement**

Reason found here was the way bytes were interpreted. It was simple and got fixed soon.

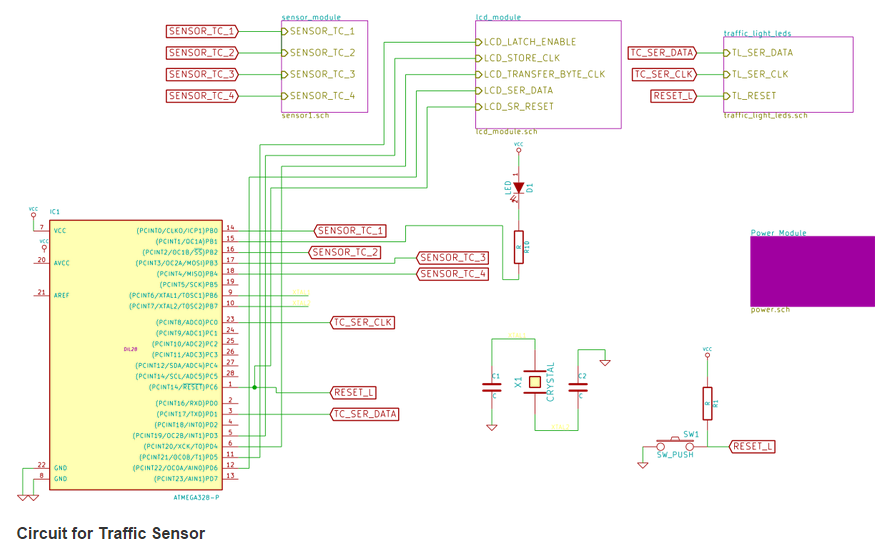
* **Serial Port was not enabled**

Reason here was that to run Arduino one has to be sudoer or member of WV dial group on Linux.

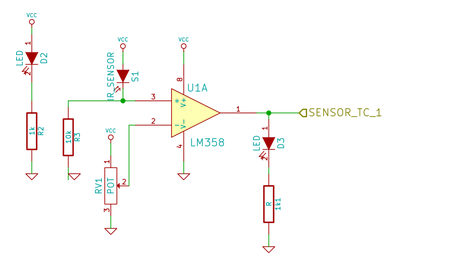
Working on Arduino was the easiest task performed during this project as all libraries are made in its IDE. We just need to put our code and set the pins for input and output.

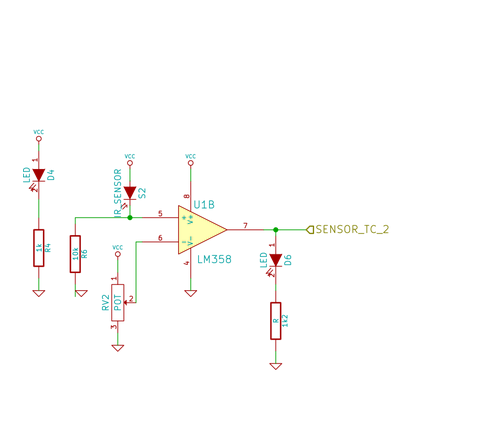
**Working on PCB, Hard Wired**

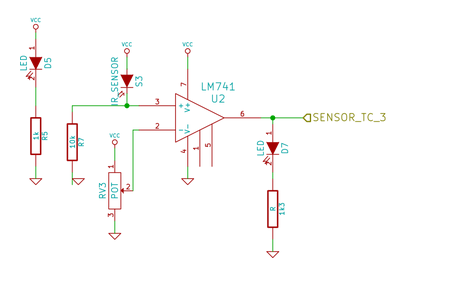
* **Circuit for ATmega 328**

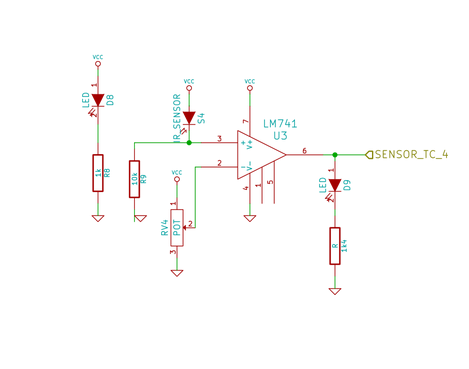


* **Circuit for Traffic Sensor**

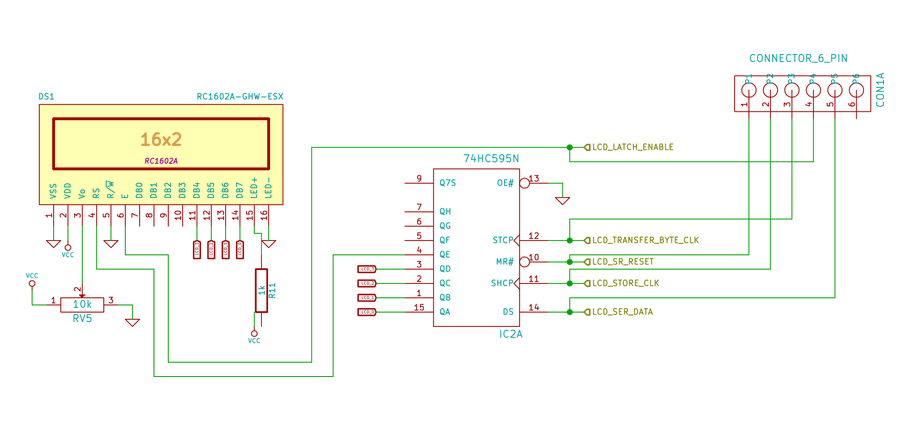




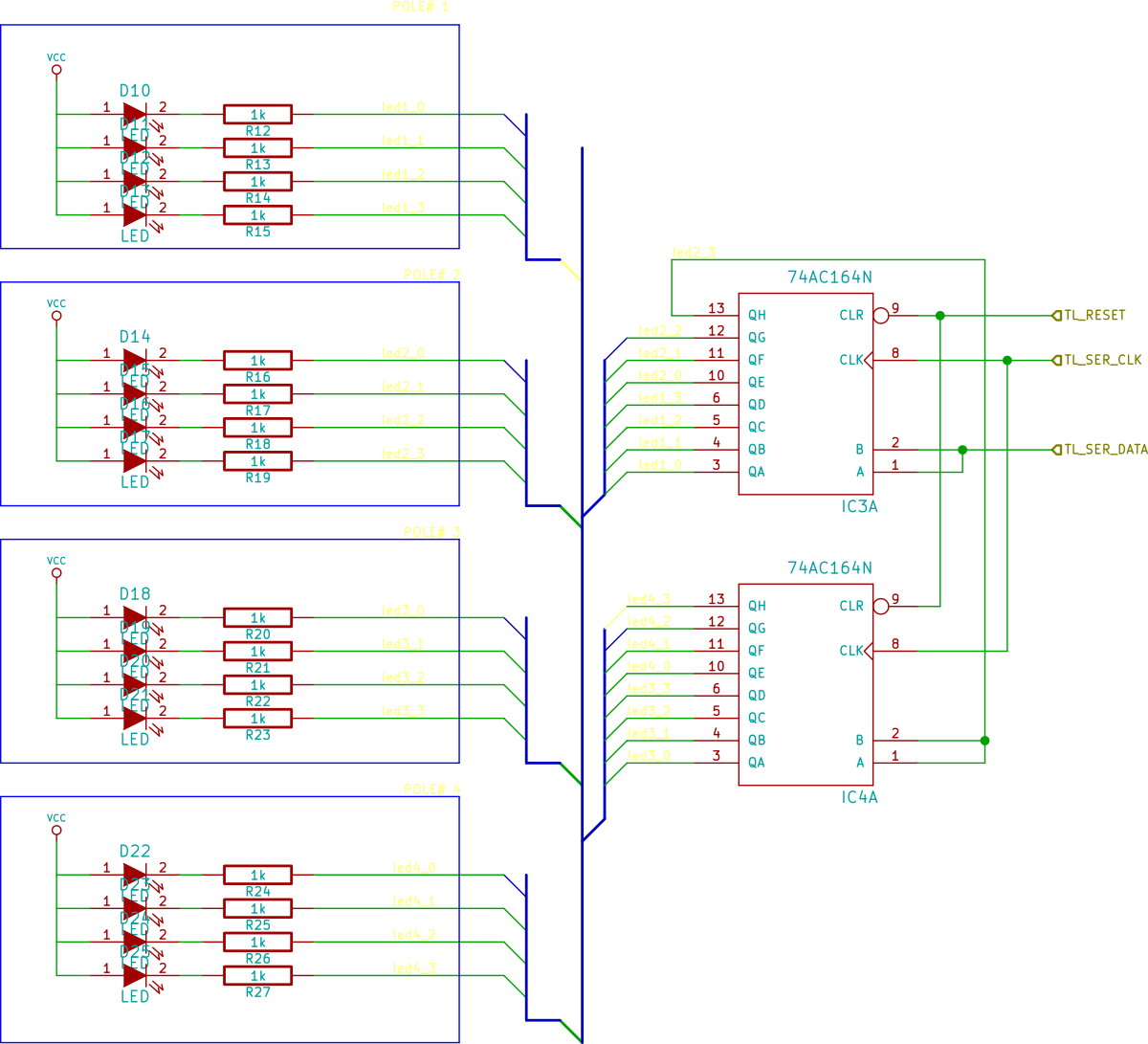




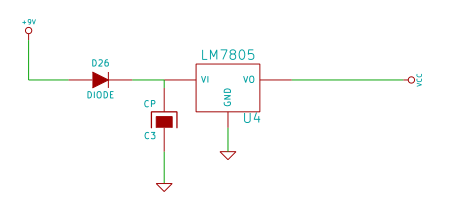
* **Circuit for LCD**



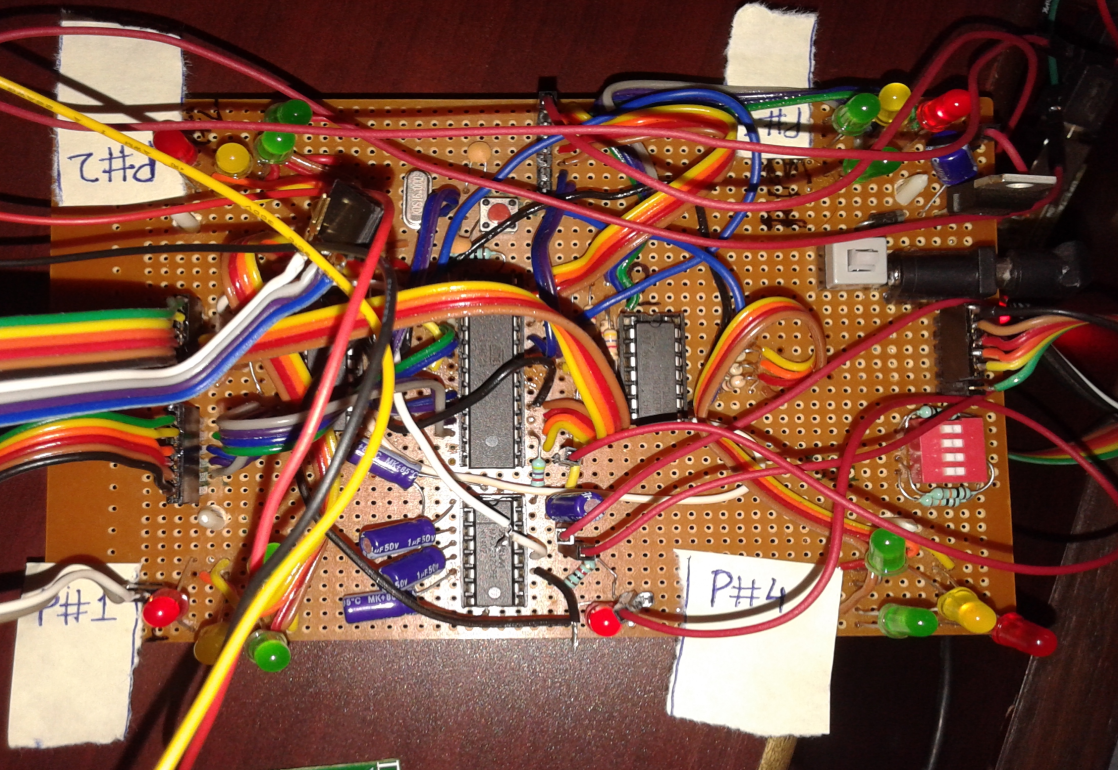
* **Circuit for Traffic LEDs**

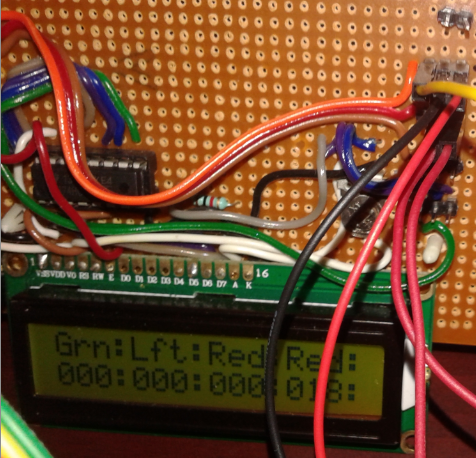


* **Circuit for Power Regulator**



* **Actual Circuitry**

****



* **Extracts of Code and Explanation**

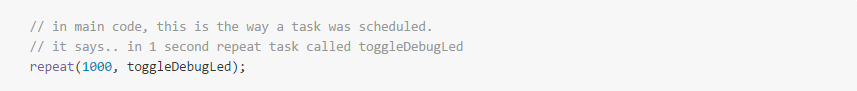
Extract of the code for pushing data to shift register has been discussed above.

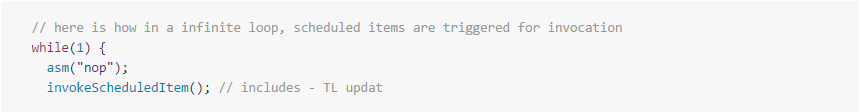
**Scheduler**

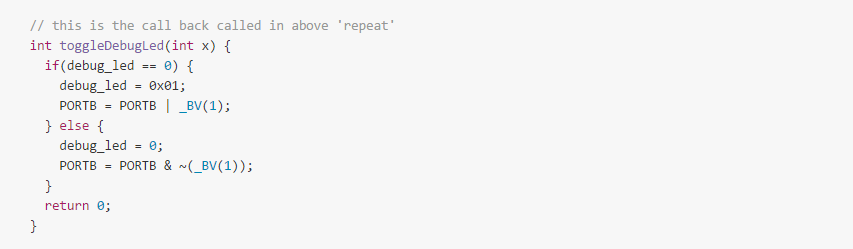
A major challenge for the system to operate without operating system and having multiple tasks at hand was to make sure how all can be done without compromising the timing requirements. For example: every second, status for incoming traffic should be checked and also status for all traffic lights to be updated. Other tasks that were to be performed are:

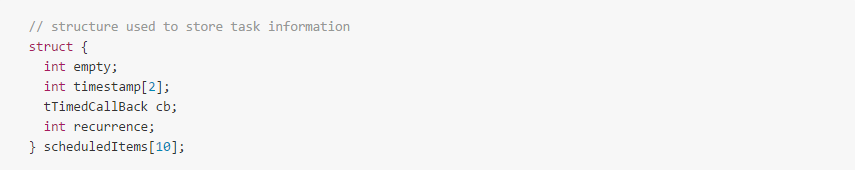
* Sending display information to remote terminal
* Receiving user commands and process when a valid command is received
* Display to LCD terminal
* Read configuration switch for any changes

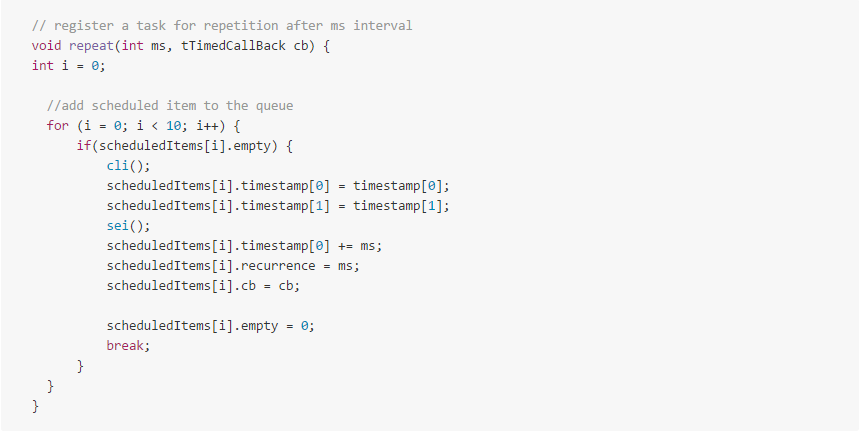
Using 16-bit timer and function pointers above requirement was fulfilled. A code extract to explain this is captured below.







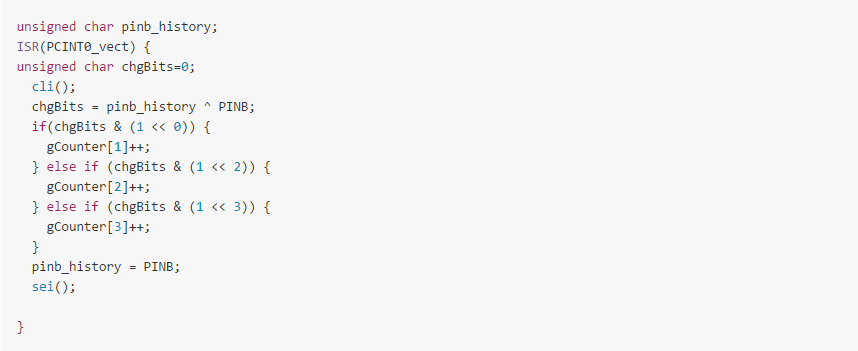




****

**ISR for Calculating Multiplexed Traffic Input**

It was tricky to find out which pin has got changed when out of three counter input pins, one has changed. As guided through manual, a history is to be maintained and we have to do exclusive OR. By doing so, whichever bit has changed will result into that bit getting set.

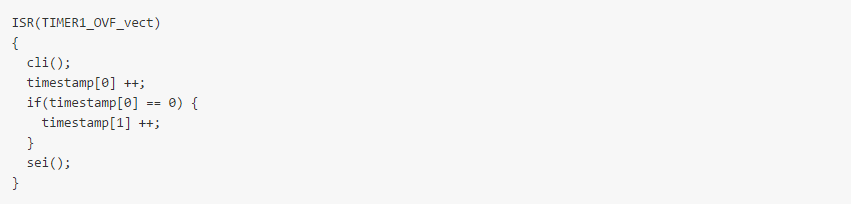


**Timer Implementation**

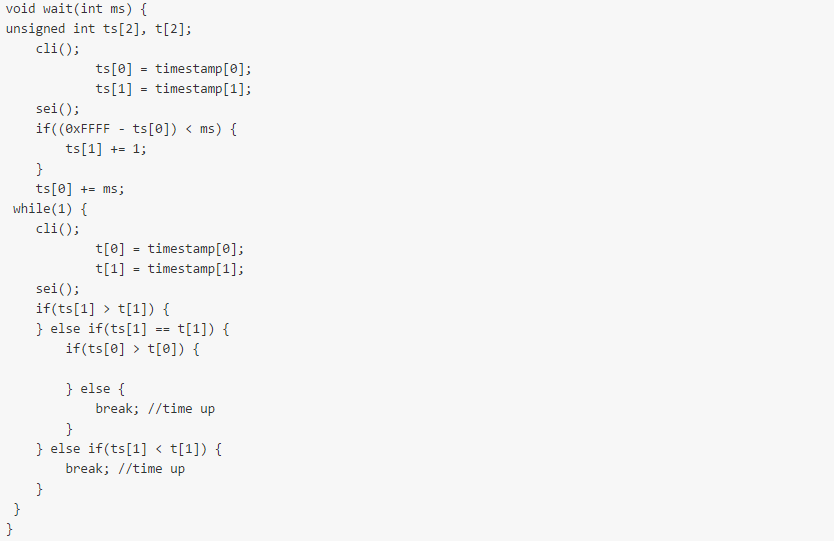
Timer1 is used for timing inside the system. Timer1 Overflow interrupt is used here. Two integer words are used for recording timestamp. On every interrupt, it is incremented, if certain number of ticks has passed by.

There is wait function, which records the ending timestamp by adding the "ms" milliseconds to wait. As soon as it expires, it returns from function call.

* **ISR for Timer**



* **Wait Function**

****

**User Interface- Serial**

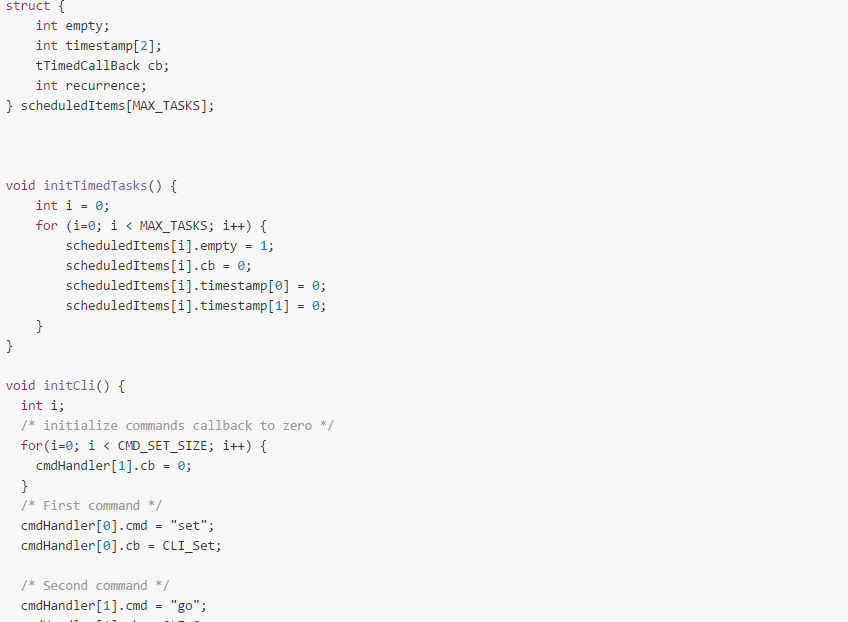
USART Based, serial interface is provided in the system to access and modify system variables. Some of them are:

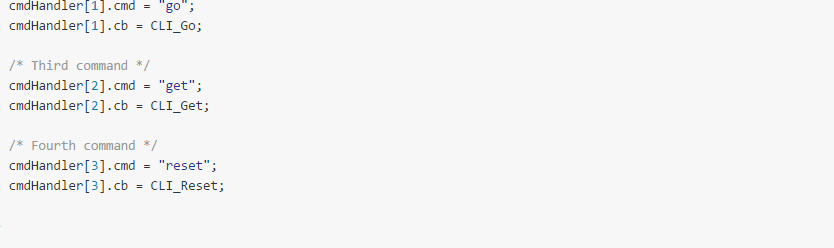
1. Traffic Counters
2. Timers
3. Traffic Threshold
4. Change Green Light instantly
5. Get Traffic Light Status

This module is implemented by breaking functionality into following sub-parts

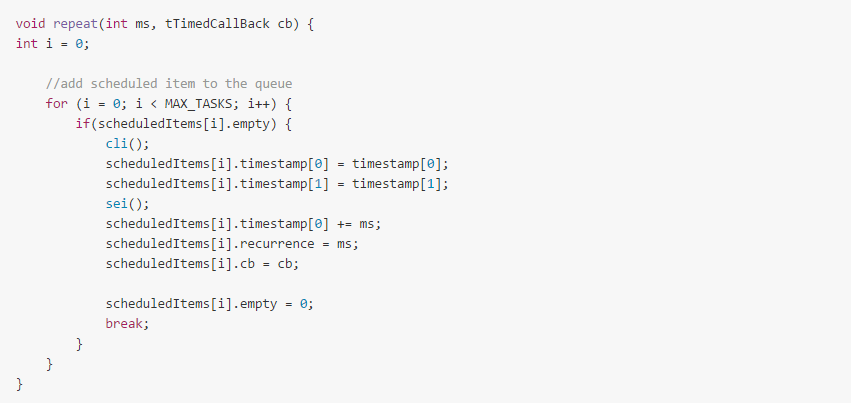
1. init timer for CLI, CLI for commands
2. Registration of new command
3. Interface / Parsing
4. Handling of a valid command received (Handler)

* **Init**

****

****

* **Registration of New Command**

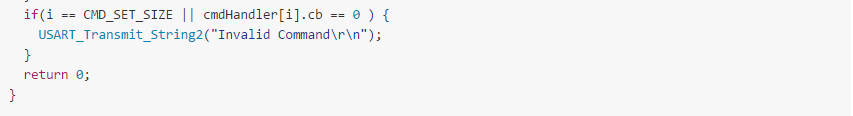
****

* **Handling of Command**



* **Parsing and Validation of Input**





**Troubleshooting**

1. **USART not working-** It is dependent on the UBRRR value to be calculated depending on the crystal placed with ATmega. Appropriate value as per calculation and information also available from various calculators was put. Circuit failed to respond on serial interface. With deeper reading on configuration of ATmega328, it was discovered, that by default, processor runs on 8MHz internal clock, scaled down by 8 internally for all practical purposes. Also, it is not a precise clock. This also correlated with response on serial interface, when baud was made 1200 on terminal side, while configuration in UBRR mapped to 9600.

As per specifications, 51 as UBRR worked, but still the desired output was not there. Tailoring it further both sides, 51 till 56 and 40 till 56. Accuracy of transmission was achieved at UBRR of 41.

1. **Fuse Bits bricked processor**

As we discovered processor works at 8MHz because of unprogrammed fuse bits. Next step was to fuse them with desired bit status using AVRdude. As we fused high byte with 0xFF, it stopped interacting even with AVRdude, and even our running program of traffic lights stopped working. Now the task at hand was to see if this processor can be recovered, somehow. As advised from internet, several links, one has to use an external clock at XTAL1 (pin 9), to clock CPU and hence try programming it for new fuse values. It worked.

We used Arduino Kit to generate clock and used AVRuPro to program the fuses.

1. **Making External 16 MHz clock work**

In previous problem controller was not responding because, it was now dependent on external Crystal clock after changing fuse settings. Crystal after putting in the circuit was not responding. Several cleaning, re-soldering were done but of no use. In the end, crystal was removed and tested independently on breadboard for its healthy condition. It was ok.

Only option left was to try it on a different location. Earlier location on PCB was little crowded, which may have led to some stray resistances and capacitances at track level. So it was moved a little bit away at cleaner place and processor started to respond at 16MHz.

With this another problem of USART not working as per UBRR calculation also started working.

1. **Heating up of MAX 232, when connected with USB**

Still unresolved. Strangely enough, this heats up all of a sudden and new chip when placed, works well. After few hours, the chip that was heated, when again placed on circuit also works. Again, all possible tracks cleaning, connection re-check was done, but of not much use.

1. **Passing String or array argument in a subroutine corrupts**

Whenever a function was called with an argument, it was causing the processor to run weirdly. This weird behavior was visible in terms of traffic lights running irregularly and weird characters coming on the serial terminal.

After long debugging, it was found that initially what was thought as relevant flag i.e. -mmcu=avr5 is not correct, -mmcu=atmega328p. Use of internet to solve one of the problems had led to induction of this fault and was never doubted in the whole process of debugging.

Now after this, even the UBRR of 51 is also working perfectly, 41 does not work anymore.

1. **Intermittent behavior of circuit when Power on**

At times, processor was not able to run program as expected. It was suspected that it has to do with power supply quality, which was taken from regular phone charger. To eliminate this, we put up 7805 regulator with polarity protection diode and a filter circuit with 100uF capacitor. With this addition, circuit was 100% responding to our requests.

Another side effect of poor quality power supply was failure of flash programming at times.

1. **Auto Incrementing Counters**

After trying in independent task for getting traffic count on input. It failed on integrated setup. All the counters kept incrementing continuously. Following suspicions came to mind and were tested negative:

* There may be spurious signal
* There is some wrong setting for the interrupts related to input compare
* There is some wrong setting for detecting edge versus levels

While doing line by line elimination and comparison with working code, it was found that in a loop, counter related interrupts were being initialized again and again. How this led to weird behavior is not known though.

1. **System hangs after an hour of running**

This problem was found after lot of code reading and analyzing various symptoms detected at the time of hang-up, some of them are:

* All transmission stops
* Interrupt specific functionality continues to work
* No corruption on display (LCD)
* Time after which it hangs was predictable

It was challenging as the time to wait for this to recur was more than an hour. In one of the experiments with PWM, system started to hang within 5 minutes and it was suspected that it is the same issue. In this case, an additional symptom surfaced. LCD display could not show up what was displayed on USART. Both were supposed to print the same. Timer overflow counter was being displayed. This had reached FFFA. This pointed to some problem related to variable overflow.

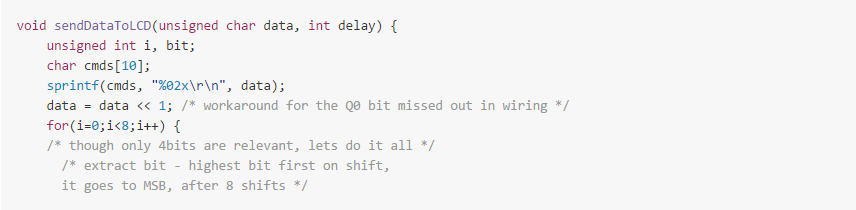
Investigation into the Interrupt Routine led to uncovering one fault, but that didn't help. It was

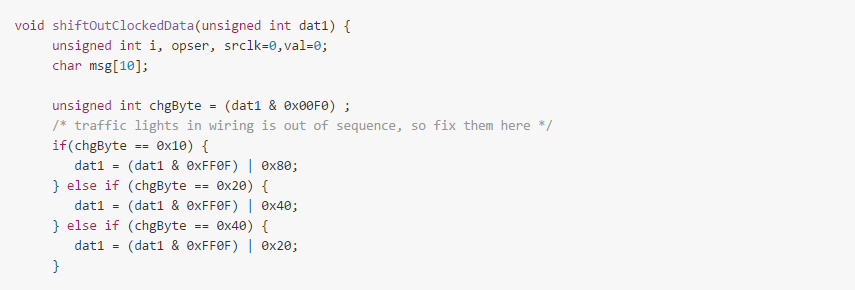
``timestamp written for assignment instead of timestamp[0]``

Next place for investigation were the users of this ISR. Wait function. Here detailed review, helped identify that a case of counter overflow, where higher word of timestamp becomes less than compared number, while lower byte was greater. This case was not handled and hence wait never used to get over when it reaches this scenario.

1. **Mistakes in Hardware, software workaround**

There were two instances, where hardware mistake occurred. One - LEDs for traffic lights were not in logical sequence. Second - Shift register output lowest bit (D0) was left unused for LCD data input. In both cases, software was enhanced to deal with situation. Clearly, it gives a performance penalty because of additional code. Code extract is captured below:

****

****

**Bibliography**

**Datasheets**

* ATmega 328
* LCD module RG1602A
* Shift Registers (74HC595 and 74HC164)
* MAX 232
* Arduino Uno Rev 3
* AVR Instruction Set

**Useful URLs**

* <http://bildr.org/2011/02/74hc595/>
* <http://bildr.org/2011/08/74hc595-breakout-arduino/>
* <https://www.arduino.cc/en/Tutorial/HomePage>
* <http://www.programiz.com/c-programming>
* <http://markdowntutorial.com/>
* <http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/>
* <https://en.wikipedia.org/wiki/Moore's_law>
* <https://help.ubuntu.com/community/UsingTheTerminal>
* [www.atmel.com/images/Atmel-0856-**AVR**-**Instruction**-**Set**-**Manual**.pdf](http://www.atmel.com/images/Atmel-0856-AVR-Instruction-Set-Manual.pdf)
* <http://www.rowebots.com/services/embedded_hardware_design_and_development>
* web.eecs.umich.edu/~mazum/F02/lectures/lec**simulation**.pdf
* <http://www.atmel.com/products/microcontrollers/avr/default.aspx>

**Learnings**

- **Technical Learnings**

Apart from the things like microcontroller, peripherals I learned (mentioned above) in this training, there are some things which a person should know before implementing a project. Some of these which I learnt during or after making this project are: -

* Layout- Before making a circuit on Vero board, the layout of the circuit on the board should be made.
* Spacing- There should be proper spacing between all the peripherals because there are chances of short circuit.
* Neatness- The circuit should be clean.
* Reading the manual- One should read the user manual or datasheets of the peripherals used in the circuit because the main idea how to implement the logic through that peripheral comes from the manual itself
* Code Review- One should review the code after writing because mainly bugs are identified after reviewing again and again.
* Moving step by step-
* Documentation- There should be a documentation of all the stuff which one is doing for further reference. Github is a nice platform for that. By this you can share information on web and others can learn from your experience.
* **Other Learnings**

Main thing which a person should know that hardware is smaller part of the industry and software is larger one. Without software there is very less scope for hardware. The reason behind is that once a hardware is manufactured, that will run for some years, but software is the thing which keeps on modifying time to time.

The one of the main values which I took from this training is management of time. One should work on time and manage it such that maximum output is generated from it. There are lots of uncertainties in today’s corporate world. So to differentiate from the world one should think differently and manage time accordingly.