Waterfall Graphic Print

Deepanath C[1], Laxmi Tirumala[2], Prasanth B J[3], Onkar Ghagare[4] and Vishal S[5].

[1] Department of Mechanical Engineering, IIT Madras (2nd Year UG)
[1] deepanathbuji@gmail.com
[2] Department of Chemical Engineering, IIT Madras (2nd Year UG)
[2] ltirumala@gmail.com
[3] Department of Electrical Engineering, IIT Madras (2nd Year UG)
[3] prasanthbj05@gmail.com
[4] Department of Bio-Technology, IIT Madras (2nd Year UG)
[4] onkarghagare@gmail.com
[5] Department of Metallurgical and Materials Engineering, IIT Madras (2nd Year UG)
[5] tgvishes@gmail.com

Table of Contents:

1. Introduction

Our project “waterfall graphic print” can be described as a water screen. In general, we use LED’s as a display element but we are using water as the display element, which is very unique. This is achieved by developing an electronic structure using Arduino-UNO microcontroller, shiftregister, triacs, resistors, SD card module, MOC 3020/21/23; a mechanical structure which includes a water tank, reservoir, pump, stand and the heart of the project ‘solenoid valves-64’. Characters which we display are the combination of water droplets, ‘ON’ state of the solenoid valve and air spaces, ‘OFF’ state of the valve. Therefore, with the help of the electronic system we developed, the set of ON and OFF states of solenoid valves are fed into valves using binary numbers created based on a given image.

The binary numbers of corresponding several images are saved in a word file and fed into the SD card module. The data is read in the form of array of 64 binary numbers. Each array represents the set of pixel values in the respective row of the image. Thus, we get a continuous falling of several images. The most important part of this project is programming and we use Arduino UNO and CVG for programming. Though the mechanism used here is simple but its output will leave a long-lasting impact upon viewers.

We take one water tank and connect the solenoid valves to its bottom surface; two pipes at its sides, one as inlet and other as outlet. The inlet is connected to the reservoir with a pump in-between and the outlet is attached to the reservoir. This mechanism is used to circulate water to the tank put at 4 meters above the reservoir. Solenoid valves switching is done by triac-bt136 and MOC’s. We have used interesting patterns and popular images involving Harry potter theme and different TV series’ to entertain our audience.

2. Motivation

This was a project part of the Techno - Cultural Entertainment Show, Envisage v4.0, Shaastra 2016. Shaastra is the annual technical fest of Indian Institute of Technology Madras, Chennai, India. It is entirely student managed and holds the distinction of being the first such event in the world to be ISO 9001:2000 certified for implementing a Quality Management System.

The aim was to bring to life a combination of microcontrollers, ICs, wires and a plethora of display elements of colors into a creative lighting display. The show involves both technical and aesthetic elements and thus can be called as ‘Technotainment’.

When we were in search of new projects we came across this ‘water graphic print’, which is developed as a technology prevalently in Japan.
Later, we realized that this is organized by event organizers at many places like malls, colleges, company to display numbers and logos and different designs in air by using the water and to increase the value of the place to make money.
It can be used at public places for enjoyment and also for publicity of any product. So, we thought why students can’t develop this and our team decided to showcase this for our audience in ENVISAGE-4.0.

3. Technical Details

3.1 Electrical Connections

3.1.1 Electrical components used

- Custom Designed Printed Circuit Boards(PCBs)
- Microcontroller : Arduino Uno
- Shift Register : 74HC595N
- TRIAC – BT136/BTA12
- MOC3020/21/23
- Resistors(150 Ω , 47 Ω)
- 2/2 way Solenoid Valves (220 V AC)
- SD Card Module
- Soldering Station (solder rod, solder wire)
- Miscellaneous requirements : Multi/Single Stranded Wires, RMC wires, male and female Berg pins, 16 pin IC bases,6 pin IC bases, General Circuit Boards (GCBs), glue gun

3.2 Basic Circuit

```
     R1                       R2
       47 ohm                   150 ohm                      Load
+---/\/\/\----------+   +----/\/\/-------------+-----------o   o-------> 220V
+Vin               1|   |6                     |           bulb         Hot
                   +=====+ IC1                 | MT2
 Control side      |MOC  | TRIAC             +-+
                   |3020 | Driver          G | | TRIAC
                   +=====+                  /| | BT136          50 Hz
                   2|   |4  Load Side      / +-+
+-------------------+   |                  |_| MT1
Ground                  +-------------------+_|
                                              |
                                              +----------------------> 220V
```

<u>ON Condition:</u>
When 5V is applied to pin 1 & 2 of the opto-coupler (MOC3020), pin 6 & 4 gets connected internally (Within opto-coupler) and allows current to flow between them. This connection provides GATE current to the TRIAC (TRIAC pin 3) and TRIAC starts conducting the main current between pin 2 & 1, which completes the circuit and makes the solenoid valve on.
<u>OFF Condition :</u>
Now, when 0V is applied to opto-coupler, opto-coupler makes pin 6 & 4 electrically opened and doesn't allow any current to flow between them, as a result it stops the GATE current to TRIAC and the main current between pin 2 & 1 also stops, which finally turns off the solenoid valve.

3.3 Circuit Design

Since Arduino Uno has limited number of pins controlling 64 solenoid valves is difficult without using shift registers. Using shift registers the data can be sent serially to get parallel outputs.

Specs of the components used:

(Note: This section contains only brief description of the ICs used. For detailed information check out the datasheets of the ICs):

- 74HC595N (Shift Register): The Shift Register is a type of sequential logic circuit that can be used for the storage or the transfer of data in the form of binary numbers. This sequential device loads the data present on its inputs and then moves or "shifts" it to its output once every clock cycle. The 74HC595N IC is an 8-stage serial shift register with a storage register and 3-state outputs. Each shift register has 8 output pins and hence can control 8 independent solenoid valves. More (In this case 8) shift registers can be connected in series to simultaneously control any number (in this case 64) of solenoid valves. The shift register gets input from the Arduino. It has 4 control pins that are controlled using output pins of Arduino. The Master reset (pin no. 10) and output enable (pin no. 13) pins of the shift register are active when low. The other two control pins include shift register clock input (SHCP/clk; pin no. 11) and storage register clock input (STCP/latch; pin no. 12). The shift register clock input pin is shorted with the serial clock pin of Arduino Uno (pin no. 11) to synchronise data transmission generated. Shift register receives input via the serial input pin (DS; pin no. 14). DS pin of the shift register is connected to MOSI (Master Output Slave Input) pin of Arduino Uno (pin no. 11). The signal from the output pins of the shift register is fed to the MOC3020/21/23 as input.

- MOC3020/21/23 (Opto Isolator): Opto-isolators or Opto-couplers, are made up of a light emitting device, and a light sensitive device, but with no electrical connection between the two, just a beam of light. When logic high is given from the shift register output pin current flows through LED from pin1 to 2. So in this process LED light falls on DIAC causing 6 & 4 to close. The MOCs are designed for interfacing between electronic controls and power triac to control resistive and inductive loads for Vac operations. The isolation between the low power and high power circuits in these optically connected devices is typically few thousand volts.

- BT136/BTA12 (TRIAC): The TRIAC is an ideal device to use for AC switching applications because it can control the current flow over both halves of an alternating cycle. TRIAC enables both halves of the AC cycle to be used. Hence in our circuit it is used after MOC to perform AC switching.

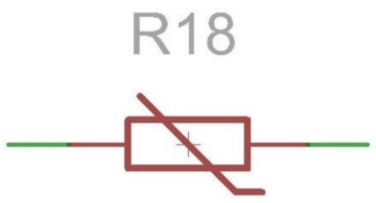
- Solenoid Valve: We used 2/2 220V AC Operated NC (Normally Closed) Solenoid Valve. The inlet and outlet diameter of the valve was ¼".

- Soldering GCBs: The circuit for the prototype was soldered on General Printed Boards (GCBs).

3.4 PCB Design

The first step when it comes to using a customized PCB is its design. The PCB's used were designed using Eagle 6.4.0 Professional as it is easily available and easy to use.

The design starts with a schematic file (with .sch extension). All the components that would be present on the board were added from the inbuilt library onto the schematic file. Sometimes, a particular model of the required component may not be available in the library, so we search for alternative models and pick the one with the same dimensions and geometry (of the soldering holes)as our required component and leave an appropriate space on the design (in the .brd file) for the component to fit while soldering. We just need the soldering holes to be of the exact dimensions and layout. Once all the components have been added to the schematic, we start connecting them. This is simple. We use the "Wire" command. One could actually draw the wires completely from one end of a component to the other end, but this would make the schematic look messy especially when there are a lot of wires running around all over the board. Instead we could just draw a small pieces of wires at the ends where they would be connected and then "Label" them with the same Name to connect them automatically as shown below.



Once all components are added and all the required connections are made on the schematic, proceed to board (.brd) file. Here is where optimization comes in. The components are placed in such a way that the area which the board occupies should be less (cost increases with area and number of layers) and also the routing should be simple. Routing is done so that no two traces on the PCB cross over. Routing is done first through the "AutoRoute" tool. If it shows 100% optimization, then there is no need for manual routing. In our case, auto routing was sufficient.

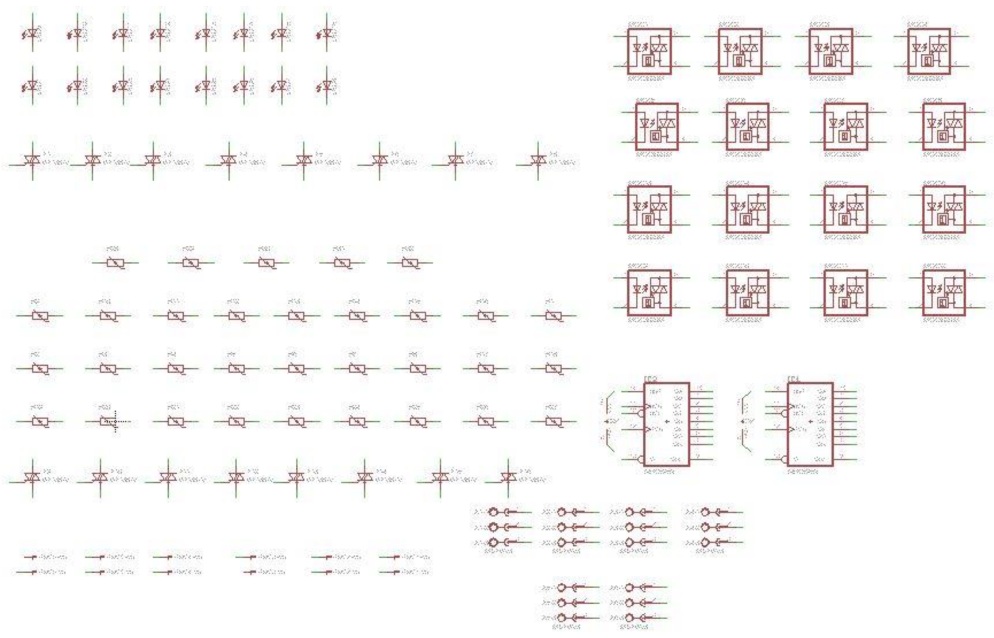
Once this is done, the board should be labelled and mount holes can be added so that the board can be fixed to surfaces easily.

To generate these files go to Camprocessor->file->open->job->gerb274x.cam. Click on 'open'. And then click on 'Process Job'.

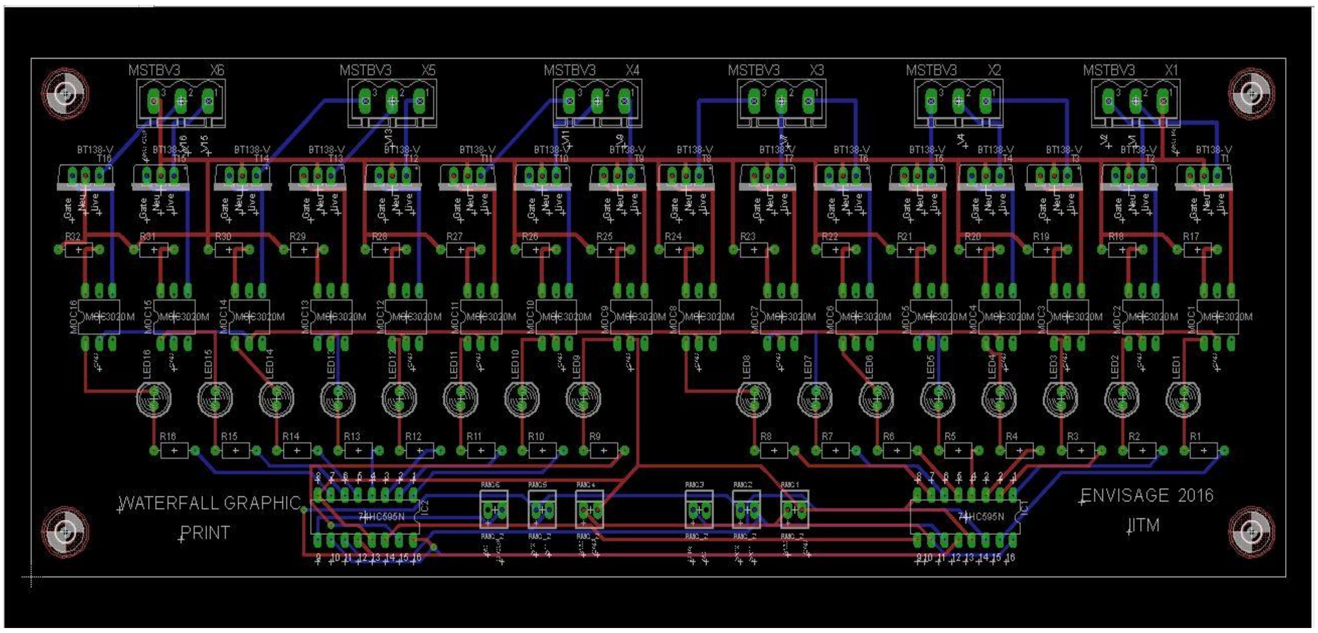
On doing this a set of files get generated in the working directory that has the schematic and board versions of the project file. Compress the folder that has these files and mail the zipped file to the PCB printing agency. The contact details of the vendor who got our boards printed can be found at the end of this section.

3.4.1 Details of our PCB

The Schematic:



The board:



It is a 2 layered board. There are 4 such boards to connect 64 valves (16 per board). There is another GCB (master) which has Arduino mounted on it, an SD Card adapter and a 3.3V regulator.

<u>Contact details of vendor:</u>

PRINTED CIRCUIT SERVICES

I/1,Dr.V.S.I.Estate,Thiruvanmiyur,Chennai-600 041

Phone: 22541641/42, Fax: 2254 1618.E-mail: zetapcs@yahoo.co.in

3.5 Coding

The coding part consists of image processing and Arduino programing.

The whole aim is to convert any image into a binary image which has only 1’s and 0’s and send it to the valves through the Arduino and the ‘1’ corresponds to the ‘ON’ state and ‘0’ corresponds to the ‘ OFF’ state of the valve.

3.5.1 Image processing

Initially the image is read and it inverted because the displayed image is going to be inverted just like the output from a printer. After inverting the image is made into binary using the function Threshold(). The function AdaptiveThreshold() can also be used but however the former one suited our purpose and hence we proceeded with it. Now the image only consists of ‘255’s and ‘0’s hence by dividing it by two hundred fifty five we would have succecfully converted our image to a matrix which has only ‘1’s and ‘0’s. This is then written into a text file and stored in the drive.

```
//*****************************************************
//                    HEADER FILES
//*****************************************************
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "opencv2/highgui/highgui.hpp"
#include "opencv2/imgproc/imgproc.hpp"


using namespace cv;
using namespace std;

//*****************************************************
//                    VARIABLES
//*****************************************************
Mat src; Mat src_inv ;Mat src_blur;Mat src_thresh;
Mat map_x, map_y;
const int rows=200;
const int cols=64;
const int cols_ascii=cols/8;
int ascii[rows][cols_ascii];
int temp;
int number=0;

int main(int argc, char **argv )
{

  //READING THE IMAGE
  src = imread(argv[1], 0);
  imshow( "Source",src );
  blur( src, src_blur, Size(3,3) );
  map_x.create( src.size(), CV_32FC1 );
  map_y.create( src.size(), CV_32FC1 );

   //INVERTING THE IMAGE

   for( int j = 0; j < src.rows; j++ )
   { for( int i = 0; i < src.cols; i++ )
       {
           map_x.at<float>(j,i) = src_blur.cols-i ;
           map_y.at<float>(j,i) = src_blur.rows - j;
           }
      }

  namedWindow( "Binary" , CV_WINDOW_AUTOSIZE );
  namedWindow( "Small", CV_WINDOW_NORMAL );
  remap( src_blur, src_inv, map_x, map_y, CV_INTER_LINEAR,
BORDER_CONSTANT, Scalar(0,0, 0) );
  resize(src_inv, src_thresh, Size(cols,rows),0,0,CV_INTER_AREA);
  imshow( "Small",src_thresh );

  //CONVERTING IT INTO BINARY

  threshold( src_thresh, src_thresh, 15, 255,THRESH_BINARY );
  imshow( "Binary",src_thresh );
     for(int i=0; i<src_thresh.rows;i++)
      {
        for(int j=0;j<src_thresh.cols;j++)
            {

src_thresh.at<uchar>(i,j)=int((src_thresh.at<uchar>(i,j))/255);
            }
      }

  //CONVERTING IT INTO ASCII

  for(int i=0; i<src_thresh.rows;i++)
      {
        for(int j=0,z=0;z<8;j+=8,z++)
            {

ascii[i][z]=(128*(int(src_thresh.at<uchar>(i,j+0))))+(64*(int(src_th
resh.at<uchar>(i,j+1))))+(32*(int(src_thresh.at<uchar>(i,j+2))))+

(16*(int(src_thresh.at<uchar>(i,j+3))))+(8*(int(src_thresh.at<uchar>
(i,j+4))))+(4*(int(src_thresh.at<uchar>(i,j+5))))+
(2*(int(src_thresh.at<uchar>(i,j+6))))+((int(src_thresh.at<uchar>(i,
j+7))));
            }
      }


   //DISPLAYING THE BINARY IMAGE

   for(int i=0; i<src_thresh.rows;i++)
      {
        for(int j=0;j<src_thresh.cols;j++)
            {
              cout<<int(src_thresh.at<uchar>(i,j));
            }
            cout<<endl;
      }
     ofstream myfile;
     cout<<"Count"<<number<<"\n";

  //WRITING IT TO THE FILE

  myfile.open (argv[2], ios::out );
  if (myfile.is_open())
   {
      for(int k=0;k<src_thresh.rows;k++)
          for(int l=0;l<src_thresh.cols;l++)
              {
                myfile << int(src_thresh.at<uchar>(k,l));
              }
   }
     myfile<<'\n';

  myfile.close();



  waitKey(0);
  return(0);
}
//******************************************************
//                    END OF PROGRAM
//******************************************************
```

A lot of thought was also given into compressing the image in such a way as to nullify the blow up of the image as the water falls down due to gravity. But, after repeated trials with different images and different compression algorithms it was decided to move forward without any compression.

3.5.2 The image file

After converting all the required images into binary, A text file is created in which the binary images are copied one after the other in the required order and is fed into the SD card.

3.5.3 Arduino Coding

The work of the Arduino code is to read the file from the SD card and send the data to the valves.

```
//***************************************************************
//   CODE TO WRITE INFORMATION INTO EIGHT SHIFT REGISTERS
//                   USING COMMON PINS
//***************************************************************

//Pin connected to ST_CP(12) of 74HC595
int latchPin = 6;
//Pin connected to SH_CP(11) of 74HC595,CLK od SD
int clockPin = 7;
//Pin connected to DS(14) of 74HC595,MOSI
int dataPin = 4;

//Pin connected to SS of SD card
int S_S = 10;


//***********************************************
//                  HEADER FILES
//***********************************************
#include <SPI.h>
#include <EEPROM.h>
#include <SD.h>



// Variables for holding values
int d=10;
File myFile;
```

```
int pinstate;
char c;

void setup()
{
  pinMode(latchPin,OUTPUT);
  pinMode(clockPin, OUTPUT );
  pinMode(dataPin, OUTPUT );
  pinMode(S_S, OUTPUT);
  Serial.begin(57600);
  while (!Serial) {
    ; // wait for serial port to connect. Needed for Leonardo only
  }
  Serial.print("Initializing SD card...");
  if (!SD.begin(S_S)) {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

    // re-open the file for reading:
  myFile = SD.open("main.TXT");
  if (myFile) {
    Serial.println("main.txt:");

    // read from the file until there's nothing else in it:
    // close the file:
  } else {
     // if the file didn't open, print an error:
    Serial.println("error opening main.txt");
  }
}//  END OF SETUP




void loop()
{
 digitalWrite(latchPin,0);
 for(int i=0;i<64;)
 {
   c=myFile.read();
   digitalWrite(clockPin,0);
   digitalWrite(dataPin,0);
   if(c=='1')
   {
     pinstate=1;
     i++;
     digitalWrite(dataPin,pinstate);
     Serial.print(pinstate);
     digitalWrite(clockPin,1);
   }
    else if(c=='0')
    {
      pinstate=0;
      i++;
      digitalWrite(dataPin,pinstate);
      Serial.print(pinstate);
      digitalWrite(clockPin,1);
     }

  }
  digitalWrite(latchPin,1);
  Serial.println();
  delay(d);
  if(myFile.available()==0)
   {
     myFile.close();
     return;
   }
}//    END OF LOOP
```

The above code reads a number from the file and then send HIGH or a LOW signal to the shift register depending upon the data received. When the next data is received firs a little and then

A shift register basically consists of several single bit D-Type Data Latches, one for each data bit, either a logic 0 or a 1, connected together in a serial type daisy-chain arrangement so that the output from one data latch becomes the input of the next latch and so on. So, once the entire byte is being transferred, the first bit reaches the last data latch of the shift register and the last bit in the sequence in at the first data latch. The speed of propagation of these bits depends on the clock frequency of the microcontroller, which is 16MHz for arduino Uno. But , in this case sufficient time has to be given so that the valves has enough time to respond and the the valves being slower than the Arduino will determine the delay time.

Now, eight shift registers are being used in this. When the second byte is being transferred, the first byte gets shifted to the second shift register. In this case the first shift register acts as a master and the second shift register acts as a slave and similar explanation applies to other shift registers too. So, finally we have transferred two bytes of data across two shift registers. The output from the shift registers resembles the data in the data latches, that is, the output is ’HIGH’ for logic ’1’ and the output is ’LOW’ for logic ’0’. The 8 output pins are labeled as Q0, Q1, Q2,....,Q7. The first bit of the first byte goes with Q7 of the second shift register. The last bit of the last byte goes with Q0 of the first shift register. The valves have to be properly connected and the code is written accordingly. The command ’delay(t)’ ensures that the output remains in the same state for ’t’ milliseconds until the next set of bytes flush out this data. There was a quite a bit of discussion in making this delay dependent on the image but later it was decided to go on a constant delay and to change the image as required.

All the codes used in the project are in the link:

https://drive.google.com/open?id=0By58FCn0UGueZl9JQlV1X0g3ajg

3.6 Mechanical Requirements

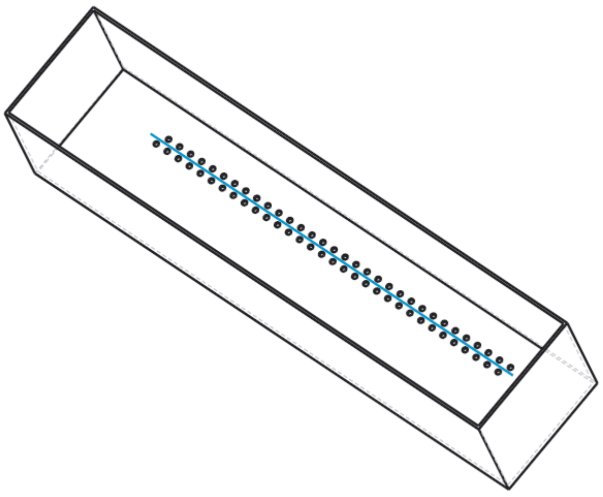
3.6.1 Coming with a proper overhead tank

Objective:

Water supply to all 64 solenoid valves at once.

Our model:

- We had an overhead tank of made of PP (Polypropylene) sheets and we used plastic welding to stick the sides and to avoid leakages we welded on both inner and outer side of the tank.
- The tank has 64 drilled holes such that 32 are in one side of the axis and the rest on other side. This arrangement helps us to arrange the valves in a zigzag manner which was done to decrease the distance between two valves to increase the pixel density for more resolution of images falling down.
- All 64 Solenoid Valves are fixed to the respective holes using proper connectors and coated with leak-proof agents.



3.6.2 Stand to hold the overhead tank

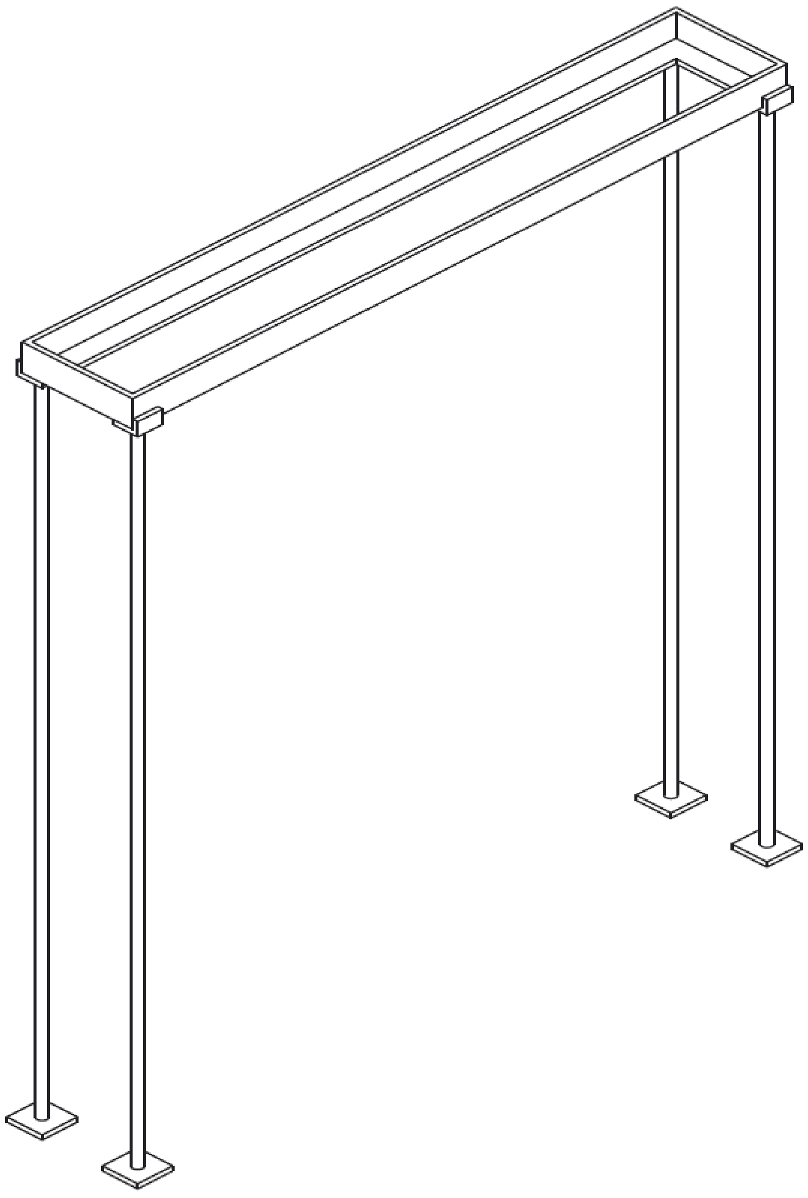
Objective:

- Hold the tank at a height of 12 feet. The model of stand we came up is simple 4-point supported stand supporting the tank at 4 corners using L-Channels.
- To ensure stability we added four pole system cover the 4-Point support system.

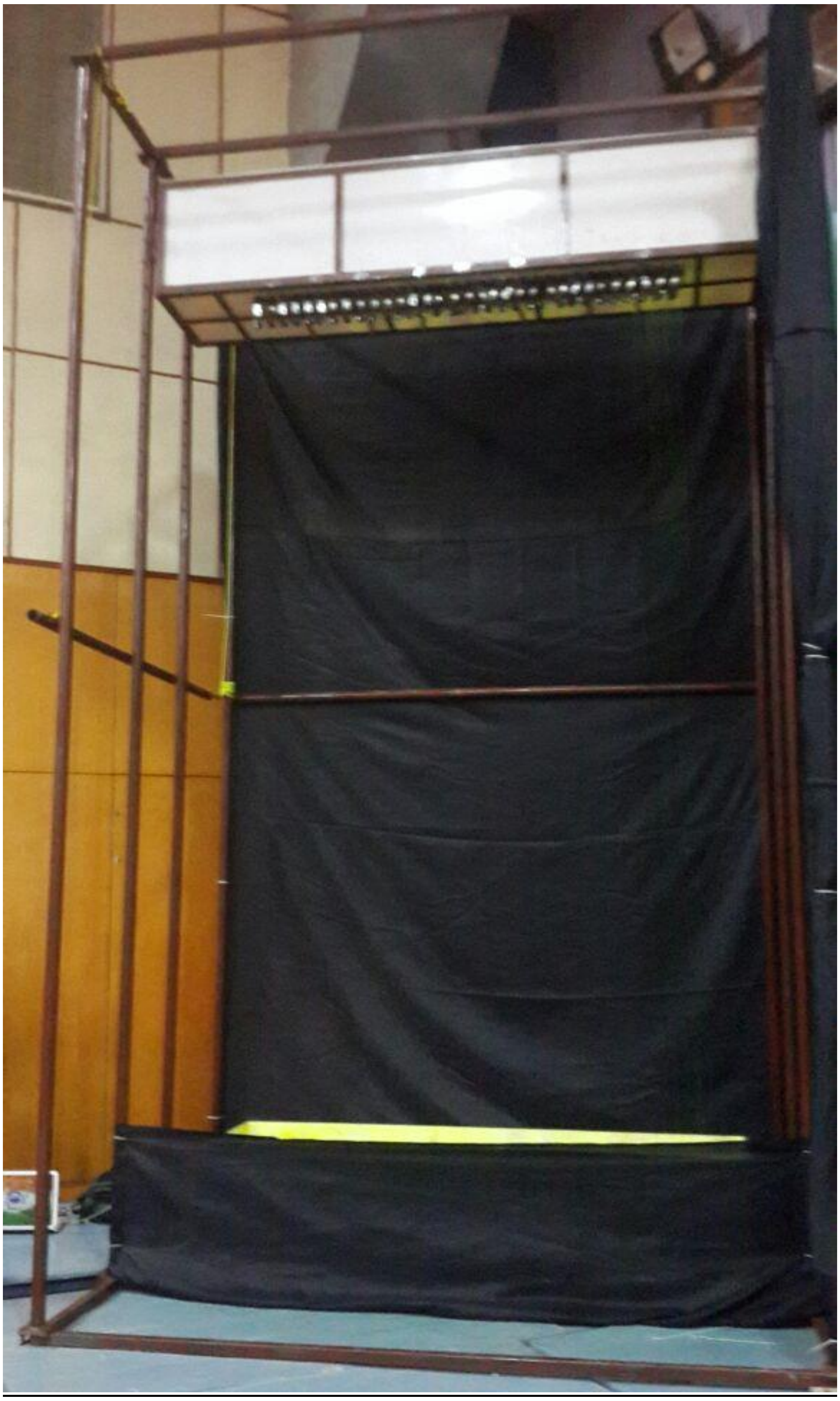
Various Plans for Stand:

Final Plan is extending the Plan 1 with one more arrangement.

Plan 1:



Final Set-Up:



3.6.3 Reservoir

Objective:

- To collect the water from solenoid valves and re-circulate them to the overhead tank.
- The dimensions were kept bit larger than the overhead tank to avoid loss of water due to spillage.
- Reservoir was made of wood and given waterproof coating.

3.6.4 Pump

Objective:

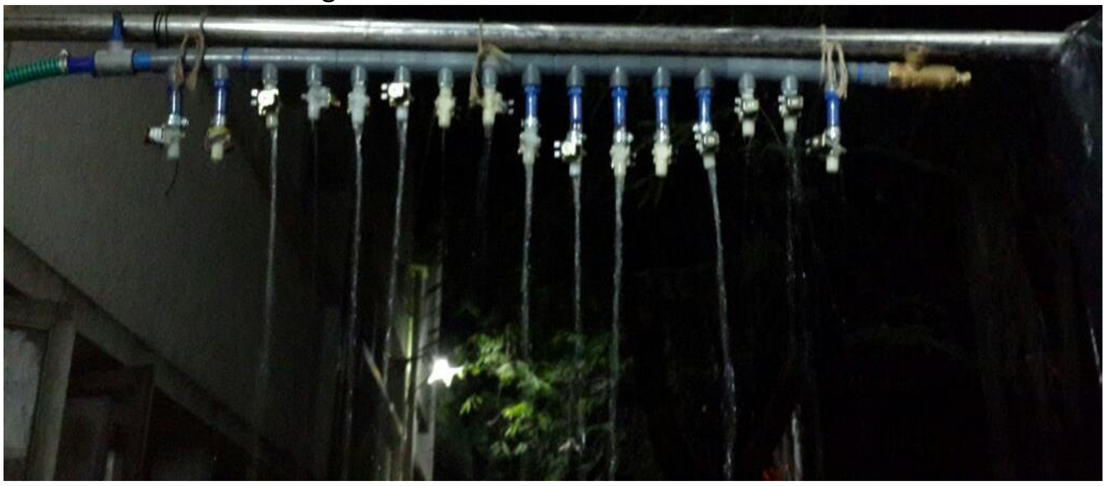
Re-circulation of water from overhead tank through the solenoid valves to reservoir again the overhead tank.

Specifications:

0.5 HP single phase AC Pump.

4. Problems

1) Working with GCB is very hard because of many loose connections and wiring errors. This is sorted out when we shifted working with PCB.
2) In GCB, triacs 2nd and 3rd pins are always shorted making the valves be always in 'ON' state. This because of excess usage of flux in board soldering. Due to this, when high voltage is given to 1st and 2nd pins the current passes through flux creating a short. This is solved by cleaning the excess flux with IPA solution and soldering without flux.
3) First we have chosen 1 inch solenoid valve, but this made the water flow as a continuous flow but not a flow with water droplets and air spaces.
4) In our prototype, we used a pipe with 12 valves fitted to it. When water is pumped into it, water was flowing through only first and the flow became thinner when compared to the first one from the inlet side. This is because, flow rate is low and the pressure is not uniform. So, we used a tank to create a uniform pressure and filled it in advance to have enough volumetric flow.



5) The excessive usage of valves made its efficiency decrease and become hot.
6) Due to the electro -magnetic effect of coils in the solenoid valves, the adjacent valves work improperly because of induction.
7) In SD Card module, we have used a logic invertor instead of a resistor for voltage dividing because the circuit with resistors wasn't working properly.
8) One of the connections were wrong. So we cut the traces on the board and soldered wires manually.
9) Also there was no provision for connecting the neutrals of all the 4 boards together. Later this connection was made externally by using the two unused pins of RMC in each board and neutrals were connected through them.
10) Also, the traces where AC current was flowing was not wide enough.
11) These changes were incorporated in the final design of the PCBs.
12) Also, while we tested out the circuit on a GCB before the use of PCBs, we figured out that flux shouldn't be used while soldering (at least for soldering the Phoenix connectors on to the PCB).The flux would start conducting when 220 AC was applied to the board.

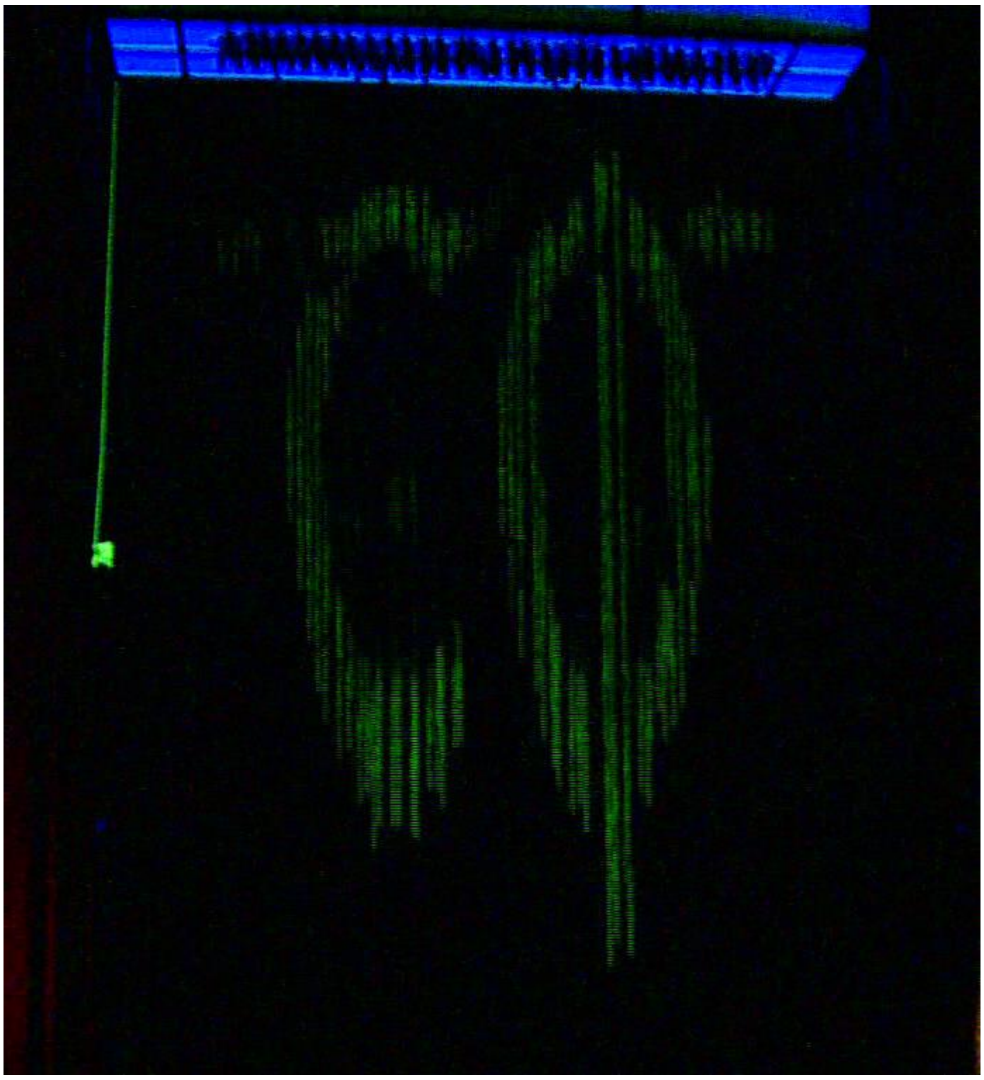
5. Final Show

Overall, there was a good response from the audience for the project. They could be heard continuously cheering throughout the show.

Here is the link for final Show video during Envisage 4.0, Shaastra - 2016:

https://drive.google.com/open?id=0B3kIyAj28xRZN0JqSHRlajljOEE

Few Glimpses of final Envisage show:



6. Acknowledgement

We would like to thank Envisage 4.0 team and Shaastra 2016 for giving us an opportunity to do this project. We would like to thank our mentors Koushik Balaji ,Netaji Babu, Karthik and Tejas Kulkarni for helping us identify the bugs whenever things didn't work the way they should have worked and offering us constant support. We would also like to thank our Envisage core Yashwanth and our Envisage QMS Coordinator Harshit for being patient and cooperative with us.

Overall, it was an amazing experience working in this project. It instilled in us enormous potential for perseverance and helped us explore the spirit of teamwork.