MINI PROJECT: BELL SYNCHRONIZATION AND AUTOMATION

INTRODUCTION

This project is named as bell synchronization and automation. This project deals with synchronization of the bells in our campus. Our campus has three different building for different programs, the given figure below gives the basic idea of the campus infrastructure.

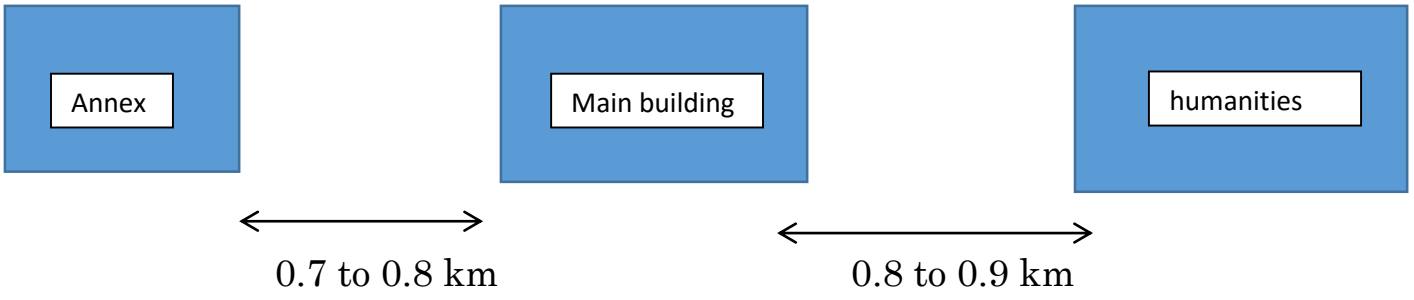

Figure 7.1: Block diagram of campus infrastructure

This distance given in the figure is the distance between the bells in the building. Each bell has an automation device which has a timer. The bell is triggered by the automation device. But all the timers are not synchronized so each bell ring at

different timings. For overcoming this problem, we have to connect each bell physically which is not cost effective and there are many other problems. To solve these real time problems, we wanted to connect the bells using the existing LAN network/Ethernet cables. Where we read the time from server and send the ringing message through Ethernet.

Basic idea for the project:

Use Arduino Uno board with Ethernet shield which can be connected to network and can be programed to trigger the relay and in the server side we can write a socket program in java language which can run in institute server, it reads time from net and sends a ringing message to Arduino Ethernet shield through Ethernet.

COMPONENT REQUIRED:

1. Arduino UNO microcontroller
2. Arduino Ethernet shield
3. Relays (5v, 7A)
4. Capacitors
5. Resistors
6. Jumpers
7. Connectors
8. 5v DC fan
9. Adaptor
10. Wooden board
11. Screws
12. A box with specific dimensions
13. Ethernet or LAN cable.

Demands:

1. Each day there are different timings for the bell.

2. Some particular timings bell should ring little longer than the normal timings like after prayers and before classes starts.

Arduino Uno board:

Figure 7.2 Typical Arduino Uno board

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board.

Over the years Arduino has been the brain of thousands of projects, from everyday objects to complex scientific instruments. Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded

environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.

It runs on Mac, Windows, and Linux. Teachers and students use it to build low cost scientific instruments, to prove chemistry and physics principles, or to get started with programming and robotics.

Arduino Ethernet Shield:

Figure 7.3 Typical Arduino Uno Ethernet shield board

The Arduino Ethernet Shield allows an Arduino board to connect to the internet. It is based on the Wiznet W5100 Ethernet chip (datasheet). The Wiznet W5100 provides a network (IP) stack capable of both TCP and UDP. It supports up to four simultaneous socket connections. Use the Ethernet library to write sketches which connect to the internet using the shield. The Ethernet shield connects to an

Arduino board using long wire-wrap headers which extend through the shield. This keeps the pin layout intact and allows another shield to be stacked on top.

Relays:

Figure 7.4: Typical Arduino Uno Ethernet shield board

A relay is an electrically operated switch. Many relays use an electromagnet to mechanically operate a switch, but other operating principles are also used, such as solid-state relays. Relays are used where it is necessary to control a circuit by a separate low-power signal, or where several circuits must be controlled by one signal. The first relays were used in long distance telegraph circuits as amplifiers: they repeated the signal coming in from one circuit and re-transmitted it on another circuit. Relays were used extensively in telephone exchanges and early computers to perform logical operations.

A type of relay that can handle the high power required to directly control an electric motor or other loads is called a contactor. Solid-state relays control power circuits with no moving parts, instead using a semiconductor device to perform switching. Relays with calibrated operating characteristics and sometimes multiple operating coils are used to protect electrical circuits from overload or faults; in modern electric power systems these functions are performed by digital instruments still called "protective relays".

Magnetic latching relays require one pulse of coil power to move their contacts in one direction, and another, redirected pulse to move them back. Repeated pulses

from the same input have no effect. Magnetic latching relays are useful in applications where interrupted power should not be able to transition the contacts.

Magnetic latching relays can have either single or dual coils. On a single coil device, the relay will operate in one direction when power is applied with one polarity, and will reset when the polarity is reversed. On a dual coil device, when polarized voltage is applied to the reset coil the contacts will transition. AC controlled magnetic latch relays have single coils that employ steering diodes to differentiate between operate and reset commands.

Designing:

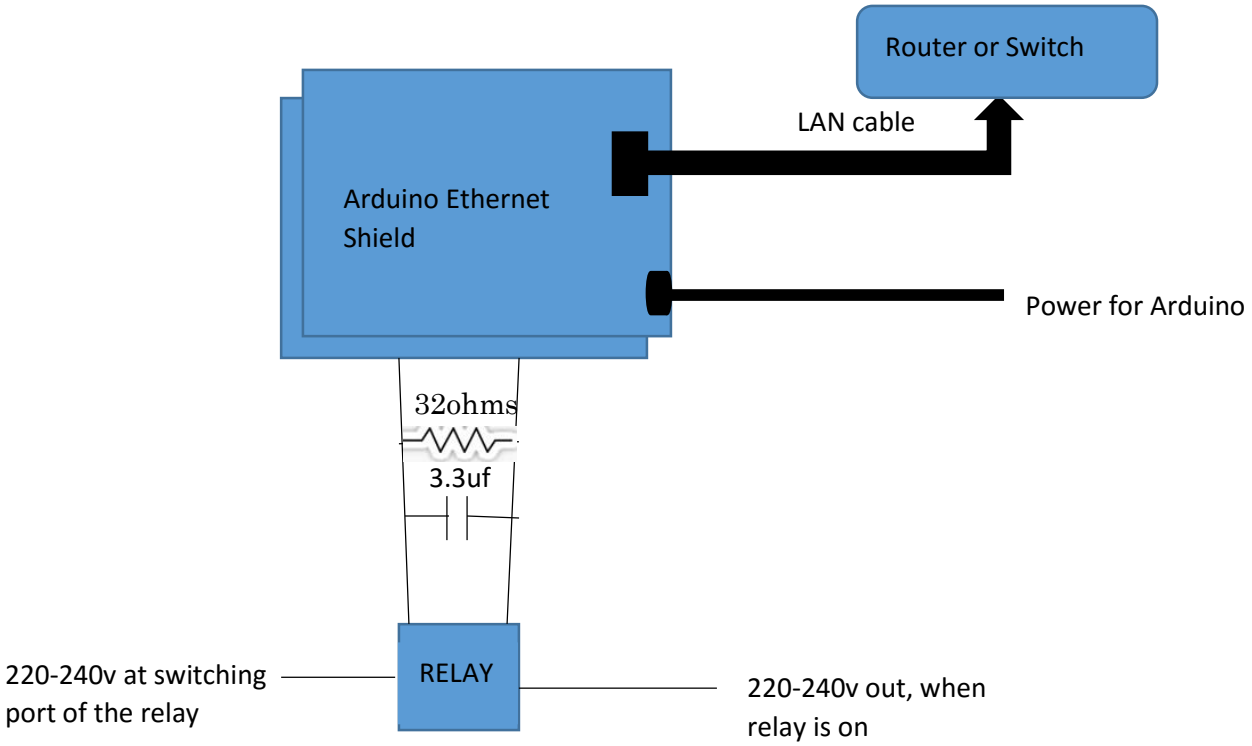

Figure 7.5: Block diagram of the internal part of the device

Arduino Ethernet shield is mounted on the Arduino UNO board and is connected to network using an Ethernet cable. Arduino is powered by a mobile

adaptor (5v, 1A). Whenever there is an input from the Ethernet port, Arduino makes one of the output port high, the same port is connected to the relay's five volts triggering pin.

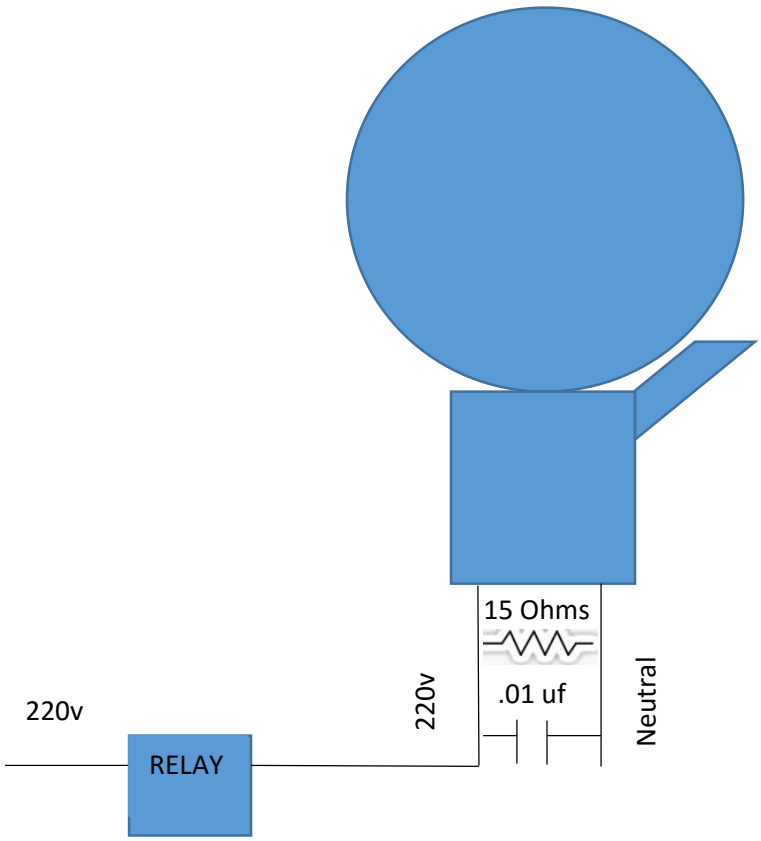

Figure 7.6: Connection at the bell end

Relay 220volts from relay is given to bell and neutral of bell is connected directly from the AC supply.

Counter electromotive force:

Counter-electromotive force (abbreviated counter EMF or simply CEMF), also known as back electromotive force (or back EMF), is the electromotive force or "voltage" that opposes the change in current which

induced it. CEMF is the EMF caused by magnetic induction (see Faraday's law of induction, electromagnetic induction, Lenz's Law).

For example, the voltage appearing across an inductor or "coil" is due to a change in current which causes a change in the magnetic field within the coil, and therefore the self-induced voltage. The polarity of the voltage at every moment opposes that of the change in applied voltage to keep the current constant.

The term back electromotive force is also commonly used to refer to the voltage that occurs in electric motors where there is relative motion between the armature and the magnetic field produced by the motor's field coils, thus also acting as a generator while running as a motor. This effect is *not* due to the motor's inductance but a separate phenomenon.

This voltage is in series with and opposes the original applied voltage and is called "back-electromotive force" (by Lenz's law). With a lower overall voltage across motor's internal resistance as the motor turns faster, the current flowing into the motor decreases. One practical application of this phenomenon is to indirectly measure motor speed and position, as the back-EMF is proportional to the rotational speed of the armature.

RC circuit:

we know that the bell is nothing but an electromagnet, since the coil inside the bell is having a high inductance it has very high back emf (~1500volts). Back emf due to the bell resets the Arduino board and due to this reason each time bell rings we need to manually reset the bell get it connect to the network. Even relay is also an electromagnet. To overcome this problem, we can use a RC circuit where R and C are parallel to each other. This also known as snubber circuit. Snubbers are an essential part of power electronics. Snubbers are small networks of parts in the power switching circuits whose function is to control the effects of circuit reactance. Snubbers enhance the performance of the switching circuits and result in higher reliability, higher efficiency, higher switching frequency, smaller size, lower weight, and lower EMI. The basic intent of a snubber is to absorb energy from the reactive elements in the circuit. The benefits of this may include circuit damping, controlling the rate of change of voltage or current, or clamping voltage overshoot. In performing these functions, a snubber limits the amount of stress which the switch must endure and this increases the reliability of the switch. When a snubber is properly designed and implemented the switch will have lower average power dissipation, much lower

peak power dissipation, lower peak operating voltage and lower peak operating current.

Further Demands requested:

1. There should be a GUI, such that anyone can run the program or rather use it without even knowing about the programming.
2. GUI should run in the background, such that it does not disturb the user when he is doing some other work in system.
3. Thursday the second bell should not ring, since the moral class continues in the auditorium just after the prayer, so there should not be any disturbance due to the bell.
4. The program should automatically run whenever server is coming on.
5. If there is some network issue, then we should be able to ring it by sending a manual ring message.
6. And there should be a manual control for the bell.

Programming:

Programming is done in such a way that it satisfies all the need of the customer.

Program for Arduino board:

This program should take care of long and short rings and also triggering of the relay whenever there is a ring message from the server.

Arduino program:

```
/*
  Web Server
```

```
 A simple web server that controls bells
 using an Arduino Wiznet Ethernet shield.

 Circuit:
 * Ethernet shield attached to pins 10, 11, 12, 13
 * Pin 8 is connected relay

 created 18 Dec 2009
 by David A. Mellis
 modified on 19 aug 2017
 by Satya prasanna

 */

#include <SPI.h>
#include <Ethernet.h>

// Enter a MAC address and IP address for your controller below.
// The IP address will be dependent on your local network:
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED
};
IPAddress ip(192, 168, 12, 90);
IPAddress gateway(192, 168, 28, 1);
IPAddress subnet(255, 255, 255, 0);
```

```
// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(5000);

void setup() {
  // Open serial communications and wait for port to open:
  Serial.begin(9600);
//  while (!Serial) {
//    ; // wait for serial port to connect. Needed for native USB port only
//  }

pinMode(8,OUTPUT);
//pinMode(4,OUTPUT);
  // start the Ethernet connection and the server:
  Ethernet.begin(mac, ip, gateway, subnet);
  server.begin();
  Serial.print("server is at ");
  Serial.println(Ethernet.localIP());
}
void loop() {
//  digitalWrite(4,HIGH);
  // listen for incoming clients
  EthernetClient client = server.available();
  if (client) {
    Serial.println("new client");
    // an http request ends with a blank line
```

```
//    boolean currentLineIsBlank = true;
digitalWrite(8,LOW);
    while (client.connected()) {
      if (client.available()) {
        char c = client.read();
        Serial.write(c);
        // if you've gotten to the end of the line (received a newline
        // character) and the line is blank, the http request has ended,
        // so you can send a reply
        if (c == 'l') {
          digitalWrite(8,HIGH);
          delay(8000);
          digitalWrite(8,LOW);
          break;
        }
         if (c == 's') {
          digitalWrite(8,HIGH);
          delay(5000);
          digitalWrite(8,LOW);
          break;
         }
      }
    }
    // give the web browser time to receive the data
    delay(1);
    // close the connection:
    client.stop();
```

```
    Serial.println("client disconnected");
  }
}
```

Java programming:

Server side programming is written in JAVA language. Server side programming takes care of all other demands whenever the conditions are satisfied it sends a ringing message (l or s, l is for a long ring and s for a short ring) to the arduino board.

JAVA program part 1:

```
import java.io.*;
public class BellServerMain
{
   public static void main(String[] args)
   {
      try
      {
         if(args.length != 2)
         {
            System.out.println("usage: java BellServerMain <ip filename> <timings filename>");
            System.exit(-1);
         }

         System.out.println("\n\n\t\t\tWelcome to Sai Bell Services");
         new BellServer(args[0],args[1]);
         new ManualControl();
         System.out.println("\n\n\t\t\tBell synchronization system is now running");
      }
      catch(FileNotFoundException f)
      {
         System.out.println("One of the files is not found quitting");
          f.printStackTrace();
      }
      catch(IOException i)
      {
         i.printStackTrace();
      }
```

```
    }

}
```

Part 2:

```
import java.io.*;
import java.util.*;
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class BellServer implements Runnable
{
   private ArrayList<String> ip;
   private ArrayList<String> timings;
   private static int nextSlot;
   private int totalTimeSlots;
   private ArrayList<String> duration;
   private Calendar cal;
   private SimpleDateFormat hr;
   private SimpleDateFormat min;
   private SimpleDateFormat sec;
   private Boolean tomorrowHoliday;

   private void exitMessage(String msg)
   {
        System.out.println("\n\n\t\t\t"+msg+"\n\n");
        System.exit(0);
   }

   private void checkIp(String ip)
   {
        String ipParts[] = ip.split("\\.");

        if(ipParts.length != 4)
             exitMessage("Invalid entry for ip address: "+ip);

        for(String s:ipParts)
             if(Integer.parseInt(s)<0 ||Integer.parseInt(s)>255)
                  exitMessage("Invalid entry for ip address: "+ip);
   }
   public BellServer(String ipFile,String timingFile) throws FileNotFoundException, IOException
   {
```

```
BufferedReader reader = new BufferedReader(new FileReader(ipFile));
ip = new ArrayList<String>();
String str;

while((str = reader.readLine())!=null)
{
    checkIp(str);
    ip.add(str);
}

reader.close();

if(ip.isEmpty())
    exitMessage("No ip details found. Please check your ip file");

reader = new BufferedReader(new FileReader(timingFile));
timings = new ArrayList<String>();
duration = new ArrayList<String>();

try
{
    while((str = reader.readLine())!=null)
    {
            timings.add(str.substring(0,str.indexOf(",")));
            duration.add(str.substring(str.indexOf(",")+1,str.length()));
    }

    reader.close();
}
catch(StringIndexOutOfBoundsException a)
{
    exitMessage("Duration missing for some timings");
}

if(timings.isEmpty() || duration.isEmpty())
{
    exitMessage("No timing or duration details found. Please check your timing file");
}

totalTimeSlots = timings.size();
cal = Calendar.getInstance();

hr = new SimpleDateFormat("HH");
min = new SimpleDateFormat("mm");
```

```
    sec = new SimpleDateFormat("ss");

    tomorrowHoliday = false;

    Thread t = new Thread(this);
    t.start();
}

private int calTimeSlotsPassed()
{
    int timeSlotsPassed = 0;
    int nowHr = Integer.parseInt(new String(hr.format(cal.getTime())));
    int nowMin = Integer.parseInt(new String(min.format(cal.getTime())));
    int hr,min;

    for(String time:timings)
    {
        hr = Integer.parseInt(time.substring(0,time.indexOf(":")));
        min = Integer.parseInt(time.substring(time.indexOf(":")+1));

        if(hr == 24)
         hr = 0;
        if(min == 60)
        {
         hr += 1;
         min=0;
        }

        if(hr < 0 || hr > 23 || min < 0 || min > 59)
         exitMessage("Invalid time entry");

        if((nowHr>hr) || (nowHr == hr && nowMin > min))
            ++timeSlotsPassed;
    }
    return timeSlotsPassed;
}

private void ring()
{
    for(String s:ip)
        new SendMessage(s,duration.get(nextSlot));
}

private long calTime2Sleep(int nextSlot,int today)
```

```
  {
        int nowHr,nowMin,nextHr,nextMin,nowSec;
        long current,upcoming;

        cal = Calendar.getInstance();
      nowHr = Integer.parseInt(new String(hr.format(cal.getTime())));
      nowMin = Integer.parseInt(new String(min.format(cal.getTime())));
      nowSec = Integer.parseInt(new String(sec.format(cal.getTime())));

      if(today == Calendar.SUNDAY)
        nextSlot = 0;

      nextHr =
Integer.parseInt(timings.get(nextSlot).substring(0,timings.get(nextSlot).indexOf(":")));
      nextMin =
Integer.parseInt(timings.get(nextSlot).substring(timings.get(nextSlot).indexOf(":")+1));

        current = (nowHr*3600*1000)+(nowMin*60*1000);//+(nowSec*1000);
        upcoming = (nextHr*3600*1000)+(nextMin*60*1000);

        if(today == Calendar.SUNDAY)
        {
                if (current < upcoming)
                return (upcoming-current) - (nowSec*1000) + (24*3600*1000);
        else if(current > upcoming)
                return upcoming + (24*3600*1000) - current - (nowSec*1000);
        else
                return (24*3600*1000);
        }

        if(tomorrowHoliday)
        {
                tomorrowHoliday = false;
                if (current < upcoming)
                return (upcoming-current) - (nowSec*1000) + (24*3600*1000);
        else if(current > upcoming)
                return upcoming + (24*3600*1000) - current - (nowSec*1000) + (24*3600*1000);
        }

        if (current < upcoming)
                return (upcoming-current) - (nowSec*1000);
        else if(current > upcoming)
                return upcoming + (24*3600*1000) - current - (nowSec*1000);
        else
```

```
                return 0;
    }

    public void run()
    {
       long time2Sleep;
       int today,nowSec;

       nextSlot = calTimeSlotsPassed();

       nextSlot %= totalTimeSlots;

       while(true)
       {
          try
          {
                today = cal.get(Calendar.DAY_OF_WEEK);

          time2Sleep = calTime2Sleep(nextSlot,today);

                                Thread.sleep(time2Sleep);

                                ring();
          cal=Calendar.getInstance();
          nowSec = Integer.parseInt(new String(sec.format(cal.getTime())));

                          Thread.sleep((60-nowSec)*1000);
                          ++nextSlot;

          if(nextSlot == timings.size() && today == Calendar.SATURDAY)
                          tomorrowHoliday = true;

          nextSlot %= totalTimeSlots;
          }
          catch (InterruptedException e)
          {
                                e.printStackTrace();
                          }

       }
    }
}
```

Part 3 for manual control:

```
import java.util.Scanner;

public class ManualControl implements Runnable
{
	public ManualControl()
	{
		Thread t = new Thread(this);
		t.start();
	}

	@Override
	public void run()
	{
		Scanner sc = new Scanner(System.in);
		String temp;
		while(true)
		{
			try
			{
			   temp = sc.nextLine();
			   if(temp.equalsIgnoreCase("quit") || temp.equalsIgnoreCase("exit"))
			   {
				System.out.println("\n\n\t\t\tsee you next time");
				sc.close();
				System.exit(0);
			   }
			   if(temp.length() > 0)
			      new
SendMessage(temp.substring(0,temp.indexOf(",")),temp.substring(temp.indexOf(",")+1)) ;
		}
		catch(ArrayIndexOutOfBoundsException a)
		{
		}
		}
	}

}
```

Part 4 for sending message to Arduino:

```
import java.net.*;
import java.io.*;
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class SendMessage implements Runnable
{
   private String ip;
   private String duration;

   public SendMessage(String ip,String duration)
   {
      this.ip = ip;
      this.duration = duration;
      Thread t = new Thread(this);
      t.start();
   }

   public void printErrMsg(String s)
   {
      synchronized(this)
      {
         System.out.println(s);
         System.out.flush();
      }
   }

   private synchronized void printLog(String s)
   {
         FileWriter writer;
                  try {
                           Calendar cal = Calendar.getInstance();
                           SimpleDateFormat sdf = new
SimpleDateFormat("dd/MM/YYYY:HH:mm:ss");
                           writer = new FileWriter("log.txt",true);
                           writer.write(new String(sdf.format(cal.getTime())) + ":: ");
                           writer.write(s);
                           writer.write("\n");
                           writer.close();
                  }
```

```
                catch (IOException e)
                {
                        System.out.println("Cannot write to log file");
                }
    }

    public void run()
    {
       int numAttempts;
       for(numAttempts = 1;numAttempts<=5;numAttempts++)
       {
          try
          {
             Socket socket = new Socket(ip,5000);
             PrintWriter sendMsg = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));
             printErrMsg("ring signal is sent to bell at ip "+ip);

             sendMsg.println(duration);
             sendMsg.flush();
             socket.close();
             break;
          }
          catch(ConnectException e)
          {
             String printStr = e.toString();
             printErrMsg("Attempt Number "+numAttempts+ ". " +printStr.substring(27)+" by " +ip);
          }
          catch(NoRouteToHostException nhe)
          {
           String printStr = nhe.toString();
           printErrMsg("Attempt Number "+numAttempts+ ". " +printStr.substring(33)+ " on ip " +
ip);
          }
          catch(UnknownHostException u)
          {
           printErrMsg(u.toString()+ " on ip " + ip);
           printLog(u.toString()+ " on ip " + ip);
           break;
          }
          catch(IOException i)
          {
           printLog(i.toString());
             break;
```

```
            }
        }

        if(numAttempts == 6)
        {
            printErrMsg("Max number of attempts tried on "+ip+". Check connection to "+ip);
            printLog("Max number of attempts tried on "+ip+". Check connection to "+ip);
        }
    }
}
```

Final design of the device:

Arduino board with Ethernet shield can be mounted on the wooden platform and relay also can be fixed on the wooden platform. Then the full setup can be fixed inside a box as shown in the figure.

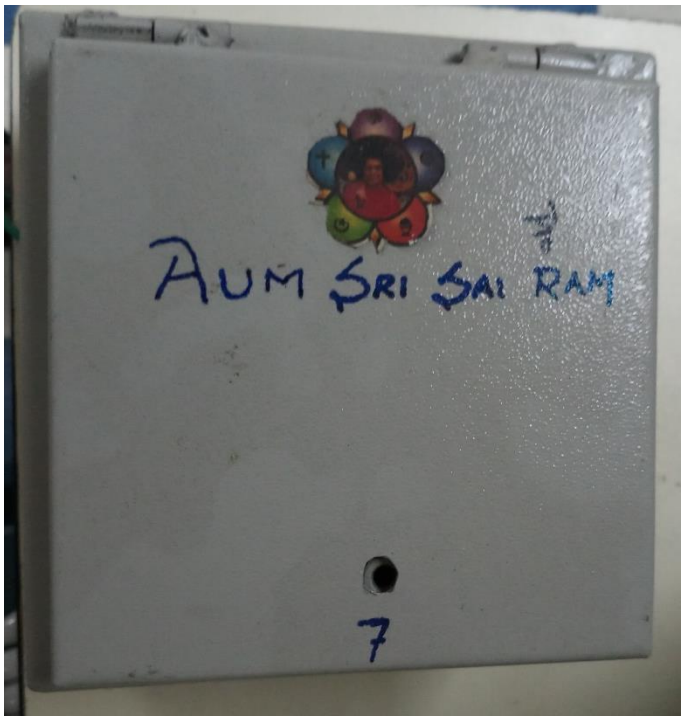

Figure 7.7: Top view of the device

Figure 7.6: Side view of the device

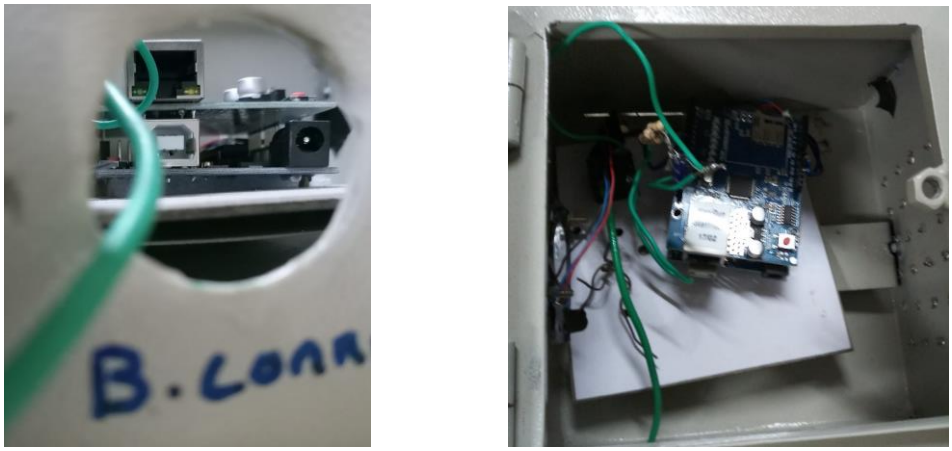

Figure 7.6: Side and inner view of the device

There can be entrance made for the cables for power and network connections on the box and the dc fan can be fixed to on the box as shown in the figure, to keep the Arduino cool.

Result and discussion:

The devices were installed at all the three bells. Performance of the devices were observed for 3 months. Minor changes in the circuit were made depending on the requirements, details of the changes are mentioned above.