Project Title:

SMS based control for electronic appliances using power line carrier communication

(EE 318: Electronics Design Lab 1)

Report

Group 4:

Saurabh Tembhurne (08d07024)
Mihir Patel (08d07025)
Saurabh Suryavanshi (08d07037)

INDEX

Page number

1. Proposal:

1.1 Aim:

Primary: To build a system that allows an individual to use his mobile phone as a remote control to control the electronic appliances.

Secondary: To expand its application to controlling all day-to-day appliances like water-taps, gas-stoves, etc. to realize smarter homes.

1.2 Motivation:

Imagine a hypothetical situation when you drive home after ending your days work. You are still 10 minutes away from your home. The service we wanted to build will enable you to switch on AC in your house. Living room is cooled before you enter home☺. This can be used for many devices in day-to-day life. Switching remotely water heater, AC, room heater, automatic answering machine, lights can be done very easily.

Our motivation is to building a cost-effective technology that can be used by a common man in his day-to-day life without making any major infrastructural. With the current volume of the mobile phone market and low-cost SMS services available, it is possible to easily adopt this technology.

1.3 Sub-parts of the project:

The entire project can be essentially divided into 3 parts:

1. Making a black-box that can be easily interfaced with the current designs of electric switch boards. This will essentially contain the wireless (say, bluetooth) receiver (if required transmitter too, in order to send the feedback of the status of the switches) that will control the switches. It will work on the nominal power fetched from the AC/DC converter in the black box***.[This communication via Bluetooth link was changed to that via PLC (power line carrier) as using Bluetooth would not include much of electronic design.(It is almost impossible to build a complete Bluetooth module by ourselves without referring to any available schematics! And using a readymade schematic is not designing.)]***
2. One base station for a home (essentially another mobile phone or an equivalent) that will act as a route between all the boards and the mobile phones of the family members. It will receive the SMS from the user mobile phone and convert it into a signal that will be transmitted (via Bluetooth ***[changed to PLC later]*** to all the electric boards.
3. Developing a mobile application that has a user friendly interface which enables an individual to use it essentially like a remote control with virtually infinite range.

1.4 Various steps to define success of the project:

(Originally)

1. We'll first try to use a computer as a base station and send a signal via its Bluetooth module to another Bluetooth receiver module that will control an led. (30%)
2. Then we will make the black-box which we had described earlier (10%)
3. Interfacing computer base-station and black-box (10%)
4. Reading a message received by a mobile phone real-time and generating a signal that will control the led via Bluetooth (30%)
5. Interfacing mobile-phone base-station and black-box (5%)
6. Final working model. (15%)

(Work done until midsem)

1. We were able to successfully write a script using python on computer that could retrieve the message received by a cell-phone (Micromax).
2. The script could decode the format in which the message was sent and relate it to the physical implication.
3. The script could automatically send this signal to control LED set via wire serial communication (BAFO).
4. We researched in detail on the internet various ICs used to make Bluetooth modules and the schematics used for the same. We finally boiled down to 3 options (1) CC2540 (free samples available with TI) (2) LMX 9820 (National Semiconductors, free samples not available, had some military clearances required) (3) uC with inbuilt rf transceiver (Atmega, no free sample available)

(Changes proposed post Midsem)

Replace Bluetooth link by wired communication to include more electronic design. Use pre-existing wires in the power mains thus making it a power line communication. After receiving the instruction through SMS, the power line will be used to establish communication in home.

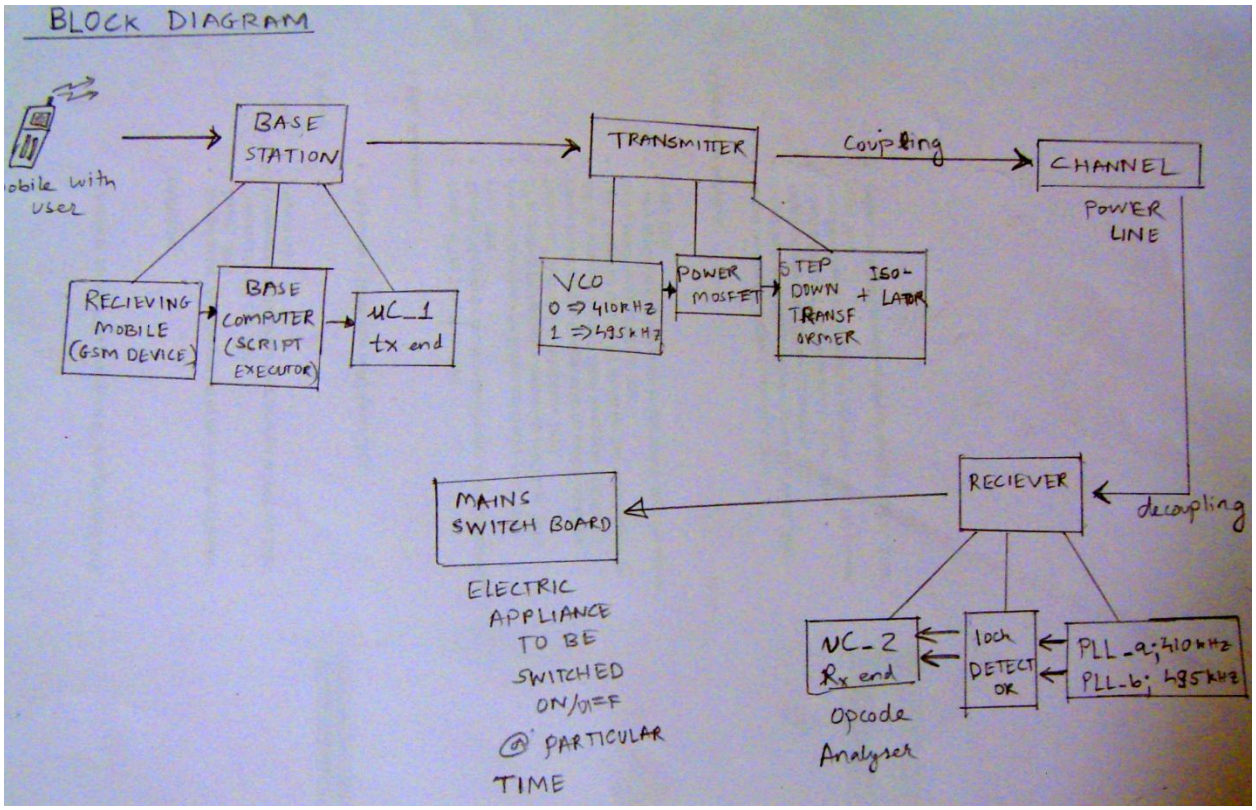
2. Introduction:

.

2.1 About the project from system perspective:

To switch a device you need to send a message from your phone to a predefined number. The SMS will be read by laptop connected to GSM module (either via Bluetooth or via serial port), in our case a Micromax mobile phone. The laptop will have a script running which will poll for the data being received by the phone. Typically, the data sent will have the name of device, what state should it be switched (on/off), what is the time after which it needs to be switched. The laptop will send this data in suitable format (opcode) to transmitter circuit. Using a FSK modulator the signal is coupled on to mains. At the receiver, the FSK modulated signal is decoupled from mains then demodulated by using PLL then the opcode is deciphered and action is taken accordingly.

3. Subsystems:



3.1 Central Server:

Central Server has the following blocks-

1.Mobile phone

2.Laptop supporting python script

3. Link between laptop and microcontroller.

3.1.1. Mobile phone:

We require a GSM module to access the SIM and keep it working. Instead of GSM module we are using Micromax X505. The phone can be used for other purpose. So practically, the additional cost of GSM module is zero.

Not all manufacturers allow the access to SIM. Micromax is one of the manufacturer that donot have security restrictions to access SIM of mobile phone.

We require Bluetooth for our project. Micromax provides cheapest phone that has Bluetooth in it. Micromax X505 smart phone with resistive touch screen. In market, it costs around 3-3.5k.

3.1.2. Laptop

We require a central unit that would keep track of message being sent. We are making use of python language to perform this operation. We require a system that could run the script generated in python. Since we had access to DELL, studio1555 laptop we used is it for this project.
On researching about the options, we later realized that the use of laptop is redundant in our project. Nokia phone which use Symbian OS can run python script .However it requires further work on the way SIM can be accessed in Nokia phones.
We have provided the necessary information about Python and AT Commands (Used to communicate with SIM) later in the report.

3.1.3. Communication with Microcontroller:

The data sent by the laptop needs some interference with the transmitter circuit. We introduce a microcontroller in between the laptop and the transmitter circuit. The communication between microcontroller and laptop was achieved using a BAFO. It is serial communication, which uses USART protocol.
We discuss the communication between laptop and microcontroller in transmitter part.

3.2 Transmitter :

The transmitter subsystem we have following blocks-

1. Microcontroller.
2. FSK modulator(VCO)
3. Power Mosfet and band pass filter
4. Coupling onto mains

3.2.1. Microcontroller:

We are using Atmega 16 microcontroller that will be interfaced with laptop by USART protocol. This would modulate the FSK signal according the instructions received from the laptop

3.2.2 FSK modulator:

We are using VCO to modulate the signal. Frequency shift keying is used for modulation. VCO has frequency dependence on resistance and capacitance connected at its input. We keep the capacitance same and switch the resistance between two values thus achieving two different frequencies

3.2.3 Power Mosfet and band pass filter:

The signal output of VCO is provided to a gate signal to power Mosfet. This would generate signal with higher power than the gate signal. We want the square wave to get converted into sine wave.
We initially tried using butter worth and sallen key topology to achieve a band pass filter to couple only the fundamental frequency into the mains.
But at such higher frequencies we were no able to filter the signal. The main problem with active ifilters

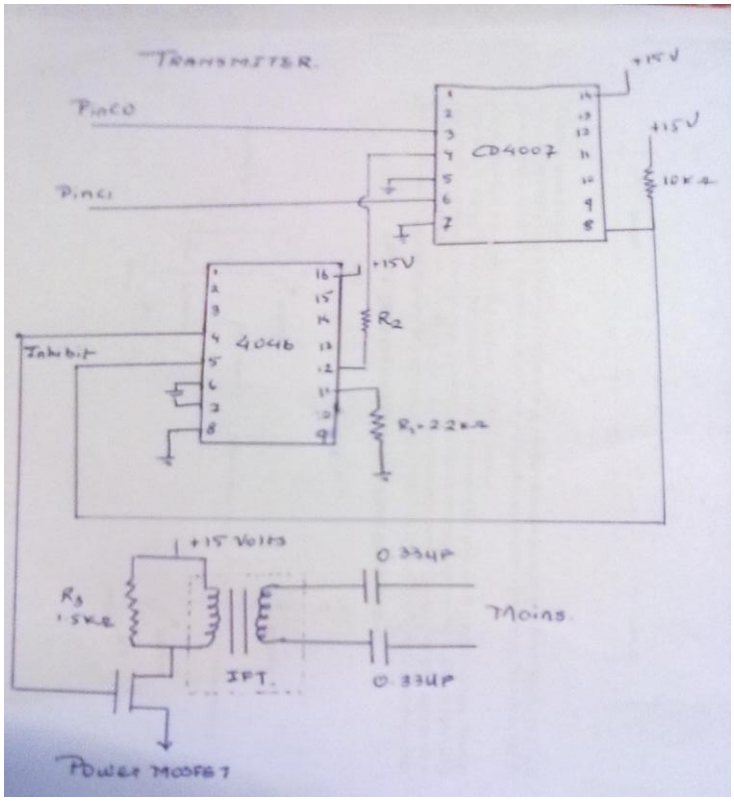
is that it cannot increase the power of the signal. Power output is very less. Therefore, we used passive filter for this purpose.

3.2.4. Coupling:

We used intermediate transformer to couple the signal onto the mains. IFT's were used in radios. They hare tune to a specific frequency. It further helps to build band pass filter using IFTs. The secondary was coupled using a pair of capacitances in series.

- Design:

Figure**:** Schematic of transmitter



1 Choice of microprocessor :

The requirement for microcontroller point of view are not very rigorous. The memory required by the code is in few kilobytes. The data rate expected is 100bps. We want 1 instruction to be per second to be performed in the system 100bps is just to be on safer side. So ideally, we do not have any specific constraints on frequency of microcontroller. Any microcontroller with USART communication would work for our purpose. We have used Atmega16 in this case.

2 Communication with laptop:

The Microcontroller is required to keep the polling the laptop for new information. As soon as any new message is received, the laptop will send the message to the microcontroller. In order to make overall

communication less erroneous it is important to reduce the error in this initial data transfer. We are using Universal Synchronous Asynchronous Receiver Transmitter (USART) in Atmega to establish a serial communication with laptop. We have used to BAFO for this purpose.

3 Transmission protocol:

The data received by the microcontroller will be in opcode. Each instruction will be transmitted in 4 bytes. The first byte is the address if the device. We have 8 devices to be controlled. The address is expressed in three bits. 000 to 111.

The next byte is the function to be performed. '1' will determine that device given by the address is to be made ON. '0' refers OFF.

The next two bytes represents the time interval we want in between receipt of the message and execution. We have limited our design to have maximum time delay of 15 minutes. This is implemented by using 4 bits.

The instruction will be 8-bit signal. This is appended with 4-start bits. The 4-start bits used are 1010. Since the signal received is of constant length (12-bits) , we have not used stop bits.

4 Signal generation:

We are using FSK modulation. 1 represents high frequency (495 kHz) and 0 represents low frequency (410 kHz). The FSK signal was generated by using VCO of 4046pll.

The lower frequency was generated fixing the value of R1 at pin11. The higher frequency was achieved by manipulating the R2.

R1= 2.2 Ohms(had to tune it to get exact frequency)

C1= 100nF

For lower frequency the R2 was open(infinite resistance). For higher frequency, the value of R2 was finite. We achieved this using a Mosfet switching.

When gate voltage is low, the Mosfet is open and R2 is infinite. With introduction of Mosfet, we introduce some series resistance in R2. For our purpose, the value of R2 observed was practically zero. We achieved higher frequency just by using Mosfet on resistance.

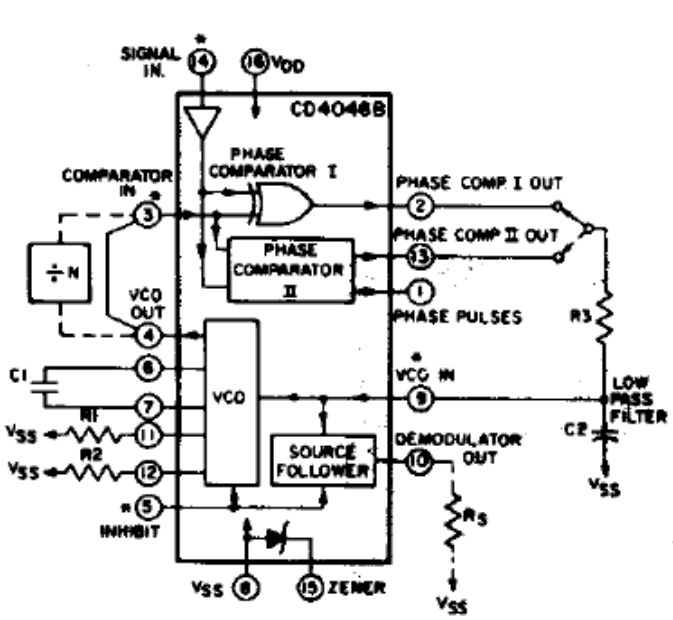
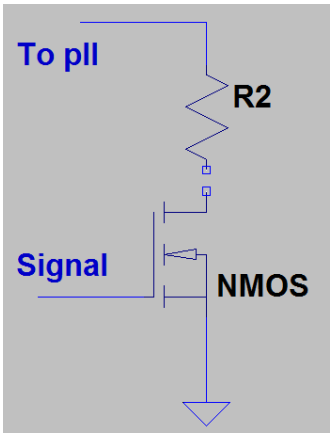


Figure: PLL 4046 pin diagram.

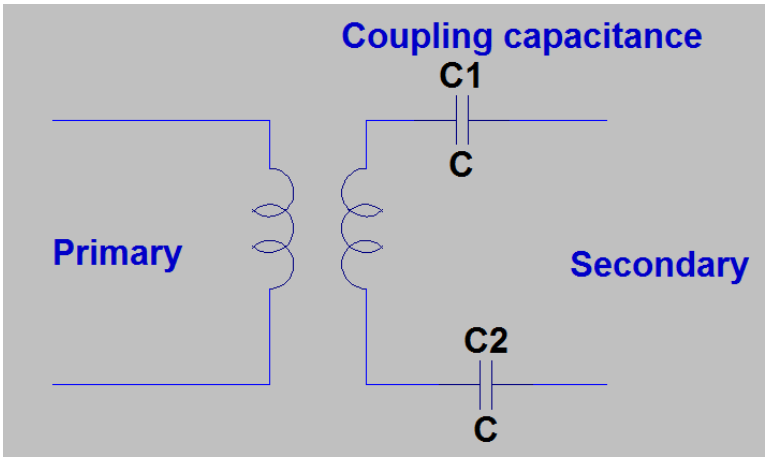


5 Coupling the signal onto mains:

We used transformer to couple onto the mains. The isolation from mains was achieved by capacitors in series with the primary and secondary. In case of inductive coupling, it is necessary to fine tune the inductance. We used Intermediate frequency transformer, which are pre-tuned at 455 kHz.

With capacitors in series, we reduced the Mains effect by almost 300 times. The signal seen on primary side was smaller than 100mVolts.Since inductor was not known; we determined the capacitor value bye trial method. Used: C1=C2=0.33 micro farad, 230 Volts.

Figure: Schematic of Transformer and isolation from Mains.



- Why choose FSK?

For communication via the power line we had three options to choose from:

a. Amplitude Modulation
b. Phase Modulation
c. Frequency Modulation

Out of the available options we discarded amplitude modulation because of following reason:

> The background voltage of the signals on the power line is of amplitude around 311V peak sinusoid. It also contains some harmonics of its natural frequency 50 Hz but of lower amplitude. The signal we are transmitting has voltage amplitude budget of around 2V peak sinusoid freq (470Hz)(We needed to step-down the signal before transmitting inorder to increase the current). When this signal reaches at the receiver end its amplitude has attenuated a lot and even after stepping it up we get around 100 mV peak sinusoid (using IFTs). With such a low amplitude it becomes really challenging to process the received signal and we need to put in various amplifiers before we can operate on the received signal. This method is quite challenging as compared to other available modulation options (explained later). So, it was a natural choice not to choose amplitude modulation.

We discarded Phase modulation because:

> It depends largely on how accurately we can model the channel properties. Over here it is power line on which various loads are present in parallel. Modelling it might in itself be a good topic of research. Too much of data mining and understanding of the phase modulation done over power line had to be done in this method.

The last option of Frequency modulation was the one we opted for because:

- Frequency modulation gives us a room to process a frequency that is independent of the background frequency and its harmonics. (i.e. we can choose frequencies that are not on the harmonics of the principal freq 50Hz.)

- We saw the FFT of the ambient noise in the mains and found that the voltage amplitude is negligible beyond 10kHz frequency. Also if we keep the operating frequency large we can design better filters that fall within desired cutoff ranges. So this enabled us to select our frequencies in the range of hundreds of Khz (***we chose 410 kHz for '0' and 495 kHz for '1'***)
- We were able to avail IFTs(intermediate frequency transformers) in the range which is in the neighbourhood of 475Khz. So, stepping down the test signal at transmitter and stepping up the test signal at receiver was feasible.

3.3 Receiver:

Our receiver end has following blocks:

1. Step-up block
2. PLL block
3. Digital lock detection

3.3.1. Step-up block

After having received the attenuated signal we first need to step-up the voltage in order to get voltage amplitude that is sufficient for pll to work. We used 1:7 IFT with center freq 455kHz. It also works as a filter that attenuates the signals, which do not fall in the range of about 50 Khz of the centre frequency. This would reduce the current but will increase the voltage signal. For Pll we require voltage signal.

3.3.2. PLL (Phase locked loop) block:

We used CD4046 IC as PLL. There are two different PLLs getting same input from the IFT. (Vdd = 15V, phase-2 comparator)
PLL_a is tuned to fc = 410 kHz, f_{min} = 390kHz, f_{max} = 430kHz.
PLL_b is tuned to fc = 495 kHz, f_{min} = 480kHz, f_{max} = 510kHz.
The output of the low pass filter V9(output measured at pin-9) varies from 3V to 12 V as we vary the signal in freq from fmin to fmax. In the range beyond the lock range, below fmin the voltage is 0.6V and above fmax it is 14.2V. This motivates us to use the output of the filter to determine the locked/unlocked state.

3.3.3. Digital lock detection block:

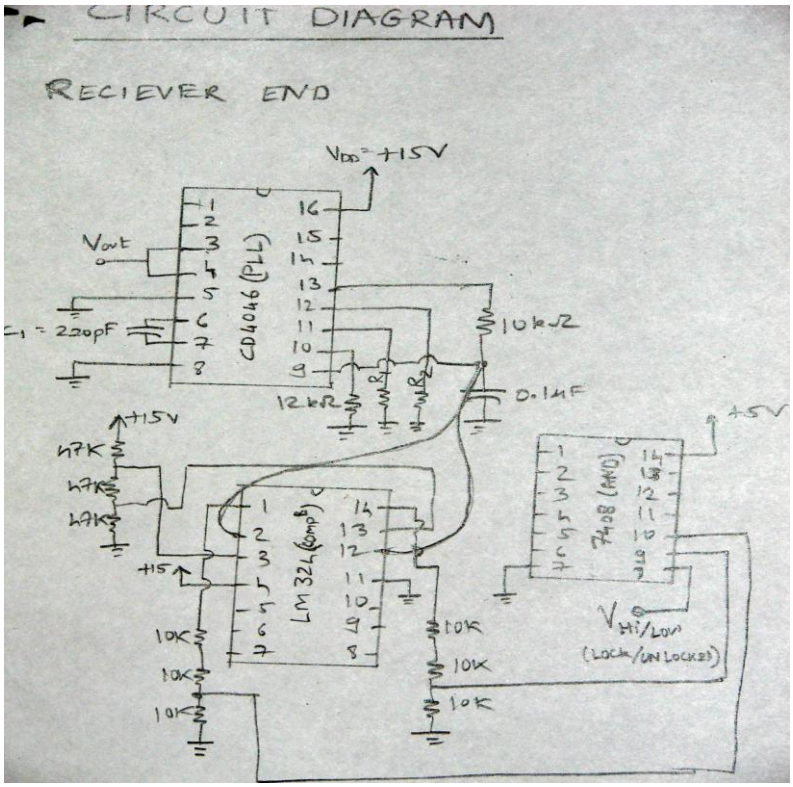
One block consists of 2 comparators (LM324). (One block each for each PLL implying 2 digital blocks)
Comparator_a: (-ve input = V9, +ve input = 10V)
Comparator_b: (+ve input = V9, -ve input = 5V)
The output from each of the comparators is divided by 3(to get it within the range 0-5V) (using 3, 10K resistors) and is supplied to AND gate (7408) inputs. When V9 is between 5V and 10V we get AND output as high indicating that the PLL is locked. Low implies the unlocked state.

The overall picture is that when
0 is sent AND_a shows HIGH, AND_b shows low.

1 is sent AND_b shows HIGH, AND_a shows low.
When nothing is sent both show low.

Design:

Figure: Schematic of receiver



1.Choosing between phase 1 and phase 2 comparators of PLL

Phase 1 has only one advantage over phase 2 comparator. It can high noise rejection capability. But it has got many drawbacks when we consider this particular application. Phase comp 1 gives output with a phase shift from 0 to 180 degrees while phase 2 comparator has output that has 0 phase difference as compared to input. Phase 2 comp has lock range equal to capture range thus demarcating only one set of boundary that is easier to be dealt with.

2. Deciding the amplitude of signal to be locked.

When we apply sinusoid as input through a small coupling capacitor we need to supply atleast 1.4 V signal-in so as to be able to lock PLL properly. In square wave we need a bit higher amplitude (so as to get the amplitude of principal sinusoid to be equal to 1.4 V)

3.Determine the values of C1, R1 and R2

C1: In order to determine C1 we select a centre frequency fc and supply voltage Vdd . Then we look at the graph of freq versus C1. For frequency range about 500 kHz we selected C1 = 220pF.

R1: For selecting R1 we first R2 = ∞ use the phase-1 comparator (at pin 2) instead of phase-2 (pin 13) for connecting R_{filter} . Now in the unlocked state the freq of the square wave (@pin 4) is the centre frequency. Change the value of R1 to set the desired centre frequency.
R2: For determining R2 we connect the R_{filter} to the phase-2 comparator (pin 13) and keep the freq of the signal in far away from the lock range. The frequency of the square wave is f_{min}. Adjust R2 to get the desired f_{min} , centre freq f_c being same as set earlier.

Over all lock range is $\mathbf{f_c + (f_c - f_{min})}$ and $\mathbf{f_c - (f_c - f_{min}).}$

4.Determine filter component values

The filter is supposed to work as a low pass filter with RC time constant sufficient enough to allow the capacitor to retain the voltage that acts as input to the VCO. We first choose C_{filter} to be very low 0.22nF but the resulting time constant was too low to allow VCO to get a stable enough input votage. Then we changed C_{filter} to 0.1uF and R_{filter} = 10k which worked well

5.Form of input to PLL

It is advisable to use sine wave as input and not square wave because if square wave is of high enough amplitude we get its component sine waves of sufficiently high amplitude to lock the PLL at undesired harmonis.

6.Choosing the Vdd for PLL

Vdd value to be used it determined by the error margin you want to tolerate in your design. Choosing Vdd = 5 V implies more tolerance (higher % error) and Vdd = 15 V implies lower tolerance (lower % error). We choose Vdd = 15 V in our design.

7.Detect the lock and unlocked state conveniently and accurately

At the beginning of locking near f_{min} we see that the output of C_{filter} is around $V_{th\ min}$ = 2.4 V (depends on the Value of R_{filter}) and at the end of locking range near f_{max} we get the output of filter (@ Pin 9) about $V_{th\ max}$ = 12.6 V. This implies that we want some circuit to indicate the lock state between $V_{th\ min}$ and $V_{th\ max}$. So we employ use of 2 comparators (LM324) with references 5 V and 10 V whose outputs are ANDed (7408) to get the desired lock state indicating signal.

8.Dealing with different Vdd on the same board in case PLL is not operated at 5 V .

Care must be taken to divide the voltage of comparator output by 3 (Max comparator output = 14.2 V; max Vin to AND gate = 5 V). Use three 10k resistors of this. DON'T USE POTS. [We tried using two 100k pots to divide the voltage by 3. When we didn't connect POT to AND gate we got the correct voltage value at the centre pin of the pot. But as soon as we connected it to the AND input it got loaded and the state that was initially 0V was measured to be 1.6 V on being connected to AND input. We used 3 equal resistors to solve this problem with ease

4. Power Line Communication:

4.1 Introduction:

Power Line Communication (PLC) uses the existing power cable infrastructure for high-speed data communication. Even with several unresolved technical challenges, it is becoming one of the strong competitors in the broadband communication market for in-building networking as well as for the last mile access. PLC enjoys the advantages of wide availability and low infrastructure cost, assuming that the power cables are already installed for energy distribution purposes. However PLC performance is limited by attenuation and background noise present in the medium. The frequency shift keying (FSK) modulation scheme is experimentally found to be more robust against such power line distortions. [1] Further it was shown earlier that the background noise in PLC follows Nakagami's *m* distribution. Thus error performance, especially the evaluation of bit error rate (BER) of FSK when corrupted with background Nakagami noise, is of considerable interest.

4.2 Types of noise in power-line[2]

1. Coloured background noise
2. Narrow-band noise
3. Periodic impulse noise asynchronous with main frequency
4. Periodic impulse noise synchronous with main frequency
5. Asynchronous impulse noise

Type 1, 2 and 3 are background noise, which last for minutes or hours.This noise is due to channel environment
Type 4 and 5 are time variant noise whose effect can be perceived over millisecond or microsecond. This noise is due to switching on/off of electronic devices.

4.3 Factors on which attenuation depends

a. Time Dependence: The attenuation seen is depends on the load present on the line. It is high during day and low during night..
b. Frequency Dependence: Very much attenuation at higher frequencies >10MHz.
c. Distance dependence: Signal attenuates with large distance. To cove larger distance the signal should carry large power.
d. Coupling mechanism: Capacitive coupling. Signal is coupled through capacitor to the mains.
e. Inductive coupling: An inductor is used to couple the signal into the network's current waveform. It is loss. Tunning of inductor is very difficult.

References:
[1]EvricleaVoudouriManiati and DemetriosSkipitaris ,'Robust Detection in Power Line Communication for OFDM and non-Rayleigh Fading Channels'
[2]Khurram Hussain Zuberi, Power line Carrie(PLC) communication system- Master Thesi

5. Python and AT Commands

5.1 Python – Discussion and Relevance to project.

The language we used for accessing the Ports on a computer and thereby receiving data from the mobile and sending data to the microcontroller; was Python. This language was chosen due its flexibility and easiness. Now we will go through the algorithm implemented on python.

Our primary aim, for accessing the SMS stored in a mobile phone , was to get access to the port on which the cell phone and microcontroller will be connected to the computer. Now as proposed, Cell phone at home is connected to the computer via Bluetooth link .The only reason why we chose Bluetooth link over other wired links was due to some flexibility in handling the cell phone as with Bluetooth the cell phone can be moved in and around home in the vicinity of about 10 meters (can be increased) from the base computer.

Bluetooth devices have a unique identification Id. It is easy to get the ID of a Bluetooth device by using device manager in windows. We can use this device id to establish communication (-using RFCOMM) between Bluetooth device. Since we needed a serial profile to transfer data between computer and mobile phone we used SPP (serial port profile) provided in Bluetooth stack. The computer identifies any device connected to it by COM port. The COM port is assigned to a device when it is connected to the computer. By looking into the properties of connected device we can find out to which COM port the device is connected. This COM port is used to address a device (in this case mobile phone) in program.

Another link which has to be made on the computer is between the computer and the microcontroller and for that we used serial communication (USART).
Lets get familiarize with python syntax and algorithm we will be using in our code.

To implement the two links, i.e. – Bluetooth link and Serial link , on python we got to include the modules which provide access to all the commands for accessing these links.
This is done in python by writing

```
Import serial                    // From pyserial module
Import Bluetooth                 //From pybluez module
```

One important thing that was noticed during the course of experimentation with python was that we actually don't need to use "Bluetooth" module in our code .What happens is that if something is sent to the port via python then since that port is connected via some link to another devices, the data gets automatically passed through the link to the other device.

Therefore, even for achieving communication on the Bluetooth link, we can just send/receive the data from the port .We will use mobile phone as a serial port once we define the connection.

For creatink a link in python following command is used.

serial_Port = serial.Serial("COM32",9600) /
-Here 9600 is the baud rate and the COM32 is the port at which the link has to be created.
- serial. Serial( ) is the syntax for creating a serial link.
- Serial_ Port is the name given by us to access that link.

The commands for writing and reading from the port are:
serial_Port.write('0') // This command is used for sending the character '0' to the port COM32

serial_Port.read(1) //This command is used for reading 1character of data from the port COM32 , here the argument (1) can be changed to any integer number of character which are to be read from the port.

The command for closing the link or port is
serial_Port.close( ) // here serial_Port is the name of the link which we created. ".close ( )" is the syntax to close any link.

5.2 AT Commands-brief discussion

The AT commands are a way to access the SIM via GSM module . Essentially when we type or do any activity on the cell phone, the activity is transformed to corresponding AT commands , and are sent to the GSM module and accordingly the response is obtained.

Now for accessing the SIM , instead of going for a GSM module (which would require buying a GSM module as it was unavailable in our labs) , we chose to go for using the GSM module of a cell phone. And this provides us with additional flexibility in terms no wasting any extra money as well as easy accessibility and mobility.

But the problem we faced during the course of experimentation was that the available cellphone companies had security control over their GSM module. Therefore after trying with many phones we choose MIcromax as its GSM module could be easily accessed.
But this does not imply that user has to buy a Micromax phone to avail this service of SMS based control of switches, AT commands are accessible at any cell phone but to make it work at all the cellphone , it requires intensive involvement into the GSM module structure and there programming which are beyond the scope of this project.

Let us look at some of the AT commands we will be using in our program.

AT command syntax

AT command start with header AT, and end with <CR>. If the format of AT command input was correct the terminal will return the corresponding request information and finally will return "OK"; Otherwise , ,the terminal will return "ERROR".

The command to tell the system to use particular format of the message is
At + CMGF =<mode>
The SMS can be read in two modes
Mode = 1 : Text mode
Mode = 0 : PDU mode
Out of these two, the text mode is of our use as it read the message as typed by the user.

Therefore, in our code we first check the mode in which the SIM gives the message output by writing

Command: at+cmgf ? Response: +CMGF<mode>
OK

Then we set the mode to text mode by writing
AT+CMGF=1

The command to list all the messages from the SIM is

At + cmgl=’’all” // which lists all the messages in the SIM memory in the sequence of the memory address.

Important thing to note here is that whenever a message is sent to the cellphone by default it will go to the memory location one of the SIM ,only if its empty. Otherwise it will go to some random location which is available.
Normally ,a SIM has the ability to store about 1-10 messages in its memory there to access the message which is currently being sent we keep the first memory location of the SIM empty , by clearing the SIM message memory each time the message is processed.

To delete an SMS message

AT+CMGD=1

We have used the command AT+CMGD=1,3 where “1” is the integer type index of the message in the SIM storage and “3” is for Deleting all read messages from the preferred message storage.

We initially used HyperTerminal to communicate with sim. Once we achieved sufficient familiarity with the process we developed a script using Python.

Following few lines explain how the command is sent using python.

```
def sendatcmd():                ///define sendatcmd()

        serial_Port.write('at+cmgf=1'+'\r')              #set the mode to receive the sms.
        Serial_Port.write('at+cmgl="all"'+'\r')          #list all the messages

    sendatcmd()          // command to send the AT command
```

Entire code with explanation has been included in appendix

6. Experiments and observations:

1. Our experiment consisted of: Writing a script that is able to automate complete process.: We experimented doing this using different languages available. Some of the options we considered were:

a. Hyperterminal: Test code from HyperTerminal. For HyperTerminal we require to send one instruction at a time. The user has to wait until the mobile replies.

"atz
OK
at+cmgr?
ERROR
at+cmgs?
ERROR
at+cnmi
ERROR
at
OK
at+cmgf?
+CMGF: 0

OK
at+cmgf=1
OK
at+cmgr=1
+CMGR: "REC READ","w4w4w25353030303",,"2010/11/28 15:42:16+22"
Tees Maar Khan, Sheila Sheila, Wallah Re Wallah, Shakira(No problem),Hum adhoore
eik ke baad eik naye NON STOP gaane Dial 55000(tollfree) sirf Rs.30/30days
(message read in text format)
OK
at+cmgf=0
OK
at+cmgr=1
+CMGR: 1,, 156
0791198996909949040ED0CD66AB5683C160000001118251246122 9BD472790E6A86C372D012
1D76
B340537439CD0E83A6E8729A1D6681AE61363B8C0649CBA06B98CD0EA359A0293ABC4ECBC32
8E71B
0497BFC5EC723BC542D6DBA03019FD7ECBCBA0727A0D5A9741E270980C2AA7D72077385F063
99F4E
D094FA8482CEE1B0BB0C22A6C36C50AD0683C150F4379B6D9697CB29D03C2D3783A473D70CF
69AC1
C8E1FC1C"
(Same message read in PDU format)

b. Pearl language: GSM module: We could not use it to generate any useful output.

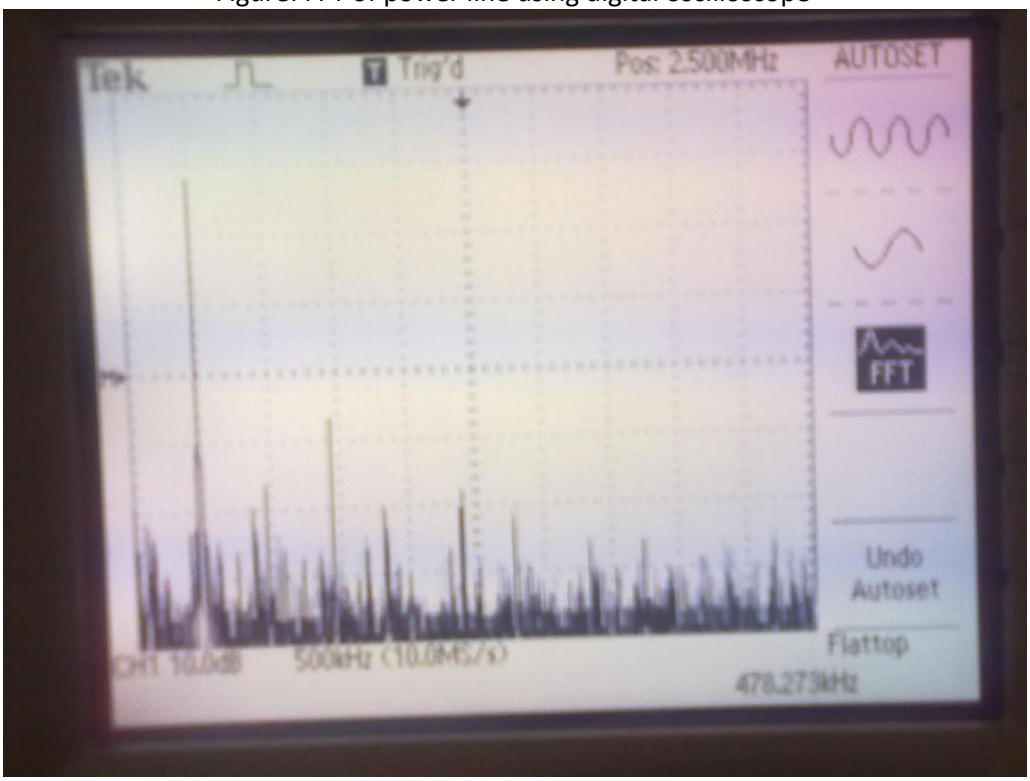
c. Python: Discussed in previous part

2. Designing a robust receiver: This consisted of making a PLL circuit that is locked in the desired frequency range and converting the output in a digital form that can be used by uC.
Experiments where based on specification we have fixed in our system design. We fine-tuned our specifications during these experiments

3. Designing a transmitter that is able so to couple power sufficient to meet the requirement of both receiver and our communication protocol. We experimented with astable multivibrator for purpose of generating the frequencies required. We could achieve stable frequencies at theses specification. In addition, it was difficult in switching between the frequencies. The output of multivirator had duty cycle different from 50 percentages. It created problem at it included effects of even harmonics. Therefore, we rejected multivibrator and used VCO instead.

4. Characterizing the power line, determining attenuation coefficient and defining coupling coefficients: It was observed that odd harmonics of 50 Hz where more dominant in the power line. The observed FFT changed with time and space. Change in higher order frequency was less predominant. Except few cases the contribution of frequencies: 410 kHz and 495 kHz was negligible. The noise however introduced some fluctuation in high frequency range.

Figure: FFT of power line using digital oscilloscope



5. Experimenting on power line if rest of the blocks work perfectly fine under the normal conditions: Due to lack of time and insufficient testing of power line we could not experiment this part of the project. However, we successfully experimented the system using a pair of single strand wires.

7. Future Prospects:

We can extend this idea to designating specific addresses to various electronic devices in home, buildings, etc. We can specify the exact time after which we want our device to be switched On/Off. We can get the feedback from each appliance about its state. All we need to do for this is to place both transmitter and receiver at both the ends and write a code in uC for that. We can then set up a schedule to control entire home (with option of manual operation always available). With feedback system available we can actually set up a system by which the usage report is sent to the electricity board of a particular area on periodic basic to check the consistency with electricity bill (and eliminating the need of manually going to each home for measuring the usage altogether).

8. Conclusion:

We were able to demonstrate all the blocks working well individually and after integration except the transmitter block. We could not characterize the channel to the extent that was required to be able to determine the exact power (Voltage and Current) we need to couple into mains. But if some more time is spent over the characterization of the channel (for example: *determining coupling efficiency of the transmitter and receiver, determining the effect of loading channel with common loads in parallel, for the same amount of power being transmitted what would be the better choice using high voltage or high current, how would the distance between transmitter and receiver change the amplitude to the received signal, and so on*) we can make the PLC communication possible using just the basic ICs and without using any readymade modules available off-shelf.

9. APPENDIX:

Python code:

```
################################################################################
import serial
import bluetooth


while True:


   flag=0
   serial_Port = serial.Serial("COM32",9600)
   def sendatcmd():
      print '#######################start#########'
      serial_Port.write('at+cmgf=1'+'\r')
      #print '2'
      serial_Port.write('at+cmgl="all"'+'\r')
      #print '3'
   sendatcmd()
   buf=''
   while True:


         q=serial_Port.read(34)
         print 'read'
         m=q[32]
         n=q[33]
         buf=buf+q
```

```
        print q
        break
    if m=='O'and n=='K':
       flag=1
    serial_Port.close()
    if flag==1:
       print "nothing"
    else:
       print 'in_else'
       serialPort = serial.Serial("COM32",9600)
       def sendatcmd():


          serialPort.write('at+cmgf=1'+'\r')     #set the mode to receive the sms.
          serialPort.write('at+cmgl="all"'+'\r') #list all the messages



       sendatcmd()
       buf=''
       while True:
          #print 'loop'
          r=serialPort.read(100) #defives the data limit for input messages.
          print 'received msg'
          m=r[96]# this stores the message which was received by the phone.
          n=r[97]
          o=r[98] #here 'm' is an array that is storing an entire fucntional code which is supposed to be of
                  the form "xxxx"
```

```
    p=r[99]
    buf=buf+r
    print r
    break

serialPort.close()

if m=='a': # this thing choses the first device which is defined by the 'a'
    print 'm ='' was detected'


    serialPort_atmega = serial.Serial("COM31",9600)
    def sendatcmd():

        serialPort_atmega.write('0')
        a=serialPort_atmega.read(1)
        serialPort_atmega.write(n)
        a=serialPort_atmega.read(1)
        serialPort_atmega.write(o)
        a=serialPort_atmega.read(1)
        serialPort_atmega.write(p)
        a=serialPort_atmega.read(1)


    sendatcmd()
```

```
    serialPort_atmega.close()


if m=='b': # this thing choses the first device which is defined by the 'a'


    serialPort_atmega = serial.Serial("COM31",9600)
    def sendatcmd():


        serialPort_atmega.write('1')
        a=serialPort_atmega.read(1)
        serialPort_atmega.write(n)
        a=serialPort_atmega.read(1)
        serialPort_atmega.write(o)
        a=serialPort_atmega.read(1)
        erialPort_atmega.write(p)
        a=serialPort_atmega.read(1)




    sendatcmd()
    serialPort_atmega.close()


print "end of the first"
########################################3
 serialPort = serial.Serial("COM32",9600)
def sendatcmd():
```

```
        serialPort.write('at+cmgd=1,3'+'\r')
    sendatcmd()
    buf=''
    while True:
        r=serialPort.read(2)
        m=r[1]
        buf=buf+r
        print r
        break
    buf=''
    while True:
        r=serialPort.read(2)
        m=r[1]
        buf=buf+r
        break
    buf=''
    while True:
        r=serialPort.read(2)
        m=r[1]
        buf=buf+r
        print r
        break
    serialPort.close()
    print 'port closed'
```

##3###############

Explaination:

- First we create a communication link for accessing the port on which the mobile is connected via bluetooth.

- After successful creation of the link , we first set the SMS message mode to text using the at+cmgf=1 command.

-Then we read all the messages in the sim memory using the at+cmgl=”all” command.

-Next we check for the “OK” response of the at+cmgl command by reading from the port and comparing the read data to the string “O” and “K”.

-Next we keep on checking for the OK response until it is obtained by putting the whole code in “while True:” loop . Basically here we will be moving forward only when we get the OK response so we poll for OK continuously.

If “OK” response is obtained then we set a flag to high and to further processing. This technique is used so that no unwanted message affects the program.

-When flag is high , we again read the SMS message from the SIM , this time we know the message we reading the contains the information we have to process as the flag was high.

Therefore we read in the message and search for text message which was sent on the cellphone.For this we read in the entire data in an array and then search for character numbers where our text SMS will be appearing.Through experimentation it was found that we read message through at+cmgl command the response is as shown below

+CMGL:1, “ REC READ”, “+9198xxxxxxxx”, “11/04/11 , 20:40:31+00” hi how u doing

+CMGL:2, “ REC UNREAD”, “+9198xxxxxxxx”, “11/04/11 , 20:40:31+00” hi there

Basically what we are trying to emphasis on here is that the actually message that we want to read is not at the starting , therefore after some experimentation we found out that in our case the message we were searching for starts at character number 96.

Then accordingly we read the four bits of our opcode in different variables.

After reading the SMS we got to send we read to the microcontroller.There fore we create another link for communication with the microcontroller port COM31(in our case microcontroller was connected at COM31 of the computer)

-Then we will send the read data by using “write” to port command (described earlier).All the four letter of the opcode are send sequentially.

Important thing to note here is that if we write the data one after the other , then the data will be written to the microcontroller at very high speed and the synchronization between sending the data from python and receiving the data on the microcontroller may be lost .Therefore to get reed off this problem, we programmed in such a way that after writing each character python awaits for a response from the microcontroller and the python code proceeds to the next write instruction only if it has got some

feedback from the microcontroller. In this way , the process of the sending from python and receiving by microcontroller are compelely in synch.

-After sending the data to the microcontroller , we have to make the programme ready to receive the next message if any.Therefore we clear the SIM message memory after a message has been processed so the next message can come in the location **"1"** of the SIM storage.

Atmega 16 Transmitter code:

Code used: (Put in appendix)

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <inttypes.h>
#include <util/delay.h>

void USARTInit(uint16_t ubrr_value)
 //Set Baud rate
  //UBRR|= ubrr_value;
  UBRRL = ubrr_value;
  UBRRH = (ubrr_value>>8);
  /*Set Frame Format
   >> Asynchronous mode
  >> No Parity
  >> 1 StopBit
  >> char size  8 bits*/

  UCSRC=(1<<URSEL)|(3<<UCSZ0);
  //Enable The receiver and transmitter
  UCSRB=(1<<RXEN)|(1<<TXEN);}



uint8_t USARTReadInt()
{  //Wait untill a data is available
 while(!(UCSRA & (1<<RXC)))
  {          //Do nothing }
//Now USART has got data from host
```

```
  //and is available is buffer
  return UDR;}

//This fuction writes the given "data" to
//the USART which then transmit it via TX line


void USARTWriteChar(char data)
{  //Wait untill the transmitter is ready
  while(!(UCSRA & (1<<UDRE)))
  {
 //Do nothing  }
 //Now write the data to USART buffer
  UDR=data;}

int main(){
USARTInit(103);
DDRC=0xFF;
PORTC=0x00;
volatile char data1,data2,data3,data4;
//data1='d';
int output[8];
//USARTWriteChar(data+1);
//data=USARTReadInt();
//PORTC=0Xff;
while(1){
//PORTC=0xaa;
data1=USARTReadInt();
USARTWriteChar(data1)

;
// _delay_ms(105);

data2=USARTReadInt();
USARTWriteChar(data2);

// _delay_ms(105);

data3=USARTReadInt();
USARTWriteChar(data3);
// _delay_ms(105);

data4=USARTReadInt();
USARTWriteChar(data4);
```

```
//          _delay_ms(105);
PORTC=0x0f;
int i=0;

//convert the received data to 8 bit for.

if (data1=='0')
{  output[0]=0;
   output[1]=0;
   output[2]=0;   }

if (data1=='1')
{  output[0]=0;
   output[1]=0;
   output[2]=1;   }

if (data1=='2')

{  output[0]=0;
output[1]=1;
 output[2]=0;

   }

if (data1=='3')
{  output[0]=0;
   output[1]=1;
   output[2]=1;
   }

if (data1=='4')

{  output[0]=1;
 output[1]=0;
output[2]=0;
 }

if (data1=='5')

{ output[0]=1;
output[1]=0;
 output[2]=1; }

if (data1=='6')
```

```
{ output[0]=1;
  output[1]=1;
  output[2]=0; }

if (data1=='7')
{  output[0]=1;
 output[1]=1;
 output[2]=1; }

if (data2=='0')
{  output[3]=0;}

if (data2=='1')
{  output[3]=1;}

if (data3=='0')
    {if (data4=='0')
{  output[4]=0;
   output[5]=0;
   output[6]=0;
   output[7]=0;}

if (data4=='1')
{  output[4]=0;
  output[5]=0;
  output[6]=0;
  output[7]=1;}

if (data4=='2')
{  output[4]=0;
 output[5]=0;
 output[6]=1;
 output[7]=0;}

if (data4=='3')

{  output[4]=0;
output[5]=0;
 output[6]=1;
 output[7]=1;}

if (data4=='4')

{  output[4]=0;
 output[5]=1;
```

```
 output[6]=0;
 output[7]=0;}

if (data4=='5')
{  output[4]=0;
 output[5]=1;
 output[6]=0;
output[7]=1;}

if (data4=='6')

{  output[4]=0;
 output[5]=1;
 output[6]=1;
 output[7]=0;}

if (data4=='7')

{ output[4]=0;
  output[5]=1;
  output[6]=1;
  output[7]=1;}

if (data4=='8')

{  output[4]=1;
 output[5]=0;
 output[6]=0;
 output[7]=0;}


if (data4=='9')

{  output[4]=1;
 output[5]=0;
 output[6]=0;
output[7]=1;}


if (data3=='1')
        {if (data4=='0')

                {  output[4]=1;
                   output[5]=0;
```

```
                output[6]=1;
                 output[7]=0;}

        if (data4=='1')

                { output[4]=1;
                 output[5]=0;
                output[6]=1;
                output[7]=1;}

        if (data4=='2')

                {  output[4]=1;
                output[5]=1;
                output[6]=0;
                output[7]=0;}

        if (data4=='3')

                { output[4]=1;
                output[5]=1;
                output[6]=0;
                 output[7]=1;}

        if (data4=='4')

                { output[4]=1;
                output[5]=1;
                output[6]=1;
                output[7]=0;}

        if (data4=='5')

                { output[4]=1;
                output[5]=1;
                output[6]=1;
                 output[7]=1;}

}//sending the start bits

for (i=0;i<4;i++){ if (i%2==0)
 {PORTC = 0b00000000;
 _delay_ms(100);
PORTC = 0b01000000;
 _delay_ms(100);
_delay_ms(100);
```

```
_delay_ms(100);
_delay_ms(100);

PORTC = 0b00000000;
 _delay_ms(100);}

  else

 {PORTC = 0b00000000;
 _delay_ms(100);
PORTC = 0b110000000;
 _delay_ms(100);
_delay_ms(100);
_delay_ms(100);_
_delay_ms(100)
  PORTC = 0b00000000;
  _delay_ms(100);}
}
//PORTC=0x00;

for (i=0;i<8;i++)
{//c[0] is the signal , which will be given to the gate of the mosfet of VCO
 //c[1] is the VCO IC inhibit
//Final 8-bit second signal serial transmitted.

if (output[i]==0)
{PORTC = 0b00000000;
 _delay_ms(100);
PORTC = 0b01000000;
for (int i=0;i<4;i++)
{_delay_ms(100);}
PORTC = 0b00000000;
  _delay_ms(100);}

 else
{PORTC = 0b00000000;
 _delay_ms(100);
 PORTC = 0b11000000;
 for (int i=0;i<4;i++)
{_delay_ms(100);}
PORTC = 0b00000000;
 _delay_ms(100);

}}}
```

Receiver code:

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

#include <avr/interrupt.h>

#include <inttypes.h>

#include <util/delay.h>

#define key_play  (PINB & _BV(PB2) )  //  PB2

#define key_play2  (PINB & _BV(PB3) ) //  PB3

#define pind  (PIND & _BV(PD1) )      //  PD1


int main(){


DDRC = 0xff;

DDRB = 0x00;// define as input

PORTC= 0x00;

DDRD=0x00;

int add[3]; // will store the adress of the receiver target

int signal=1;//this decides whether the swith should be on' or off'

int time[4]; // Delays the time after which the command is to be executed. in mins.

PORTC=0b00000000;// alldevices are off initially.


while(1){

/* The scenario : We have two inputs, one representing the '1' in the signal and other representing '0' in the signal.

The main funda is that we will have only one of the PLL giving high output at a time.

our start bits are : 0101 thefore we poll for '0' signal initially. As soon as we get the first zero we began to search for the
```

start bits. Once this start bits are received we look for the signal which is of predefined length of 8- bits and of predefined form.

*/

```
while( (PINB & _BV(PB2) )!=0){_delay_ms(155);
        _delay_ms(155);
                        _delay_ms(155);
                        _delay_ms(155);}


while((PINB & _BV(PB3) )!=0){_delay_ms(155);
       _delay_ms(155);
                        _delay_ms(155);
                        _delay_ms(155);}


while((PINB & _BV(PB2) )!=0){_delay_ms(155);
        _delay_ms(155);
                        _delay_ms(155);
                        _delay_ms(155);}


while((PINB & _BV(PB3) )!=0){_delay_ms(155);
       _delay_ms(155);
                        _delay_ms(155);
                        _delay_ms(155);}
```

// If the program reaches this point it implies that recevier has successfully decoded the the start bit and the signal comming is decently

```
// can be said to be as proper signal and not noise

// we have reached the centre of the signal pulse. **********----------**********

// here onwards we can just add 300ms in each signal to get the next bit

// we donot know what will be the exact signal therfore we will have to wait for any one of the bit to go high


 while(pind==1){};

        // the program shall enter the loop only when any one of the "PLL-0" or "PLL-1" Pin goes high.

        // we hope that the delay in reponse os signal comming to receiver and detection by PLL is less than 10 ms

        if ((key_play)!=0)//zero detected

         {      _delay_ms(150);

                                _delay_ms(150);

                                _delay_ms(150);

                                _delay_ms(150);//detect second bit

                                        add[0]=0;

                                if ((key_play )!=0)

                                  add[1]=0;

                                else

                                  add[1]=1;

                                 _delay_ms(150);

                                        _delay_ms(150);

                                        _delay_ms(150);

                                        _delay_ms(150);

                                if ((key_play )!=0)

                                  add[2]=0;
```

```
                                        else
                                          add[2]=1;
                                        _delay_ms(150);
                                        _delay_ms(150);
                                        _delay_ms(150);
                                        _delay_ms(150);
                                        }

else if ((key_play2 )!=0)//one detected detected
 {  _delay_ms(150);
                          _delay_ms(150);
                          _delay_ms(150);
                          _delay_ms(150);//detect second bit
                                  add[0]=1;

                                  if ((key_play2)!=0)
                                  add[1]=0;
                                  else
                                  add[1]=1;
                                  _delay_ms(150);
                                  _delay_ms(150);
                                  _delay_ms(150);
                                  _delay_ms(150);

                                  if ((key_play2 )!=0)//detect third bit
```

```
                                        add[2]=0;
                                        else
                                        add[2]=1;
                                        _delay_ms(150);
                                        _delay_ms(150);
                                        _delay_ms(150);
                                        _delay_ms(150);}// ready for the fourth bit
        // till here we have decided the adress.
if ((key_play)!=0)
                                signal=0;
 else
                                signal=1;
 _delay_ms(150);
                        _delay_ms(150);
                        _delay_ms(150);
                        _delay_ms(150);//this is for fifth bit being ready

 if ((key_play)!=0)
                                time[0]=0;
 else
                                time[0]=1;
_delay_ms(150);
                        _delay_ms(150);
                        _delay_ms(150);
                        _delay_ms(150);//this is for sixth bit . being ready
```

```
                    if ((key_play)!=0)
                        time[1]=0;
        else
                        time[1]=1;
                    _delay_ms(150);
                    _delay_ms(150);
                    _delay_ms(150);
                    _delay_ms(150);//this is for seventh bit
         if ((key_play)!=0)
                        time[2]=0;
        else
                        time[2]=1;
        _delay_ms(150);
                    _delay_ms(150);
                    _delay_ms(150);
                    _delay_ms(150);//this is for eight bit . being ready


        if ((key_play)!=0)
                        time[3]=0;
        else
                        time[3]=1;


        // uptil now we have decoded entire signal + start bit . since we  are sure that the total number of bits being sent is equal to
        // 18 than we can stop decoding without using any kind of stop bits. They are reducndant here
        // what is left is incoporating them into the output
```

```
//PORTC=
                        // chosing the pin of port B using address add.

                        int t=0;
                        t=add[0]*4+add[1]*2+add[2];

                        if (t=0)
                        {       if(signal==1)
                           PORTC|=0b00000001;
                                else if(signal==0)
                                PORTC&=0b11111110;
                                 }
                        if (t=1)
                        {while(1){
                                if(signal==1)
                                PORTC|=0b00000010
                                else if(signal==0)
                                PORTC&=0b11111101
                                }}
                        if (t=2)
                        {       if(signal==1)
                           PORTC|=0b000000100;
                                else if(signal==0)
                                PORTC&=0b111111011;
                                 }
```

```
                    if (t=3)
                        {       if(signal==1)
                          PORTC|=0b000001000;
                                else if(signal==0)
                                PORTC&=0b111110111;
}}}
```