Haustix - Home Automation using Power line Communication

Hariharan Mathavan and Vikram Vikram

BU Usernames: hrhrnm , gvikram

Abstract

Power Line Communication (PLC) is a communication technology that enables sending data over existing power cables. This means that with just power cables running to an electronic device one can both power it up and at the same time control/transfer data. With the use of Gumstix, we built a smart home automation system which can be controlled using Android application. We also used Passive infrared sensor(PIR Motion sensor) and I^2C(Inter-Integrated Circuit) based Light sensor(TSL2561) to get the necessary data to perform smart automation control. We built an Android application which connects with Gumstix through Bluetooth and controls two rooms. This application can be used to control light, blinds and check if a person is inside the room or not. Gumstix will communicate with each other through PLC modem(TP Link AV 200 Nano).

Keywords: PLC, Gumstix, Sensors, PIR, Light sensor, Ethernet, I^2C, Bluetooth.

1. Introduction

1.1. Motivation

With the spread of Internet of Things, technologies and easy to use APIs, Home Automation Systems have been very popular these past few years. Common examples are systems like Google's Home and Amazon's Echo. Such systems usually work with Wi-Fi as the network physical layer and a phone/web server as the front-end. Google's Assistant API helps parse and process commands from the user and 'Smart Lights' can be controlled via this interface. The devices require an existing Wi-Fi Router and also require wiring for power. In our project, we have eliminated the need for

both of these with Power line Communication. Power line Communication is easy to setup, requires no extra-wiring and is more secure than Wi-Fi.

1.2. The Big Picture

After our initial research into light sensors interfacing and socket programming, and considering the time limitations, and the requirements of the EC535 final project, we created the following design overview for the Haustix project: The Haustix design has three elements: One, a simple and easy to operate android application which can control the home appliances; two, the ability to collect the data from a variety of sensors (light and motion sensor) and three, the ability to ability to communicate through PLC and bluetooth.

1.3. Accomplishments

We designed the system to model two rooms in a house. Each room has a Haustix node which is connected to the others via PLC. A user who connects to one of the nodes using an Android application will be able to control and receive data from all the other nodes. The user can choose to turn the lights on and off, open the blinds and detect the presence of human beings. We also have an automatic light control algorithm using a light sensor to open and close the blinds as needed.

2. Design Flow

The final form of Haustix was designed to replicate the smart automation for two rooms, Room1 and Room2. Room1 had a Gumstix, light sensor, motion sensor, one led, blind, Bluetooth Antenna and a PLC modem(AV 200 Nano). Room2 had a Gumstix, motion sensor, 4 led lights, and PLC modem(AV 200 Nano). An Android application was developed to communicate with the Gumstix in Room1. The android application has toggle/slide controls for all led lights, motion sensors, blinds. This application connects with the Gumstix in Room1 and sends the control commands. Gumstix1 will implement these commands for Room1 and forward the Room2 commands to Gumstix2 through PLC. Figure 1 display the Haustix overview.

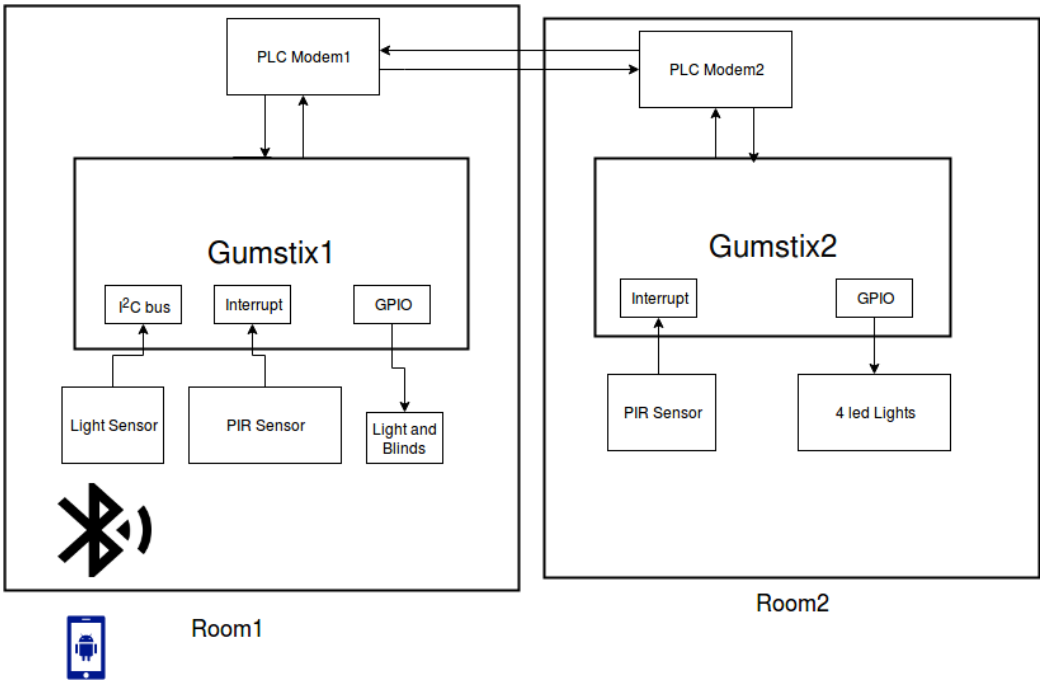

Figure 1: Haustix Overview

2.1. Work by team members

2.1.1. Android application

Hariharan developed the android application to communicate with Gumstix using Bluetooth. He created the message transfer system and check for the status option in the application.

2.1.2. Ethernet Communication

Hariharan created socket based server and client to communication through gumstix. Both of us worked on debugging for ethernet connection as we faced difficulties in communication through ethernet.

2.1.3. Bluetooth Communication

Both of us worked on writing the user-level application for Bluetooth connection on gumstix as server and client modes. Furthermore, Hariharan worked to design the data-flow for Bluetooth communication.

2.1.4. I^2C Communication

Vikram worked on I^2C communication. He wrote the kernel level module(interrupt based) and a user level program(polling-based) to communicate with the light sensor through I^2C. He also wrote different modes(automatic, manual) of light controls through light sensor data and one I^2C echo kernel module to make the data readable by ethernet and Bluetooth applications. Hariharan helped to debug the I^2C for interrupt mode operation but we could not make it work on interrupts so we went ahead with the user-level program.

2.1.5. Interrupt based PIR sensor

Vikram wrote the kernel level module to detect the output from the motion sensor.

2.1.6. Lights and Blinds Controls

Vikram wrote the kernel level module to control the light and blinds from user applications.
Hariharan contributed to 50% and Vikram contributed to 50%. We both contributed equally to the final project report.

3. Project Details

Most of the preliminary research for this project revolved around configuring the communication protocols and sensor interfacing with the gumstix on Linux 2.6.21. Existing drivers available in gumstix for ethernet and I^2C didn't work with our sensor and PLC modem. After deciding to move in a different direction research shifted to requesting data from the light sensors used in this project using existing I2C device drivers and manually reading to and writing from registers on the sensors through I2C.

3.1. Hardware

3.1.1. TSL2561 Digital Luminosity/Lux/Light Sensor

The TSL2561 luminosity sensor is an advanced digital light sensor, ideal for use in a wide range of light situations. The best part of this sensor is that it contains both infrared and full spectrum diodes. That means it can separately measure infrared, full-spectrum or human-visible light. Most sensors can only detect one or the other, which does not accurately represent what human eyes see (since we cannot perceive the IR light that is detected by most photo diodes). This sensor comes with a 3.3V regulator and level

shifting circuitry so it can be used with any 3-5V power/logic microcontroller. The current draw is extremely low, so its great for low power data-logging systems. This draws about 0.5mA when actively sensing, and less than 15uA when in power down mode.
The retailers website [1] provided links to existing drivers, sample code, and the chip data sheet. However, the existing drivers were written for a more recent Linux version (Linux 2.6.30 or above). We downloaded new "i2c-dev.h" file as the one in gumstix is outdated to use for this sensor. As per the data-sheet the TSL256x is controlled and monitored by sixteen registers (three are reserved) and a command register accessed through the serial interface. These registers provide for a variety of control functions and can be read to determine results of the ADC conversions. Figure 2 shows the address and uses of each register. For the user level program, we used dev/i2c-0 file in

ADDRESS	RESISTER NAME	REGISTER FUNCTION
--	COMMAND	Specifies register address
0h	CONTROL	Control of basic functions
1h	TIMING	Integration time/gain control
2h	THRESHLOWLOW	Low byte of low interrupt threshold
3h	THRESHLOWHIGH	High byte of low interrupt threshold
4h	THRESHHIGHLOW	Low byte of high interrupt threshold
5h	THRESHHIGHHIGH	High byte of high interrupt threshold
6h	INTERRUPT	Interrupt control
7h	--	Reserved
8h	CRC	Factory test — not a user register
9h	--	Reserved
Ah	ID	Part number/ Rev ID
Bh	--	Reserved
Ch	DATA0LOW	Low byte of ADC channel 0
Dh	DATA0HIGH	High byte of ADC channel 0
Eh	DATA1LOW	Low byte of ADC channel 1
Fh	DATA1HIGH	High byte of ADC channel 1

Figure 2: TSL2561 Registers

Linux. To read and write from I^2C we used `i2c_smbus_write_byte_data()` and `i2c_smbus_read_byte_data()` functions. We used the light sensor as the slave for reading the lux values and address used for this was 0x39. Once we get the data from `i2c_smbus_read_byte_data()` to calculate the normalized lux value we used c program provided by TAOS, Inc[2].
For kernel level module we followed the steps provided on P page 524 PXA27x Processor Family Developers Manual.

3.1.2. PIR Sensor/HC-SR501

PIR sensor is a body detection sensor module. It requires operating voltage range as DC 4.5-20V. This sensor can be triggered in two ways(jumper selectable) non-repeatable and repeatable trigger. In non-repeatable trigger the sensor outputs high on detection, once the delay time is over, the output is automatically changed from high level to low level. In the repeatable trigger, if there is human activity in its sensing range, the output will always remain high until the people left after the delay will be high level goes low. We can adjust the distance potentiometer clockwise rotation, increased sensing distance (about 7 meters), on the contrary, the sensing distance decreases (about 3 meters). To adjust the delay, potentiometer clockwise rotation sensor the delay lengthened (300S), on the contrary, shorten the induction delay(5S). To interface this sensor with gumstix we connected the output of the sensor to the gumstix GPIO as an interrupt.

3.2. Software

3.2.1. Socket Programming

Sockets are a software level interface to establish communication between two processes. They are bi-directional and they can be accessed the same way as normal files. This includes using file descriptors to set parameters and using the read() and write() system calls. Sockets are uniquely identified by an IP Address and a port number. Usually, one process acts as the server and another acts as a client. The server listens on the given port waiting for a connection. So it's a passive socket. As soon as it gets one, it services it and it may or may not use the same socket to reply to the client. The client initiates the connection and hence it's an active socket.

There are some typical steps to establish communication via sockets.

- Create socket
  Can be created using the function socket(). It requires the socket communication type (Inet or local or Bluetooth), the type (stream or datagram) and the protocol(TCP/UDP/Raw/Bluetooth) as the arguments. In our case, we used raw stream sockets (0) for the Ethernet communication and stream `BTPROTO_RFCOMM` sockets for Bluetooth. It returns the file descriptor for the socket.

- Bind
  The server needs to bind to the socket to listen for connections. Can be accomplished with the bind() function.

- Listen
  The Server should listen on the specified socket using listen(). The connection queue limit can also be specified.

- Connect
  The client connects to the server using connect(). The parameters are the client side socket, the server address and the size of the address object.

- Accept
  To confirm the connection, the server should accept it. This is a blocking function and the process is on hold until a connection is received.

- Send and Receive data
  Like files in Unix, read() and write() can be used to send and read data over an existing socket connection. Other choices are send() and recv().

- Close
  When the program wants to stop communicating, it can call close() on the socket. This will terminate the link and the server will start listening again.

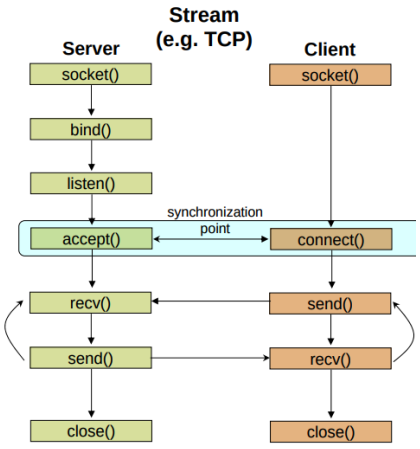

Figure 3: Typical socket connection life cycle [3]

3.3. Android

The Android application was developed using Java. The layout was designed using XML. The Java Socket library was used to establish the connection with Gumstix. Toggle buttons were used to toggle appliances on and off. Automatic and manual control for blinds can be set. Refresh button is for checking the human presence in the room. If a human is present inside the room application will update it as "room occupied" else "room empty". Figure 4 is the screen shot of the app.

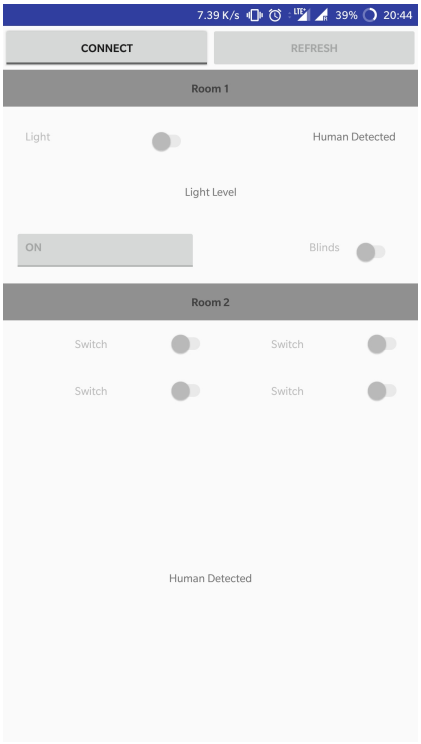

Figure 4: Screen-shot of the App

3.4. Programs and Kernel Modules

We put the codes according to gumstix1 and gumstix2. Gumstix1 and Gumstix2 folder has separate sub folder for each module and user level programs. The Bluetooth server and Ethernet client programs is located in *gumstix1/main/gumstix1.c*. I^2C is located in *gumstix1/I2C/*. Similarly motor(blinds), leds, motion module are located in *gumstix1/Motor/mymotor.c*,

gumstix1/LEDs/myleds.c and *gumstix1/Motion/mymotion.c*. For gumstix2 ethernet server is located in *gumstix2/main/gumstix2.c*. Leds control module is located in *gumstix2/LEDs/myleds.c*.

4. Summary

Haustix is only a proof of basic home automation plug n play device, we feel that the features we have incorporated into it make it well marketable. The goal of this final project was to create a low cost smart home automation system with PLC. The final Haustix achieved all of these goals. gumstixs reads information from multiple sensors and communicate with application and user can see the real time data and control. With more and better placements of sensor user can get more flexibility and precise control. Currently, we embedded one gumstix with bluetooth. For future work, all the gumstix can have bluetooth server in them so user can control from anywhere in home and can communicate with each other through PLC. PLC modem and whole gumstix circuitry can be incorporated as single device.

5. References

1. https://learn.adafruit.com/tsl2561/downloads.
2. https://github.com/alanbarr/ChibiOS_Sensors/blob/master/TSL2561/tsl2561_lux.c
3. http://www.csd.uoc.gr/ hy556/material/tutorials/cs556-3rd-tutorial.pdf