Bluetooth Communication using a Touchscreen Interface with the Raspberry Pi

Gopinath Shanmuga Sundaram, Bhanuprasad Patibandala, Harish Santhanam, Sindhura Gaddam, Vamsi Krishna Alla, Gautham Ravi Prakash, Shiva Chaitanya Vishwakarma Chandracha, Sindhu Boppana and James M. Conrad
jmconrad@uncc.edu
Electrical and Computer Engineering, University of North Carolina, Charlotte, NC, USA 28223-0001

Abstract— **The Raspberry Pi is a low cost single-board computer which has recently become very popular. In this paper we showcase our attempt at building a low cost stand-alone device which transmits data using the Raspberry Pi with Bluetooth and has a resistive touch screen display providing a user interface. The Raspberry Pi is controlled by a modified version of Debian Linux optimized for the ARM architecture. The display contains a graphical user interface which provides various fields for data entry via an onscreen keyboard. Also, various fields were provided to display data obtained from a remote host.**

Keywords—Raspberry Pi, Bluetooth, Touchscreen, Embedded systems

I. INTRODUCTION

Raspberry Pi (represented in Figure 1) is a credit-card-sized single-board computer developed in the UK by Raspberry Pi foundation with the intention of stimulating the teaching of basic computer science in schools. It has two models; Model A has 256Mb RAM, one USB port and no network connection. Model B has 512Mb RAM, 2 USB ports and an Ethernet port. It has a Broadcom BCM2835 system on a chip which includes an ARM1176JZF-S 700 MHz processor, Video Core IV GPU, and an SD card. The GPU is capable of Blu-ray quality playback, using H.264 at 40MBits/s. It has a fast 3D core accessed using the supplied OpenGL ES2.0 and OpenVG libraries. The chip specifically provides HDMI and there is no VGA support. The foundation provides Debian and Arch Linux ARM distributions and also Python as the main programming language, with the support for BBC BASIC, C and Perl.

Bluetooth is a low cost, low power, universal radio interface in the 2.45GHz frequency ISM band that enables portable electronic devices to connect and communicate wirelessly via short-range, ad hoc networks. Bluetooth radios use Frequency-hop (FH) spread spectrum which divide the frequency band into several hop channels in order to cope with severe interference. Bluetooth units that are within range of each other can set up ad hoc connections. Each unit can communicate with up to seven other units per piconet. To regulate traffic on the channel, one of the participating units become a master and all other participants are slaves. Communication in a piconet is organized so that the master polls each slave according to a polling scheme. A master-to-slave packet uses central polling scheme to eliminate collisions between slave transmissions.

A resistive touchscreen panel comprises of several layers, the most important of which are two thin, transparent electrically-resistive layers separated by a thin space. These layers face each other; with a thin gap between. The top layer (the screen that is touched) has a coating on the underside surface of the screen. Just beneath it is a similar resistive layer on top of its substrate.

One layer has conductive connections along its sides, the other along top and bottom. A voltage is applied to one layer, and sensed by the other. When an object, such as a fingertip or stylus tip, presses down on the outer surface, the two layers come into contact at that point. The panel then behaves as a pair of voltage dividers, by rapidly switching between each layer, the position of a pressure on the screen can be read one axis at a time.

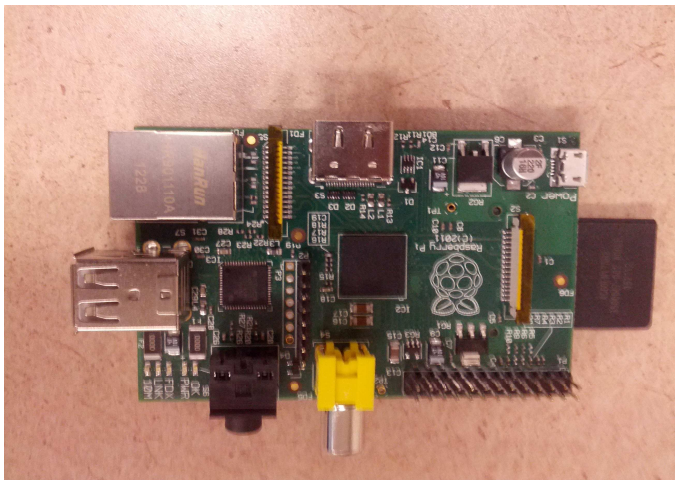

Fig. 1. Raspberry Pi board

978-1-4799-0053-4/13/$31.00 ©2013 IEEE

II. INTERFACING THE TOUCH SCREEN WITH THE RASPBERRY PI

The display is connected to the Raspberry pi via HDMI. The touch screen interface is achieved by building on top of the generic device driver provided by the manufacturer. To obtain fine precision the device is calibrated using the open source touch screen calibrator program Xinput-calibrator. The configuration settings were written to the device configuration files so that the need to run the configuration program after each boot up is eliminated. Also a shell script has been written which brings up the graphical user interface right after boot.

III. FRONT END GRAPHICAL INTERFACE

The graphical user interface forms a crucial part of the project as it is the point where the user interacts with the system. We looked at several ways of doing it such as developing an application in C and cross compiling it for ARM architecture. Another option we explored was to build an application in python taking advantage of the native python compiler included in the Raspbian Wheezy. We finally decided to build the GUI in html because of its rich graphical capabilities and the portability such an implementation would have.

The graphical user interface has been designed to work both in landscape as well as portrait modes and the user can dynamically switch between the two modes. The part of the graphical interface that remains static and has no interaction with the user has been implemented using html. These include background fields for displaying and entering the data. The dynamic functions of several elements of the GUI such as button press animations and buttons for navigating between different pages of the GUI have been implemented using JavaScript. Verification and validation of the data also has been provided in the application so that each data field can only accept its designated type of data.

IV. HARDWARE INTERFACE BETWEEN PI AND BLUETOOTH

We used a Broadcom 2046 Bluetooth 2.1 dongle with the Raspberry Pi. The first task is to burn the Debian Linux specific to Raspberry Pi controller board and install the drivers required for Bluetooth in the Raspberry Pi. PyBlueZ is a Python extension module written in C that provides access to system Bluetooth resources in an object oriented manner. The following commands will install the Bluetooth stack in linux *sudo apt-get install Bluetooth bluez-utils blueman, sudo apt-get install Pybluez.* Once this is done, check with hcitool dev to confirm whether the device is connected or not. The output of the hcitool dev command will be the MAC address of the device. The *hciconfig* command can be used to get the status of the adapters. By default the Bluetooth adapter will be down. We get the Bluetooth adapter running by using the command *hciconfig hci0 up*. For the Bluetooth adapter to be detectable by other devices and to accept incoming connection requests both inquiry scan and page scan should be enabled. To enable this following command is used *hcitool hci0 piscan. .*

V. SOCKET PROGRAMMING AT CLIENT AND SERVER END

We have used Radio Frequency Communication protocol for Bluetooth programming. RFCOMM is a simple set of transport protocols, which provides emulation of RS232 serial ports over the top of the L2CAP protocol. RFCOMM provides a simple reliable data stream to the user, similar to TCP. The RFCOMM protocol determines the need for a communication session so it requests that the L2CA layer initiate a logical connection to the Bluetooth radio. Once the logical channel is created between the client and the server, the protocol is used to establish the parameters of serial connection. The RFCOMM protocol then continually receives serial data from client, identifies and packetizes the data, and sends it to the server.

The MAC address of the server to which the Bluetooth adapter has to be connected is assigned to the variable server_address and port 1 is selected. The RFCOMM protocol is selected using the following instruction *sock = BluetoothSocket (RFCOMM).* To connect to given server MAC address using port 1 the following instruction was used *sock . connect ((server_address , port)).* After the devices are connected the data is sent using the following instruction *sock . send (Humidity)*

VI. SOCKET PROGRAMMING AT SERVER END

In the server side also RFCOMM protocol is used to make the server bind with any device which tries to connect with it, the following instruction is used.

```
server_sock.bind("",port)
```

The empty string indicates that any local Bluetooth adapter is acceptable. To put the socket into listening mode to accept incoming connection the following instruction is used.

```
server_sock.listen(backlog)
```

The method 'listen' accepts a single parameter which indicates the number of pending incoming requests the operating system should accept. Also the data sent is stored in a variable (data) and printed in the server screen is achieved using the below instructions:

```
client_sock , client_info = server_sock .
accept ( )
print "Accepted connection from " , client_info
data = client_sock . recv (1024)
```

The user enters the data at the client end in the GUI as shown in Figure 2. The acknowledgement received from the server end is displayed in the GUI. If the data received at the server end matches with the sent data, the values are displayed as shown in Figure 3.

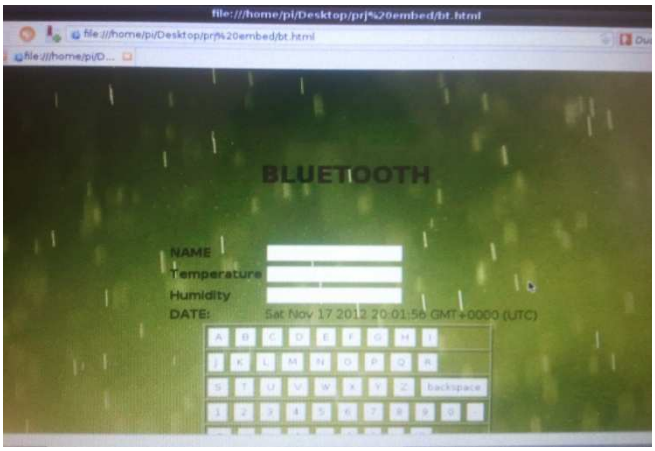

Fig. 2. Graphical user interface

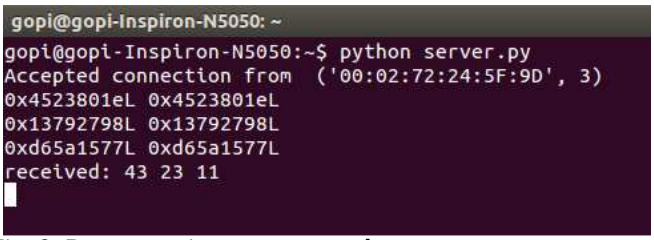

Fig. 3. Data reception on server end

VII. VALIDATING DATA CONSISTANCY

We used Cyclic Redundancy Check for achieving data integrity during the transmission. CRC is an error-detecting code used to detect any accidental changes to data in the digital networks and storage devices. They are specifically designed to protect against common types of errors on communication channels, where they can provide assurance of the integrity of messages delivered. At the client end, the user input data is converted to a unique checksum value using *binascii.crc32*. binascii.crc32 is an inbuilt function in Python to compute the 32-bit checksum of data. binascii.crc32 can be used by importing binascii package in the program. Then the actual data is string concatenated with the checksum value and transmitted.

```
crc=binascii.crc32(mtext1)&0xffffffff
crc1=hex(crc)
final1=str(mtext1)+str(crc1)
final1 = mtext1 + str(crc1)
```

At the server end, the encrypted data which is received is decoded and stored in separate variables. The CRC is recalculated and checked with the checksum value which was received. If both the CRC values are equal, then an acknowledgement is sent to the client side. The decryption of the received data is done using the following program:

```
y1,z1 = (data1[-11:], data1[:-11])
crc1 = binascii.crc32(z1) & 0xffffffff
crc_z1 = hex(crc1)
if crc_z1 == y1
print "received:", z1
```

A run-time error may takes place during the execution of the program, usually happens because of adverse system parameters or invalid input data. We have used the following instructions at the client end when trying to establish connection with the server.

```
try : sock.connect ((server_address,
port))
except BluetoothError:
msg1.set( 'Socket Error = check the
connection' )
```

VIII. RESULTS

By using Radio Frequency Communication protocol we were able to establish Bluetooth transmission in Raspberry Pi controller board with utmost accuracy. Also when there is a mismatch between the sent and received data, we were able to detect it at all instances and notify the client system. By using error handling techniques we were also able to catch the exceptions and were able to retransmit till the acknowledgement was received. When there is no data transmission for a long duration, the socket connection is broken and the connection has to be established again. Figure 6 shows the entire hardware setup.

REFERENCES

[1] J. Haartsen, M. Naghshineh, J. Inouye, O. J. Joeressen, W. Allen, "BLUETOOTH- The universal radio interface for ad hoc, wireless connectivity", *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 2, no. 4, pp. 38 – 45, October 1998.

[2] J. C. Haartsen, "Bluetooth- a new low-power radio interface providing short-range connectivity", *Ericsson Radio Syst. B. V., Emmen, Netherlands*, vol. 88, issue. 10, October 2000.

[3] J. Bray, C. F. Sturman, "Bluetooth 1.1: Connect without Cable", *Pearson Education*, edition 2, 2001.

[4] M. Frodigh, P. Johansson and P. Larsson, "Wireless ad hoc networking- The art of networking without a network", *Ericsson Review*, pp. 10-14, 2000.

[5] A. S. Huang, L. Rudolph, "Bluetooth essentials for programmers", *CambridgeUniversity Press,* 2007

[6] Powers, Shawn. "The open-source classroom: your first bite of raspberry pi." *Linux Journal* 2012.224 (2012): 7

[7] Mitchell, Gareth. "The Raspberry Pi single-board computer will revolutionise computer science teaching [For & Against]." *Engineering & Technology* 7.3 (2012): 26-26.

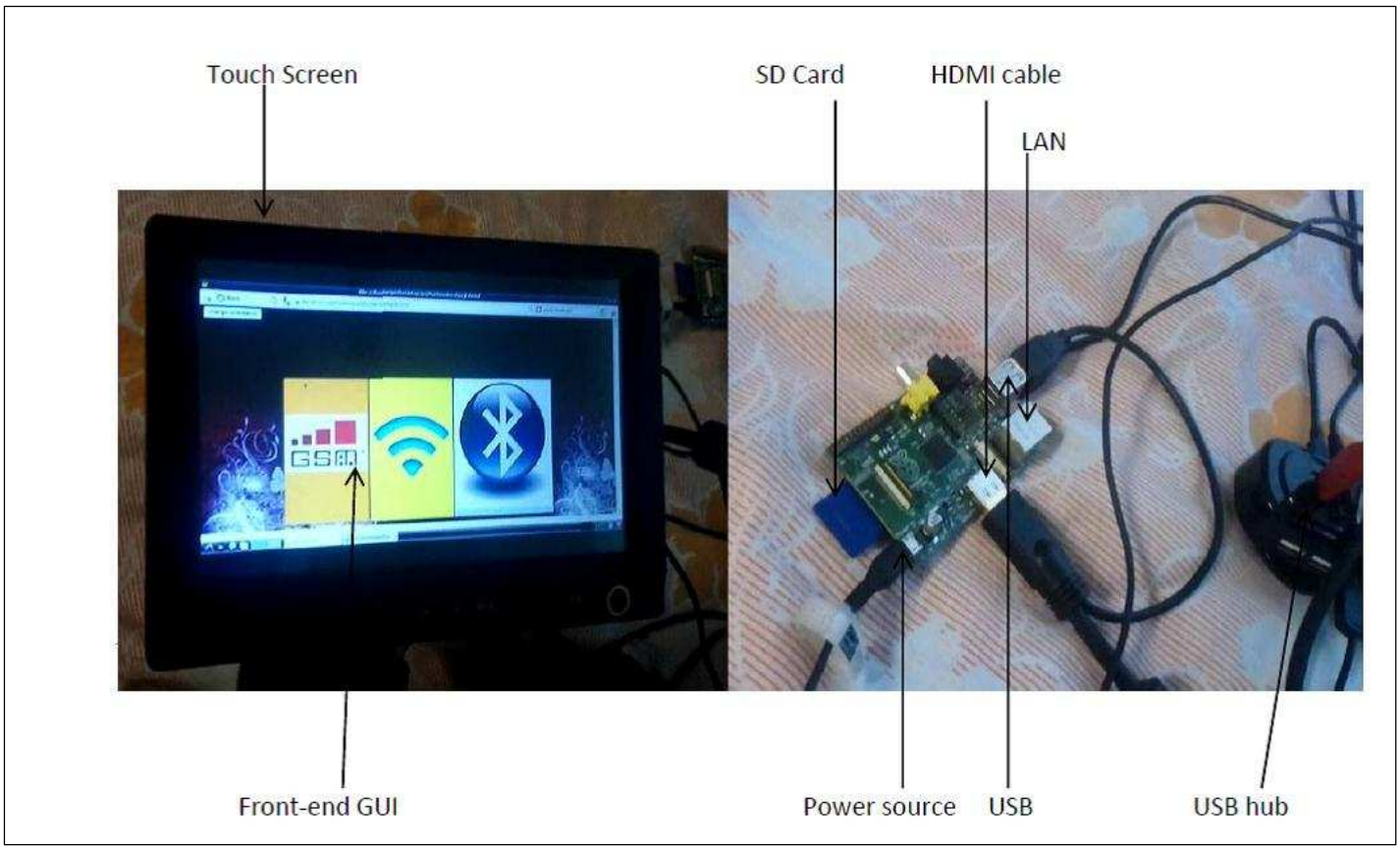

Fig. 4. System setup