User Interface for NIJ Public Safety Radio

Qian Liu, Rithirong Thandee, S.M. Hasan and S.W. Ellingson

October 22, 2008

Contents

1 Introduction **3**

2 Description of Hardware Development Kit **4**
2.1 Gumstix LCD Pack Assembly 4
2.2 SPI Communication 4
2.3 GPIO Pin Allocation 6

3 Description of Software Development Kit **8**
3.1 Getting Started 8
3.1.1 Prerequisites 8
3.1.2 Setting Up a Build Environment 8
3.1.3 Setting Up a Serial Connection 9
3.2 Creating a Bootable microSD Card 9
3.2.1 Repartitioning the microSD Card 10
3.2.2 Installing the Boot Files 10
3.3 Transferring files through Ethernet 11
3.3.1 IP Address Configuration 11
3.3.2 Transfer Command 11

4 Application Program Discussion **12**
4.1 SPI Code 12
4.1.1 GPIO Configuration 12
4.1.2 Data Format Conversion 12
4.1.3 Program Flow 12
4.2 Python Program 13

5 User Guide **14**

A Program Codes **16**
A.1 SPI Code 16
A.2 Python Program 31

B Troubleshooting Resources **34**

1 Introduction

The Gumstix LCD pack [KIT0019][1] is used as the user interface for choosing the frequency of the VT NIJ public safety radio[2]. This report describes its architecture, the software for graphic display and command dispatch, installation instructions for people who wish to study this work, some discussion of issues about how to modify the software for further development, and the user guide for people who would like to use this interface. To provide context, Figure 1 shows a conceptual block diagraph of the prototype.

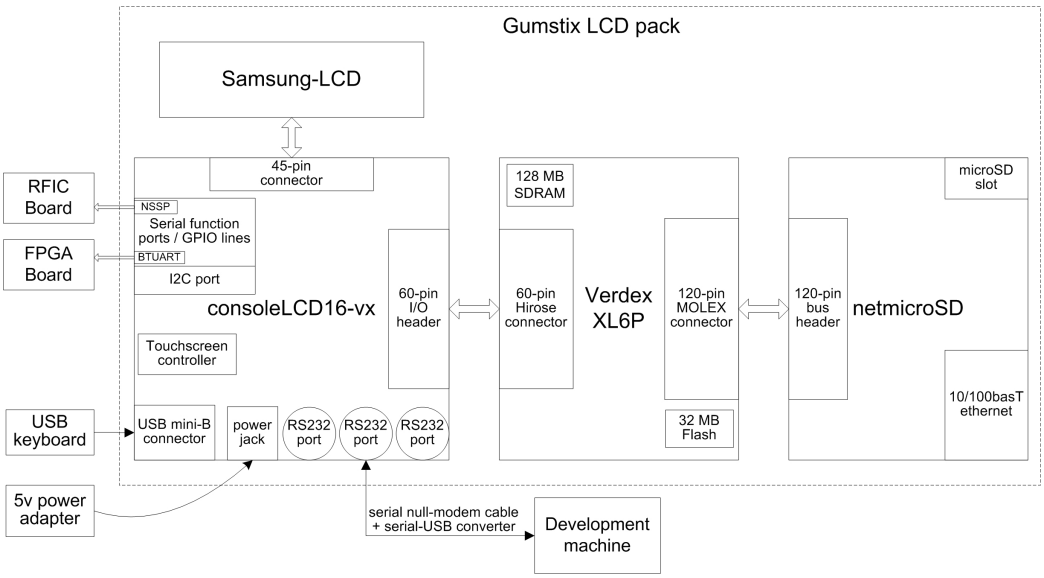

Figure 1: Block Diagram

This report is organized as follows. Section 2 describes assembly of the Gumstix LCD pack and details the interfaces between the Gumstix and the RFIC board as well as the Gumstix and the FPGA board. Section 3 presents the software development kit. Section 4 discusses the application programs. Section 5 summarize the instruction to use the graphic interface. Finally, two appendices present the source code and troubleshooting information.

[1] http://gumstix.com/store/catalog/produc_info.php?products_id=202
[2] http://www.ece.vt.edu/swe/chamrad/

2 Description of Hardware Development Kit

2.1 Gumstix LCD Pack Assembly

Figure 2 presents all the components of the LCD pack used in this report.

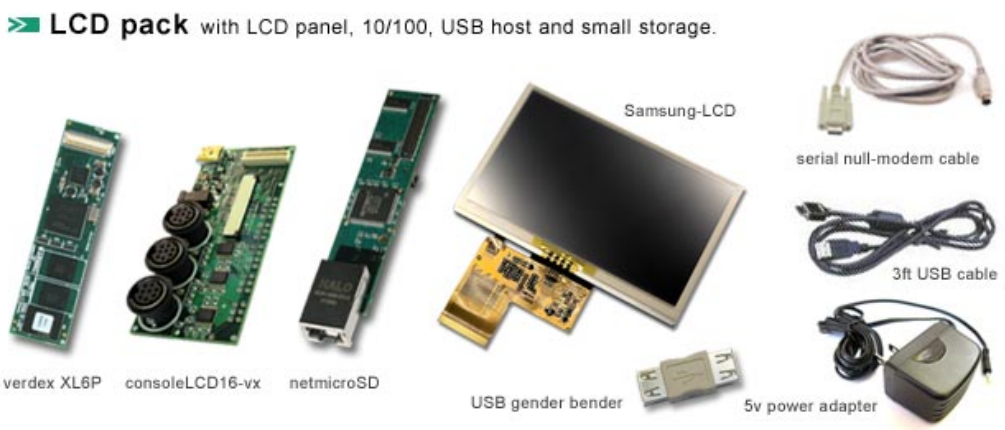

Figure 2: Gumstix LCD Pack[1]

Connect the LCD panel directly to the consoleLCD16-vx board first. Then, the netmicroSD expansion board must be securely connected to the verdex motherboard with a good pressure. Be sure that the pressure is applied only at the indicated locations printed on the verdex motherboard. It is time to attach the verdex motherboard to the consoleLCD16-vx expansion board. Finally, use screws and spacers kit to make them fixed. The online Flash animates the assembly procedure, and Figure 3 shows the model.

Since the LCD pack uses the verdex motherboard, the middle port of the three RS232 ports on miniDIN8 connectors on the consoleLCD16-vx should be used to connect to the serial port on the development machine (a desktop or laptop) through the null-modem serial cable, as Figure 1 shown. If the development machine has no open serial port, a USB serial adaptor is needed to connect to the null-modem serial cable and to the USB port on the development machine.

2.2 SPI Communication

This report uses the "bit banging" method to control the serial peripheral interface (SPI); that is, the master (Gumstix) uses the general-purpose

Figure 3: Assembly Drawing[a]

[a]http://gumstix.com/anim/LCDpack.swf

input/output (GPIO) pins to generate slave select signals in the given time intervals. Generally, data can be transferred in both directions simultaneously when the slave select signal is low active. However, SPI communication in this report is only one way at current stage. That is why the signal `MISO` in Figure 4 is connected with the dotted line. The signal `RESET` in Figure 4 is not defined in SPI bus, but it is required by the RFIC chip as the reset operation is needed before reprogramming the RFIC chip.

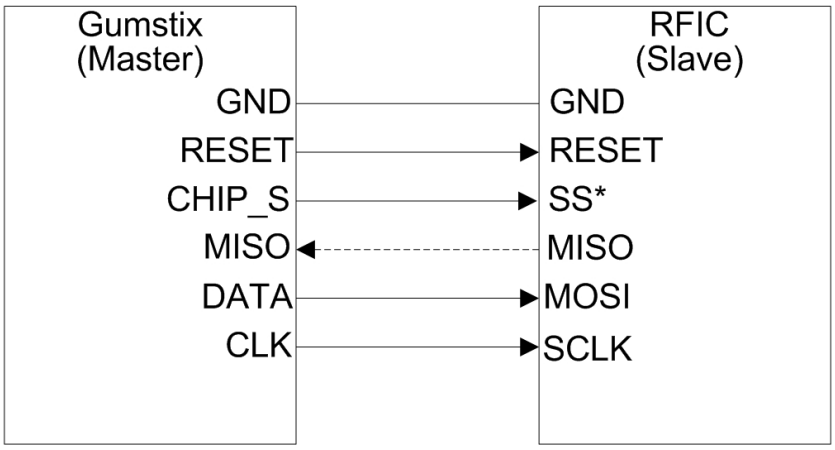

Figure 4: SPI communication between Gumstix and RFIC

The Gumstix selects a channel and sends the corresponding channel information to the FPGA board. The FPGA board processes the channel

information and then sends them to the RF Front End (RFFE) board[1]. The signals `CH_A` and `CH_B` in Figure 5 controls the channel selection. Table 1 is the true table for the channel selection, and the details of the channel information can be found in [1].

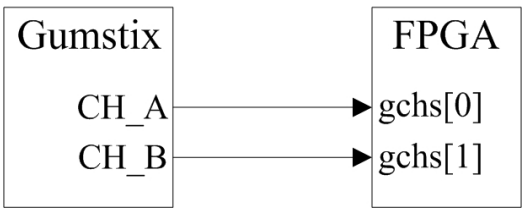

Figure 5: Connection between Gumstix and FPGA

Table 1: Channel Selection True Table

Control Signals		Channel Number
CH_A	CH_B	
Low	Low	1
Low	High	2
High	Low	3
High	High	4

2.3 GPIO Pin Allocation

Figure 6 shows the GPIO expansion interface on the consoleLCD_16vx board in the view of the bottom layer. This report has used SPI port (NSSP) for the SPI communication with the RFIC board[2, 3], and BlueTooth UART (BTUART) to send the channel information signals to the FPGA board.

According to the GPIO characteristics[4] and GPIO pin positions in Figure 6, Table 2 presents the GPIO pin mapping.

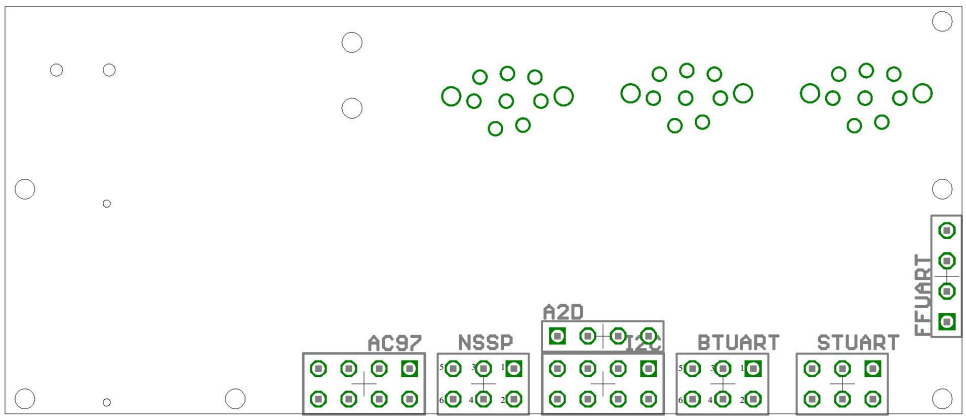

Figure 6: PCB Bottom-Side Silkscreen of consoleLCD_16vx Board[a]

[a]http://pubs.gumstix.org/boards/CONSOLE/LCD16/PCB10016-R1833/PCB10016.bot.png

Table 2: GPIO Pin Definition

Pin Name	Pin Number			Signal Name[a]
	consoleLCD16-vx Board	RFIC/FPGA Board		
	NSSP-1	RFIC Board[b]	J14-6	GND
GPIO<13>	NSSP-2		J14-2	DATA
GPIO<11>	NSSP-3		J14-5	RESET
GPIO<14>	NSSP-4		J14-4	CHIP_S
GPIO<19>	NSSP-6		J14-1	CLK
GPIO<44>	BTUART-5	FPGA Board[c]	J27-9	CH_A
GPIO<45>	BTUART-6		J27-11	CH_B

[a]They are for the Gumstix.
[b]Refers to [2] for the pin information.
[c]Refers to [5] for the pin information.

3 Description of Software Development Kit

3.1 Getting Started

3.1.1 Prerequisites

A computer running a fairly recent Linux distribution, such as Ubuntu 7.10, with at least 10 GB of free space on the hard drive is needed as the development machine. It is better that the development machine has an open serial port. If not, a USB-serial converter is required for the connection. Also an internet connection should be available for downloading packages and source code.

Run "`apt-get install`" in the terminal or use the package management tool to install the following packages: gcc, patch, help2man, diffstat, texi2html, makeinfo, ncurses-devel, cvs, gawk, python-dev, and python-pysqlite2.

This report uses an Ubuntu distribution, so it is necessary to change `/bin/sh` to link to `/bin/bash`. The method[3] is to run "`sudo dpkg-reconfigure dash`", and answer no when asked whether you want to install `dash` as `/bin/sh`.

3.1.2 Setting Up a Build Environment

The online documentation[3] gives instructions to set up a build environment. For this report, the build environment can be set up as follows:

$ svn –version
$ mkdir ∼/gumstix
$ cd ∼/gumstix
$ svn co https://gumstix.svn.sourceforge.net/svnroot/gumstix/trunk gumstix-oe
$ cat gumstix-oe/extras/profile >> ∼/.bashrc
$ sudo groupadd oe
$ sudo *username*[4] -a -G oe *username*[4].
$ mkdir /usr/share/sources
$ chgrp oe /usr/share/sources
$ chmod 0775 /usr/share/sources

[3] http://www.gumstix.net/Software/view/Getting-started/111.html

[4] It is the username of the login account to the Linux distribution on the development machine.

$ chmod ug+s /usr/share/sources

Now close the terminal window and open a new one to make the above environment changes take effect. In the new window, run "`bitbake gumstix-basic-image`" to build a basic root file system image. Depending on the downloading speed, it will take about six hours or more to download all the source code for the initial build. When it completes, the root file file system image and the kernel image can be found by running "`ls -l ~/gumstix/gumstix-oe/tmp/deploy/glibc/images/gumstix-custom-verdex/`".

3.1.3 Setting Up a Serial Connection

First run "`sudo apt-get install cKermit`" to install Kermit communication package. Then, follow online instructions[5] to set up the serial connection.

$ kermit -l /dev/ttyUSB0
C-Kermit> take ~/gumstix/gumstix-oe/extras/kermit-setup
C-Kermit> connect

When the Gumstix LCD pack has been connected and powered on, a message from U-Boot followed by the normal Gumstix boot sequence can be seen in the terminal window, showing the serial connection has been set up successfully.

3.2 Creating a Bootable microSD Card

This reports uses the X Window system (commonly X11 or X) to provide the standard toolkit and protocol with which to build graphical user interfaces (GUIs) in Ubuntu[6]. Considering the large memory size needed for the X11 and limited memory size of the on-board flash, an external 2 GB microSD card is used to store the Linux boot image, the root file image, and application program. In this case, the Gumstix boots from the microSD card, which has nothing to do with the existing U-Boot, rootfs, or kernel images in the on-board flash.

[5] http://www.gumstix.net/Software/view/Getting-started/Setting-up-a-serial-connection/111.html

[6] http://en.wikipedia.org/wiki/X_Window_System.

3.2.1 Repartitioning the microSD Card

First insert the card into the development machine's flash card slot. A microSD to SD card adaptor may be needed to fit the development machine's flash card slot. Assume that the newly inserted card shows up as `/dev/sde1`. Then it is necessary to unmount the card's existing file system by running "`sudo umount /dev/sde1`" before getting started to repartition the card. There are a couple ways to partition the memory disk. One way[7] is to run "`fdisk`" to partition the card, and then run "`mkfs.vfat`" and "`mkfs.ext2`" to format the partitions as a FAT file system and an ext2 file system respectively. However, such methods do not always work. This report uses a GUI partitioning tool, GParted, to partition and format the microSD card.

If GParted is not installed on the development machine, run "`sudo apt-get install gparted`" firstly. When completed, use "`gparted`" to launch the program, and select the media (sde1) to partition it into two partitions as a FAT16 partition and an ext2 partition. The FAT16 partition with the size of 80 MB only places the boot script and linux boot image (uimage), and the ext2 partition uses the rest of the available space of the microSD card for the root file system and application program.

3.2.2 Installing the Boot Files

Two files, gumstix-factory.script and uimage, are required on the FAT16 partition to boot the X11. Download and extract the kernel image to the FAT16 partition by the following procedure:

$ cd /media/disk[8]
$ wget http://www.sakoman.net/oe/mmc-boot/kernel-mmc.tar.gz
$ sudo tar xvf kernel-mmc.tar.gz
$ sudo rm kernel-mmc.tar.gz

Using the same method, download and extract the rootfs image to the ext2 partition as follows:

$ cd /media/disk-1[9]
$ wget http://www.sakoman.net/oe/mmc-boot/rootfs-mmc.tar.gz

[7] http://www.gumstix.net/Documentation/view/Overo/Creating-a-bootable-microSD-card/109.html

[8] Assume the FAT16 partition is mounted as disk

[9] Assume the ext2 partition is mounted as disk-1

```
$ sudo tar xvf rootfs-mmc.tar.gz
$ sudo rm rootfs-mmc.tar.gz
```

When completed, it is very important to unmount the two file systems before placing the microSD card into the slot on the Gumstix board. Skipping this step will corrupt the card data and make booting fail. After transferring the microSD card to the Gumstix board and powering it up, the Gumstix can run with the root file system on the microSD card.

3.3 Transferring files through Ethernet

It is more convenient to debug the application programs on the development machine, and then transfer them to the microSD card after the successful compilation and debugging.

3.3.1 IP Address Configuration

Due to the high speed, the Ethernet cable with a router is an ideal way to copy files on the Gumstix. However, the file transfer will fail if the IP address of the development machine is higher than that of the Gumstix. To solve this problem, run `ifconfig` to change the IP address of either the development machine or the Gumstix. Take the Gumstix as an example. We log into the Gumstix first, and then run the following command

ifconfig eth0[10] 192.168.1.103

to set the Gumstix's IP address to 192.168.1.103. The development machine's IP address can also be changed in the same way through a terminal window on the development machine.

3.3.2 Transfer Command

When the IP address of the development machine is lower than that of the Gumstix, the file can be transferred successfully with the commands, cp, rcp, scp, and so on. This report uses the secure copy (scp) command to transfer files in a remote and secure mode. After changing directory to that of the file which will be transferred, the following command

scp *program_name_exe* root@192.168.1.103:*/home/root*

can copy the file *program_name_exe* to the fold *root* in the home directory of the Gumstix.

[10] Generally, eth0 represents the LAN ethernet connection, and eth1 denotes the wireless connection.

4 Application Program Discussion

The application program is divided into two parts as SPI code and Python program, and all the source code can be found in Appendix A,

4.1 SPI Code

The SPI code is written in C language, and is called by Python to send SPI signals through the GPIO pins to command the NIJ radio[2].

4.1.1 GPIO Configuration

The GPIOs are programmed by the functions distributed by the Free Software Foundation. The following functions represents how to configure the GPIOs.

Function "gpio(OUT, SET, 59);" programs GPIO<59> as a GPIO function, sets its direction as output and sets it to a logic 1.

Function "gpio_function(59, GPIO);" programs GPIO<59> as a GPIO function.

Function "gpio_direction(59, OUT);" program GPIO<59> as an output.

Function "gpio_set(59);" programs GPIO<59> to a logic 1.

Function "gpio_clear(59);" programs GPIO<59> to a logic 0.

Command "i = gpio_status(59);" makes i be set to the logic level of GPIO<59>.

4.1.2 Data Format Conversion

The data and address read from the file are in hexadecimal format; however, SPI bus is serial. Hence, the function `putBin( )` is used to convert the hexadecimal data into binary format. The source code can be found in Appendix A.

4.1.3 Program Flow

Figure 7 shows how the Gumstix sends data to reprogram the RFIC through its GPIOs according to the SPI bus protocol. The clock for the SPI

communication is software-programmed, so the transfer speed depends on the program execution speed.

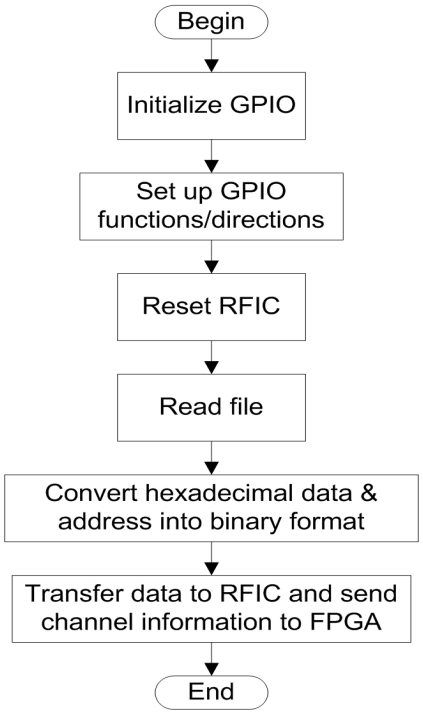

Figure 7: Flow Chart

4.2 Python Program

In this report, the Python program uses basic functions to create four buttons for the frequency choice. When the user presses a button, the Python program will call a C program with an argument, which will read the script file and then send SPI signal through GPIOs to tell the RFIC board which frequency has been selected.

Importing two Python modules as *os* and *sys*, the following command

os.system("./*myprogram myargument*")

can call a C program *myprogram* with an argument *myargument*. In this way, everything inside the quotation is executed as if from the command line.

The GUI can be easily changed by resetting the parameters of the functions such as window.set_border_width(), window.set_size_request(), and so

on[6]. For example, the function `windown.set_size_request(480,272)` creates a window with size of 480×272. The other settings can all be configured in the same way, and the details are explained in Appendix A.

Figure 8 shows the touchscreen interface for the VT NIJ public safety radio. The buttons are used for the user to choose a desired frequency. When the user touches a button, the button will whiten, and the corresponding data will be sent to the RFIC board.

Figure 8: Touchscreen Interface

5 User Guide

Once implemented, the Gumstix user interface operates as follows. Upon power-up, the Gumstix boots from the microSD card, and then the X11 will appear on the screen.

Now a touchstick is used to open the terminal window which is on the left of the taskbar, and then the USB keyboard is used to input the following commands to run the Python.

```
$ cd /home/root
$ python radio.py
```

When finished, the interface shown in Figure 8 will display on the LCD.

Acknowledgment

The authors are thankful to Philip Balister of OpenSDR for providing X Window operating system and patient support to answer the Gumstix related questions.

References

[1] S. M. Hasan and S. W. Ellingson, "Design and Development of an RF Front End Board for NIJ Public Safety Radio," Tech. Rep. 29, Virginia Tech, VA, July 22, 2008. [Online]. Available: http://www.ece.vt.edu/swe/chamrad/.

[2] S. M. Hasan and S. W. Ellingson, "Design and Development of an Evaluation Board with RFIC Ver.4," Tech. Rep. 22, Virgina Tech, VA, Sep 1, 2007. [Online]. Available: http://www.ece.vt.edu/swe/chamrad.

[3] S. M. Hasan and S. W. Ellingson, "Revision of the RFIC Board for NIJ Public Safety Radio," Tech. Rep. 30, Virgina Tech, VA, July 22, 2008. [Online]. Available: http://www.ece.vt.edu/swe/chamrad/.

[4] Intel Corporation, *Intel® PXA27x Processor Family Developer's Manual*, October 2004.

[5] Altera Corporation, *Stratix II DSP Development Board Reference Manual*, 6.0.1 ed., August 2006.

[6] J. Finlay, *PyGTK 2.0 Tutorial.* April 13, 2005.

Appendices

A Program Codes

A.1 SPI Code

```
/*************************************************************/
/* Copyright©2006 Time Crawford <timcrawford@comcast.net>
/* Modified by Rithirong Thandee <rthandee@vt.edu> on April, 2008
/* Modified by S.M. Hasan <hasan@vt.edu> on July, 2008
/*************************************************************/
#include <stdio.h>
#include <time.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <ctype.h>

//========= GPIO Controller Register ========
#define GPLR0     0x40E00000
#define GPLR1     0x40E00004
#define GPLR2     0x40E00008
#define GPDR0     0x40E0000C
#define GPDR1     0x40E00010
#define GPDR2     0x40E00014
#define GPSR0     0x40E00018
#define GPSR1     0x40E0001C
#define GPSR2     0x40E00020
#define GPCR0     0x40E00024
#define GPCR1     0x40E00028
#define GPCR2     0x40E0002C
#define GAFR0_L   0x40E00054
#define GAFR0_U   0x40E00058
#define GAFR1_L   0x40E0005C
#define GAFR1_U   0x40E00060
```

```
#define GAFR2_L  0x40E00064
#define GAFR2_U  0x40E00068
// ================================
#define MAP_SIZE     4096
#define MAP_MASK  (MAP_SIZE-1)
#define IN       250
#define OUT    251
#define GPIO   0
#define AF0     0
#define AF1     1
#define AF2     2
#define AF3     3
#define SET     252
#define CLEAR 253
// ======== GPIO pins number for SPI ========
#define DATA 13
#define CLK   19
#define CHIP_S 14
#define RESET 11
// ==== GPIO pins number for Channel Information ===
#define CH_A   44
#define CH_B   45

typedef unsigned int u32;
void *map, *regaddr;
void putBin(int type, int digit, int val, char bin[ ]);

static void putmem(u32 addr, u32 val)
{
    regaddr = (void*)((u32)map + (addr & MAP_MASK));
    *(u32*)regaddr = val;
}

static int getmem(u32 addr)
{
    u32 val;
    regaddr = (void*)((u32)map + (addr & MAP_MASK));
    val = *(u32*)regaddr;
    return val;
}
```

```
void gpio_set(u32 gpio)
{
    u32 pos;
    u32 bit = 1;
    pos = gpio / 32;
    bit <<= gpio % 32;
    putmem(GPSR0+(pos*4), bit);
}

void gpio_clear(u32 gpio)
{
    u32 pos;
    u32 bit = 1;
    pos = gpio / 32;
    bit <<= gpio % 32;
    putmem(GPCR0 + (pos * 4), bit);
}

u32 gpio_status(u32 gpio)
{
    u32 pos;
    u32 bit = 1;
    u32 data;
    pos = gpio / 32;
    bit <<= gpio % 32;
    data = getmem(GPLR0 + (pos * 4));
    data &= bit;
    if(data == 0)
        return(0);
    else
        return(1);
}

void gpio_direction(u32 gpio, u32 dir)
{
    u32 pos;
    u32 bit = 1;
    u32 data;
    pos = gpio / 32;
    bit <<= gpio % 32;
    data = getmem(GPDR0 + (pos * 4));
```

```
    data &=  bit;
    if(dir==OUT)
            data |= bit;
            putmem(GPDR0 + (pos * 4), data);
}

void gpio_function(u32 gpio, u32 fun)
{
    u32 pos;
    u32 bit = 3;
    u32 data;
    pos = gpio / 16;
    bit <<= (gpio % 16) * 2;
    fun <<= (gpio % 16) * 2;
    data = getmem(GAFR0_L + (pos * 4));
    data &=  bit;
    data |= fun;
    putmem(GAFR0_L + (pos * 4), data);
}

u32 gpio(u32 dir, u32 set, u32 reg)
{
    if((dir != IN) & (dir != OUT)) {
        printf("ERROR: must specify a valid direction\n");
        return(1);
    }
    if((set != SET) & (set != CLEAR)) {
        printf("ERROR: must specify a valid level\n");
        return(1);
    }
    if(reg > 84) {
        printf("ERROR: not a valid register -> %d\n", reg);
        return(1);
    }
    gpio_function(reg, GPIO);
    gpio_direction(reg, dir);
    if(dir == OUT){
        if(set == SET)
            gpio_set(reg);
        else
            gpio_clear(reg);
```

```
    }
    return;
}

void putBin(int type, int digit, int val, char bin[ ])
{
    int one = 0;
    int two = 0;
    int three = 0;
    int four = 0;
    if(type == 1)
    {
        if(digit == 1)
        {
            four = 13; three = 12; two = 11; one = 10;
        }
        else if(digit == 2)
        {
            four = 9; three = 8; two = 7; one = 6;
        }
        else if(digit == 3)
        {
            four = 5; three = 4; two = 3; one = 2;
        }
        else if(digit == 4)
        {
            four = 1; three = 0;
        }
    }
    else if(type == 2)
    {
        if(digit == 1)
        {
            four = 7; three = 6; two = 5; one = 4;
        }
        else if(digit == 2)
        {
            four = 3; three = 2; two = 1; one = 0;
        }
    }
```

```
if(val==‘0’)
{
      if(digit!=4) {bin[one]=‘0’;bin[two]=‘0’;}
      bin[three] = ‘0’; bin[four] = ‘0’;
}
else if(val==‘1’)
{
      if(digit!=4) {bin[one]=‘0’;bin[two]=‘0’;}
      bin[three] = ‘0’; bin[four] = ‘1’;
}
else if(val==‘2’)
{
      if(digit!=4) {bin[one]=‘0’;bin[two]=‘0’;}
      bin[three] = ‘1’; bin[four] = ‘0’;
}
else if(val==‘3’)
{
      if(digit!=4) {bin[one]=‘0’;bin[two]=‘0’;}
      bin[three] = ‘1’; bin[four] = ‘1’;
}
else if(val==‘4’)
{
      if(digit!=4) {bin[one]=‘0’;bin[two]=‘1’;}
      bin[three] = ‘0’; bin[four] = ‘0’;
}
else if(val==‘5’)
{
      if(digit!=4) {bin[one]=‘0’;bin[two]=‘1’;}
      bin[three] = ‘0’; bin[four] = ‘1’;
}
else if(val==‘6’)
{
      if(digit!=4) {bin[one]=‘0’;bin[two]=‘1’;}
      bin[three] = ‘1’; bin[four] = ‘0’;
}
else if(val==‘7’)
{
      if(digit!=4) {bin[one]=‘0’;bin[two]=‘1’;}
      bin[three] = ‘1’; bin[four] = ‘1’;
}
else if(val==‘8’)
```

```
    {
        if(digit!=4){bin[one]='1';bin[two]='0';}
        bin[three] = '0'; bin[four] = '0';
    }
    else if(val=='9')
    {
        if(digit!=4) {bin[one]='1';bin[two]='0';}
        bin[three] = '0'; bin[four] = '1';
    }
    else if(val=='A' || val == 'a')
    {
        if(digit!=4) {bin[one]='1';bin[two]='0';}
        bin[three] = '1'; bin[four] = '0';
    }
    else if(val=='B' || val == 'b')
    {
        if(digit!=4) {bin[one]='1';bin[two]='0';}
        bin[three] = '1'; bin[four] = '1';
    }
    else if(val=='C' || val == 'c')
    {
        if(digit!=4){bin[one]='1';bin[two]='1';}
        bin[three] = '0'; bin[four] = '0';
    }
    else if(val=='D' || val == 'd')
    {
        if(digit!=4) {bin[one]='1';bin[two]='1';}
        bin[three] = '0'; bin[four] = '1';
    }
    else if(val=='E' || val == 'e')
    {
        if(digit!=4) {bin[one]='1';bin[two]='1';}
        bin[three] = '1'; bin[four] = '0';
    }
    else if(val=='F' || val == 'f')
    {
        if(digit!=4) {bin[one]='1';bin[two]='1';}
        bin[three] = '1'; bin[four] = '1';
    }
}
```

```
int main(int argc, char *argv[] )
{
    // take the argument determining the file i'm opening
    printf("the argument is %\n",argv[1]);
    // setting up for bit-banging GPIO
    unsigned int iii, ii, rval, speed, count, tmp;
    int fd;
    fd = open("/dev/mem",O_RDWR | O_SYNC);
    if(fd<0) {
        perror("open(\"/dev/mem\")");
        exit(1);
    }
    map = mmap(0,
        MAP_SIZE,
        PROT_READ | PROT_WRITE,
        MAP_SHARED,
        fd,
        0x40E00000 & ~MAP_MASK
        );
    if(map == (void*) -1) {
        perror("mmap()");
        exit(1);
    }

    // setting up the GPIO functions
    gpio_function(DATA, GPIO);
    gpio_function(CLK, GPIO);
    gpio_function(CHIP_S, GPIO);
    gpio_function(RESET, GPIO);
    gpio_function(CH_A, GPIO);
    gpio_function(CH_B, GPIO);

    // setting up the GPIO directions
    gpio_direction(DATA, OUT);
    gpio_direction(CLK, OUT);
    gpio_direction(CHIP_S, OUT);
    gpio_direction(RESET, OUT);
    gpio_direction(CH_A, OUT);
    gpio_direction(CH_B, OUT);

    // start the RESET (and stay this way)
```

```
gpio_clear(RESET);
gpio_set(RESET);

// start reading file
static const char filename[ ]="test_146.txt[11]";
FILE *file = fopen(filename,"r");
if(file != NULL)
{
    char line [128]; // or other suitable maximum line size
    char addrBin[14] = "00000000000000";
    char valBin[8] = "00000000";
    int i = 0;
    int count198 = 0;

    // read a line
    while((fgets(line, sizeofline, file)!=NULL) && count198<198)
    {
        count198++; // ensuring the program
        char *pch;
        pch = strtok(line,"\t");
        int olddec,remain;
        int counter = 13;
        int address = atoi(pch);
        printf("======the number is %d======\n",address);
        //converting decimal to 14-bit binary
        while(address>0){
            olddec = address;
            remain = address % 2;
            address /= 2;
            if(remain == 1)
                addrBin[counter] = '1';
            else
                addrBin[counter] = '0';
            counter--;
        }
        pch = strtok(NULL, "\t");
        // converting hex to 8-bit binary
        if(pch[1] != NULL) {
```

[11]The file read here is "test_146.txt" for channel 1, "text_223.txt" for channel 2, "text_467.txt" for channel 3, and "text_800.txt" for channel 4.

```
        putBin(2,2,pch[0],valBin);
        putBin(2,1,pch[1],valBin);
}
else {
        putBin(2,2,'0',valBin);
        putBin(2,1,pch[0],valBin);
}
// now we have the address and value

gpio_set(CHIP_S);
gpio_clear(DATA);
gpio_clear(CLK);
gpio_clear(CHIP_S);
gpio_clear(DATA);
gpio_clear(CLK);
gpio_set(CLK);
gpio_clear(CLK);

printf("====setting address======\n");
for(i=0; i<14; i++)
{
        if(addrBin[i] == '0') {
                printf("0");
                gpio_clear(DATA);
        }
        else {
                printf("1");
                gpio_set(DATA);
        }
        gpio_set(CLK);
        gpio_clear(CLK);
}
printf("\n");

gpio_clear(DATA);
gpio_set(CLK);
gpio_clear(CLK);
printf("====setting value======\n");
for(i=0; i<8; i++)
        if(valBin[i] == '0'){
                printf("0");
```

```
                gpio_clear(DATA);
            }
            else{
                printf("1");
                gpio_set(DATA);
            }
            gpio_set(CLK);
            gpio_clear(CLK);
        }
        printf("\n");
        gpio_set(CHIP_S);
    }
    fclose(file); // close the file
//==============end big loop (of 198 times)=================//

    // Receive 1/2x mode initialization
    printf("====normal address complete=======\n");
    putBin(1,3,'0',addrBin);
    putBin(1,2,'8',addrBin);
    putBin(1,1,'C',addrBin);
    putBin(2,2,'3',valBin);
    putBin(2,1,'E',valBin);

    gpio_set(CHIP_S);
    gpio_clear(CHIP_S);
    gpio_clear(DATA);
    gpio_clear(CLK);
    gpio_set(CLK);
    gpio_clear(CLK);

    printf("=====setting address======\n");
    for(i=0; i<14; i++)
    {
        if(addrBin[i] == '0') {
            printf("0");
            gpio_clear(DATA);
        }
        else {
            printf("1");
            gpio_set(DATA);
        }
```

```
        gpio_set(CLK);
        gpio_clear(CLK);
    }
    printf("\n");

    gpio_clear(DATA);
    gpio_set(CLK);
    gpio_clear(CLK);
    printf("====setting value======\n");
    for(i=0; i<8; i++)
    {
        if(valBin[i]=='0') {
            printf("0");
            gpio_clear(DATA);
        }
        else{
            printf("1");
            gpio_set(DATA);
        }
        gpio_set(CLK);
        gpio_clear(CLK);
    }
    printf("\n");

    putBin(2,2,'3',valBin);
    putBin(2,1,'F',valBin);

    gpio_set(CHIP_S);
    gpio_clear(CHIP_S);
    gpio_clear(DATA);
    gpio_clear(CLK);
    gpio_set(CLK);
    gpio_clear(CLK);

    printf("====setting address======\n");
    for(i=0; i<14; i++)
    {
        if(addrBin[i] == '0') {
            printf("0");
            gpio_clear(DATA);
        }
```

```
        else {
                printf("1");
                gpio_set(DATA);
        }
        gpio_set(CLK);
        gpio_clear(CLK);
}
printf("\n");

gpio_clear(DATA);
gpio_set(CLK);
gpio_clear(CLK);

printf("====setting value======\n");
for(i=0; i<8; i++)
{
        if(valBin[i] == '0') {
                printf("0");
                gpio_clear(DATA);
        }
        else{
                printf("1");
                gpio_clear(DATA);
        }
        gpio_set(CLK);
        gpio_clear(CLK);
 }
 printf("\n");
 gpio_set(CHIP_S);

 // Transmit 1/2x mode initialization
 putBin(1,3,'0',addrBin);
 putBin(1,2,'2',addrBin);
 putBin(1,1,'4',addrBin);
 putBin(2,2,'3',valBin);
 putBin(2,1,'E',valBin);

 gpio_set(CHIP_S);
 gpio_clear(CHIP_S);
 gpio_clear(DATA);
 gpio_clear(CLK);
```

```
gpio_set(CLK);
gpio_clear(CLK);

printf("====setting address======\n");
for(i=0; i<14; i++)
{
	if(addrBin[i] == '0') {
		printf("0");
		gpio_clear(DATA);
	}
	else {
		printf("1");
		gpio_set(DATA);
	}
	gpio_set(CLK);
	gpio_clear(CLK);
}
printf("\n");

gpio_clear(DATA);
gpio_set(CLK);
gpio_clear(CLK);

printf("====setting value======\n");
for(i=0; i<8; i++)
{
	if(valBin[i] == '0')
	{
		printf("0");
		gpio_clear(DATA);
	}
	else {
		printf("1");
		gpio_set(DATA);
	}
	gpio_set(CLK);
	gpio_clear(CLK);
}
printf("\n");

putBin(2,2,'3',valBin);
```

```
putBin(2,1,'F',valBin);

gpio_set(CHIP_S);
gpio_clear(CHIP_S);
gpio_clear(DATA);
gpio_clear(CLK);
gpio_set(CLK);
gpio_clear(CLK);

printf("====Setting Address =======\n");
for(i=0;i<14;i++)
{
	if(addrBin[i]=='0'){
		printf("0");
		gpio_clear(DATA);
	}
	else{
		printf("1");
		gpio_set(DATA);
	}
	gpio_set(CLK);
	gpio_clear(CLK);
}
printf("\n");

gpio_clear(DATA);
gpio_set(CLK);
gpio_clear(CLK);

printf("====Setting Value======\n");
for(i=0;i<8;i++)
{
	if(valBin[i]=='0'){
		printf("0");
		gpio_clear(DATA);
	}
	else{
		printf("1");
		gpio_set(DATA);
	}
	gpio_set(CLK);
```

```
            gpio_clear(CLK);
        }
        printf("\n");
        gpio_set(CHIP_S);

        // sending channel information
        gpio_clear(CH_A);[12]
        gpio_clear(CH_B);[13]
    }
    else{
        printf("file does not exist");
        printf("\n");
    }

    return 0;
}
```

A.2 Python Program

```
/*************************************************************/
/* Copyright ©April, 2008 Rithirong Thandee <rthandee@vt.edu>
/*************************************************************/
#!/usr/bin/env python

import pygtk
pygtk.require('2.0')
import gtk
import os, sys, string

#-------------------------------------------------------------
#    Memory Location Definitions
#    Refer to Inter® PXA270 Processor Developer's Manual
#-------------------------------------------------------------
GPIO_BASE_OFFSET = 0x40E00000
SPIDirROff = 0x00000014
```

[12] Use this command for channels 1 and 2, and use the command "gpio_set(CH_A);" for channels 3 and 4.

[13] Use this command for channels 1 and 3, and use the command "gpio_set(CH_B);" for channels 2 and 4.

```
SPIAfROff  = 0x00000068
# Register definitions in byte locations
NSSPCR0_BL = 0x41400000
NSSPCR1_BL = 0x00000004
NSSSR_BL     = 0x00000008
NSSITR_BL   = 0x0000000C
NSSDR_BL    = 0x00000010
NSST0_BL     = 0x00000028
NSSPSP_BL   = 0x0000002C
# Register definitions in word locations
NSSPCR0 = 0x41400000
NSSPCR1 = 0x00000001
NSSSR     = 0x00000002
NSSITR   = 0x00000003
NSSDR    = 0x00000004
NSST0     = 0x0000000a
NSSPSP   = 0x0000000b

class Demo04:

    def Denary2Binary(self, n):
        '''convert denary (base 10) integer n to binary string bStr'''
        bStr = ''
        if n<0: raise ValueError, "must be a positive"
        if n==0: return '0'
        while n>0:
            bStr = str(n % 2) + bStr
            n = n >> 1
        return bStr

    def sendData(self, data):
        #getting the data from the callback function
        #myfile will import the data file
        if data == 146:
            os.system("./spi_read_146")
        elif data = 223.5:
            os.system("./spi_read_223.5")
        elif data = 467:
            os.system("./spi_read_467")
        elif data == 800:
            os.system("./spi_read_800")
```

```
    #function callback to get one thing done anymore
    def callback(self, widget, data):
        if data == "146":
            self.sendData(146)
        elif data == "223.5":
            self.sendData(223.5)
        elif data == "467":
            self.sendData(467)
        elif data == "800":
            self.sendData(800)

    def delete_event(self, widget, event, data=None):
        gtk.main_quit()
        return False

#-----------------------------------------------------------
#     These functions are straightly from the pygtk tutorial
#     and almost self explainatory
#-----------------------------------------------------------
    def_init_(self):
        self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)
        self.window.set_size_request(480,272)
        self.window.set_title("VT Public Safety Radio")
        self.window.connet("delete_event",self.delete_event)
        self.window.set_border_width(10)

        self.image4 = gtk.Image()
        self.image4.set_from_file("banner.gif")
        self.myHBox = gtk.HBox(False, 0)
        self.myVBox = gtk.VBox(False, 0)
        self.window.add(self.myVBoxAll)
        self.myVBoxAll.pack_start(self.imag4,False,False,0)
        self.image4.show()
        self.myVBoxAll.pack_start(self.myHBox)

        self.button1 = gtk.Button("146.5MHz")
        self.button1.modify_bg(gtk.STATE_NORMAL,gtk.gdk.color_parse("#962A1C"))
        self.button1.connect("pressed",self.callback,"146")
        self.myHBox.pack_start(self.button1,True,True,0)
        self.button1.show()
```

```
        self.button2 = gtk.Button("223.5MHz")
        self.button2.modify_bg(gtk.STATE_NORMAL,gtk.gdk.color_parse("#228b22"))
        self.button2.connect("pressed",self.callback,"223.5")
        self.myHBox.pack_start(self.button2,True,True,0)
        self.button2.show()

        self.button3 = gtk.Button("467.6125MHz")
        self.button3.modify_bg(gtk.STATE_NORMAL,gtk.gdk.color_parse("#ffa500"))
        self.button3.connect("pressed",self.callback,"467")
        self.myHBox.pack_start(self.button3,True,True,0)
        self.button3.show()

        self.button4 = gtk.Button("800MHz")
        self.button4.modify_bg(gtk.STATE_NORMAL,gtk.gdk.color_parse("#b03060"))
        self.button4.connect("pressed",self.callback,"800")
        self.myHBox.pack_start(self.button4,True,True,0)
        self.button4.show()

        self.myHBox.show()
        self.myVBoxAll.show()
        self.window.show()

def main():
        gtk.main()

if __name__ == "__main__":
        hello = Demo04()
        main()
```

B Troubleshooting Resources

Website: http://www.gumstix.com/support.html
Forum: http://www.nabble.com/Gumstix-f22543.html
Mailing List: gumstix-users@lists.sourceforge.net
IRC's channel: #gumstix