Dpto de Telemática y Electrónica
Universidad Politécnica de Madrid

TempApp

DESIGNING AN APPLICATION FOR RASPBERRY-PI TEMPERATURE AND HUMIDITY MEASUREMENT AND VISUALIZATION IN A TOUCH SCREEN

Ovidiu Mircea Moldovan

2015

ETSIS.Telecomunicación –Universidad Politécnica de Madrid

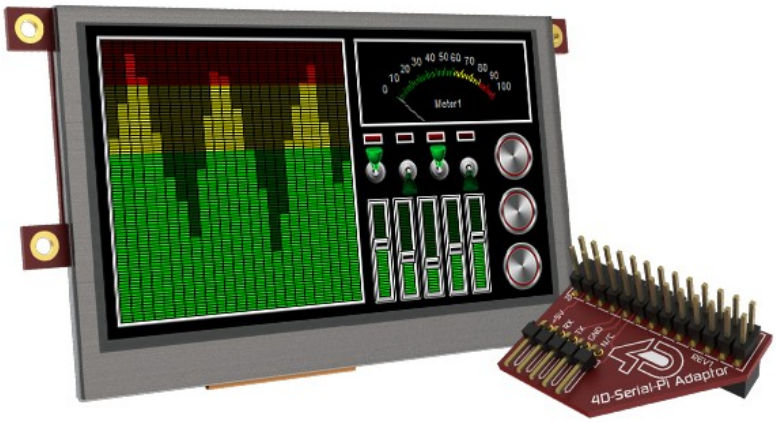
Meter1
4D-Serial-Pi Adaptor

Table of contents

TABLE OF FIGURES

1 SCOPE

1.1 Document Overview

This document reviews the making of and the results of a practical exercise that pretends to build a system that acts like a temperature and humidity alarm. This system is managed by a touchable display.

1.2 Material used

- RaspberryPi and accessories (SD card with Raspian OS)
- circuit board
- red and RGB leds
- HIH-6120-021-001 sensor
- Display 4D uLCD-43PT and accessories
- Windows 7/8 host

[Time to complete the project]: 3/4 days

1.3 Acronyms

IDE	Integrated Development Environment
LCD	Liquid Crystal Display
LED	Light-emitting diode
RGB LED	Consist of one red, one green, and one blue LED
GUI	Graphical user interface

2 REFERENCED DOCUMENTS

2.1 Reference documentation.

The following documentation have to be read and checked.

[RD1] http://www.4dsystems.com.au/downloads/Application-Notes/4D-AN-00001_R_1_0.pdf

[RD2] http://www.4dsystems.com.au/downloads/Application-Notes/4D-AN-00023_R_1_0.pdf

[RD3] https://github.com/4dsystems/ViSi-Genie-RaspPi-Library

[RD4] https://sites.google.com/site/ece3724/Home

[RD5] http://sensing.honeywell.com/honeywell-sensing-humidicon-hih6100-series-product-sheet-009059-6-en.pdf

3 SPECIFICATIONS

3.1 Minimal requirements for your design.

We want to build a basic system to measure the temperature and humidity. This system is part of a more home automation system for complex applications. The system is based on the use of a Raspberry-PI as basic microprocessor-based system. The specifics esta requirements for application are:

- *Measurement of temperature and humidity using an I2C sensor HIH-6120-021-001 Honeywell form.*
- *A red LED will be ON Exceeds When a threshold temperature. A RGB LED Indicates the range of the relative humidity.*
- *Implementation of the HMI using a touch screen. The touch screen is the ULCD-43PT-PI from 4D Systems.*
- *The system Should be autonomous in the sense That it does not need user intervention for booting or starting the applications. Once you apply the power supply it must boots and starts the operation.*
- *The system has to be connected to an Ethernet / TCP-IP based network Allowing the developer to Interact with it.*
- *The system must be Implemented using CodeBlocks with the exception of the software used in the Windows platforms for the HMI design.*

3.2 Deliverables of the project

- The system with all the requirements fulfilled.
- Documentation (this document).
- A presentation describing the system and the building process.
- Source code.

3.2.1 Additional functionalities implemented.

In order to assure the system is started along with the RaspberryPi board a bash script was implemented. It's functionalities are described at #4.5.4.Management script|outline.

4 STEPS FOLLOWED TO IMPLEMENT THE APPLICATION

4.1 Description

The implementation of the required system imply some basic tasks such as:

- Decomposition or modularization of the system (professors way-bill was very useful for that).
- Get some basic wiring skills and physically connect the hardware components.
- Become familiar and implement the controls needed for reading humidity sensor's measures.
- Become familiar and implement the GUI and the back end of the touchable display.
- Integrate the modules.
- A script that manages the system.
- This figure express well the decomposition:

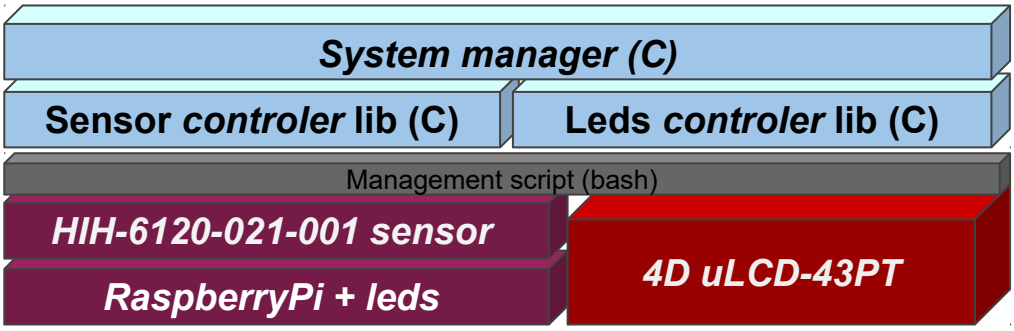

Fig 1: System Architecture

4.2 Task decomposition

1. Connect the LEDs to the RaspberryPi.
2. Implementation of LEDs control interface (C application) and required logic.
3. Connect the sensor to the RaspberryPi.
4. Implementation of Sensor control interface (C application) and required logic..
5. Implementation of touchable display required screens.
6. Implementation of the C application control code for the touchable display.
7. Integration of all this implementation both physical (touch screen to RaspberryPi) and logical (source code).

4.3 Problems found and solutions provided

1. The latest version of Debian (Raspbian Jessie) has problems with managing serial connection the way we need in order to connect the touchscreen to the RaspberryPi.
    1. The solution was a downgrade of the Raspbian version which is the same as a Raspbian Wheezy installation on RaspberryPi SD card.
2. The Raspbian Wheezy comes with no preconfigured i2c devices in configuration files.
    1. A manual configuration of I2C was required. (following the steps mentioned here: https://learn.adafruit.com/adafruits-raspberry-pi-lesson-4-gpio-setup/configuring-i2c)
3. Working with limited resources (RaspberryPi HW) for development and HW connection, some times the SD card may be corrupted.
    1. In order to deal with that there were made periodical backups.

2. If the SD is corrupted then this procedure is useful most of the time:*"So add the SD to a linux, open a console and do* `dmesg` *to view what is the last disck (example sdb) discovered, do* `fdisk -l /dev/sdb` *to see what partitions exist. The linux one should be called linux (example sdb2). finally do* `fsck /dev/sdb2` *to scan and fix the filesystem. "* (source: http://raspberrypi.stackexchange.com/questions/7489/recover-files-from-broken-sd-card-no-boot)

4. The 4D systems are not trending topic so there were some problems to solve with no documentation and no examples.
    1. Solve them using the imagination and the proof error method.

4.4 Tests developed to demonstrate that system is working correctly

1. Test statement were developed for each module (sensor bus connections and correct reading and so on).
2. Screen output of the main procedures such as sensor reading, procedures performed and program status.
3. Both C source code and the management script have their correspondent function tests. This test were not include in the source code because of the continuous changes and the resulting disorder (in code).

4.5 Description (basic) of the source code developed.

As it may be seen in the system architecture diagram there are 4 main software modules.

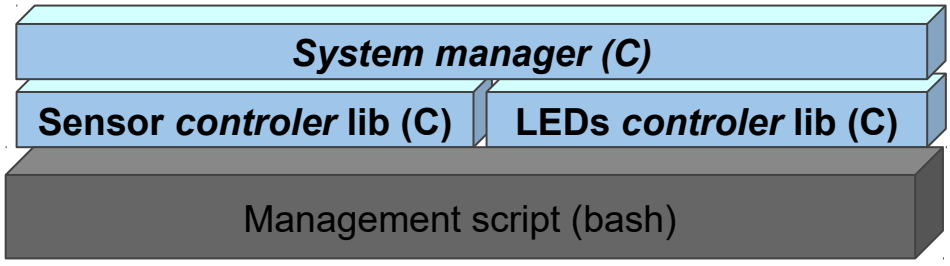

Fig 2: Software Architecture

Note: The code is accessible at this link: https://github.com/OvidiuMM/iot-class

4.5.1 Main (System Manager)

There is a C implementation of an orchestrator or a manager that integrates the system software.

The following is a list of its functions:

```
//***FUNCTIONS DEC.****//
void alarm_led_switch();
void rgb_led_management(int hum_perc);
void system_management(SENSOR *sens);
int handleGenieEvent(struct genieReplyStruct * reply);
```

```
static void *handleLeds(void *data);
void err_f(char mess[]);
void write_string(char mess[]);
```

As specified by the requirements there are some task the system must fulfill such as turning on and off the red led, `alarm_led_switch`, and give the RGB led different colors for certain ranges (% of humidity), `rgb_led_management`. There is also a need to recover GUI user actions such as new temperature bonds, `handleGenieEvent` and so on.
For a better understanding of the different phases of the program and for error checking there were implemented 2 functions, err_f and write_string, which prints to standard output the program messages.
This task are implemented inside the upper functions which are managed by the system_management function.
It is important to realize that this part of the program uses two other modules implemented as libraries. They will be described below.

4.5.2 Sensor controller library

To control the humidity and temperature sensor a software module has been implemented.
It's header file contains the following:

```
//#include "temphum.h"

#define DEVICE_ADDRESS 0x27


//****vars defintion****
//structure that will hold  sensor data
struct temp_hum{
char *fichero;
int fd;
unsigned char buf[10];
float temp,hum;
};
//for ease of use a typedef is defined
typedef struct temp_hum SENSOR;

/********fuctions definitions***/
int start_inquire(SENSOR *sens);
int sens_inquire(SENSOR *sens);
void print_status(SENSOR *sens);
int check_sensor(int file, char *fichero);
void read_request(int file, unsigned char buf[10]);
unsigned char *sensor_data(int file, unsigned char buf[10] );

float get_humidity(unsigned char seis_h, unsigned char ocho_h);
float get_temperature(unsigned char ocho_t, unsigned char seis_t);
void close_connection(int file, char *fichero);
```

The device address is hard-coded since this implementation responds to a specific hardware device.
For a better data management a “struct” type was defined.
Before any reading of measures can be done there is a need to check the connection to the physical device is correct and the information that gets through the channel is meaningful. This is accomplished by the `check_sensor` and `sensor_data` functions. The data obtained is parsed to information, a meaningful value for humidity and temperature, by `get_humidity` and `get_temperature`.
This actions are started and managed by `start_inquire` function.

4.5.3 LED controller library

This module controls the LEDs lightning.

It has a simple structure:

```
void rgb_all_off();
void led_up(int pin);
void led_off(int pin);
void connect_leds();
//consts
#define       RED_LED       17
#define RED_RGB 25
#define GREEN_RGB 27
#define BLUE_RGB 22
```

The pins (of the RaspberryPi board) defined here were used.
There was no need to use other than 2 function for on/off function since each LED (or color for the RGB LED) has his own pin.
In order to connect and set the pin mode the `connect_leds` function is implemented.

4.5.4 Management script

This script is in charge of starting and terminating the program instances. If the RaspberryPi is rebooted or is starting new then it will boot along side and will launch the program. It will not start a new program execution if one is already opened, but It can shut down the old one and then bring up a new one. It's main functions are described by it's “usage” statement:

```
8  #functions *************
9  function usage()
10 {
11 name=$0
12 echo -e "USAGE: \n      $name <-s|-k|-r|-f]"
13 echo -e "       -s start app program"
14 echo -e "       -k kill app program"
15 echo -e "       -r print log file content"
16 echo -e "       -f start app program and write output
17  to log file (in the same directory)"
18 exit 1
19 }
```

Fig 3: Management script usage

To make this script but along with the system it was included inside the “/etc/rc.local” file and added to the system PATH.

4.5.5 Makefile configuration

It's worth mention that the executable file of the program is build through Make. The Makefile was added this fields:

```
main:  main.o
       @echo [link]
       @$(CC) -o $@ main.o temphum.o blinkie.o $(LDFLAGS) $(LIBS)
tempAPP: main.o
```

The temphum.o and blinkie.o were already generated.

5 CONCLUSIONS AND SELF EVALUATION

This assignment was a great practice for basic IoT way of working and principles. Using a single-board computer such as RaspberryPi and other hardware resources such as touch-screens, LEDs and multi sensors it may be the “Hello world” implementation of this field. And as many know building a fairly complex IoT system or any other kind it has his hurdles dose.

It would have been a good practice the development of a real test environment, for both function and system, but due to the lack of time that wasn't possible.

Apart from the already mentioned I think the development of a PoC or an experimental system has proven to be accessible both in time and resources and have great potential in finding easy solutions for many problems of human societies.

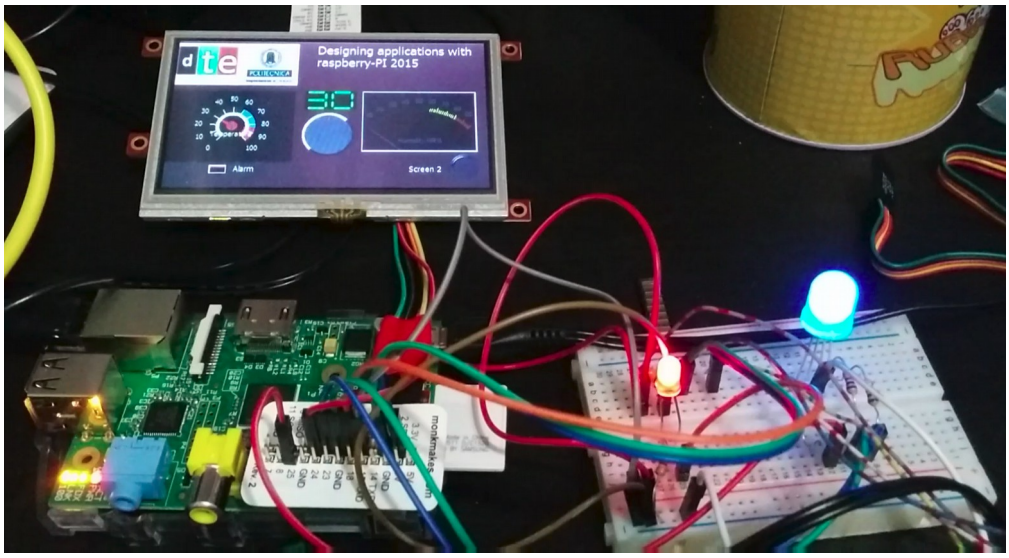

Fig 4: System overview (physical)