The WeatherPi

IFB102 Mini-Project Report

Nicholas Havilah and Josh Horswill

**Statement of Contributions:**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Student Number | Percentage | Signature |
| Nicholas Havilah | N10469231 |  |  |
| Josh Horswill | N10213635 |  |  |

**Project Objectives:**

The aim of the project was to create a weather clock that lights up a selection of LEDs and play sounds from a speaker that correspond with the current weather. The WeatherPi achieves this by connecting to a network to retrieve weather, piping data into text files and running Python code that interfaces with hardware to display the time and weather.

**Review and Discussion of Technologies Used:**

The WeatherPi utilized several key concepts from the lectures. These included:

* Piping data into other files
* Interfacing between the Raspberry Pi and the peripherals
* Languages and libraries
* Networking

The WeatherPi features a range of components that each combine to deliver its objectives. These components include the Raspberry Pi itself, 16x2 LCD, 10K potentiometer, jumper wires and breadboard, speaker, USB drive, varying resistors and LEDs. Along with the hardware, several functions and code were used to bring the project together which include the inbuilt weather-util function and python code that retrieves the data and implements it on the LCD screen.

The WeatherPi utilizes several different peripherals to perform its function as a weather clock. The peripherals used are as follows:

* Raspberry Model 3B+
* 16x2 LCD (Backlit display)
* 3 LEDs (Red, green and a blue one)
* Speaker (3.5mm audio jack connection)
* 10K potentiometer and 3 330Ω resistors
* Breadboard and jumper wires (mix of male to female and male to male wires)

The software used for the project included Python (to run the main code) and the weather-util function. Their implementation will be described later in the report.

The core of the WeatherPi is the Raspberry Pi. To provide an interface between the software and the hardware, the following features were used:

* BCM2837B0 SoC (to run the code)
* GPIO pins (to attach/run the display and the LEDs)
* 3.5mm audio jack (to run the speaker)

As stated before, the primary output for the WeatherPi was an LCD. There were other alternative displays that could be used such as a capacitive touch screen (it could be used as an app interface-allowing the user to select different time zones and locations or have animations to display the weather) or an LCD with a higher resolution. However, 16x2 LCDs are more available and are easier to use in a variety of different ways due to the high volume of internet tutorials, so this was selected as the display for the WeatherPi. To use the LCD for the WeatherPi the following pins were used:

* 1: Ground
* 2:
* 3:
* 4: Register Select
* 5: Read/Write
* 6: Enable
* 7-14: 8-bit data pins
* 15: LED+
* 16: LED-

The wiring configuration for the Raspberry Pi and LCD was as follows:

**LCD WIRING DIAGRAM**

The rest of the system configuration is as shows:

**SYSTEM WIRING DIAGRAM**

In the above diagram, 330Ω resistors were used with the LEDs to prevent them from drawing excessive power from the Raspberry Pi, which could potentially damage it, rendering the WeatherPi unusable. A 10K potentiometer was also used on the LCD to control contrast on the display. Alternatively, a fixed resistor could have been used (provided that the ideal resistance that gave suitable contrast was known) but this would allow for the contrast to be adjusted for different users. Furthermore a 14 pin LCD could also be used instead; it wouldn’t have backlighting meaning that a potentiometer is not needed as it controls the contrast on a backlit display.

To output any information to the display, some form of code was required to power the WeatherPi. There were several options available to use, such as the command line, Python or C. A combination of the command line and Python were used as they can interface easily with each other and the hardware used in the WeatherPi.

In the code, there were two main functions:

1. Function to retrieve the current weather and outputs it to a text file
2. Function to retrieve current time and output to the display based on the data in the text file from the first function

The first function was used to retrieve the current weather and runs from the command line. There were several options available to achieve this, but a function called weather-util was chosen. It was chosen for its output into the console (shown below) as it displayed a name for the weather as well as specific data for it such as temperature and humidity. This meant that less code had to be developed specifically for the project (as the weather patterns had already been defined). The main advantage to using the weather-util function is that the database is updated every 30 minutes (some of the alternate options did not update as frequently), allowing the WeatherPi to become more accurate as a weather clock.

One of the benefits to using weather-util is that the location of the weather could be changed. Weather-util queries a database of airports and retrieves the weather from those locations, so the user can choose the closest airport to them and input its ICAO code to find the weather at that location (Brisbane Airport’s ICAO is ybbn). This result can then be piped into a .txt file where it can be read by the Python code using the following command:

weather ybbn >> output.txt

The second function reads the data contained in the .txt file and outputs commands to the hardware based on the data it finds. To achieve this, Python was used. It was chosen for its ability to easily interface with hardware and as many online resources were available for Python that could be used to achieve the project goals. First, the code opened the .txt and read its contents using a for loop. If it found a weather pattern name (e.g. clear, cloudy, rain, etc.) it recognized, it sent a weather name to the LCD (through the Adafruit CharLCD library), lit up an LED and played a specific sound to the speaker. (a different colour and sound for each weather pattern) In order to fulfill its purpose as a weather clock, the Python code running on the WeatherPi also needed to output the current time to the LCD. This was achieved through the following code:

**WEATHER CODE**

To keep this code running continuously as a clock, a while loop that was set as permanently true was used to keep the code continuously updating. (shown below)

**TIME CODE FOR LOOP**

There were several alternatives that could have been used in this project. Instead of using an LCD, the WeatherPi could have used a touchscreen with a higher resolution that would have acted as an app interface (the user could configure their time zones and locations) Instead of using three separate LEDs of differing colour, a single RGB LED could have been used instead, which could save space when the WeatherPi was built into a small form factor such as a clock shell. Finally, instead of using a wired speaker, the WeatherPi could utilize the Raspberry Pi Model 3B+’s Bluetooth capabilities and connect itself to a Bluetooth speaker. This would be useful for further integration with smart speakers such as Amazon Alexa to potentially transform the WeatherPi into an IoT device.

**Design and Implementation**

* Details in the design implementation
* How it works
* Challenges faced
* Configuration
* Experimentation
* Results
* System diagram
* Future direction and improvements

**A close up of a logo

Description automatically generated**Each sub-system involved in developing the project is designed independently before being pieced together to bring the WeatherPi to fruition. The design itself is branched into two sub-systems with different categories under each branch which can be described by the following:

These branches identified how the project needed to be managed and what needed to be done in order for the hardware and software sub-systems to be complete and implemented into the WeatherPi. With this design ideology in mind, clear and sequential steps were made which allowed the completion of the project.

Explained in brief and independently earlier, each different component to the WeatherPi has a different role which comes together as one to complete its task. From an overall perspective, the WeatherPi operates by doing the following. Firstly, the Weather-Util function is used to network and collect weather data from the desired airport and is piped into a text file after changing several permissions. This text file is then run as a variable in a python script that outputs the time, sky conditions and controls the outputs of the LEDs and speaker. Once the code is run, the Raspberry Pi runs the data it is given through to its destination via the 16-pin connection it has with the breadboard. Based on this received data, a variety of sky conditions could appear on the LCD, with one of three LEDs and sound effects turning on.

Many challenges arose during the set-up which lead to many experimental trials and ultimately to the current configuration. Firstly, finding and using an accurate and consistent weather data base was one of the earlier but important issues that the group faced. Initially, the group intended to use Openweathermap, which was a website that returns data to an API Key based off of its location. However, it proved itself to be complicated and often temperamental when trying to access the weather data, let alone piping it into a text file. The Raspberry Pi website recommended the use of its weather utility which had instructions on its set up. Again, its complexity and lack of nearby stations made the WeatherPi less consistent as it was intended to be. Thus, after multiple attempts at experimenting different configurations, the group lead to finding and ultimately implementing the Weather-Util function which is operable in a Linux command line for easy use and consistent data.

With the weather data organized, the final component to the coding aspect of the project was piping the data into a python script. The text file used required permission changes which halted progress briefly, before finally being implemented into a python script as a variable. The script itself has been arranged to output what the group had chosen after going through several iterations of outputting different data and incorporating different looks.

In terms of hardware, there was no shortage of variety. With the main displays, the initial LCD screen was burnt out when first set up, which lead to the integration of the 10k potentiometer. Without risking the same fate for the LEDs, 330 Ohm resistors were placed in parallel on the breadboard to avoid damaging them. The resistor and LED choices were provided by QUT and happened to be a sufficient supplement for the design. The 2x16 LCD screen was a provided hardware component that was able to provide an ample display for its users. The groups experimentation with other display components was limited due to the lack of availability and high costs, especially with the speaker and LCD choice.

The jumper wire and breadboard configuration took some time to set up with the magnitude of different set ups and possibilities. After multiple experimental circuits, the final configuration was implemented and enabled consistent operation and control over the hardware components of the WeatherPi. Overall, a lot of trial and error was undertaken before finding the final design for the WeatherPi. The software elements took upon multiple different means of retrieving data and outputting to the displays and the variety of hardware made it challenging to decide on what to incorporate into the system. The result of the different design iterations, debugging and experimentation ultimately lead to the final design that the WeatherPi offers to its user.

Despite the final iteration of the project’s design being able to supply an effective and useful weather display, recommendations can be made to allow for its improvement. Firstly, hardware choices could have been more complex with the substitution of the Bluetooth speaker or touchscreen display but due to time and cost it was not a considered option. Coding wise there were a variety of different output options that could have been used instead of the time and sky conditions.

**Reference**