CS12020 Morse Assignment Documentation

Charlie Robinson

7th December 2016

1 Introduction

The purpose of this docmentation is to, I hope to; explain what I have done and why I decided to create the program that way, show which tasks I have completed and how I tested those sections of code. I believe that I have completed all the tasks and they all fulfill the functional requirements. Due to this I believe that I should have a grade in the region of 60-69%.

2 Functional Requirements

2.1 Encoding and decoding CAM code

For this section I identified that all the CAM codes could be grouped into three categories, depending on whether they started with a, %, @ or =. These CAM codes where then put into an array and a matching array was created for their ascii counterparts. This should result in much faster CAM to ascii conversion times. However, it wasn't without its problems, for example the cam2char function couldn't convert multiple spaces, this resulted in me having to implement lots of checks for whether there were spaces at the start or end of the string. which resulted in me having to include lots of

```
#ifdef DEBUG_CAM_CONV
```

and

```
#ifdef DEBUG_INPUT
```

statements so that I could properly test my code which allowed me to find out where I had gone wrong so I could then fix my code. I split the debug statements up because when I was testing the code I was being bombarded with lots of information about sections that I had previously completed and this was unnecessary. This was tested using the automated testing tool and has passed all the tests.

2.2 Transmitting CAM code by LED

In this section I used two functions, ledDecision() and sendLed(). ledDecision() was responsible for checking the input and sending the CAM string to the correct LED. sendLed() read the potentiometer value, calculated what one time unit was equal to and then sent the data to the LED passed to it from ledDecision(). This was also tested using the automated testing tool.

2.3 Responding to CAM data messages

The hardest part of this section was the LP message. Originally, I had a very long if statement to check for each number, this was obviously a bad way of doing it. I then moved the number checking into its own function which allowed me to have much more flexibility about how I tackled the problem. I decided to create a string made purely of the numbers and then iterate through that string checking whether each character was a number.To test this part I flashed the lights at their specified PWM values for two seconds. The IR* and RXIR messages were easy to implement but couldn't really be properly tested. Likewise PV was alright to implement and was tested myself, the output of this can be seen in the provided screenshots.