Raspberry Pi - Morse Code Reader
Software Specifications
Author: Yashiv Fakir
Date: 20 January 2020

1   Introduction

This document aims to outline all the software aspects of the Morse Code project. The report begins with the programming language used for the project. Then moves onto the algorithm and sequence of the code. Finally concludes with the libraries that would be required as well as the software environment setup that would be needed on the Raspberry Pi.

2   Programming Language

The decision for choosing a programming language was between C/C++ and Python. The project is to be coded using C/C++. This decision was made as C/C++ offers much faster speeds during program execution once compiled. There is also much better memory control which is advantageous given that the memory on a Raspberry Pi is limited. A combination of the two would have been a better solution as interfacing with with the Pi is much simpler to do so in Python but offers many potential problems that would draw one's focus from the main object that is to decipher a Morse coded message. Lastly the C code would be compiled using the GNU GCC compiler.

2.1   C Libraries

- wiringPi.h - This library is used to interface between the C code and the GPIO ports on the raspberry Pi. WiringPi also has functionality to deal with button interrupts.

- mcp3008.h - This library is part of the wiringPi.h library that will interface between the ADC chip and the C code

- stdio.h - This library will allow for the decoded message to be displayed on the console as well as any progress or failed errors

- string.h - This library will aid in the conversion and combination of each character to form the deciphered string.

3 Program Flow

The overall sequence of events will be pretty similar only differing to cater for the various input encoded messages being the paper input and the LED input.

The actual algorithm for converting from the measured LDR voltage values to alphanumeric symbols will be the same for both. The only exception would be that a high voltage will mean that and black mark is detected for the paper input message, where as, a low voltage will mean the light emitted from the LED and part of the LED input message. The program is also initiated to begin reading only when a push button is pressed. This was done to prevent the constant use of memory to see if there is an input being fed to the sensor or not. Instead due to the button, memory used to store the input signal will thus only be used when there is a input to be stored.
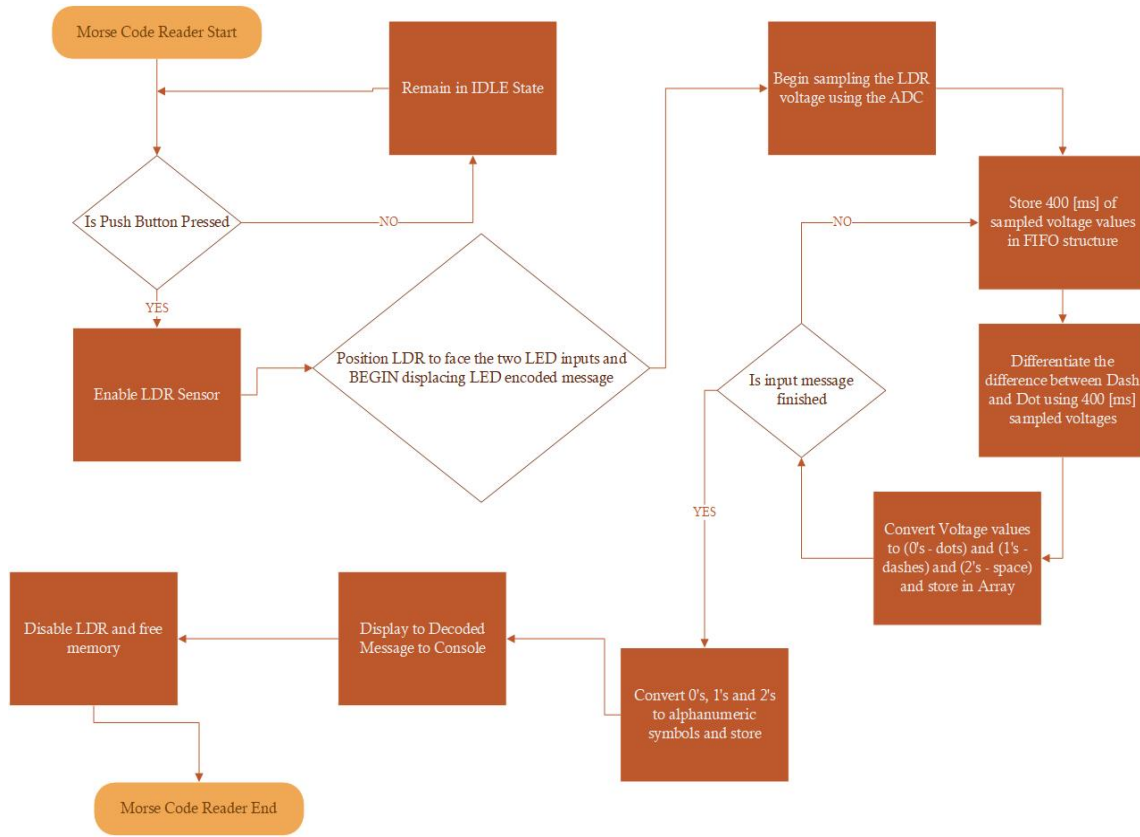
3.1 Program sequence for Paper Input

Below is a flow chart showing the program sequence for a paper encoded message.
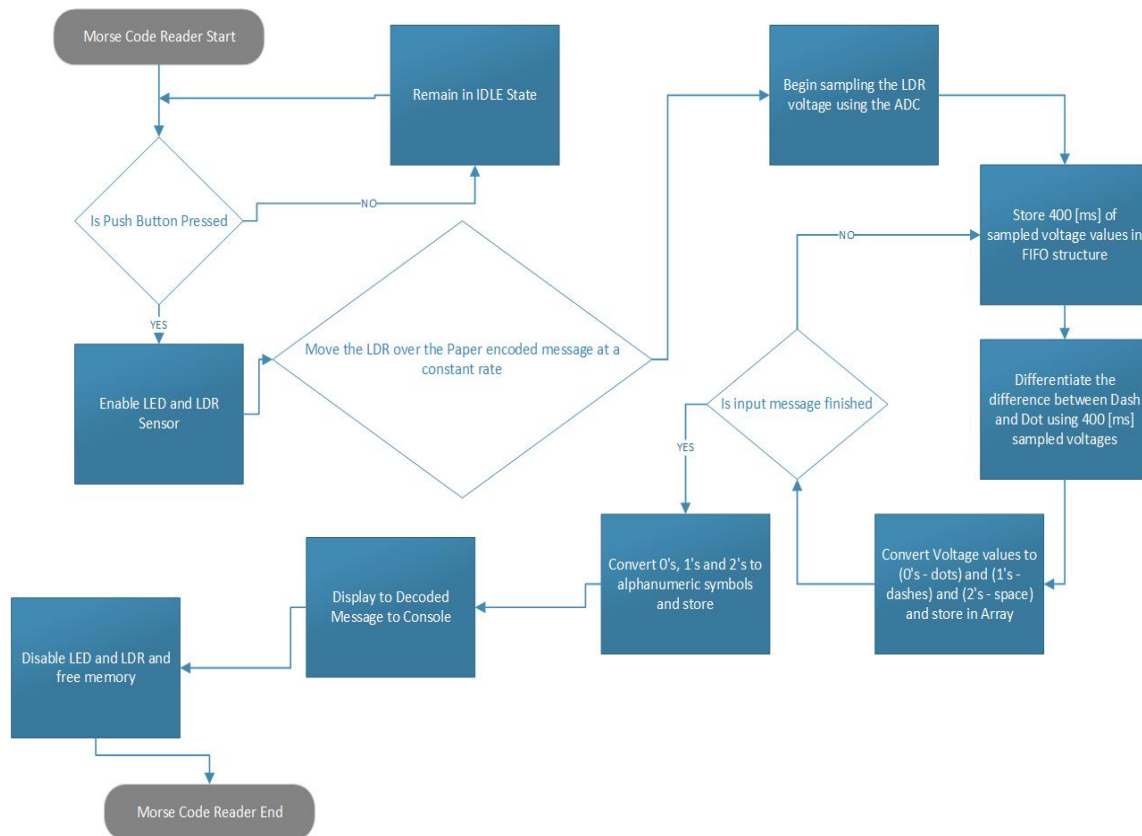
3.2 Program sequence for LED Input

Below is a flow chart showing the program sequence for an LED encoded message.

```
Morse Code Reader Start
    ↓
Is Push Button Pressed ──NO──→ Remain in IDLE State
    │ YES                              ↑
    ↓                                  │
Enable LDR Sensor ──→ Position LDR to face the two LED inputs and
                      BEGIN displacing LED encoded message
                                  ↓
                      Begin sampling the LDR voltage using the ADC
                                  ↓
                      Store 400 [ms] of sampled voltage values in FIFO structure
                                  ↓
                      Differentiate the difference between Dash and Dot using 400 [ms] sampled voltages
                                  ↓
                      Convert Voltage values to (0's - dots) and (1's - dashes) and (2's - space) and store in Array
                                  ↓
Is input message finished ──NO──→ Store 400 [ms]...
    │ YES
    ↓
Convert 0's, 1's and 2's to alphanumeric symbols and store
    ↓
Display to Decoded Message to Console
    ↓
Disable LDR and free memory
    ↓
Morse Code Reader End
```

4 Algorithm

4.1 Error Handling

1. Error: If either input being fed is blank (if a blank sheet of paper or if the LED's aren't illuminated as input to the system).

   Solution: Will have to have a check function that analyses the input voltages by the LDR. If no changes after a period of time say 2-3 [s] then stop the reading and display an error message.

2. Error: If the rate that the paper input is fed to the system is either extremely fast or slow to record the results. This error poses challenges when at a very fast rate.

   Solution: An extremely slow input would still work with the main algorithm however would have to have upper limit in the amount of voltage values measured to avoid running out of memory, after which the system times out. For a fast input the voltage signal would either be a constant signal or only have a single spike. Thus if constant the previous error would handle this. But if a single spike is detected then this could either be a 'T' = dash or 'E' = dot in Morse code (due to the input rate), or could be an error this making it difficult to differentiate.

3. Error: If the space between LED flashes is two quick or to slow.

Solution: The same solution for the paper input can be used here, however and LED flash will have a much greater effect on the LDR thus making it easier to detect.

4. Error: If there is insufficient high and low voltage samples measured to determine the difference between a dash or dot or even the difference between white or black (Lit LED or unlit LED).

   Solution: If the measured data is full of all high voltages or all low voltages then a difference between the message and a space/break would not be established. Also if half of the samples are low voltages and the other half all high, then a difference would be established but the length of a dot or dash would not be established. If either of the 2 conditions are met, then the previous data is to be kept and the remainder of the voltages that were measured after the initial data set are then to be analysed.

4.2   Sudo Algorithm

Still under development...

References