Monorail Emulator

Module Specification

Contents

[System Configuration 3](#_Toc497242850)

[Station Number Entry 3](#_Toc497242851)

[Station Name Entry 3](#_Toc497242852)

[Station Travel Time Entry 3](#_Toc497242853)

[Stop Time Entry: programCounter 4](#_Toc497242854)

[Monorail Emulation 4](#_Toc497242855)

[Travelling Between Stations 4](#_Toc497242856)

[Stopping At A Station 4](#_Toc497242857)

[Stopped At A Station 4](#_Toc497242858)

# System Configuration

The system configuration section of the program uses four different steps to complete the configuration. The overall flow goes: detailing the required input -> waiting for the user to enter their input -> accepting (and storing in memory) or rejecting the input. A variable called programCounter is used to determine what part of the program is running, as different functionality is required at different points in the program, i.e. keypad scanner only stores the entered string to memory if the program has passed the Station Number Entry module. The value of programCounter at each point is given at the start of each paragraph.

## Station Number Entry: programCounter==0

The first function to be used is *INIT*, where the screen is cleared, along with the number of stations, and the string “Please type the max number of stations: ” is printed to the screen. The printing uses the function *LOAD\_STRING* which loads the z pointer the beginning of the string that is stored in SRAM. It then uses the function *PRINT\_STR* to print the string one byte at a time, and to scroll the text when the string goes past the edge of the screen. This is the ‘detailing the required input part’ of the flow. It then moves in *KEYPAD\_SCANNER.* This is the ‘waiting for user to enter input’ part of the flow. The entered number is stored in a register, and if further numbers are inputted, the value in the register is multiplied by ten before the new input is added. This is to ensure that the final input (which is entered using the hash key) is not greater than ten. If it is greater than 10, it is rejected. The program runs the function *TOO\_MANY\_STATIONS*, which sends an error message to the screen, returns the program to INIT, and the user is given another opportunity to enter a number from 1 to 10. If it is less than or equal to 10, the input is accepted and the program moves to the Station Name Entry part of the program.

## Station Name Entry: programCounter==1

In the function *PRINT\_NAMING\_STRINGS,* the temporary variable called holder is incremented every time a string asking for a station name is printed, until it is greater than the number of stations previously entered by the user, which tells the program that all names have been entered and to move on. The screen is cleared before displaying the string “Please type the name of Station “, then the variable holder is printed (it gives the number of the station that the program is up to), then it prints a colon, so that the whole string would be (i.e. for station 4) “Please enter the name of station 4:“. Again the program jumps to keypad scanner, and the string the user enters is stored in SRAM. This continues until every station has a name, and then the program moves onto the Station Travel Time Entry part of the program.

## Station Travel Time Entry: programCounter==2

In the function ENTER\_TIMES, the holder variable is again cleared and the x pointer pointed to the empty memory space where the travel times between the stations will be stored. *INC\_STATION* increments the value of holder in order to cycle through the stations. *CHECK\_TO\_STATION* defines holder2, which holds the value of the ‘To’ station in the statement: “Time from Station 1 to Station 2 is: “, i.e. here Station 1 is the ‘From’ station and Station 2 is the ‘To’ station. *PRINT\_TIMING\_STRINGS* then prints the string “Time from Station “, then the value of holder, then the string “ to Station “, then the value of holder2, then the string “is: “. It then jumps to *KEYPAD\_SCANNER* to accept user input if it is less than or equal to 10. If not, it prints an error message and allows the user to re-input the value. This continues until holder2 is fully incremented, at which point it increments holder as normal, then sets the value of holder2 to 1, so that the monorail loop is closed i.e. “Time from Station 5 to Station 1”. It then moves on to the Stop Time Entry part of the program.

# Stop Time Entry: programCounter==3

The *MONO\_STOP\_TIME* function loads a two-part string, which has been split up because it is too long to print on one line. It prints the first half on the top line, scrolling the screen so that the user can read it, and then the second half of the string on the bottom line. It then stores the user input in SRAM, if it is less than or equal to 10. If not, it prints an error message and asks again for the input. It then prints the ‘configuration complete’ string and makes the user wait for 5 seconds.

# Monorail Emulation

The function ­*RUN\_MONORAIL* clears all variables required for operating the variable and also stores a 1 to a memory location which indicates the monorail is running. The station number is incremented in *INC\_STATION2* and the speed is set to 60rps.

## Travelling Between Stations

As the monorail is now running, the “Next Station: “ string is displayed on the LCD, along with the name of the next station for the length of time previously chosen for travel between the two stations. Since the holder variable is used to specify where the monorail is currently, it has to be incremented to find the name of the next station, and then decremented again so that the correct time is selected from memory. The correct about of travel time is achieved used repeated calls of the wait\_one\_sec macro, which waits one second.

## Stopping At A Station

While the monorail is running, if a push button is pressed, an EXT\_INT interrupt occurs (same for both PB0 and PB1 since they result in the same behaviour) and the stopAtStat variable is set to one. The state of this variable is checked as the monorail reaches the end of its nominated travel time, the program runs the function *SHOULD\_MONO\_STOP* and if it is set to 1, the monorail stops at the next station.

## Stopped At A Station

While stopped at a station, the current station name is printed using the *PRINTING\_CUR\_STAT* method. The program then enters *WAIT\_AT\_STAT*, where the monorail stops for the nominated amount of time. It also sets a variable in TIMER0OVF to 1, which indicates the monorail is stopped at the station, and so that the LEDs should flash 3 times a second. After the stop time is completed at the station, the monorail jumps up to *INC2\_STATION* and moves onto the next station infinitely.