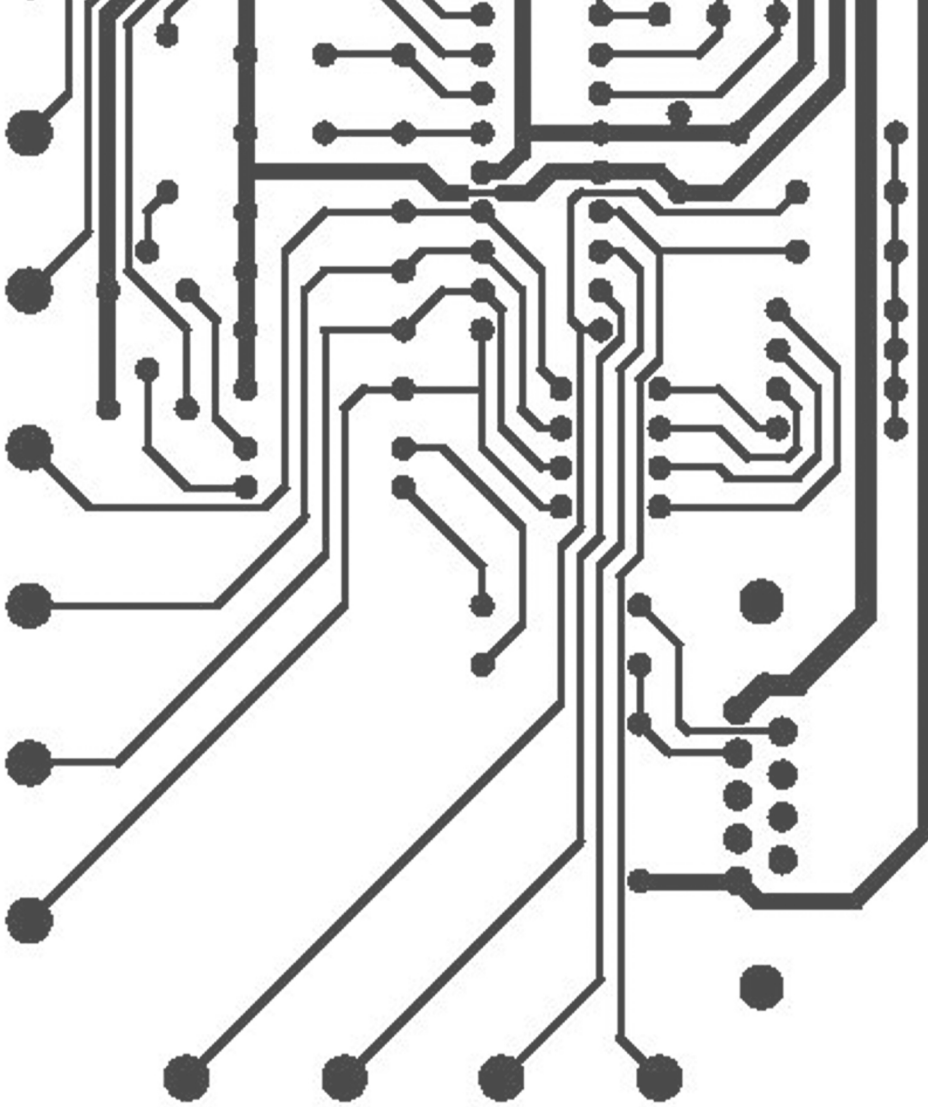
COMP2121 -
Microprocessors and Interfacing



A Monorail Emulator

Design Manual

Contributers

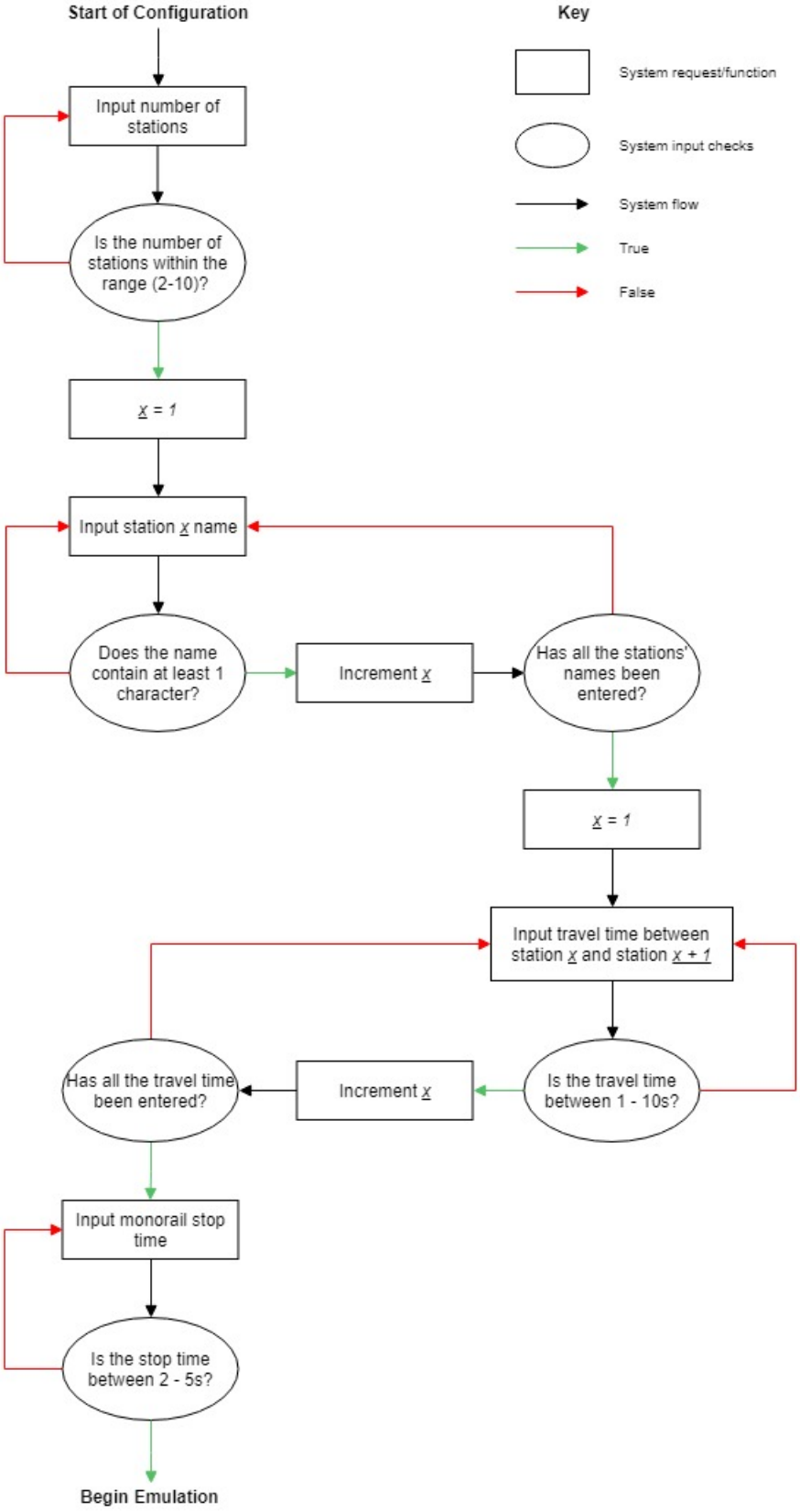
Quynh Boi Phan (z5205690) (55%)
Julie Truong (z5207716) (45%)

Table of Contents

1. System Control Flow

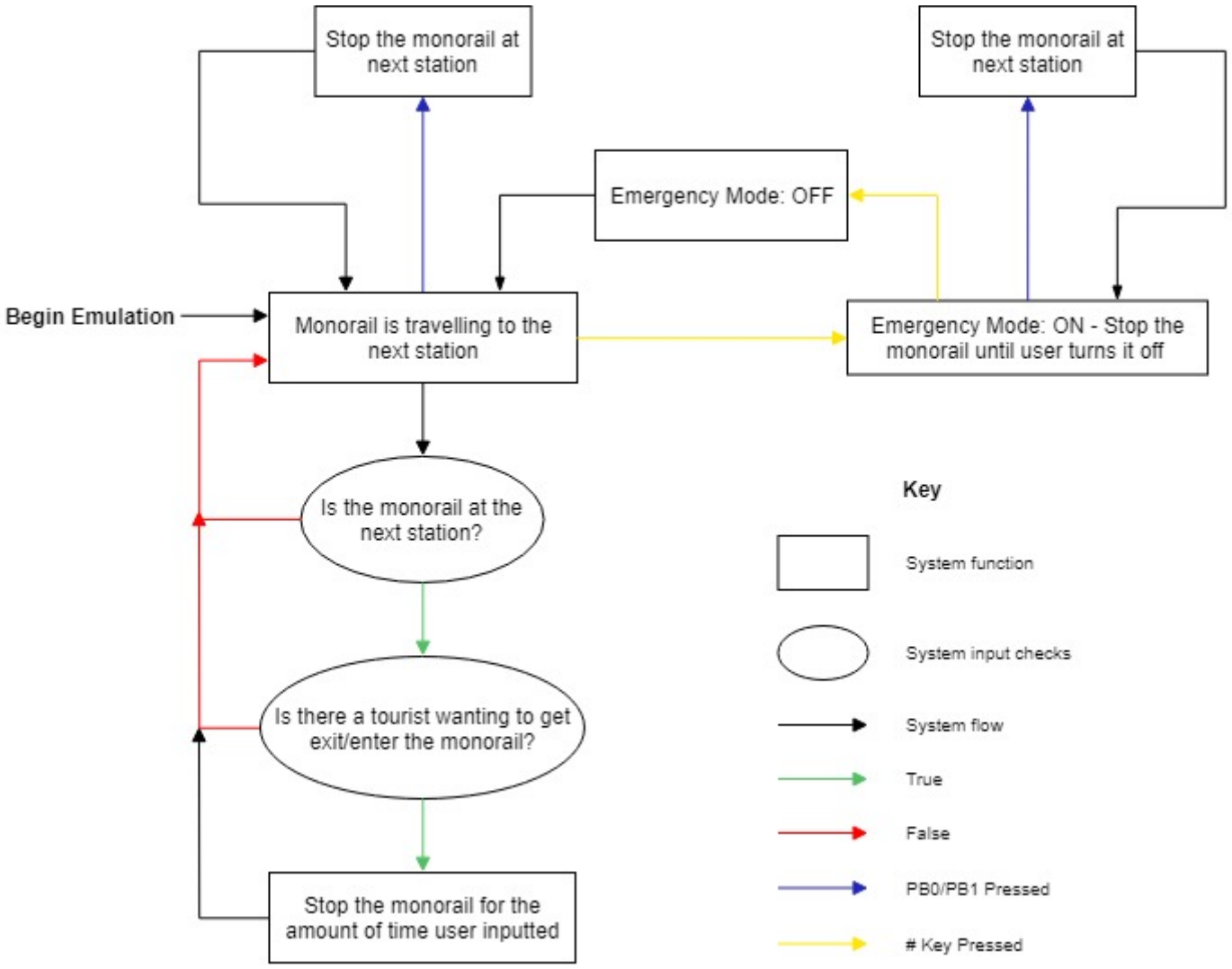
1.1 System Control Flow in Monorail Configuration

The following flow chart illustrates the system control flow of the monorail emulator when configuring the monorail emulation:



1.2 System Control Flow in Monorail Emulation

The following flow chart illustrates the system control flow of the monorail emulator when emulating the monorail system:



2. Data Structures

2.1 Registers

The following registers are used, with some defined with a name using .def:

Name	Register	Description
	R0	Contains the lower byte of the result of two registers multiplied
curr_station	R5	Contains the station position the monorail is travelling to/stopped at in the emulation
	R14	Temporary register used to navigate through the array of station names and load the next station name to display
	R15	Temporary register used to load Timer0 Interrupt counter
temp	R16	Temporary register
row_keypad	R17	Current row number when scanning keypad
col_keypad	R18	Current column number when scanning keypad
mask_keypad	R19	Mask for column/row whilst scanning keypad
temp2	R20	Temporary register
temp3	R21	Temporary register
counter	R22	For looping: indicates what number the loop is up to before exiting the loop
timer	R23	Countdown how much time is left
	R24	Contains the lower byte of DELAY_1MS to countdown to 0
	R25	Contains the higher byte of DELAY_1MS to countdown to 0
xl	R26	Pointer used for the station names to display during the emulation
xh	R27	
yl	R28	Pointer used for the travel time between adjacent stations during the emulation
yh	R29	
zl	R30	Pointers used for the station names, travel time between adjacent stations, and loading up a string during configuration
zh	R31	

2.2 Keypad

2.2.1 Keypad Constants

The following constants are used for the keypad, defined using .equ:

Name	Constant	Description
PORTLDIR	0xF0	Sets all columns as output, rows as input
INITCOLMASK	0xEF	Mask used to analyse which key is pressed in the keypad column
INITROWMASK	0x01	Mask used to analyse which key is pressed in the keypad row
ROWMASK	0x0F	Mask the column bits to read only the row bits

2.3 LCD

2.3.1 LCD Constants

The following constants are used for the LCD, defined using .equ:

Name	Constant	Description
LCD_RS	7	Bit number in Port A to trigger its functions for LCD
LCD_E	6	
LCD_RW	5	
LCD_BE	4	
F_CPU	16000000	CPU Frequency (16MHz)
DELAY_1MS	F_CPU/4/1000 - 4	1ms delay
LCD_MAX_CHAR	16	Maximum characters LCD can display in one line

2.4 System Mode

2.4.1 System Mode Flag Constants

The following flag constants are used for the system mode, defined using .equ:

Name	Constant	Description
CONFIGURATION	0	Indicates that the system is currently configuring the emulation
EMULATION	1	Indicates that the system is currently in the emulation

2.4.2 System Mode Data Segment

The following data segments are used for the system mode, defined using .byte:

Name	Bytes	Description
mode	1	Stores the system mode flag constants

2.5 Monorail Configuration

2.5.1 Monorail Configuration Constants

The following constants are used for the monorail configuration, defined using .equ:

Name	Constant	Description
MAX_STATIONS	10	Maximum number of stations in the emulation
MAX_STATION_NAME	10	Maximum number of characters in a station's name
MAX_TRAVEL_TIME	10	Longest travel time between 2 stations (s)
MIN_STOP_TIME	2	Minimum stop time at a station
MAX_STOP_TIME	5	Maximum stop time at a station

2.5.2 Monorail Configuration Flag Constants

The following flag constants are used for the monorail configuration, defined using .equ:

Name	Constant	Description
ENT_STATION	0	Indicates that the user is inputting the number of stations
ENT_NAME	1	Indicates that the user is inputting the station's name
ENT_TRAV	2	Indicates that the user is inputting travel time between 2 stations
ENT_STOP	3	Indicates that the user is inputting the monorail stop time
NO_OFFS_ENT	0xFF	Indicates that the user has not pressed an offset key (A - C)[1]

2.5.3 Monorail Configuration Data Segment

The following data segments are used for the monorail configuration, defined using .byte:

Name	Bytes	Description
config_stage	1	Stores the monorail configuration flag constants (except NO_OFFS_ENT) to mark which part of configuration the user is at
num_station	1	Stores the number of stations for the emulation
station	100	Stores all stations' name
travel_time	10	Stores all the travel time between 2 stations
stop_time	1	Stores the monorail stop time
upto_station	1	Stores which station user is up to whilst inputting the station's name and travel time between 2 stations
letter_offs	1	Stores the offset key value[1]

[1] *Offset is used to indicate the position of the letter on top of number keys with letters*

2.6 Monorail Emulation

2.6.1 Monorail Emulation Constants

The following constants are used for the monorail emulation, defined using .equ:

Name	Constant	Description
MOT_60RPS	75	Motor speed PWM constant for 60rps

2.6.2 Monorail Emulation Flag Constants

The following flag constants are used for the monorail emulation, defined using .equ:

Name	Constant	Description
MONO_STOP	0	Indicates that the monorail is currently stopped
MONO_TRAVEL	1	Indicates that the monorail is travelling to the next station
MONO_EMERG	2	Indicates that the monorail is in emergency mode

2.6.3 Monorail Emulation Data Segment

The following data segments are used for the monorail emulation, defined using .byte:

Name	Bytes	Description
emul_stage	1	Stores the monorail emulation flag constants (except NO_OFFS_ENT) to mark which part of configuration the user is at
next_stop	1	Stores 0 if the monorail should not stop at the next station Stores 1 if the monorail should stop at the next station
button_flag	1	Indicates if the button (PB0/PB1) has been pressed already so interrupt does not happen twice

2.7 Timer0 Overflow Interrupt

2.7.1 Timer0 Overflow Interrupt Constants

The following constants are used for the Timer0 Overflow Interrupt, defined using .equ:

Name	Constant	Description
ONE_SEC	61	The number of timer0 overflow interrupts to occur to in 1 second
SIX_HERTZ	10	The number of timer0 overflow interrupts to occur in ⅙ second, used for blinking lights at 3Hz (3 blinks per second)

2.7.2 Timer0 Overflow Interrupt Data Segment

The following data segments are used for the Timer0 Overflow Interrupt, defined using .byte:

Name	Bytes	Description
t0_int	1	Stores the number of timer0 overflow interrupts left to occur

2.8 String Displays

The following strings are used to display onto the LCD, defined using .db in code segment:

Name	String	Description
num_sation_str	“# of Stations: ”	Used when the number of stations is required
station_name_str	“Name for stationS”	Used when the name of a station is required. Full string: “Name for station S*x*” (*x* - station number)
travel_time_str	“Time taken from S”	Used when the travel time between 2 stations is required. Full string: “Time taken from S*x* to S*x+1*” (*x* - station number)
stop_time_str	“Monorail stop time: ”	Used when the monorail stop time is required
next_station_str	“Next station: ”	Used before displaying the name of the station the monorail is travelling to

3. Algorithm

The program will always first start by configuring the emulation. When all inputs are correct, the monorail emulation will begin.

3.1 Monorail Configuration

Throughout the configuration stage, the keyboard is continuously being scanned to check if a key has been pressed. If a key is pressed, the program will handle it according to the stage it is upto.

3.1.1 Entering the Number of Stations

Named as *config_stage_1*, this stage is used to record the number of stations for the emulation. To indicate this to the user, *num_station_str*[2] is displayed on the LCD.

As a numeric input is required in this stage, pressing keys *, and A - D does nothing to the program. By pressing a number, the number will be recorded into *num_station*[3] and is displayed on the LCD. The user can only enter a two digit number in the system; any number larger than this does nothing to the program. Pressing the # key will indicate end of input.

Key	Functions
0 - 9	Stores the number into *num_station*
#	End of input
*	Does nothing; will not affect the program
A - D	

At end of input, the program will first check if the user input is correct before continuing onto the next stage. As a number from 2 - 10 is expected as an input, any number outside this range will be considered as incorrect input (a *red LED* will light up to indicate incorrect input). In this case, the user will have to re-enter the number of stations to a number that fits within the requirements.

3.1.2 Entering the Stations' Names

Named as *config_stage_2*, this stage is used to record the number of stations for the emulation. To indicate this to the user, *station_name_str*[2] is displayed on the LCD.

Throughout this stage, the program will iterate through the 2D array *station*[3] and store the user inputs until it records down the name of the last station.

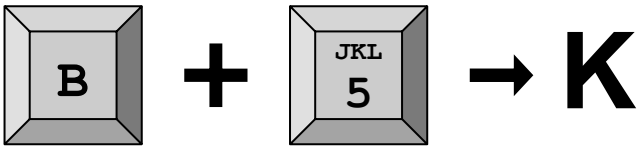
As an alphabetic input is required in this stage, all keys are used except for keys *1*, *D*, and *, in which pressing these keys will not affect the program. The *offset keys* (A - C) are used to map the relative position of the letters on top of the letter/number keys (2 - 9). The offsets are as follows:

- A - offset *0*
- B - offset *1*
- C - offset *2*

[2] *Refer to Section 2.8 for string displays*

[3] *Refer to Section 2.5.2 for monorail configuration data segment*

For example, to input the letter “K”, the key *B* is first pressed, followed by the key *5*.



To indicate that an offset key has been pressed, a *yellow LED* will light up. The LED will turn off after a letter/number key is pressed. If the offset keys are not first pressed, pressing the letter/number keys will do nothing to the program, and hence will not store the user’s input. If multiple offset keys are pressed before pressing a number key, the program will record the most recent offset key pressed.
Pressing the key *0* will introduce a whitespace character ‘ ’, however the whitespace can only be used after a letter is inputted by the user. Pressing the key # will signal end of input.
If done correctly, all letters and whitespaces that are inputted will be recorded into the byte it is upto in the 2D array *station* and displayed on the LCD.

Key	Functions
2 - 9	Stores the letter into the byte it is up to in *station*
0	Stores a whitespace into the byte it is up to in *station*
A - C	Offset keys
#	End of input
1	Does nothing; will not affect the program
*	
D	

At end of input, the program will first check if the user input is correct before continuing on with the program. As the station’s name has to contain at least one character, an empty string returned will be considered as incorrect input (a *red LED* will light up to indicate incorrect input). In this case, the user will have to re-enter the station’s name.
The user will have to enter every station's name before continuing to the next stage.
The user cannot enter more than 10 characters in the station’s name; once the tenth character is reached, the system will automatically go to the end of input.

3.1.3 Entering the Travel Time between Adjacent Stations

Named as *config_stage_3*, this stage is used to record the number of stations for the emulation. To indicate this to the user, *travel_time_str*[4] is displayed on the LCD.

Throughout this stage, the program will iterate through the array *travel_time*[5] and store the user inputs until it records down the time taken from the last station to the first station.

As a numeric input is required in this stage, pressing keys *, and A - D does nothing to the program. By pressing a number, the number will be recorded into the byte it is up to in the *travel_time* array and is displayed on the LCD. For example, if the time taken between the third and fourth station is required, the

[4] *Refer to Section 2.8 for string displays*
[5] *Refer to Section 2.5.2 for monorail configuration data segment*

user input is recorded into the third byte. The user can only enter a two digit number in the system; any number larger than this does nothing to the program. Pressing the # key will indicate end of input.

Key	Functions
0 - 9	Stores the number into the byte it is up to in *travel_time*
#	End of input
*	Does nothing; will not affect the program
A - D	

At end of input, the program will first check if the user input is correct before continuing onto the next stage. As a number from 1 - 10 is expected as an input, any number outside this range will be considered as incorrect input (a *red LED* will light up to indicate incorrect input). In this case, the user will have to re-enter the number of stations to a number that fits within the requirements.

3.1.4 Entering the Monorail Stop Time

Named as *config_stage_4*, this stage is used to record the number of stations for the emulation. To indicate this to the user, *stop_time_str*[6] is displayed on the LCD.
As a numeric input is required in this stage, pressing keys *, and A - D does nothing to the program. By pressing a number, the number will be recorded into *stop_time*[5] and is displayed on the LCD. The user can only enter a two digit number in the system; any number larger than this does nothing to the program. Pressing the # key will indicate end of input.

Key	Functions
0 - 9	Stores the number into *stop_time*
#	End of input
*	Does nothing; will not affect the program
A - D	

At end of input, the program will first check if the user input is correct before entering the emulation. As a number from 2 - 5 is expected as an input, any number outside this range will be considered as incorrect input (a *red LED* will light up to indicate incorrect input). In this case, the user will have to re-enter the monorail stop time to a number that fits within the requirements.

3.2 Monorail Emulation

3.2.1 Emulation Implementation

Immediately after configuration, the monorail will start to travel, indicated by the moving motor spinning at 60rps. At all times, the LCD displays the name of the next station.

While in emulation, the monorail will always be travelling unless it is given a signal to stop, that is if a tourist wants to get on at the next station, a tourist wants to get off at the next station or the emergency mode has

[6] *Refer to Section 2.8 for string displays*

been enabled. Otherwise, the monorail will continue to travel through its route, continuing to loop back to the first station and so on.

The push buttons, PB0 and PB1 are used to stop the monorail where PB0 simulates that a tourist wants to get off at the next station and is indicated by LED2. Similarly PB1 simulates a tourist wants to get on at the next station and is indicated by LED3. Once the monorail has reached the next station, after a push button has been pressed, the motor will stop and 2 LEDs, LED0 and LED1 will blink at

Emergency mode is enabled by pressing the ‘#’ key on the keypad, in which the monorail will stop between two adjacent stations. This stops the motor immediately and lights up the strobe LED located next to the motor. Pressing the ‘#’ key again will disable emergency mode and the monorail will continue to travel again from when it had stopped.

4. Module Specification

4.1 Macros

Macros are generally used for LCD commands and displays. Other than that, it is used to set variables to a certain value before initialise an array.

4.2 Configuration Stage

4.1.1 Program Initialisation

As this is the start of the program, all initialisations are made here except for setting up interrupt modes, which are later set up in the emulation stage. This includes setting up hardwares as input or output, initialising the LCD, loading up flags (such as mode and stages), and initialising data segments.

To configure a pin connected to a hardware as *input*, DDRx and PORTxn is written as a logical 0 and 1 respectively, where *x* is the number letter for the port and *n* represents the bit number. This also activates the *pull-up resistor*. Hardwares such as PB0/1 are configured as an input.

To configure a pin connected to a hardware as *output*, DDRx and PORTxn is written as a logical 1 and 0 respectively, where *x* is the number letter for the port and *n* represents the bit number. This however will not turn on the device (as the output is driven *low*) unless a logical 1 is also written in PORTxn (causing the output to be driven *high*). Hardwares like LEDs, strobe light, and LCDs are configured as an output. Note that setting the LCD as output low or high will not affect its function.

DDRx	PORTxn	Hardware Configuration
0	1	Input (with pull-up resistors activated)
1	0	Output Low
1	1	Output High

The pins connected to the keyboard have the columns of the keypad set as output and rows as input.
To initialise the LCD to begin displaying, the following is done in the program:

Code	Description

`lcd_do_command 0b00111000`
`rcall sleep_5ms`
`lcd_do_command 0b00111000`
`rcall sleep_1ms`
`lcd_do_command 0b00111000`
 	
`lcd_do_command 0b00111000`
`lcd_do_command 0b00001000`
`lcd_do_command 0b00000001`
`lcd_do_command 0b00000110`
 	
`lcd_do_command 0b00001110`
	- Function set: 5 × 7 dots, 2 line display
	- Sleep for 5ms
	- Function set: 5 × 7 dots, 2 line display
	- Sleep for 1ms
	- Function set: 5 × 7 dots, 2 line display
	
	- Function set: 5 × 7 dots, 2 line display
	- Display OFF
	- Clear display
	- Entry mode set: Increment DD RAM address by 1, no display shift
	- Display ON: Cursor on, no blinking.

where *lcd_do_command* is the macro used for setting up the LCD and the binary numbers are used to set bits DB_7 - DB_0 in the LCD code.

RS	R/W	DB_7	DB_6	DB_5	DB_4	DB_3	DB_2	DB_1	DB_0

Note that 0 is written to RS and R/W bits throughout the initialisation.

All data segments in *monorail configuration* are set to 0, including *config_stage* (ENT_STATION = 0). *station* however has all the bytes initialised as a whitespace character ‘ ’. The mode is set to CONFIGURATION (0).

After everything is initialised, the *num_station_str* will be displayed on the LCD and users are now able to start inputting values.

4.2.2 Keypad Scanning

The keypad will always be scanning until a key is pressed, in which it will then jump to the stage it is up to. To begin, the columns are scanned by setting all column keys are set as output high except for the first column *C0*. The row pins are then read to see if any rows are grounded. If not, the program will continue by scanning the next column. After the last column *C3* is being scanned, the program will be configured to start scanning from *C0* again.

If a row was grounded, the program then finds the bit that was grounded.

By obtaining the row and column of the key that was pressed, the keypad will convert the key so that it can be used for the configuration stages.

4.2.3 Numeric Inputs using the Keypad

Applied to every stage except for stage 2 (entering station names), the program will record down the number user inputted. Hence, pressing keys *, and *A - D* will do nothing to the program. In order to convert user input in the program, the following calculations are done:

$$num = num \times 10 + key$$

where *num* will be the number stored into the byte of the data segment the program is up to.
For example, if the user pressed the key *1* and then *2*, *num* will contain the number 12.
As all numeric inputs must be less than 10 for the emulation to work, only up to 2 digits numbers will be recorded in *num*; any number keys pressed after will be rejected.

The program will behave as follows when a numeric input is required:

```
if (key < 10 && num < 10) {
    display key on LCD
    num = required_data  // from config data segment
    num = num * 10 + key
    required_data = num
}

return to keypad
```

4.2.4 Alphabetic Inputs using the Keypad

Applied exclusively for stage 2 (entering station names), the program will record down the letters user inputted. In this case, all keys are required except for keys *, *D*, and *1*, in which it will do nothing to the program.

Keys *A* - *C* are used as *offset keys*, in which they map the relative position of the letters on top of the letter/number keys (2 -9). Offset keys must be pressed before a letter/number key so that the program can figure out which letter is requested by user. A yellow LED will turn on to indicate that an offset key has been pressed. The yellow LED will turn off after a letter/number key is pressed, in result storing the letter requested by user.

Key 0 will store a whitespace character ‘ ’ into the byte the program is up to. However, when inputting station names, a letter must be stored first before a whitespace can be stored.

The program will behave as follows when an alphabetic input is required:

```
letter_offs = NO_OFFS_ENT
if (key == letter) {
    key -= ‘A’
    if (key < 3) {
        letter_offs = key
        turn on yellow LED
    }
} else if (key == number && key != 1) {
    if (key == 0) {
        store ‘ ’ into byte of required_data
    } else if (letter_offs != NO_OFFS_ENT) {
        letter = (key - 2) * 3 + letter_offs + ‘A’
        if (letter >= ‘Q’) letter++; // This is because ‘Q’ and ‘Z’ is not
                                     // labelled on the keypad

        display letter on LCD
        store letter into byte of required_data
        letter_offs = NO_OFFS_ENT
        turn off yellow LED
    }
}

return to keypad
```

4.2.5 End of Input

In any stage of configuration, the # key indicates end of input from the user. When this key is pressed, the program will check for any errors in the user’s input before continuing.

- In stage 1 (entering number of stations), the program can only continue to stage 2 if the number of stations is between 2 - 10, else the input is considered as an *incorrect input*[7].
- In stage 2 (entering stations’ name), the program will only continue to request input for other stations’ name only if the name of the current station is not an empty string, else the input is

[7] *Refer to Section 4.2.5 for Incorrect User Input*

considered as an *incorrect input*[8]. Once the name of the last station is recorded, the program will continue to stage 3.

- In stage 3 (entering travel time between adjacent stations), the program will only continue to request input for other travel times between adjacent stations only if the current travel time is between 1 - 10 seconds, else the input is considered as an *incorrect input*[8]. Once the time travelled between the last station and first station is recorded, the program will continue to stage 4.
- In stage 4 (entering monorail stop time), the program can only begin emulating the monorail if the monorail stop time is between 2 - 5 seconds, else the input is considered as an *incorrect input*[8].

4.2.5 Incorrect User Input

For any incorrect input, a red LED will turn on and user will have to re-enter the data. Only when the input is correct, the red LED will turn off and the user can continue inputting other data.

4.3 Emulation Stage

4.3.1 Emulation Initialisation

Here, any extra configuration was made, including configuration of ports, external interrupts are setup and any required registers or memory spaces are cleared to be used. The LEDs are made sure to be turned off also. For the entire emulation, interrupts were used for all implementation of the emulation and thus the global interrupt flag was enabled.

The external interrupts (INT0, INT1), for the push buttons were enabled with the use of the External Interrupt Mask Register (EIMSK) in which the corresponding bits were set to 1. These interrupts were set to be falling-edge triggered, as they are active-low, meaning that when pressed, the output is logic low. This interrupt sense control (ISC) is configured in External Interrupt Control Register A (EICRA).

ISCn1	ISCn0	Triggering in Generation of Interrupt
0	0	Low level
0	1	Any edge
1	0	Falling-edge
1	1	Rising-edge

In regards to the timer0 overflow interrupt, normal operation mode was set, configured in TCCR0A. The prescaler value was set to 1024 so that less interrupts were required to occur to achieve the desired time interval. In doing this, only one byte is required for the number of interrupts rather than 2 or more, depending on the prescaler value.

WGM2	WGM1	WGM0	Operation Mode	TOP	TOV Flag Set on
0	0	0	Normal	0xFF	0xFF
0	0	1	PWM, Phase Correct	0xFF	0X00
0	1	0	CTC	OCRA	0xFF

[8] *Refer to Section 4.2.5 for Incorrect User Input*

0	1	1	Fast PWM	0xFF	0xFF

In setting the clock prescaler value:

CS02	CS01	CS00	Prescaling value
0	0	1	No prescaling
0	1	0	8
0	1	1	64
1	0	0	256
1	0	1	1024

To enable the Overflow Interrupt, the Timer/Counter Interrupt Mask Register (TIMSK0) was configured,

OCIE0B	OCIE0A	TOIE0	Description
-	-	1	Timer/Counter0 overflow interrupt is enabled when the I-bit in the status register is enabled

Timer 3 was configured to be used for the motor in which fast PWM 8-bit mode was used in order to control the speed of the motor. To turn on the motor as the monorail automatically starts travelling, the constant for the motor to travel at approximately 60rps was stored into the compare register (OCR3BL). Because the timer was set to fast PWM 8-bit mode, non-inverting mode, the comparator only compares 8 bits of the compare register and the timer. This was configured in 2 registers, Timer/Counter 3 Control Registers A and B.

WGMn3	WGMn2	WGMn1	WGMn0	Operation Mode	TOP	Update of OCRnX	TOVn Flag Set on
0	1	0	1	Fast PWM, 8-bit	0x00FF	0X0000	TOP

COMnB1	COMnB0	Description
1	0	Non-inverting mode, clear output on compare match and set output at BOTTOM of OCRnx (0x0000)

4.3.2 External Interrupts

The external interrupts only occur when the push buttons have been pressed, however switch debouncing issues arise due to the mechanical springiness of the buttons. In order to resolve this, a 135ms delay (button_delay) was used to delay multiple interrupts being generated from one push from the user. Another method in resolving the issue was introducing a flag (button_flag) in which it must be cleared in order for the interrupt to occur, else the flag will be cleared for the next external interrupt to occur. If the flag was cleared when the interrupt occurred, the flag was set to 1 so that another interrupt cannot occur immediately after.

Both external interrupts essentially caused the same thing to happen, making the monorail stop at the next station in which next_station was set to 1 and an LED turned on, indicating that the user had pressed the button.

This was implemented as follows:

```
call button_delay function (wait for 135ms)
// Check button_flag
if (button_flag != 0) {
    clear button_flag
    end interrupt
}
button_flag = 1
// Check current mode
if (mode == configuration) clear button_flag
} else { // in emulation mode
    Turn on LED
    next_stop = 1
}
end interrupt
```

4.3.3 Timer0 Overflow Interrupt

In the timer0 overflow interrupt, the entire emulation implementation takes place, based on the current mode of the monorail (travel, stop, emergency) and the remaining time until a change will occur, whether that change is continuing to travel through the next station, stopping at the next station or starting to travel again. The first thing checked is if it is currently in emergency mode, this is because nothing should continue if in emergency mode and thus if it is, the interrupt will end. Otherwise, the number of interrupts to occur for the desired time interval, stored in (t0_int) is decremented.

Once this value has reached 0, meaning the desired time interval has passed (e.g. 1 second), the timer register is decremented. This timer register stores the current time to be going through, this would either be the travel time from one station to the next while travelling, or the stopping time at a station. After decrementing, the interrupt would end and continue through this until the timer reaches 0.

Once the timer reaches 0, the system will undergo at least some change. Here, the current mode must be checked. If it is currently in stop mode, it will need to load up everything needed to start travelling to the next station. If it is in travel mode, it can either continue to travel or stop at the station it has just reached, in order to check this, next_stop is checked. This is always cleared unless a push button had been pressed, in which next_stop will be set to 1. If this value is still cleared, then the system will continue to travel to the next station, essentially doing the same thing as if the train had just stopped and is about to start travelling to the next station. Else, the monorail will now need to stop. In this case, the motor needs to turn off, the timer register must be updated to the stop time and the lights will need to blink at 3Hz, blinking 3 times per second. The next_stop variable also needs to be cleared.

When the monorail starts travelling to the next station, whether it had just finished stopping or the monorail is just continuing through the route without stopping, a few things need to be set up. The current station position needs to be incremented, the LEDs must be turned off and the motor needs to be on. In order to display the next station name on the LCD, the current station position is used to navigate through the array of station names (station). It must be checked if the monorail is currently back at the first station in which the travel time to load must be reset to the beginning of the array with the travel times. Another special

case that must be checked is if the monorail is currently at the last station before looping back to the start, in which it must display the first station name and thus the pointer must be reset to the start.

The logic used to display the next station was to use the x pointer registers to point at the start of the array with the station names, station. The variable curr_station was copied to another register and incremented to not tamper with the position value as it is used for locating the next travel time. This register was multiplied by 10 as that was the maximum string length for all stations, giving a value of (curr_station+1)*10. This was then added to the x pointer to find the start of the next station name string to display.

This logic was implemented as follows:

```
    // Check current emul_stage (monorail mode)
    if (emul_stage == MONO_EMERG) end interrupt

    // else emul_stage == (MONO_TRAVEL || MONO_STOP)
    dec t0_int
    if (t0_int != 0) end interrupt

    // Else, time interval has passed
    dec timer // (current travel time/stop time to pass)
    if (timer != 0) { // timer will continue to count down through interrupt
        if (emul_stage == MONO_TRAVEL) {
            t0_int = ONE_SEC (number of interrupts to occur in 1s)
        } else { // emul_stage == MONO_STOP
            Blink lights
            t0_int = SIX_HZ (to continue to achieve 3 blinks per s)
        }
    } else { // timer == 0, monorail will go through a change
        // Check mode
        if (emul_stage==MONO_STOP) || (emul_stage== MONO_TRAVEL && next_stop==0) {
            // (Load next travel state)
            Turn off LEDs
            next_stop = 0
            emul_stage = MONO_TRAVEL
            Turn on motor to spin at 60 rps (store constant MOT_60RPS in OCR3BL)
            inc curr_station // (current position)
            t0_int = ONE_SEC // Load next travel time in timer register
            if (curr_station == num_station) { // (num_station = total stations)
                curr_station = 0
                load in first travel time in timer register
                load next station name string to display
            } else if (curr_station == num_station - 1) // (at last station)
                load in first station name string to display
            } else { // (emul_stage == MONO_TRAVEL && next_stop == 1)
                emul_stage = MONO_STOP
                Turn off motor, (clear OCR3BL)
                t0_int = SIX_HZ
                Load up monorail stop time into timer
                Multiply timer by 6, since new interrupt number is for 1/6s
                Next_stop = 0
           }
```

```
}

end interrupt
```