COMP2121 Lift Emulator Design Manual

By Mitchell John (z3418315), Ian Wong (z3463696)

Introduction

This document details the design of the lift emulator system. There are four components to this document: System Flow Diagram, Module Specification, Algorithms, and Data Structures.

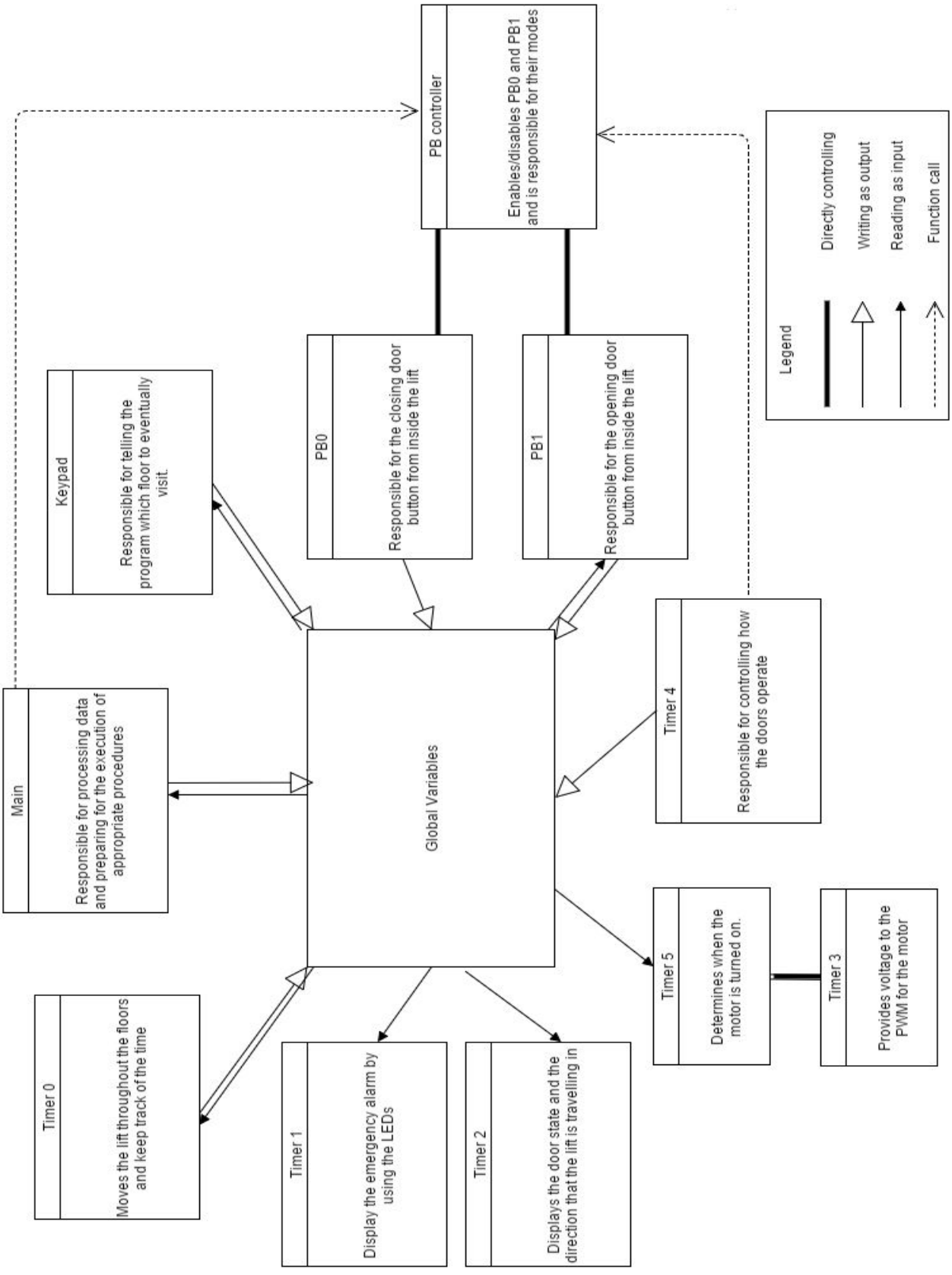
System Flow Diagram outlines how the system was implemented - in particular how each system component interacts with each other and their flow in order to emulate the lift behaviour.

Module Specification shows in more detail how each system component functions, and their interaction with other specific components. In a way, more specific information regarding the flow between each system components is provided. Some of the things discussed are the module responsibilities, and what variables are involved.

Algorithms will describe the main algorithms used in this lift emulator. Namely, processing and ordering of requested floors, execution of different timers, push buttons debouncing, keypad reading and debouncing, and displaying LED's.

Data Structures contains information on where and how data is processed. Namely, registers, data memory, and use of stack. Decisions of their use will involve brief discussions due to limitations in the AVR memory capabilities.

System Flow Diagram



MODULE SPECIFICATION

The following details the responsibilities, interactions, and processes that pertain to each of the system components described previously in the system flowchart. Note: All timers throw an interrupt when their timer counter overflows.

Main

'Main' is the brain of the system - it keeps track and displays the current floor, updates the destination floor, determines whether the lift should be stationary or move in a particular direction, and determines mode of lift operation (normal, or emergency). To fulfil its role, 'Main' needs to make decisions (by reading variables) and manipulate variables prior to the appropriate procedure.

Keypad

The keypad is responsible for checking for any keypresses, and if there are, acknowledging the request to visit a floor (if floor numbers are pressed), or request for emergency (if '*' was pressed). When a floor is requested to be visited, keypad checks whether the floor has been requested previously, before setting the corresponding index in the array of flags "floor_array".

Timer 0

'Timer0' is responsible for moving the lift through the floors, by keeping track of time. Once the duration to reach a floor has been elapsed, it will set a flag indicating the floor has changed (which is to be acknowledged and reset by 'Main'). It uses "timer0_TimeCounter" as a local variable to keep track of the time progressed. It will only start timing when the lift is in motion AND if a floor change hasn't been requested, so it must check both conditions prior to carrying out its procedure. 'Timer0' has a prescaling of CLK/8.

Timer 1

'Timer1' is responsible for displaying the emergency alarm signal through the LED's. It does so by checking whether the emergency alarm flag is set, prior to displaying the alarm pattern, and flashing the appropriate strobe LED lights (see "Algorithms" section for more detail with displaying LED's). It uses "timer1_TimeCounter" as a local variable to keep track of the timing. 'Timer1' has a prescaling of CLK/8.

Timer 2

'Timer2' is responsible for displaying the state of the doors and the direction the lift is travelling. It takes in only two inputs from the set of global variables: "door_state" and "lift_direction". Based on their values, it loads two patterns into two local variables, which are then displayed through the LED's. When door is opened/closed, the strobe LED lights turn on. For closing/opening, the strobe LED lights flash (see "Algorithms" section for more detail with displaying LED's). 'Timer2' has a prescaling of CLK/8, and uses “timer2_TimeCounter” to keep track of the timing.

Timer4

'Timer4' is responsible for carrying out the "stop at floor" procedure, by keeping track of the time elapsed and counting the progress. This procedure is only executed if there is a "stop at floor" request, indicated by the flag "stop_at_floor". The time is tracked using the local variable "timer4_TimeCounter", and the progress is managed using another local variable "stop_at_floor_progress". Depending on the progress, 'Timer4' changes the door state ("door_state" in global variables). 'Timer4' can also recognise and accept any open/close door requests (this is described in further detail in “Algorithms”). Timer4 has a prescaling of CLK/8.

Timers 3 and 5

These two timers form the unit for controlling the motor. 'Timer3' is responsible for generating a PWM waveform to control the motor, whilst 'Timer5' is responsible for turning the motor on/off, depending on the value in "door_state" from global variables. 'Timer5' is able to manipulate the motor by changing the value in Timer3's 16-bit output compare register B (OCR3B). 'Timer3' is configured to be in fast PWM mode.

Push buttons

PB0 is responsible for triggering a "close door" request when pushed", and PB1 is responsible for triggering an "open door" request when pushed/held. Both involve changing the value of the global variable "door_state_change_request", which can then read by the 'Timer4' component. PB1 is slightly different - it could also send a “stop at floor” request (instead of a “door open” request), in cases where there is no current “stop at floor” procedure being carried out.

Push Button (PB) Controller

The ‘PB Controller’ is a method responsible for enabling or disabling the push buttons after checking the global variables “lift direction” and “door state”. It disables/enables and changes the modes of the push buttons by directly altering the EIMSK register, and the EICRA register.

DATA STRUCTURES

Registers

Register	Alias	Description
R0	return_register	Contains the return value after a function has executed. Usually moved to another register, such as “temp1”, for faster processing.
R1-R7	--	*Unused*
R8	parameter_register	Contains the parameter whose value is processed by a function. Usually loaded from another register, such as “temp1” (r24), in those situations where immediate values must be loaded.
R9-R15	--	*Unused*
R16	current_floor	Contains the current floor the lift is on. Initialised to 0 in the beginning and will update when the lift moves to a different floor. This register’s value is displayed on the LCD such that the user knows where they are at that moment. This register is only used by ‘Main’
R17	final_dest	Contains the floor that the lift is currently travelling to. It is initialised to -1 so that it doesn’t interfere with any of the floor numbers. The lift will only start moving when final destination is set to one of the floor numbers. ‘Main’ updates this register from the queue for when it needs a new value for its destination. Floor between current floor and destination floor can still be visited, provided they are set to true in the corresponding element of the floor array.
R18	lift_direction	Stores the direction that the floor is currently travelling in, or whether the lift is stationary. This register may only contain the values -1, 0 or 1 where -1 represents that the lift is travelling downwards, 0 represents that the lift is stationary and 1 represents that the lift is travelling upwards. It is set through the function “set_lift_direction”, and checked by other system components (such as Timer0).
R19	door_state	Stores the state that the door is currently in. The register is allowed to hold only 4 values, either 0 (door is closed), 1 (door is currently opening), 2 (door is opened) and 3 (door is currently closing). This register is manipulated through ‘Timer4’, where the “stop at floor” procedure requires changing the door state.
R20	row	Contains the current row that is being scanned for a keypress on the keypad. Used only when keypresses are being polled.
R21	col	Contains the current current column that is being scanned for a

		keypress on the keypad. Used only when keypresses are being polled.
R22	rowmask	Allows the detection of a keypress in a particular row, by performing a logical 'AND' with the value through Port L.
R23	colmask	Allows the detection of a keypress in a particular column, by performing a logical 'AND' with the value through Port L.
R24	temp1	A general purpose register used to be processed and loaded effiicienty. It can be used in conjunction with "temp2" (r25) to represent a 16-bit value - mainly in the context of timers whose overflows require 16-bits.
R25	temp2	A general purpose register used to be processed and loaded effiicienty. It can be used in conjunction with "temp1" (r24) to represent a 16-bit value - mainly in the context of timers whose overflows require 16-bits.
R26-R31	--	*general purpose registers*

Note: All registers can be cleared on initialisation, except the following: door_state (R16), current_floor (), final_dest (), and lift_direction(). This is to preserve proper functionality of the lift emulator.

Other variables in data memory

Some variables were stored in data memory instead of the general purpose registers since we are very limited to the number of registers that we are able to utilise.

Label Name	Size (byte)	Description
floor_array	10	Array of flags used to keep track of which floors are requested to be visited
floor_queue	10	Queue used to keep track of which floors are requested to be visited
queue_start	1	Where the queue begins
queue_end	1	Where the queue ends
floor_changed	1	Flag used to indicate that floor has changed
stop_at_floor	1	Flag used to indicate a "stop at current floor" request

stop_at_floor_progress	1	Progress value of “stop at floor” procedure
door_state_change_request	1	Used to indicate whether there is a “door open” or “door close” request
emegency_flag	1	Indicate whether emergency mode was requested
emergency_alarm	1	Indicate whether emergency alarm is triggered
final_dest_saved	1	Indicate whether a destination floor was preserved prior to going to emergency floor
LED_lift_direction_output	1	Used to store the pattern describing the lift direction
LED_door_state_output	1	Used to indicate the door state (closed, opening, opened, closed)
LED_emergency_alarm_output	1	Used to indicate the emergency alarm display
timer0_TimeCounter	2	Used to store the number of timer0 overflows
timer1_TimeCounter	1	Used to store the number of timer1 overflows
timer2_TimeCounter	2	Used to store the number of timer2 overflows
timer4_TimeCounter	1	Used to store the number of timer4 overflows
pb0_button_pushed	1	Used to indicate the PB0 interrupt has been executed
pb1_button_pushed	1	Used to indicate the PB1 interrupt has been executed
oldCol	1	Stores the column location of the previous key pressed. Used together with oldRow to deduce the previous key pressed.
oldRow	1	Stores the column location of the previous key pressed. Used together with oldCol to deduce the previous key pressed.

Note: All data in dseg are (or can be) cleared on initialisation, except the following: floor_changed, stop_at_floor, emergfency_flag, emergency_alarm, and final_dest_saved. This is to preserve proper functionality of the lift emulator.

X Register (R27:R26)
This register is used exclusively to load particular values from the floor array into registers using the command "**ld temp1, X**". Since the floor levels could be variable, the location of the array element is also variable, and must first be calculated (with floor level as the offset from the array's starting address). The final address is eventually stored into X, and the **ld** command can be carried out. This is useful for checking whether the floor is requested to be visited (in which case, the loaded value from address X would be true), and also for setting that particular floor value to be false in the array.

Y Register (R29:R28)
This register was used for clearing 2-byte values located in dseg. It is mainly used to clear the "TimeCounter" variables of Timers 0 and 2 that represent number of overflows, so timing can be done again.

Array of flags

This array is a 10 celled array where each cell represents each floor that the lift is able to travel to. The purpose of this array is to keep track of which floors have been requested to be visited. When the requested floor is visited, the main program checks to see if the cell for this floor in the array is checked and then will stop at the floor. After the floor has been checked the array cell should be cleared which indicates the floor has been visited. It is consulted when 'Main' is trying to find a new destination floor.

Queue

The purpose of the queue is that it stores the key presses from the keypad. Whenever an appropriate button is pressed on the keypad (except the star key) the main function checks to see if the floor is already checked in the array of flags. If the floor is already checked in the array then the main function won't put the floor in the queue since it has already been requested to be visited. It is used by 'Main' when The relevant algorithm is described under "Algorithms".

Stack usage

The stack is used extensively by the system as a method of preserving data, and is initialised when the system is RESET Prior to procedures executed by functions and interrupts, registers that would be used have their values pushed onto the stack, and popped off for them to be restored. This allows the same registers to be used by other components whilst their original data can be retained. Prior to the lift's emergency

mode, the destination floor (if defined) is stored onto the stack so that it can be set again once emergency mode has exited.

Algorithms

Travelling to floors

First a destination floor must be set. ‘Main’ will recognise this, and set the direction in which the lift must move. At this point, ‘Timer0’ will notice that the lift is moving, and so starts timing. Once the duration for travel time between a single floor has been elapsed, ‘Timer0’ will request a floor change. This is read by ‘Main’, which will then clear the floor change request, and update the current floor in the direction it’s travelling. This repeats until current floor is equal to destination floor, then the “stop at floor” procedure is carried out - ‘Main’ sets the “stop_at_floor” flag, which is realised by ‘Timer4’ who carries out the procedure.

Processing and ordering of requested floors

When no destination floor is set (destination floor is undefined), ‘Main’ will recognise this, and call a function to update the queue. What this will do is try and find the first item from the queue that is valid as a final destination (valid means that floor has not yet been visited). This involves constantly removing the first item from the queue, checking whether the corresponding index in the floor array is set to true - if it’s true, then the destination floor is set to that item, otherwise it will ignore it, and attempt to remove the first item from queue again. If the queue is empty, it will return an undefined floor value.

At this point, ‘Main’ checks whether the destination floor is set. If it’s still not set after updating the queue, then no floors have been queued, and the lift can remain idle. Otherwise ‘Main’ will proceed to travel to that floor.

Timing with timers

All timers that require timing do so using a counter that records the number of timer overflows that have occurred. This counter is a local variable for that timer. Prescaling is configured so that the number of overflows that have occurred will correspond to a particular amount of time (this number of overflows is determined theoretically, and is an approximate estimation). Once the number of overflows is reached, the counter is reset.

Push buttons debouncing

The procedures in the push buttons are triggered by low signals. When a button is pushed, it will recognise that low signal, wait for a small duration (using the function

delay), and check again whether the signal is still low. If it's still low, then the button can be safely assumed to be pushed by the user, and the procedure can be carried out.

For unknown reasons, the push buttons on the provided AVR board send a second interrupt after the first. To reliably deal with this, a flag in dseg called "pbn_button_pushed" is always checked before carrying out the interrupt. If the flag is set, then the second interrupt is being executed, which will set the flag back to false and exit immediately. Note that this procedure may not be necessary for other boards.

Keypad reading and debouncing

In normal mode, the keypad polling procedure involves scanning each column and row sequentially for a low value (a key is identified by its column/row combination). First it checks the column - a low signal is indicated when the result of a logical "AND" operation is carried out, between the input from PORTL and the column mask, is equal to 0xF. Then each row is examined. A logical "AND" is performed between PINL and the row mask - if the result is 0, then something has been pressed in that row. At this point, the key can be identified. If there is no presses in a particular row, it will increment the row and change the mask appropriately. If there are no low-value in a column, it will increment to the next column and change the mask appropriately.

In emergency mode, the keyboard polling will only poll the '*' button, which involves the use of the special polling masks "0xEF" and "0x08". The procedure for detecting whether the key is pressed is the same process described previously (normal mode).

For debouncing, the keypad only accepts the input if it is an input that is different to what was pressed before, whether it be nothing or a different key. The previous key is stored as a location into two variables named "oldCol" and "oldRow", which are checked immediately after the signal that a key has been pressed - if they're the same, then ignore. If they're different, then accept. This approach supports recording SINGLE keypresses reliably.

Processing a "door state change" request

Prior to processing the request, Timer4 will check the current door state (through the value in “door_state”) in order to determine when the request can be carried out.

In the case of a “close door” request: If door is opening, the request is preserved (until the point where the doors are opened). If door is opened, the request is accepted, and the “stop at floor” progress is set at the point where the door begins to close, of which Timer4 will react to by closing the door. If door is closed, then the request is simply cleared.

In the case of an “open door” request: If door is opening, then the request is simply accepted (ie cleared). If door is opened, then Timer4 will change the “stop at floor” progress to a point where the door is opened but not yet closing. If door is closing, the progress is set to opening, and Timer4’s “TimeCounter” variable is set to a value such that the opening duration is the same as how much the door has closed.

Displaying LED’s

There is a variable in dseg that contains the pattern to output using the LED's. When the pattern is flashed through LED's (eg door is closing) the pattern is constantly being initialised and reset (to zero). When pattern is moving (eg displaying lift direction), the pattern is shifted (using lsr or lsl), and once it gets to 0, becomes reset to the appropriate pattern.

APPENDIX

Data Dictionary

Name	Type	Location	Description
current_floor	8-bit integer	R16	Used to store the lift's current floor level
col	8-bit integer	R21	Used to store the current column number. Used in keypad scanning
colmask	8-bit integer	R23	Mask for current column during keypad scan
door_state	8-bit integer	R19	Indicate the state of the doors. Takes on 4 values: 0 (closed), 1 (opening), 2 (opened), 3 (closed)
door_state_change_request	8-bit integer	dseg	Used to describe whether there are any requests for opening or closing the door. Takes 3 values: 0 (no request), 1 (close request), 2 (open request)
emergency_alarm	flag	dseg	Trigger the emergency alarm
emergency_flag	flag	dseg	Request for emergency mode
final_dest	8-bit integer	R17	Used to store the destination floor that the lift is moving towards
final_dest_saved	flag	dseg	Indicate that a destination floor has been preserved
floor_array	array of flags	dseg	Contains requests to visit floors. eg if index 5 is set, then floor 5 has been requested to be visited
floor_changed	flag	dseg	Indicate whether the floor level has changed
floor_queue	queue of integers	dseg	Contains the floors to be visited in a particular order
LED_door_state_output	8-bit integer	dseg	LED pattern for the door state component
LED_lift_direction_output	8-bit integer	dseg	LED pattern for the lift direction component
LED_emergency_alarm_output	8-bit integer	dseg	LED pattern
lift_direction	8-bit integer	R18	Used to indicate the direction lift is moving. Takes on 3 values: -1 (down), 0 (stationary), 1 (up)
oldCol	8-bit integer	dseg	Used for keypad debouncing
oldRow	8-bit integer	dseg	Used for keypad debouncing

parameter_register	general purpose	R8	General register used to store parameter values to function calls
pb0_button_pushed	flag	dseg	Indicates whether pb0 button was pushed. Used for debouncing.
pb1_button_pushed	flag	dseg	Indicates whether pb1 button was pushed. Used for debouncing.
queue_end	8-bit integer	dseg	An offset value indicating the end position in the floor_queue
queue_start	8-bit integer	dseg	An offset value indicating the start position in the floor_queue
return_register	general purpose	R0	General register used to store return values from function calls
row	8-bit integer	R20	Contains the current row number. Used in keypad scanning
rowmask	8-bit integer	R22	Mask for current row during keypad scan
stop_at_floor	flag	dseg	Used to indicate a "stop at current floor" request
stop_at_floor_progress	8-bit integer	dseg	Used to keep track of the different stages within the "stop at current floor" procedure
temp1	general purpose	R24	Temporary register used for general processing. Sometimes used with temp2 for 16-bit processing.
temp2	general purpose	R25	Temporary register used for general processing. Sometimes used with temp1 for 16-bit processing.
timer0_TimeCounter	8-bit integer	dseg	Used to count number of timer 0 overflows
timer1_TimeCounter	8-bit integer	dseg	Used to count number of timer 1 overflows
timer2_TimeCounter	16-bit integer	dseg	Used to count number of timer 2 overflows
timer4_TimeCounter	8-bit integer	dseg	Used to count number of timer 4 overflows