COMP9032 Assignment

Overview

Our group was tasked with developing a simulation system for the Atmel ATmega2560 microprocessor board, emulating the control of a drone to search an accident scene in a mountainous area.

This project had many complexities including: handling the hardware and related interfacing; designing software to interact with the hardware; and working collaboratively across different locations, different operating systems, and with a limited number of microprocessors.

Project timeline

The timeline of our project was as follows:

- **(25/10)**: Discussed the assignment specification and confirmed scope with lab tutors
- **(27/10)**: Had a group call with the team to brainstorm assignment approach and ideas
- **(30/10 - 12/11)**: Developed code
    - **(31/10)**: Added LCD helper functions and code to display map rows and columns, drone attributes (speed, state, position)
    - **(03/11)**: Added timer interrupt for drone movement
    - **(03/11)**: Performed major code refactoring to split codebase across multiple files
    - **(05/11)**: Added push button interrupts, speed handling, and crash detection
    - **(06/11)**: Added visibility and accident detection, and hover logic
    - **(08/11)**: Performed bug fixes and edge case handling
- **(09/11)**: Had a group call to align on completed work and communicate how the code works
- **(10/11)**: Commenced writing report
- **(15/11)**: Demonstrated board and final simulation in lab time
- **(16/11)**: Finished writing report

Communication strategy

A major challenge of group projects, especially software development projects, is

communication. Our group maintained clear communication by:

- Using early weekly labs to confirm specifications and scope with lab tutors
- Using weekly labs to check in with lab tutors to ensure that our work remained aligned with the project goal
- Using Discord messaging to communicate with group members to ensure that work was being completed correctly and on time, and to provide a platform for group members to collaborate with each other (e.g. ask for help/advice)
- Using Discord calls to check in with group member work progress, and have knowledge share sessions to ensure that all group members understand the entire code
- Using Git version control (branching, PRs, PR reviews) to facilitate software development collaboration
- Separating project tasks into functions and macros such that each member can work on tasks without conflicts or double ups

Contributions

Hongfei

- Git setup
- Code modularisation
- Timer interrupt setup
- Movement logic
- Crash/detecting accident logic

Luke

- Code skeleton
- LCD setup
- Display logic
- Terrain setup

Tina

- Code reviewer
- Software and hardware support
- External interrupt setup
- Speed modification logic
- LED logic

Alan

- Project management
- Keypad setup

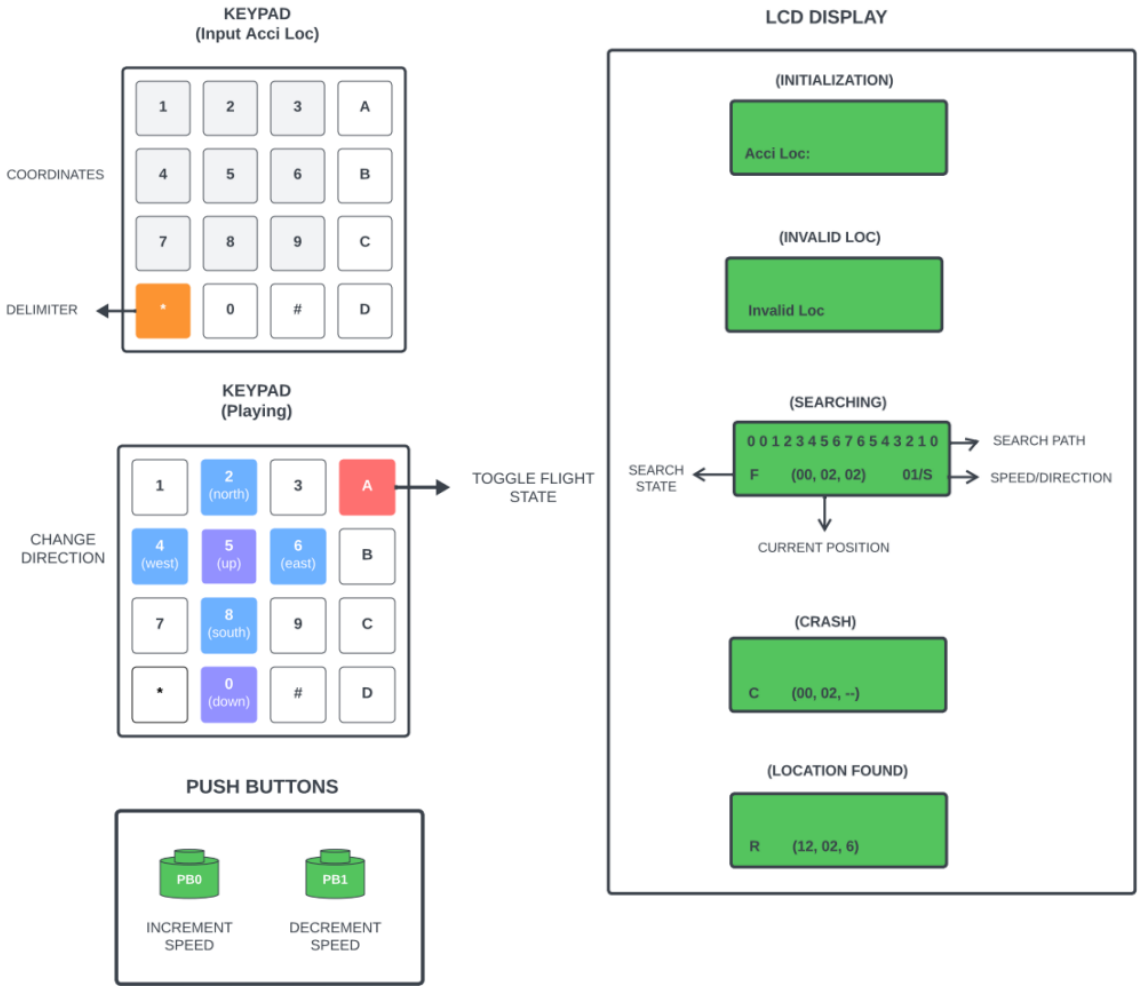
- Accident location setup and input validation
- Input handling and logic

Hardware components and interfacing design

AVR ATmega2560 Microprocessor:

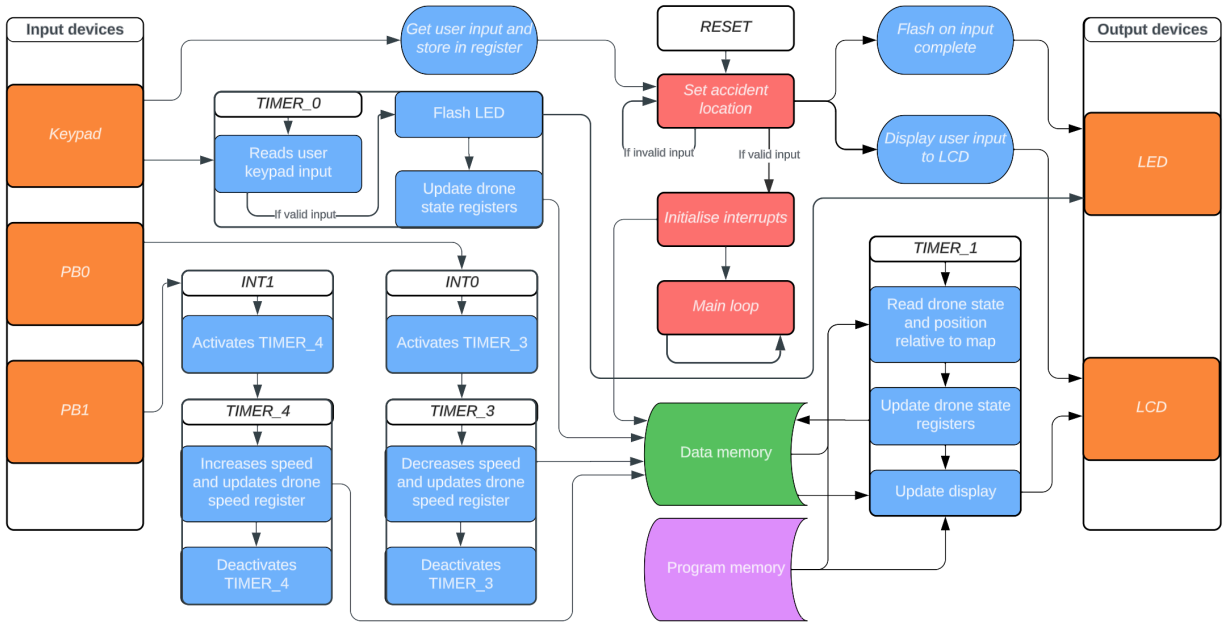
Hardware Component	Direction	AVR Port (Pins)
Keypad	Input	Port L (PL0-PL7)
PB0 button	Input	Port D (RDX3)
PB1 button	Input	Port D (RDX4)
RESET button	Input	N/A
LCD	Output	Port F (PF0-PF7)
LED	Output	Port C (PC0-PC7)

Componenets and interfaces:



Software code structure and execution flow

Execution flow:



General design choices:

- Code modularised into multiple files that are loaded in the `main.asm` file
    - Separate files for definition of macros, functions, and variables
    - Split by I/O device
- Macros capitalised, prefixed with `M_`, and written in snake case as per convention

File structure:

- `bcd.asm`:
    - Defines the `display_decimal` function which uses the double dabble algorithm to convert a binary number to decimal and displays it on the LCD
- `keypad_defs.asm`:
    - Defines variables and values required for operating the keypad
- `keypad_functions.asm`:
    - Defines the `scan_key_pad` function which scans the keypad for input and saves the result to a data register
- `keypad_macros.asm`:

    - Defines the `M_KEYPAD_INIT` macro for setting up the keypad to receive input
    - Defines the `M_MULT_TEN` macro for correctly storing multi digit keypad inputs

- `lcd_defs.asm`:
    - Defines variables and values required for operating the LCD

- `lcd_functions.asm`:
    - Defines the `lcd_command` function which accepts commands to be sent to the LCD
    - Defines the `lcd_data` function which accepts data to be displayed on the LCD
    - Defines the `lcd_wait_busy` function which ensures correct timing and independent execution of each operation

- `lcd_macros.asm`:
    - Defines the `M_DO_LCD_COMMAND` macro which uses the `lcd_command` function to send a command to the LCD
    - Defines the `M_DO_LCD_DATA` macro which uses the `lcd_data` function to display data on the LCD
    - Defines the `M_CLEAR_LCD` macro which clears the LCD
    - Defines the `M_LCD_SET_CURSOR_OFFSET` macro which sets the cursor to current drone position on the path
    - Defines the `M_LCD_SET_CURSOR_TO_SECOND_LINE_START` macro which sets the cursor to the start of the second line on the LCD
    - Defines the `M_LCD_SET_CURSOR_TO_FIRST_LINE_START` macro which sets the cursor to the start of the first line on the LCD
    - Defines the `M_LCD_INIT` macro which initialises the LCD

- `led_bar_functions.asm`
    - Defines the `flash_three_times` function which causes the lights on the LED bar to flash three times successively

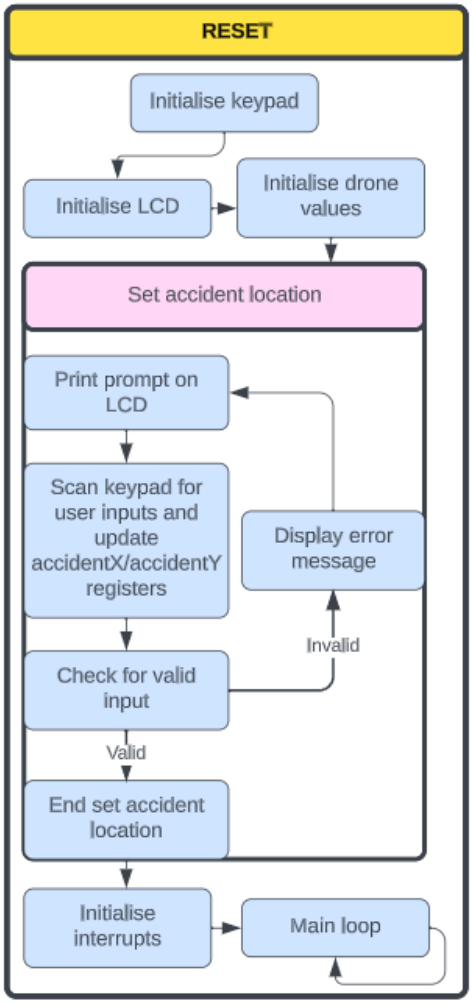
- `sleep_functions.asm`
    - Defines various functions to add delays at various lengths

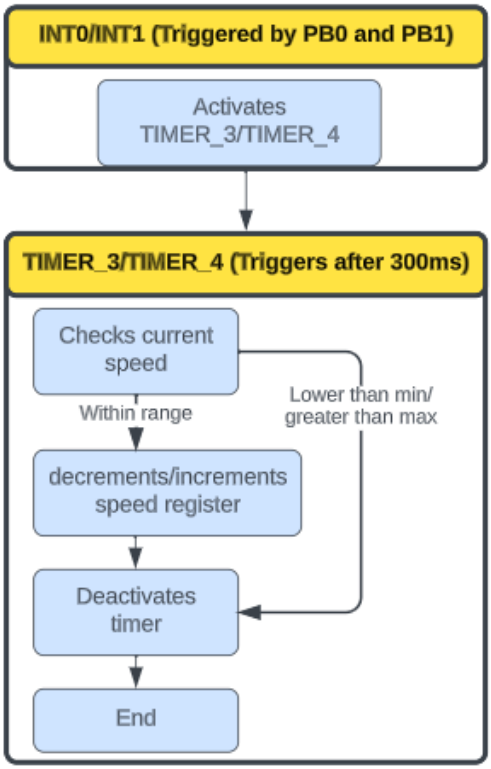
- `main.asm`
    - Uses all of the aforementioned files to execute the main body of code to satisfy the assessment criteria

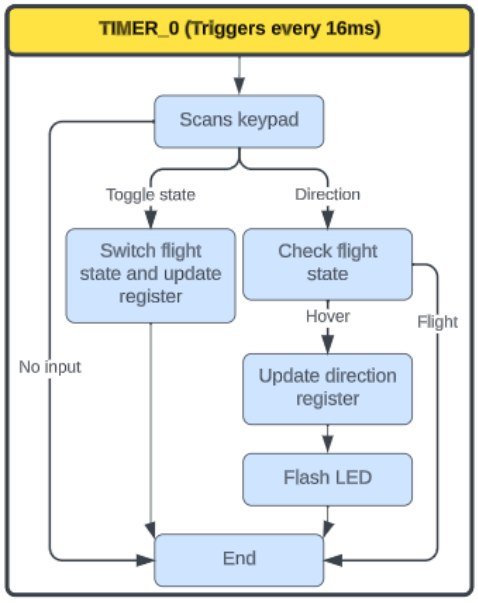
Interrupts:

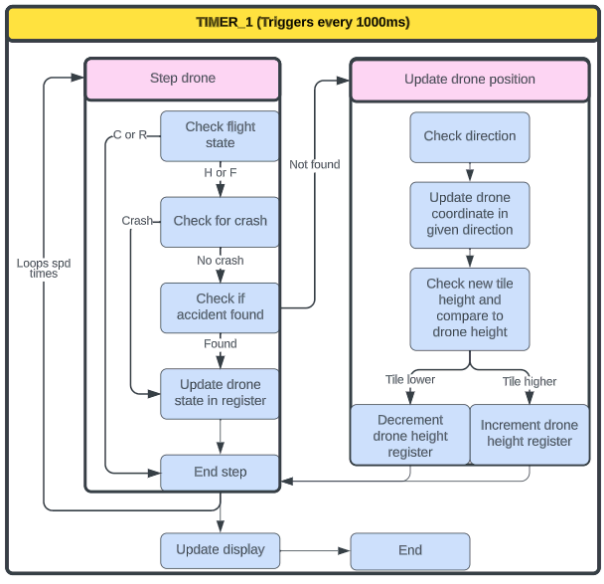
- `RESET`
  - Triggered by RESET button
  - Reset all pins and I/O devices and re-initialise starting values (e.g. drone position)
- `EXT_INT0`
  - Triggered by the `PB0` button
  - Decreases drone speed by activating `TIMER_3_COMPA_VECT`
- `EXT_INT1`
  - Triggered by the `PB1` button
  - Increases drone speed by activating `TIMER_4_COMPA_VECT`
- `TIMER_0_COMPA_VECT`
  - Timer interrupt that happens every 16ms
  - Polls the keypad to check if any key has been pressed
  - On valid input, processes user input and updates drone state accordingly
- `TIMER_1_COMPA_VECT`
  - Timer interrupt that happens every 1000ms
  - Movement handler which updates the drone position based on the values in the drone state registers
  - Updates LCD based on drone state
- `TIMER_3_COMPA_VECT`
  - Timer interrupt activated by `EXT_INT0`
  - Timer stops after 300ms and decreases speed
  - Acts as a "debouncer" for when the `PB0` button is pressed
- `TIMER_4_COMPA_VECT`
  - Timer interrupt activated by `EXT_INT1`
  - Timer stops after 300ms and increases speed
  - Acts as a "debouncer" for when the `PB1` button is pressed

Interrupt logic:









Software and hardware interaction

The software and hardware interacts as per the following:

1. A map of the terrain is stored into program memory via software

2. Upon simulation start, the software triggers the LCD to display a message stored in program memory, prompting the user to input an accident location. Subsequent inputs to the keypad are detected and the software displays them onto the LCD.
3. The software checks whether the inputted accident location is valid, and if valid the software triggers the LED bar to flash three times and the simulation begins. If not valid, the software triggers an error message to be displayed onto the LCD, and the user is prompted to input an accident location again
4. The software displays the drones current row/column onto the top row of the LCD, as well as its position in that row/column. The software displays the current state of the drone as well as its direction and speed onto the bottom row of the LCD. All of this information is retrieved from predefined drone state registers
5. The `TIMER_0_COMPA_VECT` timer interrupt polls the keypad every 16ms to see if there is any user input
6. If there is user input, the software triggers the LED bar to flash three times; and converts the inputs into instructions for the drone (e.g. change direction or flight status) which are stored into predefined drone state data registers
7. If a user presses the `PB0` push button the `EXT_INT0` interrupt is triggered, which will trigger the `TIMER_3_COMPA_VECT` interrupt to act as a "doebouncer", and then will decrement the value in the drone speed data register
8. If a user presses the `PB1` push button the `EXT_INT1` interrupt is triggered, which will trigger the `TIMER_4_COMPA_VECT` interrupt to act as a "debouncer", and then will increment the value in the drone speed data register
9. The `TIMER_1_COMPA_VECT` timer interrupt handles drone movement. It processes a number of drone steps based on the drone's current speed. If the drone is flying, it will move the drone to the next tile along the path and automatically makes small adjustments to drone height according to the terrain. Each drone step will also check whether the drone has crashed or found the accident in both hover mode, and in flight mode after the movement step has been processed. Finding the accident is based on the drone's vertical visibility which is set as a variable. Once the drone steps have been processed, the LCD is updated accordingly
10. If the drone crashes or finds the accident, the simulation ends and it will be displayed on the LCD. If the drone finds the accident, the LED bar will also flash 3 times.

Concluding remarks

We succeeded in completing the assignment specification, and in successfully demonstrating our working solution to the lab tutors.

We learnt a number of skills in the project, including:

- Writing AVR Assembly code
- Using an Atmel ATMega2560 microprocessor board

- Project management (including version control)
- Teamwork and communication