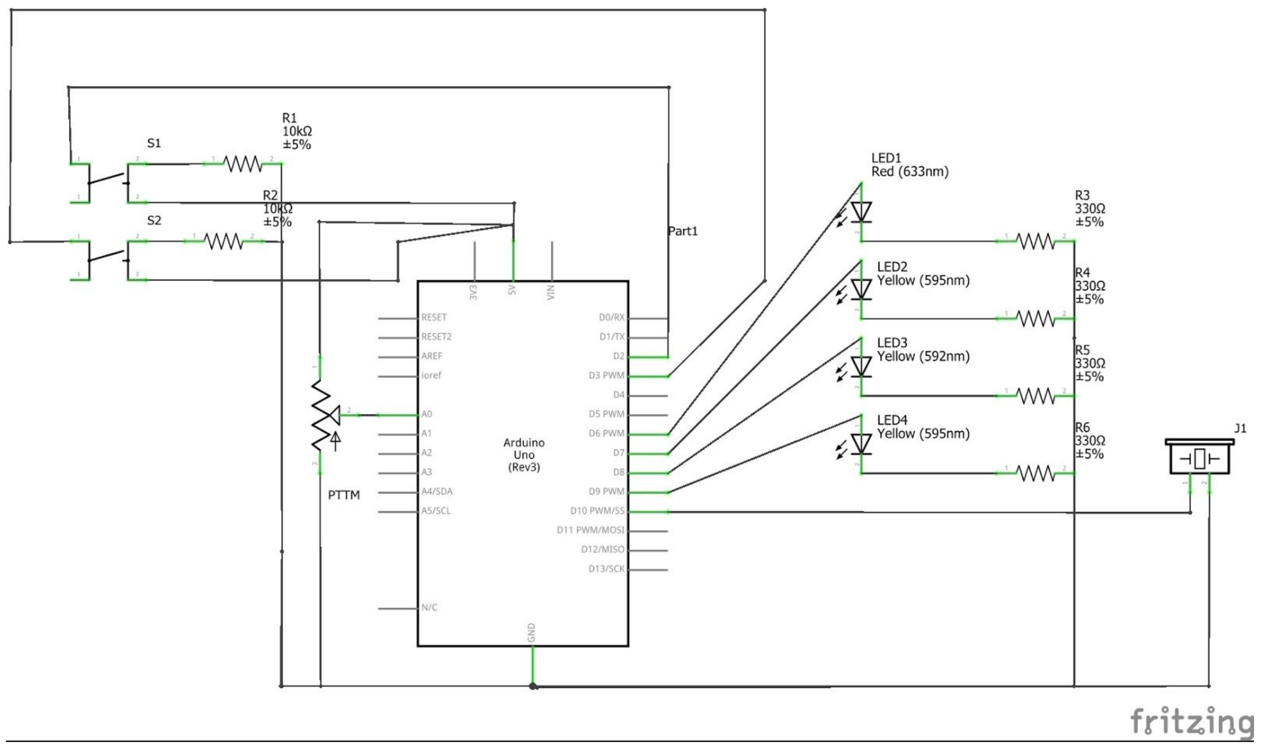
CG2271 LAB 6 REPORT

Automated Driving System

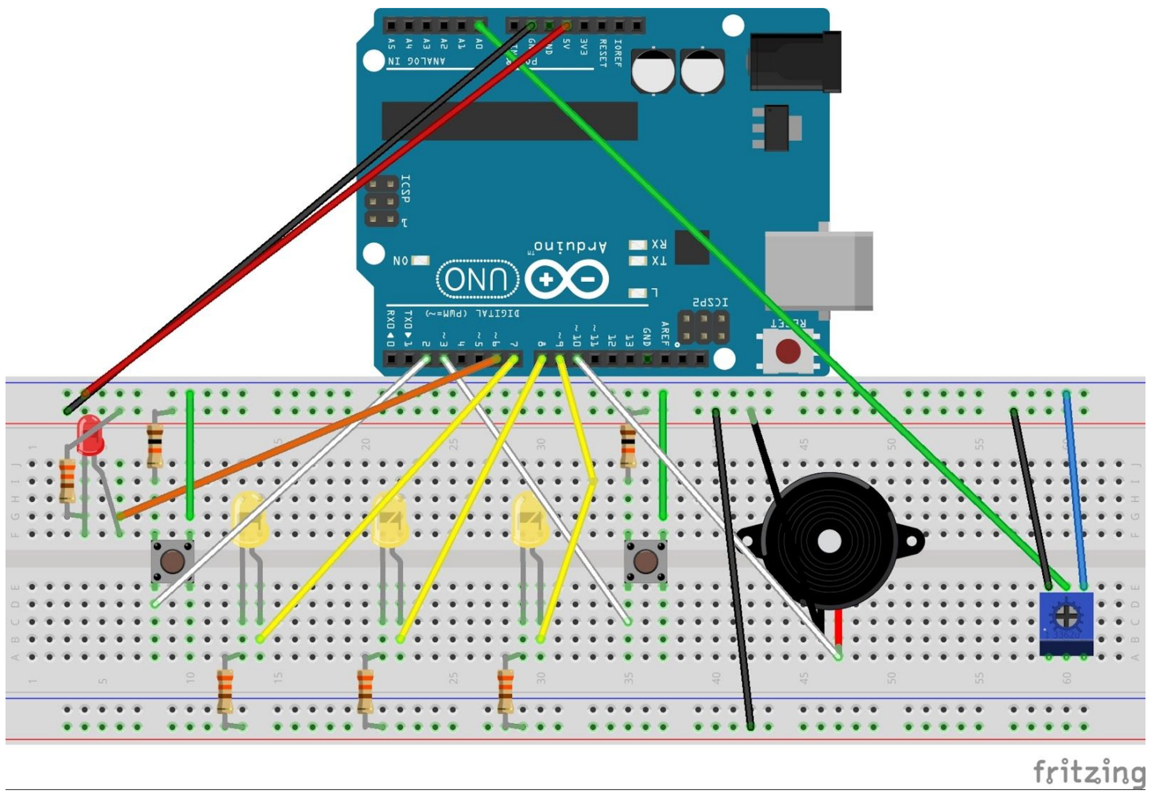
Students	:	PANKAJ BHOOTRA	A0144919W
		CHAN JUN XUN	A0139313L
Lab Group	:	01	
Teaching Assistant	:	NISHANT SHYAMAL BUDHDEV	

We understand what plagiarism is and have ensured we did not plagiarise for this assignment. This assignment is in partial fulfilment of the requirements for the module CG2271 Real Time Operating Systems.

Circuit in Schematic view:-



Circuit in Breadboard view:-



<u>Software Architecture</u>:-

This system is designed to simulate an automatic moving vehicle with safety brakes. The program is setup with 3 RTOS tasks and 7 message queues. Every 0.5s, the potentiometer value is read, and is printed on the UART terminal as the distance between the vehicle and the next vehicle in front, along with the current set speed and the speed desired by the driver. On the user front, the speed is signified by the number of Yellow LEDs that are turned ON; 1 = speed 1, 2 = speed 2, etc. Along with the speed, a Piezo speaker is used as a buzzer that echoes the speed increment with increasing volume. Two pushbuttons are being utilised with interrupt service routines for each to increase or decrease speed respectively. The data such as distance, current speed and driver desired speed are communicated amongst different tasks with the help of message queues. To adjust values of local variables, their pointers are passed to the helper methods (as is the case for currentSpeed and desiredSpeed variables, whose pointers are passed to increaseSpeed and decreaseSpeed helper methods).

To simulate the braking system, a Red LED is used as an indicator. If the distance is decreased or the current speed is increased beyond the safe speed limit for the current distance, then the automated braking system ensures that the speed is decreased to maintain safe driving. In this scenario, the Red LED is also turned ON for one second to indicate that automated safety feature was activated.

<u>Constants used</u>:-

Chosen range for potentiometer/distance: 0-1023.
Value of D taken: 256.
Stack Size for tasks: 100.
Queue Size for message queues: 2.

<u>Helper methods used</u>:-

1. **playBuzzer:** This helper method outputs sound of a certain volume to the speaker. The volume is proportional to the current speed of the car.

2. **lightLEDs:** This helper method switches on the Yellow LEDs corresponding to the current speed. [O: On, X: Off]

    Level 1: O X X
    Level 2: O O X
    Level 3: O O O
    else X X X

3. **increaseSpeed:** This helper method increases the speed of the car by 1 unit. It also takes in a safety variable, which is the safe speed for the current distance. For speed to be increased, it checks if the automated safety feature is turned off, and the speed is still below 3. If the conditions are achieved, both the current speed and the driver desired speed are increased by 1 unit. If automatedSafetyFeature is turned ON, the speed is adjusted to the maximum safe speed for the current distance. This condition enforces that the speed cannot be increased beyond an acceptable value for the current distance. It also calls the playBuzzer and lightLEDs method to denote a change in speed on the user front by adjusting the number of Yellow LEDs turned ON and the volume of the speaker.

4. **decreaseSpeed:** This method decreases the speed of the car. No safety checks are required as it is decreasing the speed. As long as the speed of the car is not 0, it will decrease. Both the driver desired speed and the current speed are decreased by 1 unit. It also calls the playBuzzer and lightLEDs method to denote a change in speed on the user front by adjusting the number of Yellow LEDs turned ON and the volume of the speaker.

5. **getSafeSpeed:** Returns the safe speed for a given value of distance. Distance can range from 0 to 1023.

Role of ISRs:-

1. **btn1ISR:** This ISR is called when the increase pushbutton is pressed. Debouncing is used for button consistency, with a delay of 250 ms. It also passes the value 1 into the checkSpeedQueue to denote that the increase pushbutton was pressed. This message queue is used to wake up the checkSpeedTask. If the value 1 is received in checkSpeedTask, then checkSpeedTask performs increase of speed.

2. **btn2ISR:** This ISR is called when the decrease pushbutton is pressed. Debouncing is used for button consistency, with a delay of 250 ms. It also passes the value 2 into the checkSpeedQueue to denote that the decrease pushbutton was pressed. This message queue is used to wake up the checkSpeedTask. If the value 2 is received in checkSpeedTask, then checkSpeedTask performs decrease of speed.

Role of Tasks:-

1. **readDistanceTask**: This is a periodic task that executes every 500 ms. This task reads the distance from the potentiometer and updates the braking system as the value of the potentiometer changes. A variable distance is created to store the potentiometer value. The task then checks if the current speed is greater than the safe speed for current distance, to determine if there is a need to reduce the speed. If so, the speed is set to the safe speed value and the Red LED is turned ON for 1 second to indicate that the automated safety feature was activated. This

task also checks if the current speed is less than equal to the safe speed for a given distance and in that case, it updates the driver desired speed to the current speed. For both cases, the values in the current speed and driver desired speed message queues are also overwritten with new values to update the queues. The same is done for distance value as well. Finally, the task uses vTaskDelayUntil to delay the task by 500 ms, so as to execute periodically every 500 ms.

2. **checkSpeedTask**: This is a periodic task that executes every 1000 ms. This task utilises the ISRs in the code to determine an increase or decrease in speed. A variable flag is used to receive the value from checkSpeedQueue. Depending on the flag value received from the checkSpeedQueue (1 or 2), the increase speed or decrease speed function is called respectively. It then does a check if speed was increased and the current speed became higher than the safe speed. In that case, the Red LED is turned ON and the current speed is adjusted to the safe speed. The values for current speed and driver desired speed are then updated in the relevant message queues by overwriting with the new values. Finally, the task uses vTaskDelayUntil to delay the task by 1000 ms, so as to execute periodically every 1000 ms. Following this, the Red LED, irrespective of its state, is turned OFF, so as to ensure that if the Red LED lights up, it stays ON for exactly 1 second.

3. **printTask**: This is a periodic task that executes every 500 ms. This task receives the values of distance, current speed and driver desired speed from the relevant message queues and prints them on the UART/Serial Monitor. The task uses vTaskDelayUntil to delay the task by 500 ms, so as to execute periodically every 500 ms.

Role of Message Queues:-

1. **distanceQueue:** Communicates distance value from readDistanceTask (sender) to printTask (receiver).

2. **driverDesiredSpeedQueue:** Communicates driver desired speed value from checkSpeedTask (sender) to printTask (receiver).

3. **currentSpeedQueue:** Communicates current speed value from checkSpeedTask (sender) to printTask (receiver).

4. **checkSpeedQueue:** Communicates whether the increase pushbutton was pressed (value 1) or decrease pushbutton was pressed (value 2) from the ISRs of the pushbuttons (senders) to the checkSpeedTask (receiver).

5. **distQueue:** Communicates distance from readDistanceTask (sender) to checkSpeedTask (receiver) for use in the logic component of the code.

6. **ddsQueue:** Communicates driver desired speed between readDistanceTask and checkSpeedTask. This value may be overwritten/updated in either of the tasks if the driver desired speed is updated based on conditions described earlier.

7. **csQueue:** Communciates current speed between readDistanceTask and checkSpeedTask. This value may be overwritten/updated in either of the tasks if the current speed is updated based on conditions described earlier.

<u>Periods/Priorities of Tasks</u>:-

1. **readDistanceTask:** 500 ms, 1
2. **checkSpeedTask:** 1000 ms, 1
3. **printTask:** 500 ms, 1

Hence, all tasks are periodic and have the same priority of 1.

<u>Scheduling Policy</u>:-

0.5s: readDistanceTask and printTask are called.

↓

1s: readDistanceTask, printTask and checkSpeedTask are called.

System repeats indefinitely.

---END OF REPORT---