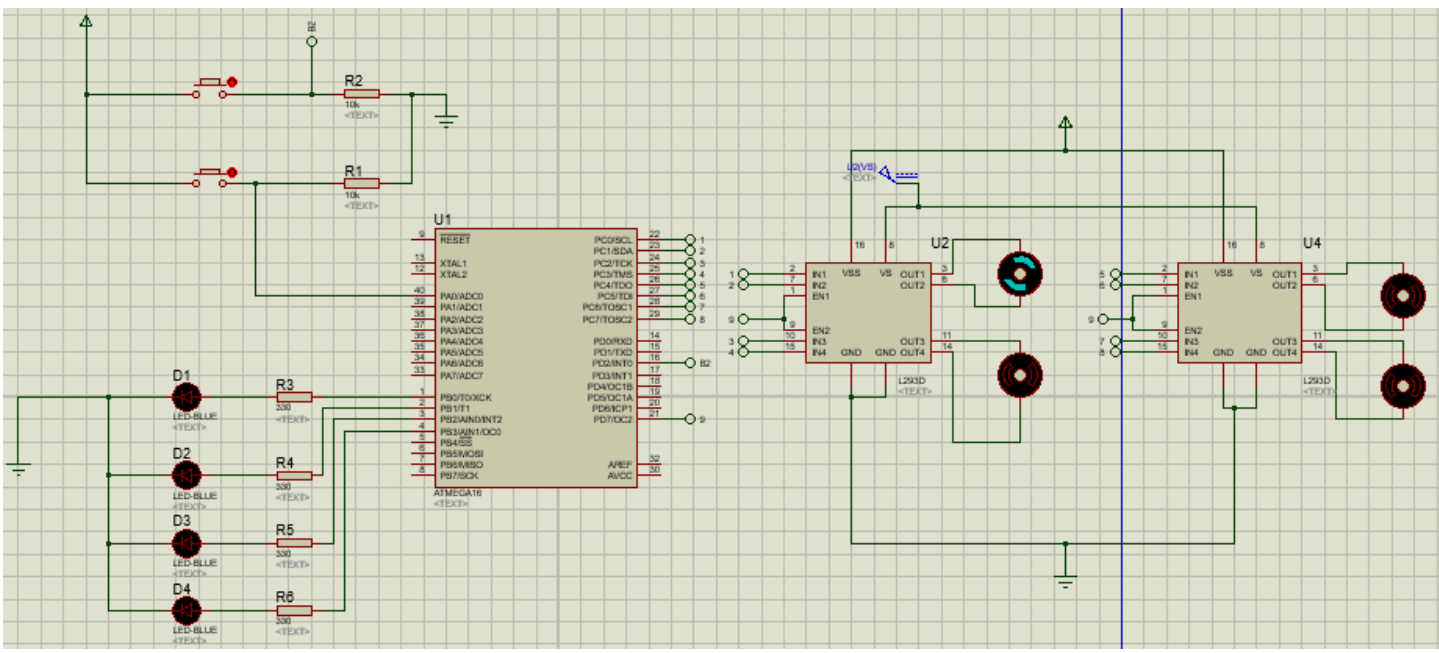
Design document

project title: Moving Car project

Name: Hazem Ashraf

Project Description

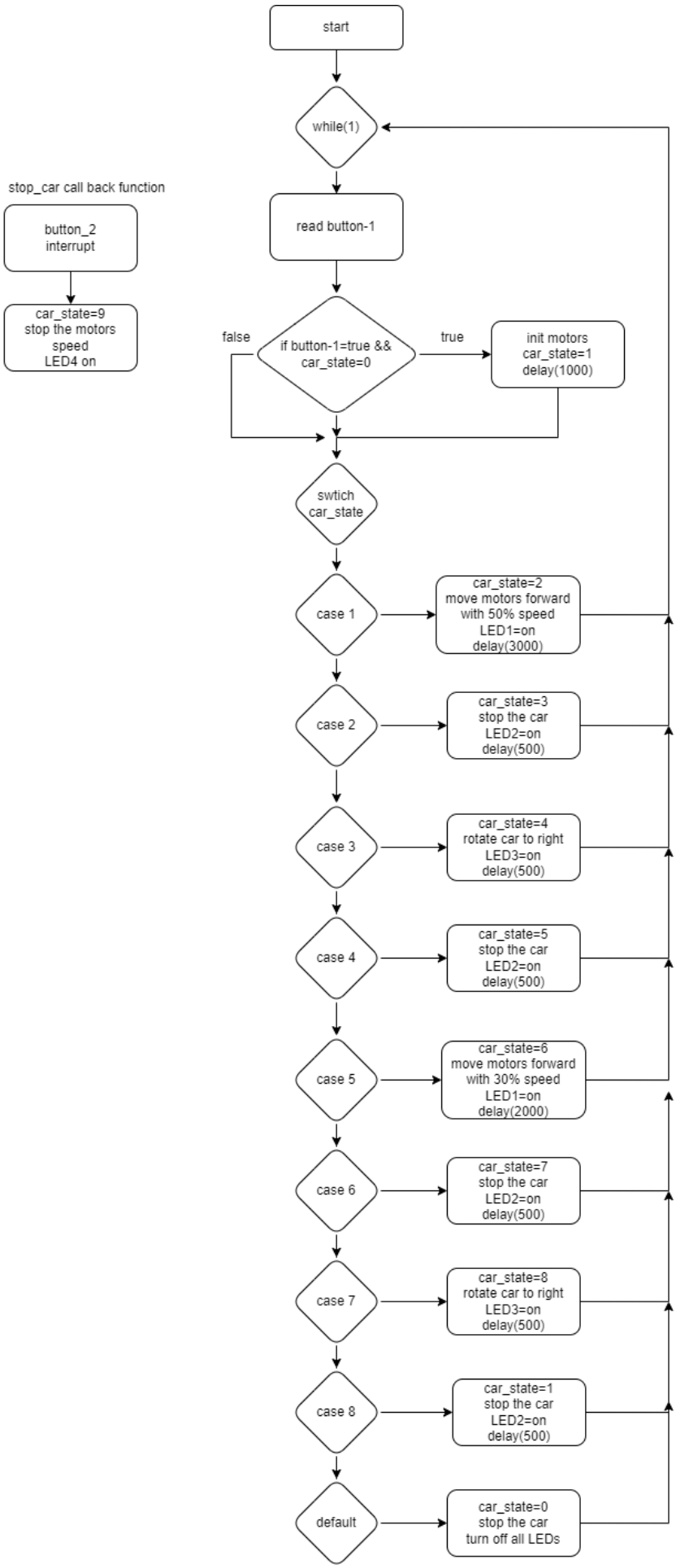
1. **Car Components**:
    1. **Four** motors (**M1**, **M2**, **M3**, **M4**)
    2. **One** button to start (**PB1**)
    3. **One** button for stop (**PB2**)
    4. **Four** LEDs (**LED1**, **LED2**, **LED3**, **LED4**)
2. **System Requirements**:
    1. The car **starts initially** from **0 speed**
    2. When **PB1** is **pressed**, the car will **move forward after 1 second**
    3. The car will move forward to **create the longest side of the rectangle for 3 seconds with 50% of its maximum speed**
    4. After finishing the first longest side the car will **stop for 0.5 seconds**, **rotate 90 degrees to the right**, and **stop for 0.5 second**
    5. The car will move to **create the short side** of the rectangle at **30% of its speed for 2 seconds**
    6. After finishing the shortest side, the car will stop for **0.5 seconds**, **rotate 90 degrees to the right**, and **stop for 0.5 second**
    7. Steps **3 to 6** will be **repeated infinitely** until you press the **stop button (PB2)**
    8. **PB2** acts as a **sudden break**, and it has the highest priority
    9. **LEDs Operations**
        1. **LED1:** On means moving forward on the long side
        2. **LED2:** On means moving forward on the short side
        3. **LED3:** On means stop
        4. **LED4:** On means Rotating



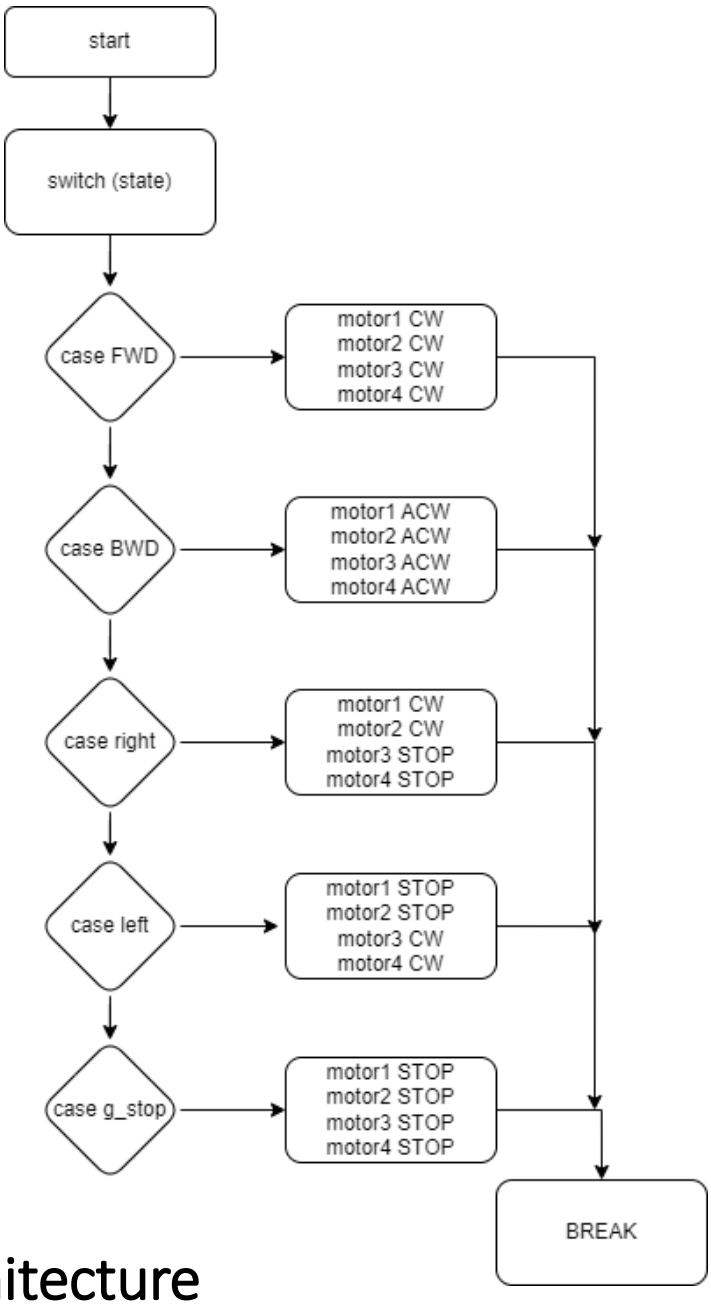
Circuit wiring

Project flowchart diagram

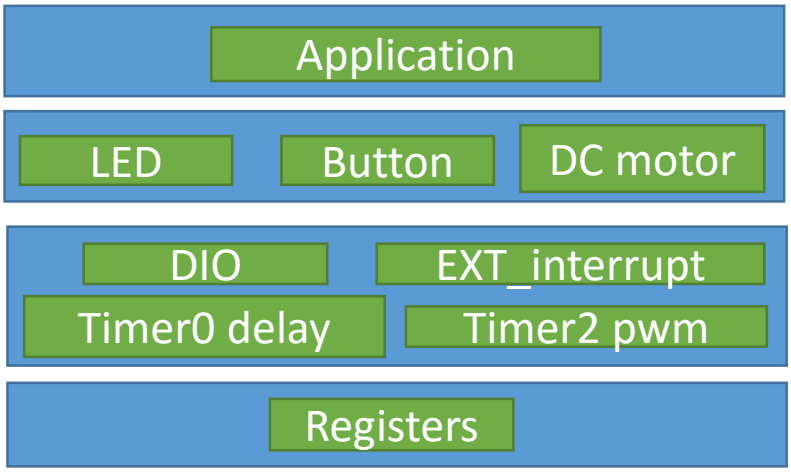
Main.c



```
void _4_motors_control(DCmotor_configtype
m1,DCmotor_configtype m2,DCmotor_configtype
m3,DCmotor_configtype m4,EN_state state,uint8 speed)
```



Layered architecture



Project Modules APIs

- GPIO module APIs

/*============= TYPE DEFINITION =============*/

typedef enum{
PIN_INPUT,PIN_OUTPUT
}EN_PIN_DIRECTION;

typedef enum{
Low,High
}EN_PIN_VALUE;

typedef enum{
FAILED,SUCCESS
}EN_STATE;

/*============= FUNCTION PROTOTYPE =============*/

EN_STATE **DIO_pinDirection(uint8 port,uint8 pin,EN_PIN_DIRECTION direction);**
EN_STATE **DIO_writePin(uint8 port,uint8 pin,EN_PIN_VALUE value);**
EN_STATE **DIO_readPin(uint8 port,uint8 pin,uint8* value);**
EN_STATE **DIO_togglePin(uint8 port,uint8 pin);**
void check_pin(uint8* pin_no,uint8* port);

/*============= FUNCTION PROTOTYPE=============*/

EN_STATE **DIO_pinDirection(uint8 port,uint8 pin,EN_PIN_DIRECTION direction);**

Description:

- **DIO_pinDirection**:used to set pin direction input/output
- function parameters
- Port: port ID number
- pin: pin number to set
- direction: direction of the pin
- Return success pin number is in the range, FAILED if pin number out of the range

EN_STATE **DIO_writePin(uint8 port,uint8 pin,EN_PIN_VALUE value);**

Description:

- **DIO_writePin:** used to write high/low to specific pin
- function parameters
- Port: port ID number
- pin: pin number to write
- value:output value high / low
- Return success pin number is in the range, FAILED if pin number out of the range

Project Modules APIs

- GPIO module APIs

EN_STATE **DIO_readPin(uint8 port,uint8 pin,uint8* value);**

Description:

- **DIO_readPin: used** to read specific pin value
- function parameters
- Port: port ID number
- pin: pin number to set
- value: address to variable of the return reading
- Return success pin number is in the range, FAILED if pin number out of the range

EN_STATE **DIO_togglePin(uint8 port,uint8 pin);**

Description:

- **DIO_togglePin:** used to toggle specific pin
- function parameters
- Port: port ID number
- pin: pin number to toggle
- Return success pin number is in the range, FAILED if pin number out of the range

void check_pin(uint8* pin_no,uint8* port);

Description:

- **check_pin:** used to check port ID for specific pin
- function parameters
- Port: port ID number
- pin: pin number
- Return success pin number is in the range, FAILED if pin number out of the range

Project Modules APIs

- EXT-interrupt module APIs

/*============= TYPE DEFINITION =============*/

typedef enum{
EN_INT0,EN_INT1,EN_INT2
}EN_INT_source;

typedef enum{
LOW_LEVEL,ANY_CHANGE,FALLING,RISING
}EN_INT_TRIGGER;

typedef enum{
INT_FAILED,INT_SUCCESS
}EN_INT_error;

typedef struct{
EN_INT_source source;
EN_INT_TRIGGER trigger;
}ST_INT_Config;

#define INT0_pin 2 //PD2
#define INT1_pin 3 //PD3
#define INT2_pin 3 //PB2

/*============= FUNCTION PROTOTYPE =============*/

EN_INT_error **INT_init(ST_INT_Config* Int_config)**

Description

- INT_init: used to initialize the interrupt by:
- disable global interrupt
- enable external interrupt source and set pin to input
- set external interrupt trigger signal type
- enable global interrupt
- Function parameters
- Int_config: pointer to structure of ST_INT_Config
- Return : FAILED if passing parameters is not correct, SUCCESS if the passing parameters is correct

void INT0_setCallBack(void(*a_ptr)(void));

Description:

INT0_setCallBack:used to set call back function for external INT_0

void INT1_setCallBack(void(*a_ptr)(void));

Description:

INT1_setCallBack:used to set call back function for external INT_1

Project Modules APIs

- EXT-interrupt module APIs

void INT2_setCallBack(void(*a_ptr)(void));

Description:

INT2_setCallBack:used to set call back function for external INT_2

void INT_Deinit(ST_INT_Config* Int_config);

Description

- INT_init: used to initialize the interrupt by:
- Disable specific external interrupt source

Project Modules APIs

- Timer0_delay module APIs (TIMER_0.h)

/*============= TYPE DEFINITION =============*/

```
typedef struct{
   float delay;
   uint16 prescaler;
   uint8 init_value;
   float NO_OF_OV;
}ST_timer0_config;
```

Description:

the structure is used to implement delay object, to define delay variable:

ST_timer0_config delay_on={100};

The remaining members don't care about initialization

/*============= MACRO DEFINITION =============*/

```
#define TCCR0 (*((volatile uint8*)0x53))
#define TCNT0 (*((volatile uint8*)0x52))
#define OCR0  (*((volatile uint8*)0x5C))
#define TIFR  (*((volatile uint8*)0x58))
#define TIMSK (*((volatile uint8*)0x59))
//TCCR0 timer counter control register
#define CS00  0
#define CS01  1
#define CS02  2
#define WGM01 3
#define COM00 4
#define COM01 5
#define WGM00 6
#define FOC0  7
//TIMSK interrupt mask register
#define TOIE0  0
#define OCIE0  1
#define TOIE1  2
#define OCIE1B 3
#define OCIE1A 4
#define TICIE1 5
#define TOIE2  6
#define OCIE2  7
//TIFR interrupt flag register
#define TOV0  0
#define OCF0  1
#define TOV1  2
#define OCF1B 3
#define OCF1A 4
#define ICF1  5
#define TOV2  6
#define OCF2  7
```

Project Modules APIs

- Timer0_delay module APIs (TIMER0_Utilities.h)

/*=============MACRO DEFINITION =============*/

```
#define max_count 256
#define min_count  1
#define init_value(T_max,T_delay,tick)  (((float)T_max-T_delay)/tick)

//pre_scaler values for TIMER0
#define N0    0
#define N1    1
#define N8    8
#define N64   64
#define N256  256
#define N1024 1024

//T_max in (ms) delay for each pre_scaler
#define Tmax_N1    0.26F
#define Tmax_N8    2.05F
#define Tmax_N64   16.38F
#define Tmax_N256  65.54F
#define Tmax_N1024 262.14F

//T_min in (ms) delay for each pre_scaler
#define Tmin_N1    0.001F
#define Tmin_N8    0.008F
#define Tmin_N64   0.064F
#define Tmin_N256  0.256F
#define Tmin_N1024 1.024F
```

Timer0_delay module APIs (TIMER_0.h)

/*============= FUNCTION PROTOTYPE =============*/

void Timer0_Delay(float delay);

Description:

- used to apply delay using polling technique
- it convert number of overflows to integer number to implement the required delay correctly
- example: if number of overflows=3.8
- mean perform 3 overflows and calculate the remaining time to complete the delay

Project Modules APIs

Timer2_PWM module APIs (Timer2_PWM.h)

/*============= FUNCTION PROTOTYPE =============*/

void PWM_Timer2_Init(void);

Description:

Used to initialize PWM mode to generate 500HZ for motor speed control

void PWM_Timer2_Start(uint8 duty_cycle);

Description:

Used to start generating PWM signal by enable the clock source

void PWM_Timer2_Stop(void);

Description:

Used to stop generating PWM signal by disable the clock source

Project Modules APIs

❑ LED module APIs

/*============= MACRO DEFINITION =============*/

#define LED_logic 1 //1:positive logic , 2:negative logic

Description : Macro used to configure LED logic connection

/*============= FUNCTION PROTOTYPE =============*/

EN_STATE **LED_init(uint8 led);**

Description:

- LED_init: used to initialize LED direction and initial value for the pin
- function parameters
- Led: pin number to be set
- Return success pin number is in the range, FAILED if pin number out of the range

EN_STATE **LED_digitalwrite(uint8 led,EN_PIN_VALUE value);**

Description

- LED_digitalwrite**:** used to write high/low to specific led
- function parameters
- Led: pin number to be set
- Value: led high/low
- Return success pin number is in the range, FAILED if pin number out of the range

❑ BUTTON module APIs

/*============= TYPE DEFINITION =============*/

typedef enum{

disable,enable

}EN_internal_pullup;

/*============= FUNCTION PROTOTYPE =============*/

EN_STATE Button_init(uint8 pin,EN_internal_pullup state);

Description

- Button_init: used to initialize BUTTON direction and set internal pullup resistor
- function parameters
- pin: pin number to be set
- State: to disable/enable internal pullup resistor
- Return success pin number is in the range, FAILED if pin number out of the range

EN_STATE **Button_Read(uint8 pin,uint8* value)**

Description

- Button_Read: used to read button state high/low
- function parameters
- pin: pin number to read
- Return success pin number is in the range, FAILED if pin number out of the range

Project Modules APIs

❑ DC motor module APIs

/*============= TYPE DEFINITION =============*/

typedef struct{

uint8 input_pin_1;

uint8 input_pin_2;

}DCmotor_configtype;

typedef enum{

CW,ACW,stop

}DCMotor_state;

/*============= FUNCTION PROTOTYPE =============*/

EN_STATE **DCmotor_init(DCmotor_configtype *motor);**

Description:

Used to initialize motor direction and set to low as initial value

Enable PWM module

Return success pin number is in the range, FAILED if pin number out of the range

void DCmotor_start(DCmotor_configtype *motor,DCMotor_state State,uint8 speed);

Description:

Used to control motor direction and speed using PWM

void DCmotor_stop(DCmotor_configtype *motor);

Description:

Used to stop the motor by disable PWM module and stop motor rotating