DriveSim

Driver Simulator for
Next Generation Driver Training

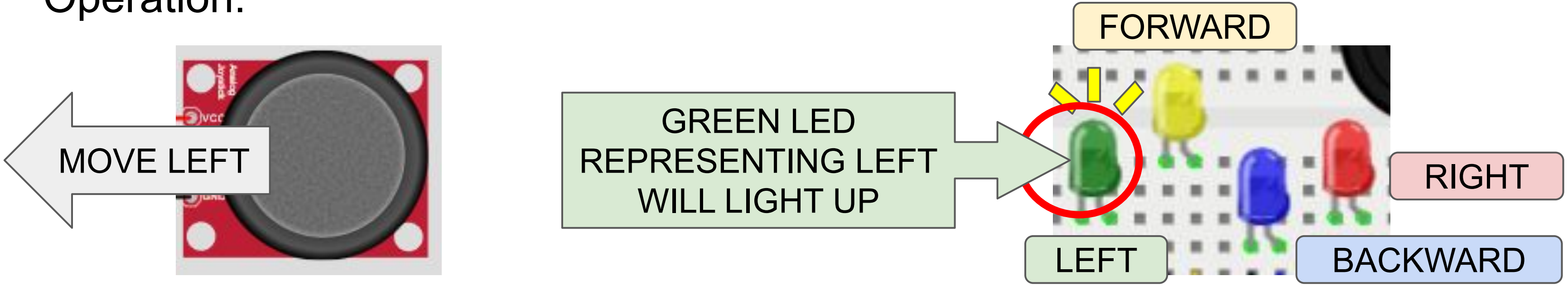
~ Group 8 ~

Project Description (Overall System)

3 SYSTEMS:

VEHICLE STEERING SYSTEM | HORN SYSTEM | EMERGENCY BRAKING SYSTEM

1. **VEHICLE STEERING SYSTEM** (JOYSTICK = STEERING DEVICE)
   a. Components: KY-023 Joystick Module, LED
   b. Operation:



MOVE LEFT

GREEN LED REPRESENTING LEFT WILL LIGHT UP

FORWARD

RIGHT

LEFT

BACKWARD

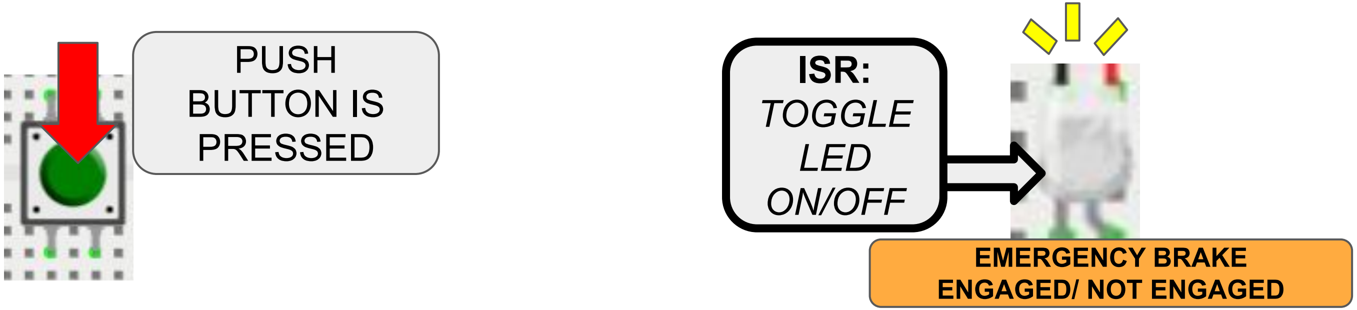
   c. Joystick is analog sensor, so analog input values will be converted from analog to digital value via ADC in analog input port.

Project Description (Overall System)

2. **EMERGENCY BRAKING**
   (PUSH BUTTON = EMERGENCY BRAKE)

   a. Components: Push Button, LED

   b. Operation:



PUSH BUTTON IS PRESSED

ISR:
TOGGLE LED ON/OFF

EMERGENCY BRAKE ENGAGED/ NOT ENGAGED

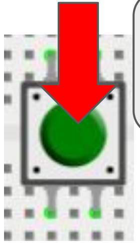
   c. This system is an external interrupt system, it can be triggered anytime regardless of other systems when push button is pressed. (***ESPECIALLY WHEN DRIVER LOST CONTROL OF VEHICLE**)

Project Description (Overall System)

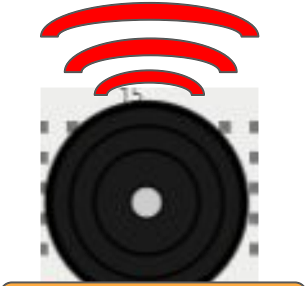
3. **HORN SYSTEM**
   (PUSH BUTTON = HORN BUTTON)

   a. Components: Push Button Module, Active Buzzer

   b. Operation:



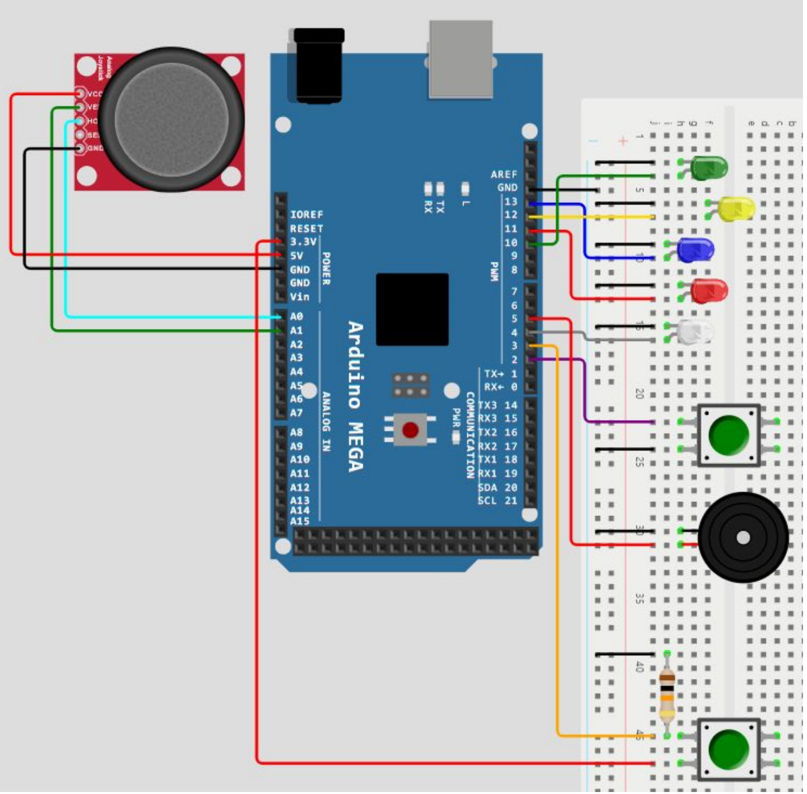
PUSH BUTTON IS PRESSED



HORN SOUNDED

   c. The pushbutton module is set in a pull down resistor configuration. When pushbutton is pressed, the circuit completes and a HIGH signal is sent to Arduino Mega. Which will then signal buzzer to sound.

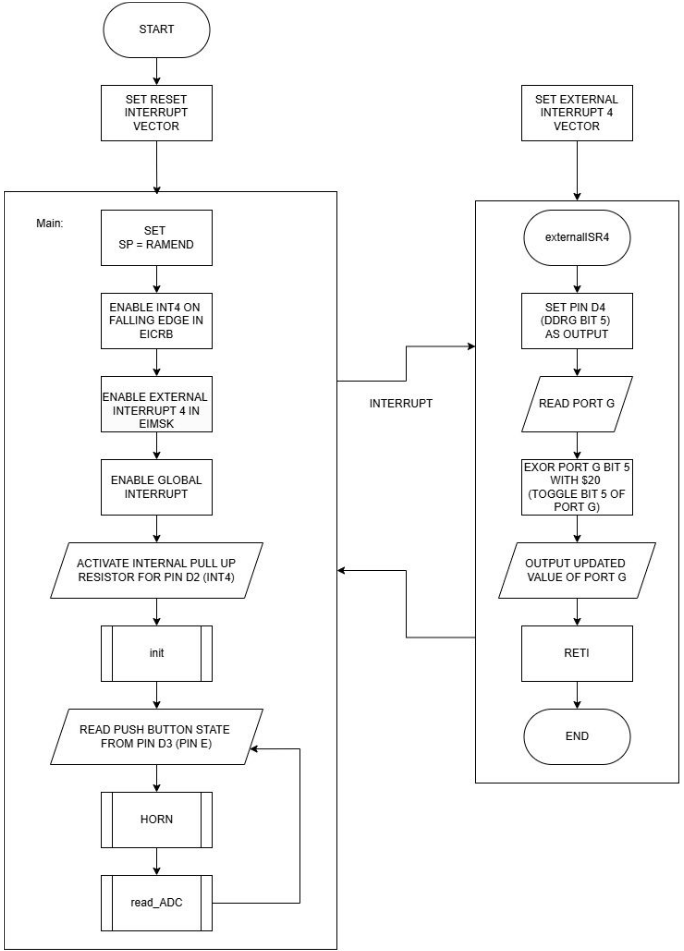
Project Description (Overall System)



Project Motivation

1. **Rising Road Accidents:**
    a. Significant increase in road accidents in Malaysia (**370286 to 545588 cases** from 2021 to 2022, Ministry of Transport Malaysia).
    b. Highlights the need for better driving education.
2. **Challenges in Learning to Drive:**
    a. Limited training hours + Overwhelming content = Improper mastery of driving skills.
    b. Stressful learning environment: Learner anxiety, yelling & cussing of some driving instructors.
3. **Benefits of Driving Simulators:**
    a. Safe, low-stress environment for learners to build confidence and practice without physical risks.

Flowchart of work:



START

SET RESET INTERRUPT VECTOR

Main:

SET SP = RAMEND

ENABLE INT4 ON FALLING EDGE IN EICRB

ENABLE EXTERNAL INTERRUPT 4 IN EIMSK

ENABLE GLOBAL INTERRUPT

ACTIVATE INTERNAL PULL UP RESISTOR FOR PIN D2 (INT4)

init

READ PUSH BUTTON STATE FROM PIN D3 (PIN E)

HORN

read_ADC

INTERRUPT

SET EXTERNAL INTERRUPT 4 VECTOR

externalISR4

SET PIN D4 (DDRG BIT 5) AS OUTPUT

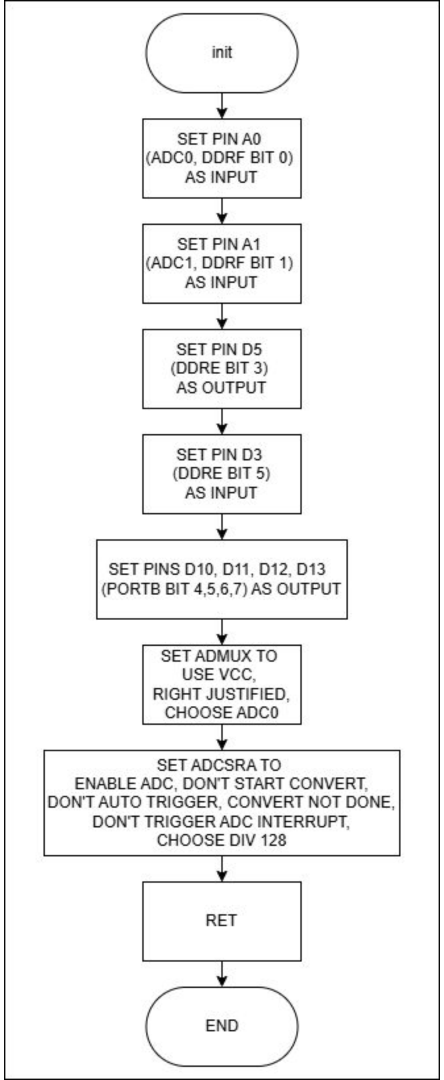
READ PORT G

EXOR PORT G BIT 5 WITH $20 (TOGGLE BIT 5 OF PORT G)

OUTPUT UPDATED VALUE OF PORT G

RETI

END



init

SET PIN A0 (ADC0, DDRF BIT 0) AS INPUT

SET PIN A1 (ADC1, DDRF BIT 1) AS INPUT

SET PIN D5 (DDRE BIT 3) AS OUTPUT

SET PIN D3 (DDRE BIT 5) AS INPUT

SET PINS D10, D11, D12, D13 (PORTB BIT 4,5,6,7) AS OUTPUT

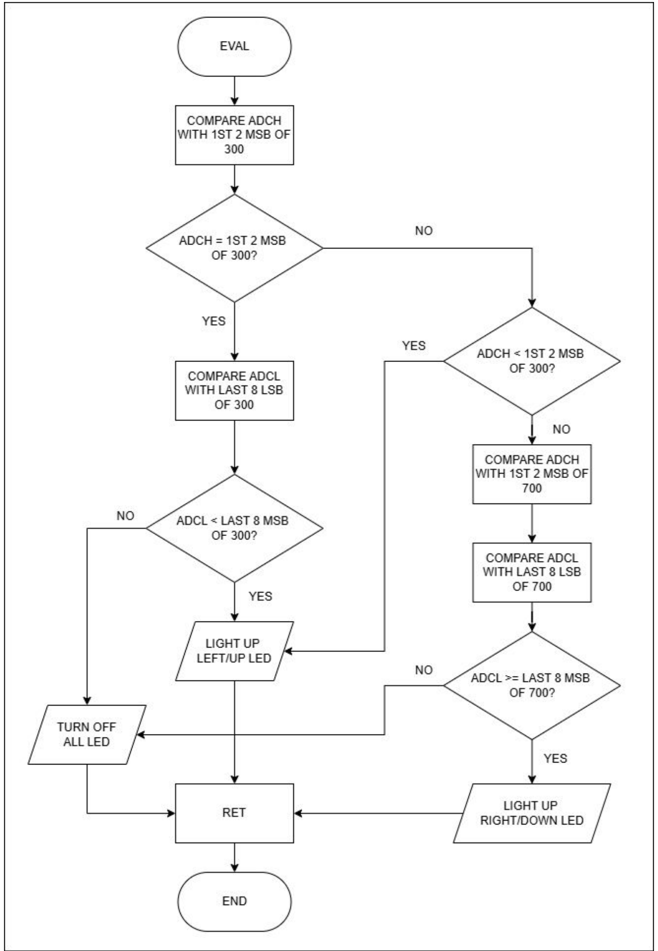
SET ADMUX TO USE VCC, RIGHT JUSTIFIED, CHOOSE ADC0

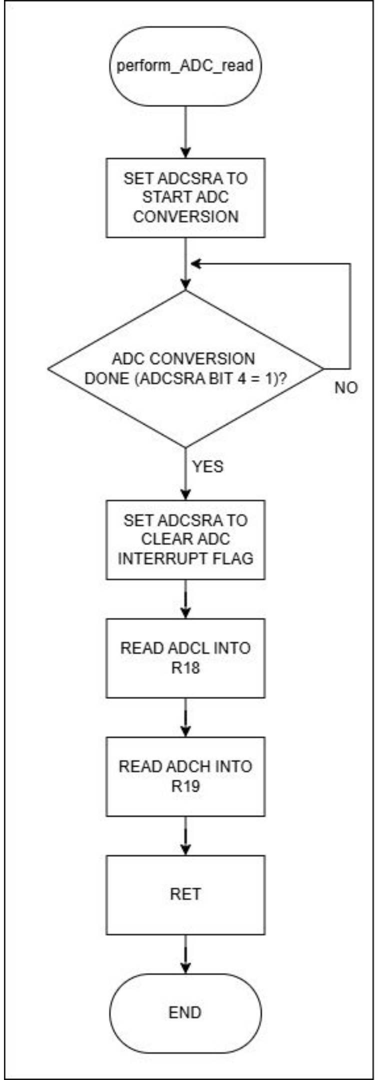
SET ADCSRA TO ENABLE ADC, DON'T START CONVERT, DON'T AUTO TRIGGER, CONVERT NOT DONE, DON'T TRIGGER ADC INTERRUPT, CHOOSE DIV 128

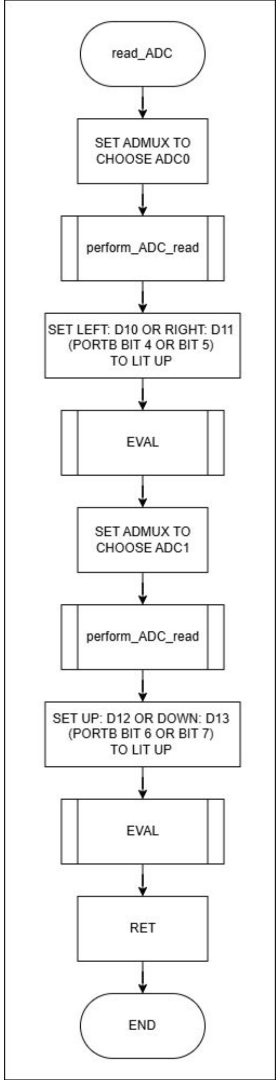
RET

END

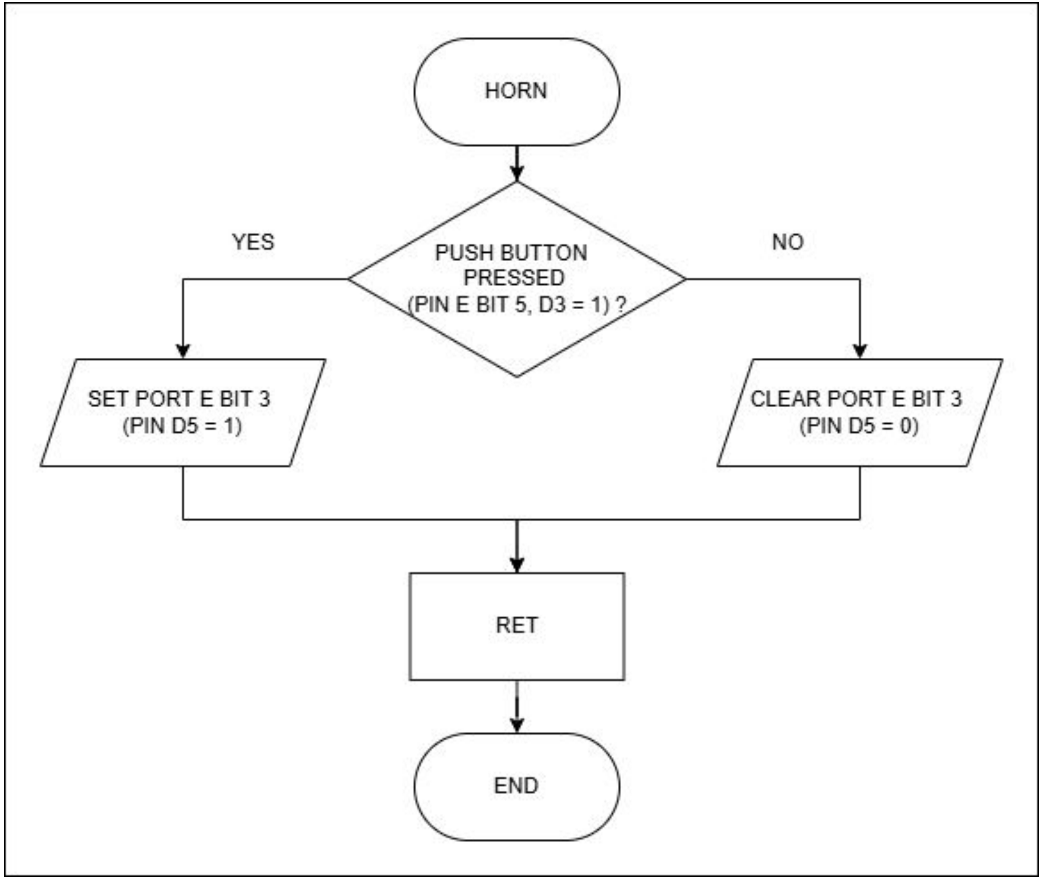
Flowchart of work:


EVAL
COMPARE ADCH WITH 1ST 2 MSB OF 300
ADCH = 1ST 2 MSB OF 300?
NO
YES
COMPARE ADCL WITH LAST 8 LSB OF 300
ADCH < 1ST 2 MSB OF 300?
YES
NO
COMPARE ADCH WITH 1ST 2 MSB OF 700
COMPARE ADCL WITH LAST 8 LSB OF 700
ADCL < LAST 8 MSB OF 300?
NO
YES
LIGHT UP LEFT/UP LED
ADCL >= LAST 8 MSB OF 700?
NO
YES
TURN OFF ALL LED
LIGHT UP RIGHT/DOWN LED
RET
END

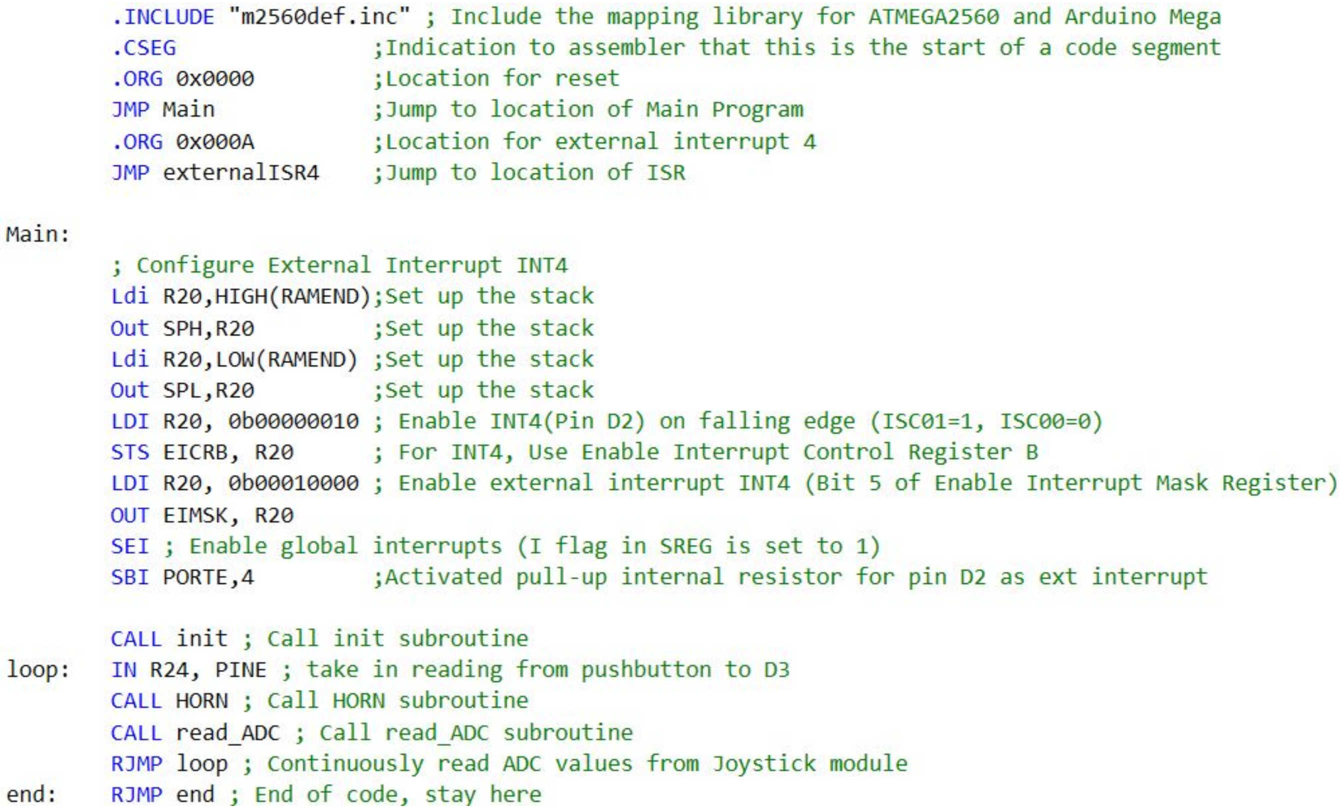

perform_ADC_read
SET ADCSRA TO START ADC CONVERSION
ADC CONVERSION DONE (ADCSRA BIT 4 = 1)?
NO
YES
SET ADCSRA TO CLEAR ADC INTERRUPT FLAG
READ ADCL INTO R18
READ ADCH INTO R19
RET
END


read_ADC
SET ADMUX TO CHOOSE ADC0
perform_ADC_read
SET LEFT: D10 OR RIGHT: D11 (PORTB BIT 4 OR BIT 5) TO LIT UP
EVAL
SET ADMUX TO CHOOSE ADC1
perform_ADC_read
SET UP: D12 OR DOWN: D13 (PORTB BIT 6 OR BIT 7) TO LIT UP
EVAL
RET
END

Flowchart of work:

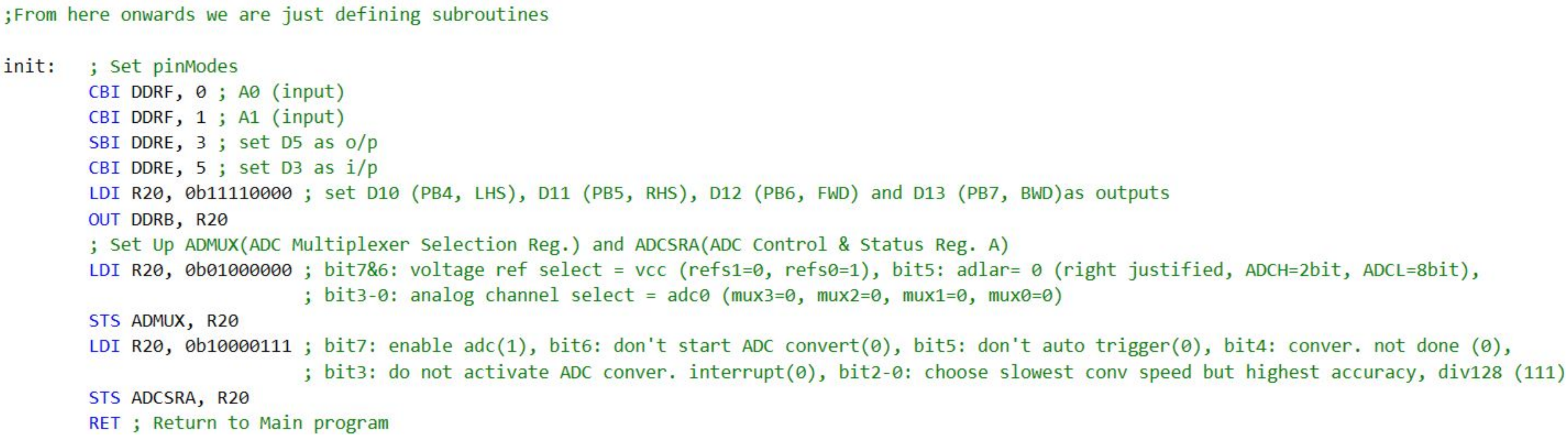

HORN
YES
PUSH BUTTON PRESSED (PIN E BIT 5, D3 = 1) ?
NO
SET PORT E BIT 3 (PIN D5 = 1)
CLEAR PORT E BIT 3 (PIN D5 = 0)
RET
END

Project Coding (Main Program)



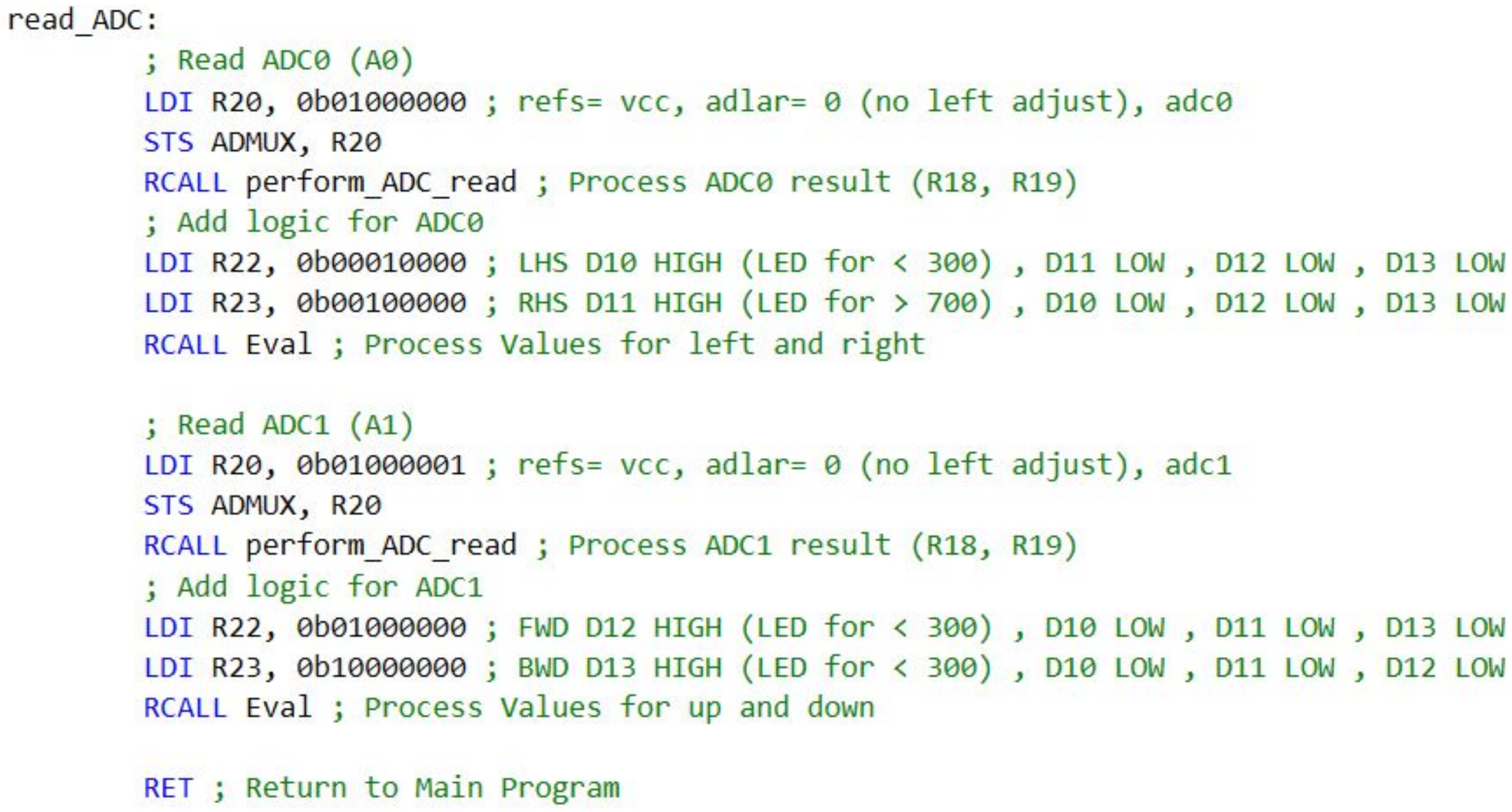
```
        .INCLUDE "m2560def.inc" ; Include the mapping library for ATMEGA2560 and Arduino Mega
        .CSEG               ;Indication to assembler that this is the start of a code segment
        .ORG 0x0000         ;Location for reset
        JMP Main            ;Jump to location of Main Program
        .ORG 0x000A         ;Location for external interrupt 4
        JMP externalISR4    ;Jump to location of ISR

Main:
        ; Configure External Interrupt INT4
        Ldi R20,HIGH(RAMEND);Set up the stack
        Out SPH,R20         ;Set up the stack
        Ldi R20,LOW(RAMEND) ;Set up the stack
        Out SPL,R20         ;Set up the stack
        LDI R20, 0b00000010 ; Enable INT4(Pin D2) on falling edge (ISC01=1, ISC00=0)
        STS EICRB, R20      ; For INT4, Use Enable Interrupt Control Register B
        LDI R20, 0b00010000 ; Enable external interrupt INT4 (Bit 5 of Enable Interrupt Mask Register)
        OUT EIMSK, R20
        SEI ; Enable global interrupts (I flag in SREG is set to 1)
        SBI PORTE,4         ;Activated pull-up internal resistor for pin D2 as ext interrupt

        CALL init ; Call init subroutine
loop:   IN R24, PINE ; take in reading from pushbutton to D3
        CALL HORN ; Call HORN subroutine
        CALL read_ADC ; Call read_ADC subroutine
        RJMP loop ; Continuously read ADC values from Joystick module
end:    RJMP end ; End of code, stay here
```

Project Coding (Define subroutine init)



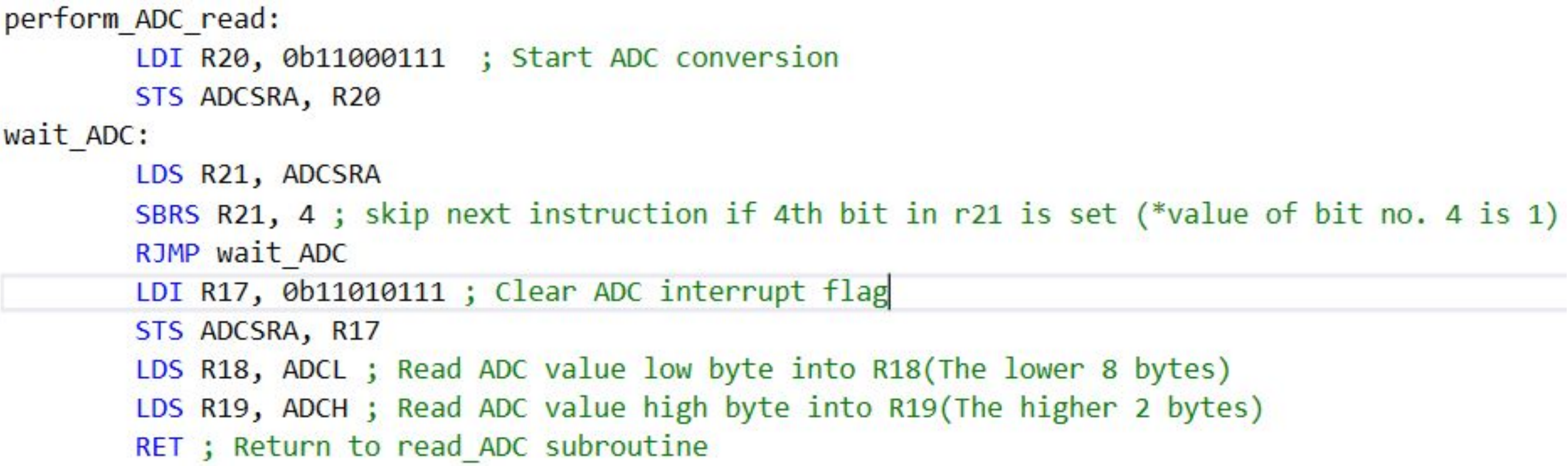
```
;From here onwards we are just defining subroutines

init:   ; Set pinModes
        CBI DDRF, 0 ; A0 (input)
        CBI DDRF, 1 ; A1 (input)
        SBI DDRE, 3 ; set D5 as o/p
        CBI DDRE, 5 ; set D3 as i/p
        LDI R20, 0b11110000 ; set D10 (PB4, LHS), D11 (PB5, RHS), D12 (PB6, FWD) and D13 (PB7, BWD)as outputs
        OUT DDRB, R20
        ; Set Up ADMUX(ADC Multiplexer Selection Reg.) and ADCSRA(ADC Control & Status Reg. A)
        LDI R20, 0b01000000 ; bit7&6: voltage ref select = vcc (refs1=0, refs0=1), bit5: adlar= 0 (right justified, ADCH=2bit, ADCL=8bit),
                            ; bit3-0: analog channel select = adc0 (mux3=0, mux2=0, mux1=0, mux0=0)
        STS ADMUX, R20
        LDI R20, 0b10000111 ; bit7: enable adc(1), bit6: don't start ADC convert(0), bit5: don't auto trigger(0), bit4: conver. not done (0),
                            ; bit3: do not activate ADC conver. interrupt(0), bit2-0: choose slowest conv speed but highest accuracy, div128 (111)
        STS ADCSRA, R20
        RET ; Return to Main program
```

Project Coding (Define subroutine read_ADC)



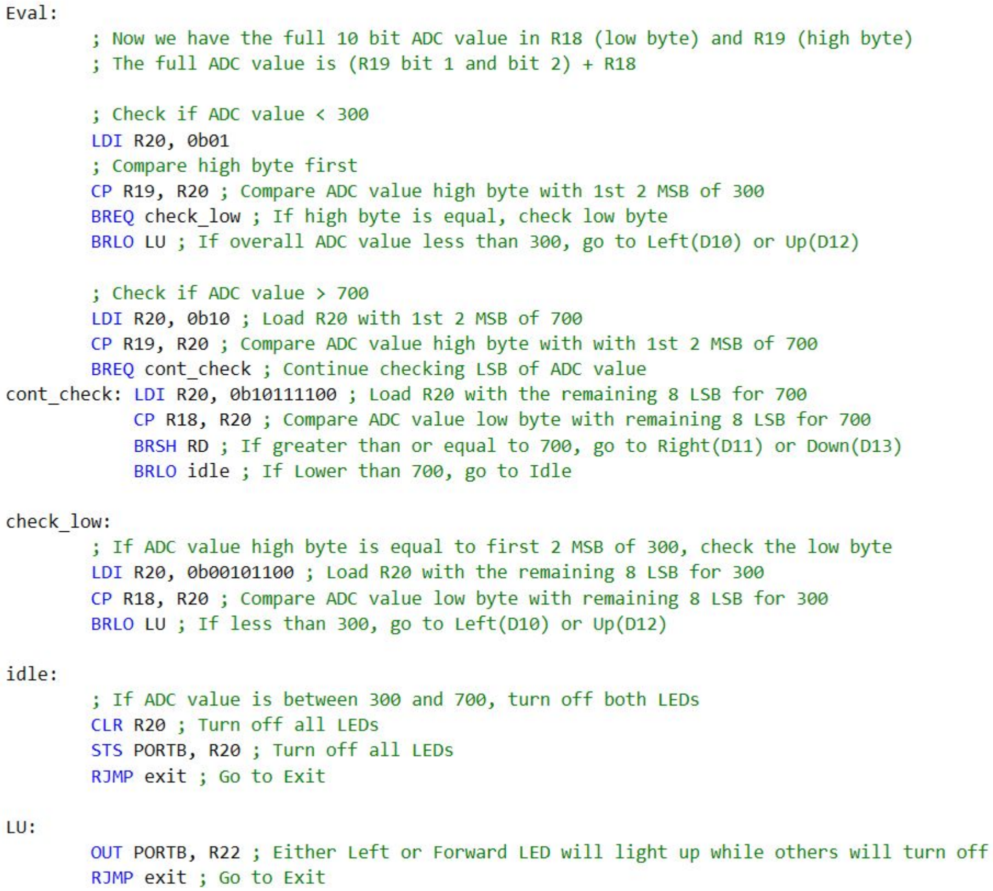
```
read_ADC:
        ; Read ADC0 (A0)
        LDI R20, 0b01000000 ; refs= vcc, adlar= 0 (no left adjust), adc0
        STS ADMUX, R20
        RCALL perform_ADC_read ; Process ADC0 result (R18, R19)
        ; Add logic for ADC0
        LDI R22, 0b00010000 ; LHS D10 HIGH (LED for < 300) , D11 LOW , D12 LOW , D13 LOW
        LDI R23, 0b00100000 ; RHS D11 HIGH (LED for > 700) , D10 LOW , D12 LOW , D13 LOW
        RCALL Eval ; Process Values for left and right

        ; Read ADC1 (A1)
        LDI R20, 0b01000001 ; refs= vcc, adlar= 0 (no left adjust), adc1
        STS ADMUX, R20
        RCALL perform_ADC_read ; Process ADC1 result (R18, R19)
        ; Add logic for ADC1
        LDI R22, 0b01000000 ; FWD D12 HIGH (LED for < 300) , D10 LOW , D11 LOW , D13 LOW
        LDI R23, 0b10000000 ; BWD D13 HIGH (LED for < 300) , D10 LOW , D11 LOW , D12 LOW
        RCALL Eval ; Process Values for up and down

        RET ; Return to Main Program
```

Project Coding (Define subroutine perform_read_ADC)

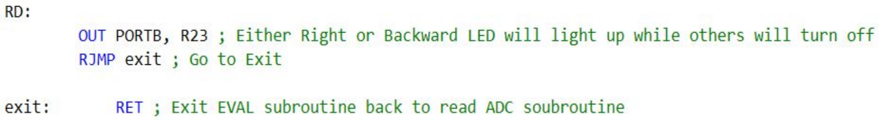


```
perform_ADC_read:
        LDI R20, 0b11000111  ; Start ADC conversion
        STS ADCSRA, R20
wait_ADC:
        LDS R21, ADCSRA
        SBRS R21, 4 ; skip next instruction if 4th bit in r21 is set (*value of bit no. 4 is 1)
        RJMP wait_ADC
        LDI R17, 0b11010111 ; Clear ADC interrupt flag
        STS ADCSRA, R17
        LDS R18, ADCL ; Read ADC value low byte into R18(The lower 8 bytes)
        LDS R19, ADCH ; Read ADC value high byte into R19(The higher 2 bytes)
        RET ; Return to read_ADC subroutine
```

Project Coding (Define subroutine EVAL)

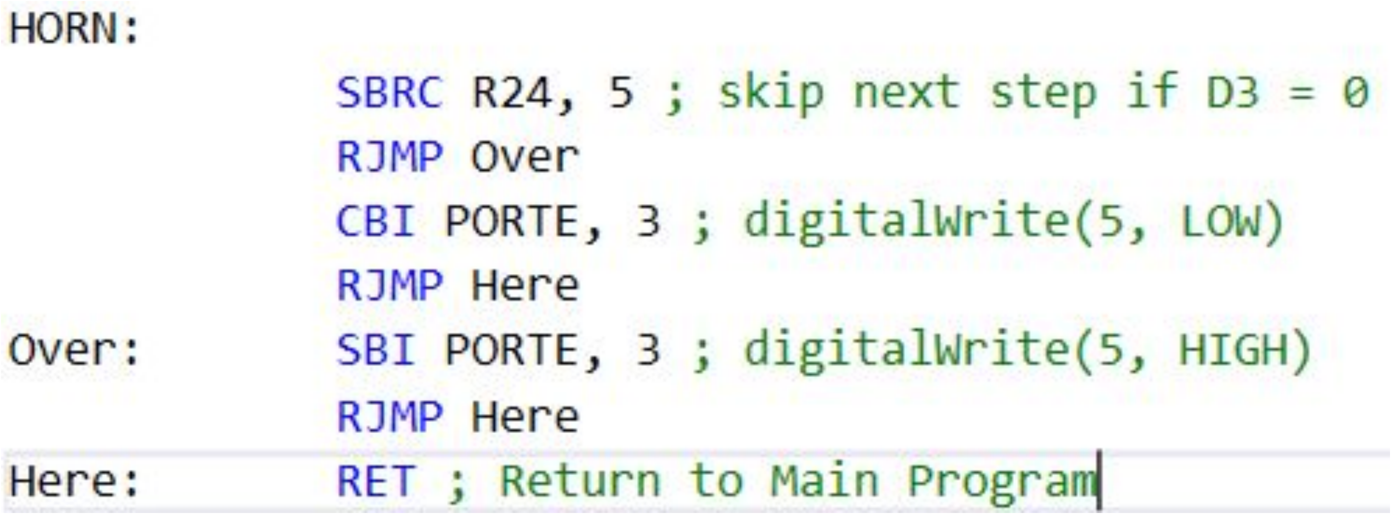


```
Eval:
        ; Now we have the full 10 bit ADC value in R18 (low byte) and R19 (high byte)
        ; The full ADC value is (R19 bit 1 and bit 2) + R18

        ; Check if ADC value < 300
        LDI R20, 0b01
        ; Compare high byte first
        CP R19, R20 ; Compare ADC value high byte with 1st 2 MSB of 300
        BREQ check_low ; If high byte is equal, check low byte
        BRLO LU ; If overall ADC value less than 300, go to Left(D10) or Up(D12)

        ; Check if ADC value > 700
        LDI R20, 0b10 ; Load R20 with 1st 2 MSB of 700
        CP R19, R20 ; Compare ADC value high byte with with 1st 2 MSB of 700
        BREQ cont_check ; Continue checking LSB of ADC value
cont_check: LDI R20, 0b10111100 ; Load R20 with the remaining 8 LSB for 700
            CP R18, R20 ; Compare ADC value low byte with remaining 8 LSB for 700
            BRSH RD ; If greater than or equal to 700, go to Right(D11) or Down(D13)
            BRLO idle ; If Lower than 700, go to Idle

check_low:
        ; If ADC value high byte is equal to first 2 MSB of 300, check the low byte
        LDI R20, 0b00101100 ; Load R20 with the remaining 8 LSB for 300
        CP R18, R20 ; Compare ADC value low byte with remaining 8 LSB for 300
        BRLO LU ; If less than 300, go to Left(D10) or Up(D12)

idle:
        ; If ADC value is between 300 and 700, turn off both LEDs
        CLR R20 ; Turn off all LEDs
        STS PORTB, R20 ; Turn off all LEDs
        RJMP exit ; Go to Exit

LU:
        OUT PORTB, R22 ; Either Left or Forward LED will light up while others will turn off
        RJMP exit ; Go to Exit
```



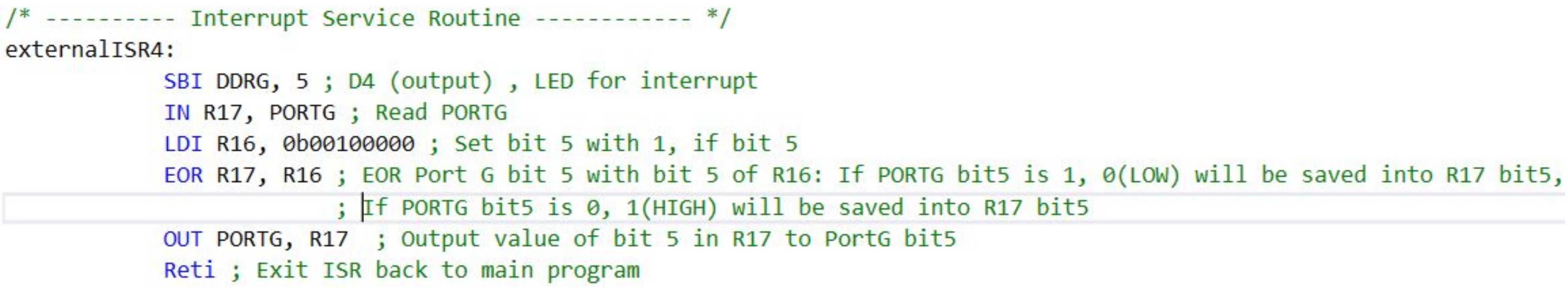
```
RD:
        OUT PORTB, R23 ; Either Right or Backward LED will light up while others will turn off
        RJMP exit ; Go to Exit

exit:       RET ; Exit EVAL subroutine back to read ADC soubroutine
```

Project Coding (Define subroutine HORN)



```
HORN:
            SBRC R24, 5 ; skip next step if D3 = 0
            RJMP Over
            CBI PORTE, 3 ; digitalWrite(5, LOW)
            RJMP Here
Over:       SBI PORTE, 3 ; digitalWrite(5, HIGH)
            RJMP Here
Here:       RET ; Return to Main Program
```

Project Coding (Define ISR)



```
/* ---------- Interrupt Service Routine ------------ */
externalISR4:
            SBI DDRG, 5 ; D4 (output) , LED for interrupt
            IN R17, PORTG ; Read PORTG
            LDI R16, 0b00100000 ; Set bit 5 with 1, if bit 5
            EOR R17, R16 ; EOR Port G bit 5 with bit 5 of R16: If PORTG bit5 is 1, 0(LOW) will be saved into R17 bit5,
                         ; If PORTG bit5 is 0, 1(HIGH) will be saved into R17 bit5
            OUT PORTG, R17  ; Output value of bit 5 in R17 to PortG bit5
            Reti ; Exit ISR back to main program
```