UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI
INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA
E SISTEMISTICA
DIMES

University degree in
Computer Engineering for the Internet of Things

Course of
Low level and embedded system programming

Project
"Steering wheel and pedals with ATmega328p"

Professor
Furfaro Angelo

Teaching assistant
Felicetti Carmelo

Candidate
Fabio Capparelli 214490

Session 2019/2020

Introduction

The aim of this project has been to develop a steering wheel and pedals for playing to car games.

The game used for testing the prototype has been F1 2017 (formula uno 2017) and the MCU used belongs to ATmega family, the ATmega328p installed on the Arduino platform, that we studied during the practical lectures.

The main goal is to implement a sort of gamepad that provides a lot of functionalities and properties, such as a joystick and button to move inside the menu game, the steering and pedals for driving, and at the end the most important property has been the power reduction of the prototype. Essentially, I wanted to develop a prototype that communicates with the game through bluetooth connection and by using interrupt programming I can reduce so much as possible the power consumption of the board by putting the CPU in sleep mode.

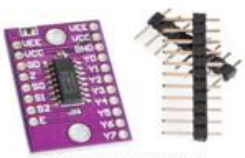
Used components

Arduino board
(Atmega328p)

A0 PCF8574 (I2C)

HC-05 serial
bluetooth

Joystick shield V1.4



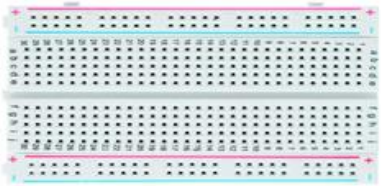
WCMCU 4051
Analog mux

74HC595N

7-segments led

Push button switch

10kΩ
potentiometers



Breadboard

Diode leds

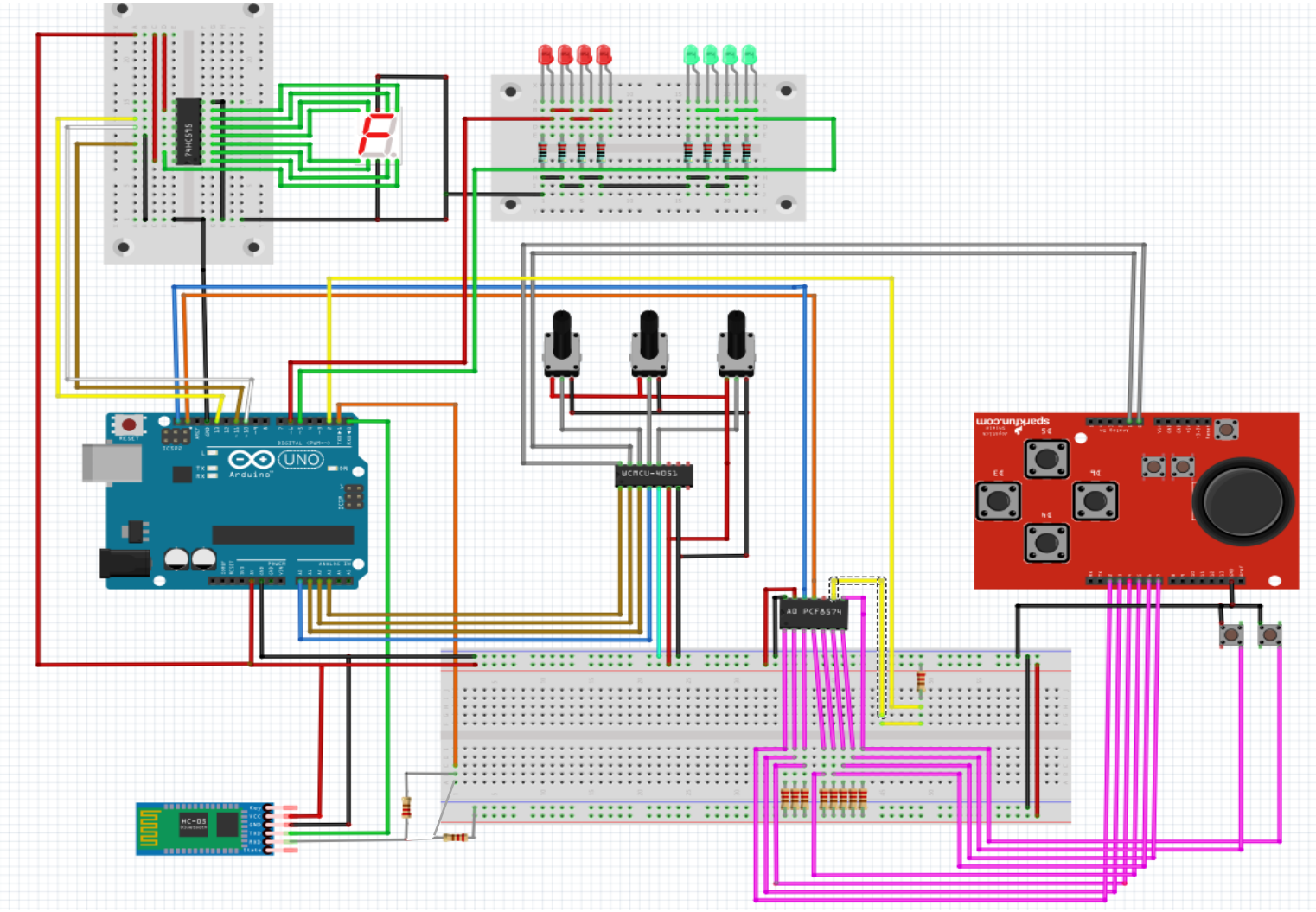
Resistors

Wires

Mini breadboards

- 1 x Arduino board with ATmega328p.
- 1 x A0 PCF8574 (i2c).
- 1 x HC-05 bluetooth serial.
- 1 x Joystick shield V1.A.
- 1 x WCMCU 4051.
- 1 x 74HC595N.
- 1 x 7-segments led.
- 2 x switch buttons.
- 3 x 10kΩ potentiometers.
- 1 x breadboard.
- 4 x green leds.
- 4 x red leds.
- N x resistors (220, 330, 1k, 2k 10k)
- N x colored wires.
- 4 x mini breadboards.

Schema



- ATmega328p → this is the MCU used to implement the entire project.
- PCF8574 → this is an I2C device based on the extending of digital pins of Arduino using the i2c bus interface, a very nice feature of this component is the INT pin, that stands for interrupt and allow to us to check if a change of the value of D0 , … , D7 on the device occurs. This pin follows the open drain policy and it is kept to high level through a 10kΩ resistor and should be used with ATmega328p external interrupt on the falling edge or low level of the signal.
- HC-05 → it is a bluetooth module that allow to us a bluetooth serial communication on COM8 of my PC.
- Joystick shield V1.A → this is a shield compatible with Arduino board, it is used to simulate a real controller with a joystick and 6 buttons similar to A, B, X, Y, start and select of Xbox controller.
- WCMCU 4051 → like 7HC4051, essentially it is an analog multiplexer, it can take 3 selection bits and can extends almost 8 analog pins. For the prototype is very useful because there are 5 analog potentiometers to control, and the analog pins available are only four, due to the usage of the SDA and SCL i2c pins that are located on the PORTC where we can find the A4 and A5 pins.
- 7-segments led → used to display the gear.
- Switch buttons → useful to implement L1 and R1 bumper of Xbox controller, typically for the gear up and gear down.
- Potentiometers → they are most important components, 2 are used to implement the accelerator and the brake/reverse pedals and the last is used for controlling the steering wheel.
- Leds → green leds used with Fast PWM mode of timer0, when the accelerator pedal is pushed down the leds intensity increases, when I release it, they are decreased. Similar for the brake/revers pedal but with red leds.

Circuit behavior

The principle behavior of the circuit and project is to implement Xbox controller, for this reason is useful the following table in order to understand better which components are adopted for which Xbox buttons.

Components	Xbox		Driving behavior
1st button on Joystick shield	Y	Button 1	
2nd button on Joystick shield	B	Button 2	
3rd button on Joystick shield	X	Button 3	
4th button on Joystick shield	A	Button 4	
5th button on Joystick shield	Start	Button 5	
6th button on Joystick shield	Select/back	Button 6	
Analog joystick on Joystick shield	Left X, Left Y	HAxis1, HAxis2	Move up/down and left/right
1st switch button	L1 bumper	Button 7	Gear down
2nd switch button	R1 bumper	Button 8	Gear up
1st potentiometer	L2 trigger	Axis 5	Break/reverse
2nd potentiometer	R2 trigger	Axis 4	Accelerate
3rd potentiometer	Right X	IHAxis 3	Steering

This picture shows how our components are translated into a an xbox360 controller.
Now will be explained the entire behavior of the circuit, I'm going to explain components per components following the previous circuital schema from left to right.

The first component that we can see is the bluetooth module, the HC-05 device. This module is based on seriale communication and it can operate in both slave and master mode. For my purpose it is a slave, that takes values from the circuit and put them into a serial channel. It has from 4 pins (VCC, GND, TX and RX) the VCC is connected to 5V of breadboard and also GND to GND of breadboard. Particular attention on TX and RX pins, the TX pin can be connected directly to RX pin of Arduino, but the RX pin of HC-05 has to connect in a differ way to TX of Arduino, because it works at 3.3 volt, and if we connect them directly it takes the 5 volt. For this reason, is useful to perform a voltage divider or using a voltage level device. What is a voltage divider? Essentially, I'm going to use 2 resistors connected to RX pin of HC-05, one of 1kΩ goes to TX pin of Arduino and the second of 2kΩ goes to GND. In this way only 2/3 of 5 volt goes into RX pin and 1/3 goes to GND. At the end the RX take 3.33 volt.

The second and third components used are the 7-segments led and the 74HC595, so essentially I'm using the 74HC595 device for controlling which segments has to be on or off, this two components are used to display the actual gear. It is essentially an analog

multiplexer, but it is used in SPI mode, it takes a packet of a byte and based on the value of the bits puts on or of the segments. It is displayed value from 0 to 7, related to gear state of a formula uno game.

On the right we can find 8 leds, 4 red and 4 green. The greens are connected to OC0A the 6th pin of Arduino whereby using fast PWM mode of Timer0 the intensity of these leds increases or decreases based on the value of the accelerator pedal. Similar for red leds, but they are connected OC0B the 5th pin of Arduino and based on the value of the break/reverse pedal their intensity changes. Each leds takes the GND through a 220 Ω resistor.

Now we are going to see the WMCMU 4051 or the 74HC4051, it is an analog multiplexer, takes three selection signals from PORTC (PC1, PC2, PC3) to select which of the 5 potentiometers have to put value on the ADC module. This components in connected to 5 volt and GND and has an enable pin $\bar{E}$ connected to GND in this way it is always enabled. The Z pin is the output pin, where the selected potentiometer puts his analog value. In the software part we are going to enable the ADC peripheral only for A0 pin the PC0 on port C. The potentiometers connected are:

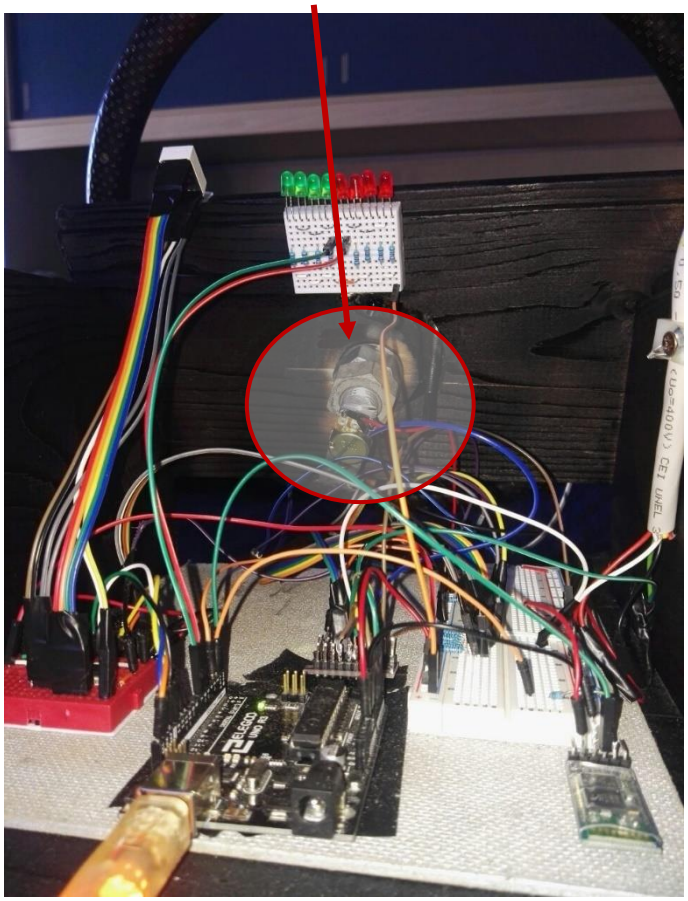
- 000 - Y0 pin → left X axis from the joystick shield
- 001 - Y1 pin → left Y axis from the joystick shield
- 010 - Y2 pin → accelerator from a 10kΩ potentiometer (R2 trigger)
- 011 - Y3 pin → break/revers from 10kΩ potentiometer (L2 trigger)
- 100 - Y4 pin → steering wheel from 10kΩ potentiometer (right X joystick)

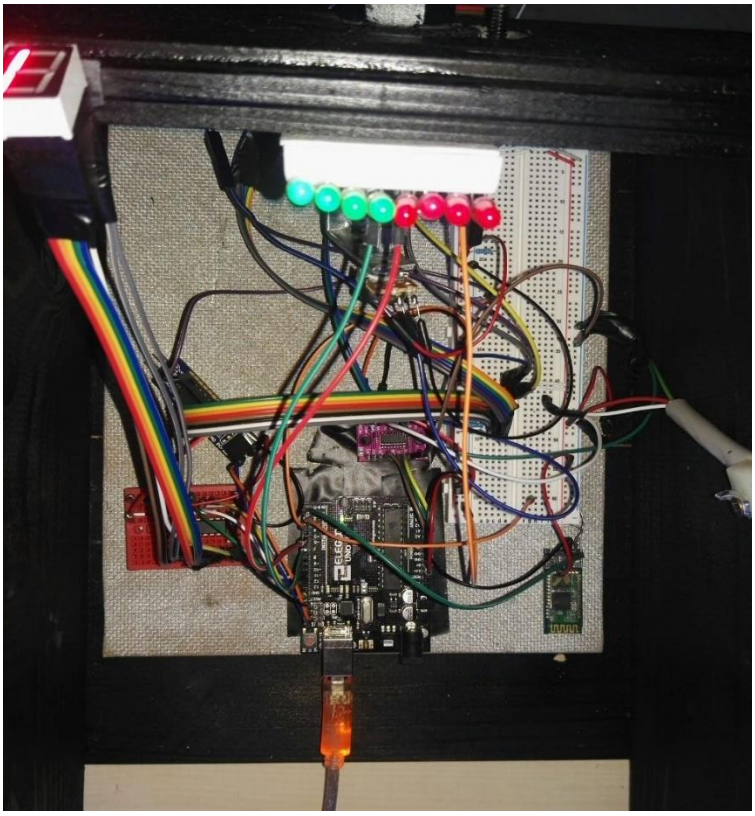
At the end we have the PCF8574 i2c device. It is connected through the i2c bus interface, as a slave device. It is used to check if a button is pushed by using an external interrupt. As we say it send a packet of 8 bits (kept always high through 330Ω resistor to 5v, 1111 1111) when a button is pushed the referring bit goes to 0, the device check that a change occurred and enable the INT pin, put it to low (open drain) and the MCU sees that a falling edge is occurred on the INT signal and will read the packet send out by the slave. This slave has 0x20 salve address. The buttons connected to it is the 6 buttons of the joystick shield and the 2 switch buttons for the gear.

Prototype structure

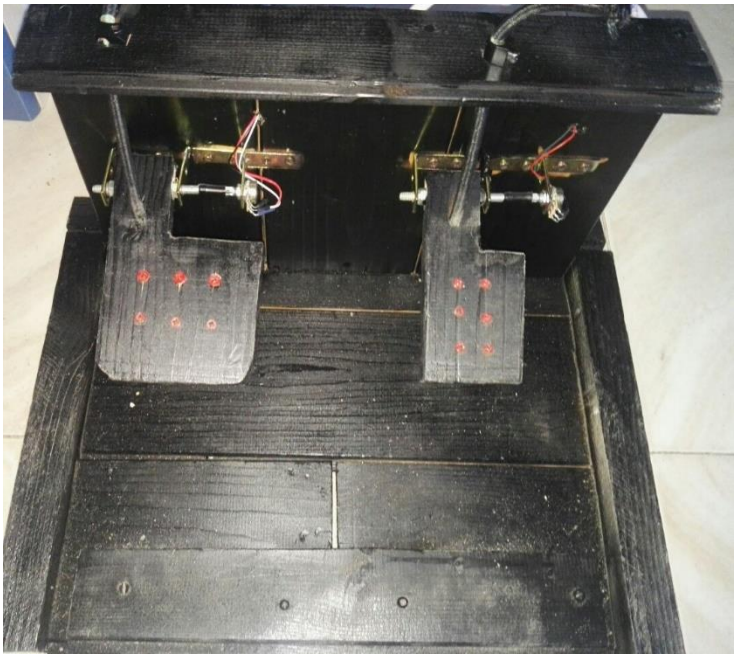
Steering

Potentiometer





Pedals



Potentiometers

The whole structure is built entirely in wood, only the steering is not in wood, it is an old steering wheel of a Fiat PANDA. At the end all has been colored in black.

Software part

Now will be provided a detailed explanation of software part of the project. All circuit code is written in C, and the MCU peripheral programmed have been:

- ADC peripheral → used to convert the analog value of the potentiometer.
- USART peripheral → used to send out the value caught from the components.
- SPI interface → used for 7-segments led.
- I2c interface → used for PCF8574 device.
- Interrupt Unit → used to program:
    - ADC_vect interrupt
    - USART_UDR0_vect interrupt
    - INT0_vect interrupt
    - TIMER1_OVF_vect interrupt
    - SPI_STC_vect interrupt
- Timer/Counter peripheral → Timer1 used for counts 0.032 sec, this is the period to used to send out the value from USART, the Timer0 used to wave form generator in fast PWM mode with OC0A and OC0B.

Essentially the behavior of the C program is that: takes value each 0.032 from the analog potentiometer and send it out. When a button is pushed an external interrupt occurs and send out the value of the pushed button through PCF8574 and i2c bus interface, if will be pushed the gear up / down button the 7segments led, through the SPI peripheral, increment or decrement the value from 0 to 7.

All the data is sent through bluetooth serial using COM8 port on my windows machine, a python script takes and read the value, manipulate them and by using a virtual joystick configured by vJoy app the events are mapped.

C program

Now will be provided the C code for ATmega328p.

```
#define F_CPU 16000000

#define ADC0 0x00
#define ADC1 0x01
#define ADC2 0x02
#define ADC3 0x03
#define ADC4 0x04

#define DOT 0x01  //used for 7 seg led, the value coming from SPI interface 74HC595 device
#define M0 0x7E   // first gear
#define M1 0x12
#define M2 0xBC
#define M3 0xB6
#define M4 0xD2
#define M5 0xE6
#define M6 0xEE
#define M7 0x32   //last gear

#define SS 0x02 //PB2  //pins for SPI interface
#define SCK 0x05 //PB5
#define MOSI 0x03 //PB3
#define MISO 0x04 //PB4

#define SLAVEBASEADDRESSi2c 0x20  //first available slave address on PCF8574
#define R 0x01                    //I2C read mode
#define W 0x00                    //I2C write mode

#define  URSEL 0x07 //last bit of UCSR0C register

#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <avr/sleep.h>

volatile const uint8_t marce[] = {M0, M1, M2, M3, M4, M5, M6, M7}; //array used to handle the gear changes

volatile uint8_t ADCval, sel, ADCx, I2Cval, m; //ADCval is the analog value sent out,
                                               //sel is a selection bit for usart transmission
                                               //ADCx is the analog source of 5 potentiometer actually used
                                               //I2Cval is the value read from PCF8574
                                               //m is the actual gear, the index of the marce[] array

volatile unsigned char charToSend, valToSend;  //firstly is sent to PC a character that identifies the button or potentiometer used
                                               //then the value is sent out
                                               /* x = leftX,
                                                  y = leftY,
                                                  s = steering wheel,
                                                  a = accelerate,
                                                  f = break/reverse,
                                                  b = button */

//ports for fastPWM and for external interrupt
void initPort(void){
    DDRD |= (1<<PORTD6)|(1<<PORTD5); //Port for FastPWM as output port
    DDRD &= ~(1<<PORTD2); //Port for external interrupt coming from PC8574 int pin as input port
    PORTD |= (1<<PORTD2); //activates the pull-up resistor
}

void initPortSPI(void){
    DDRB=(1<<SS)|(1<<MOSI)|(1<<SCK); //SS, MOSI and SCK pin as output pin
    PORTB=(0<<SS)|(0<<MOSI); //SS to 0, and MOSI to 0
}
```

```
//ports for analog mux 4051
void initAMux(void){ //output selector of AMUX
    DDRC |=(1<<PORTC1)|(1<<PORTC2)|(1<<PORTC3); //s0, s1, s2 of 4051 mux as output pins
    DDRC &= ~(1<<PORTC0); //z pin of PC0 as input, where we put the analog value to convert
    DDRC |= (1<<PORTC0);//pull-up activated
    PORTC &= (~(1<<PORTC1)) & (~(1<<PORTC2)) & (~(1<<PORTC3)); //write 000 to s0,s1,s3
}

//initializing ADC converter
void initADC(void){
    ADMUX = (1<<REFS0); //using AVCC with external capacitor
    ADCSRA = (1<<ADPS2)|(1<<ADIE); // prescaler 16 for greater precision, enable interrupt
}

//init timer1 with OVF interrupt for counting 0,032 sec
void initTimer1(){
    TCNT1 = 0; //initialize to 0 the timer value
    TIMSK1 = (1<<TOIE1); //enable timer overflow interrupt
    TCCR1B = (1<<CS11); //set a prescaler of 8
}

void disableTimer1(void){
    TCCR1B = 0x00;
}

//init timer 0 for fastPWM
void initTimer0(){
    TCCR0A = (1<<COM0A1)|(1<<COM0B1)|(1<<WGM00)|(1<<WGM01);//Fast PWM mode //clear OC0A-B on compare match, set OC0A-B at BOTTOM
    TIMSK0 = 0x00;
    TCCR0B = (1<<CS00) | (1<<CS02);// prescaler of 1024
}

void disableTimer0(void){
    TCCR0B = 0x00;
}

//init INT0 external interrupt
void  initExternalInterrupt0(void){
    EIMSK |= (1<<INT0);  //enable INT0
    EICRA |= (1<<ISC01); //falling edge
}

void disableINT0(void){
    EIMSK &= ~(1<<INT0);
}

//init serial with 38400 of baud
void initUSART(void){
    UBRR0H = 0x19>>8; //baud 38400
    UBRR0L = 0x19;
    UCSR0C |= (1<<URSEL)|(1<<UCSZ00)|(1<<UCSZ01); //no parity, 8 bit char, 1 stop bit
}

//enable serial and TX enabling and UDR0 interrupt
void enableUSART(void){
    UCSR0B |= (1<<TXEN0)|(1<<UDRIE0); //enable transmission, enable UDRE interrupt
}

//disable USART serial
void disableUSART(void){
    UCSR0B &= ((~(1<<TXEN0)) & (~(1<<UDRIE0)));
}

//transmit a character to identify which components has to send out value
void usart_TransmitChar(unsigned char c){
    charToSend = c;
    enableUSART(); //enable usart here
}
```

```
//transmit a value of the components
void usart_TransmitVal(unsigned char v){
    valToSend = v;
}

//init ATmega328p SPI as master
void init_MasterSPI(void){
    SPCR |= (1<<MSTR)|(1<<SPR1); //in master mode, prescaler of 64 at 250khz
}

void enableSPI(void){
    SPCR |= (1<<SPIE)|(1<<SPE); //enable interrupt, enable spi
}

void disableSPI(void){
    SPCR &= (~(1<<SPIE)) & (~(1<<SPE)); //disable interrupt, disable spi
}

//send value to 74HC595 device
void send_SPI(unsigned char data){
    enableSPI();
    PORTB &= ~(1<<SS); //put to 0 SS pin
    SPDR = data;
}

//init I2C serial interface
void initI2C(void){
    //frequency of i2c SCL of 333kHz
    TWSR = (0<<TWPS0)|(0<<TWPS0);//puts 0 to TWSR
    TWBR = (1<<TWBR4); //16
}

//start bit of I2C
void startI2C(void){
    TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN); //|(1<<TWIE);
    while ((TWCR & (1<<TWINT))==0);
}

//addressing the slave device PCF8574
void addressI2C(uint8_t slaveAddress, uint8_t RW){
    TWDR = (slaveAddress<<1) | RW; //put 0x20 slave address and read bit
    TWCR = (1<<TWINT)|(1<<TWEN);
    while ((TWCR & (1<<TWINT))==0);
}

//read the value coming from the slave
unsigned char readI2C(void){
    TWCR = (1<<TWINT)|(1<<TWEN); //without ACK
    while ((TWCR & (1<<TWINT))==0);
    return TWDR;
}

void disableI2C (void){
    TWCR = 0x00;
}


//stop bit of I2C communication
void stopI2C (void) {
    TWCR = (1<<TWINT)|(1<<TWSTO)|(1<<TWEN);//|(1<<TWIE);
    disableI2C();
}


//go to sleep CPU and Flash
void goToSleep()
{
    set_sleep_mode(SLEEP_MODE_IDLE); //sleep mode for CPU and Flash
    cli();
    sleep_enable();
    sei();
    sleep_cpu();
    sleep_disable();
}
```

```
void startADC(){
    ADMUX = (ADMUX & 0xF0) | (0x00 & 0x0F); //Z pin of 4051 connected to A0 PC0 of ATmega328p
    ADCSRA |= (1 << ADSC); //start conversion
}

//enable ADC peripheral
void enableADC(void){
    ADCSRA |= (1 << ADEN);
}

//disable ADC peripheral
void disableADC(void){
    ADCSRA &=  ~(1 << ADEN);
}
 //setup ALL
void enableALL(void){
    ADCx = ADC0;
    sel = 0x00;
    m = 0x00;

    initPortSPI();
    init_MasterSPI();

    initPort();
    initAMux();
    initADC();

    initTimer1(); //start timer OVF interrupt for counting 0.032 sec
    initTimer0();

    initUSART();  //start Serial 38400
    initI2C();
    initExternalInterrupt0();
    send_SPI(marce[m]);
}
 //used to fastPWM because I put the pedals potentiometer fixed to the greatest value
 uint8_t rangeFastPWM_FRENO(uint8_t val){
    if(val <= 150)
        return 9*28;
    if ((val <= 155) & (val > 150))
        return 8*28;
    if ((val <= 160) & (val > 155))
        return 7*28;
    if ((val <= 165) & (val > 160))
        return 6*28;
    if ((val <= 170) & (val > 165))
        return 5*28;
    if ((val <= 175) & (val > 170))
        return 4*28;
    if ((val <= 180) & (val > 175))
        return 3*28;
    if ((val <= 185) & (val > 180))
        return 2*28;
    if ((val <= 195) & (val > 185))
        return 1*28;
    if (val > 195)
        return 0*28;
    return 0;
}
```

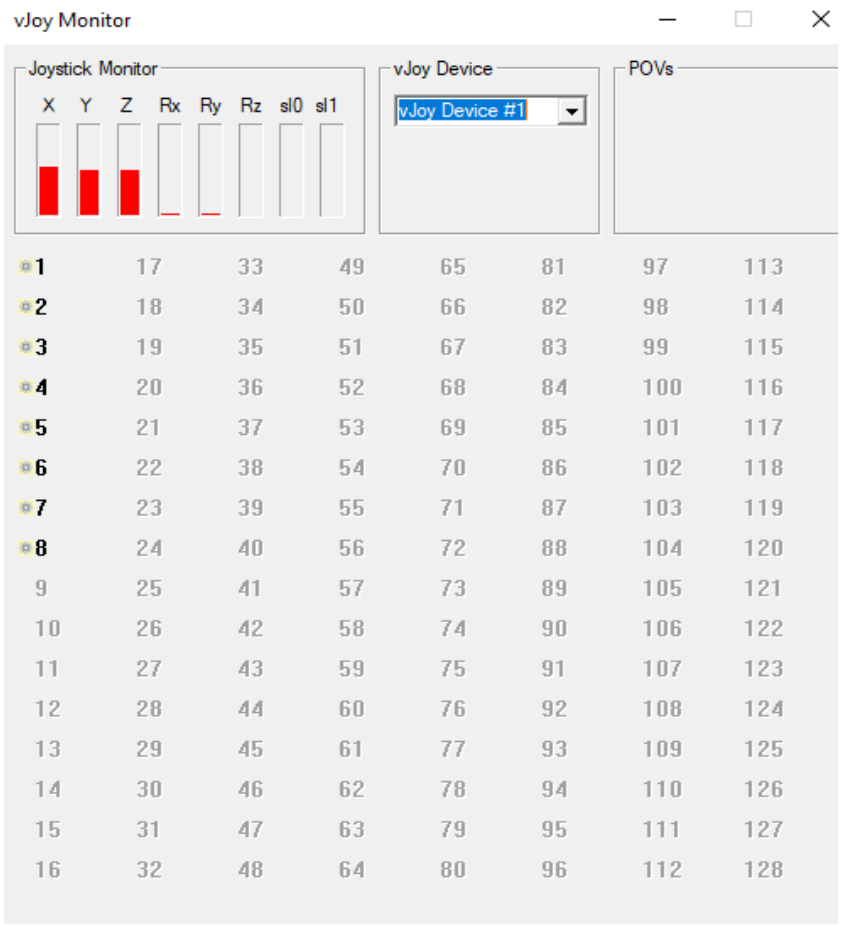
```
uint8_t rangeFastPWM_ACCELLERATORE(uint8_t val){
    if(val <= 130)
        return 9*28;
    if ((val <= 135) & (val > 130))
        return 8*28;
    if ((val <= 140) & (val > 135))
        return 7*28;
    if ((val <= 145) & (val > 140))
        return 6*28;
    if ((val <= 150) & (val > 145))
        return 5*28;
    if ((val <= 155) & (val > 150))
        return 4*28;
    if ((val <= 160) & (val > 155))
        return 3*28;
    if ((val <= 165) & (val > 160))
        return 2*28;
    if ((val <= 170) & (val > 165))
        return 1*28;
    if (val > 170)
        return 0*28;
    return 0;
}

//MAIN
int main(void)
{
    enableALL();
    sei();

    while (1)
    {
        goToSleep();
    }
}

//i2c starts and reads from PFC8574 when the external int occurs
ISR(INT0_vect){
    startI2C();
    addressI2C(SLAVEBASEADDRESSi2c, R);
    unsigned char I2Cval = readI2C();
    stopI2C();

    if((UCSR0A | ~(1<<UDRE0))){
        switch (I2Cval){
            case 0xFE: //Y
                usart_TransmitChar('b'); //send b character first to recognize that a button is pushed
                usart_TransmitVal(1+48); //send value
                break;
            case 0xFD://B
                usart_TransmitChar('b');
                usart_TransmitVal(2+48);
                break;
            case 0xFB://X
                usart_TransmitChar('b');
                usart_TransmitVal(3+48);
                break;
            case 0xF7://A
                usart_TransmitChar('b');
                usart_TransmitVal(4+48);
                break;
            case 0xEF://start
                usart_TransmitChar('b');
                usart_TransmitVal(5+48);
                break;
```

```
        case 0xDF://select
            usart_TransmitChar('b');
            usart_TransmitVal(6+48);
            break;
        case 0xBF://L1 gear dowm
            usart_TransmitChar('b');
            usart_TransmitVal(7+48);
            if (m!=0) m--;
            send_SPI(marce[m]);
            break;
        case 0x7F://R1 gear UP
            usart_TransmitChar('b');
            usart_TransmitVal(8+48);
            if (m!=7) m++;
            send_SPI(marce[m]);
            break;
        default://release buttons
            usart_TransmitChar('b');
            usart_TransmitVal(0+48);
            break;
    }
}
//used to check when UDR0 register is empty and it is ready to receive a data to send out throught serial
ISR(USART_UDRE_vect){
    switch (sel)
    {
        case 0: //case 0 send the character
            UDR0 = charToSend;
            sel=1;
            break;
        case 1: //case 1 send the related value and disable USART, an entire packer char+value is sent
            UDR0 = valToSend;
            disableUSART();
            sel=0;
        break;
    }
}

//used to check when a SPI packet is sent out
ISR (SPI_STC_vect){
    PORTB |= (1<<SS); //put to 1 SS pin
    disableSPI(); //disable SPI
}

//used when a convention is terminated
ISR (ADC_vect){

    if((UCSR0A | ~(1<<UDRE0))){
        ADCval=ADC>>2; //read the 10 bit value from ADC and shift it by 2 to right ADLAR = 0
        switch (ADCx)
        {
        case ADC0: //reading from X axis of Joystick shield
            usart_TransmitChar('x'); //send c character
            ADCx=ADC1;
            PORTC = (PORTC & (~(1<<PORTC3) & ~(1<<PORTC2))) | (1<<PORTC1); //sel 0x01 ADC1
            break;
        case ADC1: //reading from Y axis of Joystick shield
            usart_TransmitChar('y');
            ADCx=ADC2;
            PORTC = (PORTC & (~(1<<PORTC3) & ~(1<<PORTC1))) |(1<<PORTC2); //sel 0x02 ADC2
            break;
        case ADC2: //reading from f potentiometer for break/reverse
            usart_TransmitChar('f');
            ADCx=ADC3;
            PORTC = (PORTC & ~(1<<PORTC3))|((1<<PORTC2)|(1<<PORTC1)); //sel 0x03 ADC3
            OCR0B=rangeFastPWM_FRENO(ADCval); //set OCR0B for fast PWM
            break;
```

```
            case ADC3:
                usart_TransmitChar('a');//reading from a potentiometer for accelerate
                ADCx=ADC4;
                PORTC = (PORTC & (~(1<<PORTC1) & ~(1<<PORTC2))) | (1<<PORTC3); //sel 0x04 ADC4
                OCR0A=rangeFastPWM_ACCELLERATORE(ADCval);//set OCR0A for fast PWM
                break;
            case ADC4:
                usart_TransmitChar('s');//reading from s potentiometer for steering wheel
                ADCx=ADC0;
                PORTC = (PORTC & (~(1<<PORTC2) & ~(1<<PORTC1) & ~(1<<PORTC3))); //sel 0x04 ADC0
                break;
                }
            usart_TransmitVal(ADCval); //send out the value after the character
        }
        disableADC();
    }
```

All is programmed by using interrupt, the only polling mode is in the i2c interface communication, I tried a lot to use the TWI_vect but the code doesn't work well, also I searched online this problem and I found that it is very common, the start conditions, sending of address, read/write data and stop condition with TWI_vect don't work fine, due to the fact that there are many interrupt service routines that can occur concurrently that have a greater priority.

vJoy Monitor used to check the behavior of the controller.



Now it is provided the python script that takes value from bluetooth serial and put them inside virtual joystick.

```
import time
import serial                                       #libary for serial comunication
from vjoy import vj, setJoy                         #library for vJoy to send value to
                                                    #virtual joystick
ATmega328p = serial.Serial('COM8', 38400, timeout=.1)#connect script to serial COM8 with 38400 baud
print("connected on COM8")

print("vj opening", flush=True)
vj.open()                                           #open virstual joystick

time.sleep(3)                                       #setup wainting 3 seconds

x=0                                                 #left x value
y=0                                                 #left y value
b=0                                                 #buttons value
a=0                                                 #accellerator value
f=0                                                 #break reverse value
s=0                                                 #steering wheel value

def freno(val):                                     #fuction used for handling the break/reverse value
                                                    #the potentiometer is fixed to high value near to 255
    if(val <= 150):
        return 9
    if ((val <= 155) & (val > 150)):
        return 8
    if ((val <= 160) & (val > 155)):
        return 7
    if ((val <= 165) & (val > 160)):
        return 6
    if ((val <= 170) & (val > 165)):
        return 5
    if ((val <= 175) & (val > 170)):
        return 4
    if ((val <= 180) & (val > 175)):
        return 3
    if ((val <= 185) & (val > 180)):
        return 2
    if ((val <= 195) & (val > 185)):
        return 1
    if (val > 195):
        return 0
    return 0

def accelleratore(val):                              #fuction used for handling the accelerator value
                                                     #the potentiometer is fixed to high value near to 255
    if(val <= 130):
        return 9
    if ((val <= 135) & (val > 130)):
        return 8
    if ((val <= 140) & (val > 135)):
        return 7
    if ((val <= 145) & (val > 140)):
        return 6
    if ((val <= 150) & (val > 145)):
        return 5
    if ((val <= 155) & (val > 150)):
        return 4
    if ((val <= 160) & (val > 155)):
        return 3
    if ((val <= 165) & (val > 160)):
        return 2

    if ((val <= 170) & (val > 165)):
        return 1
    if (val > 170):
        return 0
    return 0
```

```
while True:

    data = ATmega328p.read()                        #read incoming byte from serial

    if(b != 0):
        vj.setButton(b, 1)                          #send to vJoy the pushed button value if 0 sends nothing
        print(b)
    else:
        print(x, y, s, a, f, b)
        setJoy(x, y, s, a, f, 30.0)                 #send to vJoy the potentiometers value

    if(len(data) > 0):                              #reads data if it receives the following char
        if chr(data[0]) == 'b':                     #read immediately the next value
            b = int(ATmega328p.read())
        elif chr(data[0]) == 'x':
            x = ATmega328p.read()
            x = int(x[0])*5                         #mutiplied by 5 because it receives an 8 bit value
        elif chr(data[0]) == 'y':                   #and has to shift to 10 bit value
            y = ATmega328p.read()
            y = int(y[0])*5
        elif chr(data[0]) == 's':
            s = ATmega328p.read()                   #the steering wheel has 2 ends point it starts from 40 and arrives to 180
            s = (int(s[0])-40)*6.75                 #it never reaches the 0 value or 255, so I apply this formula to take proportional value
        elif chr(data[0]) == 'a':
            a = ATmega328p.read()
            a = accelleratore(int(a[0]))*115
        elif chr(data[0]) == 'f':
            f = ATmega328p.read()
            f = freno(int(f[0]))*115
```