UNIVERSITATEA TITU MAIORESCU

FACULTATEA: INFORMATICĂ
DEPARTAMENT: INFORMATICĂ
Programa de studii: INFORMATICĂ
DISCIPLINA: INTELIGENȚĂ ARTIFICIALĂ

IA - *Testul de evaluare nr.* 20

Dronă terestră – REMOTE CONTROL CAR ELECTRIC

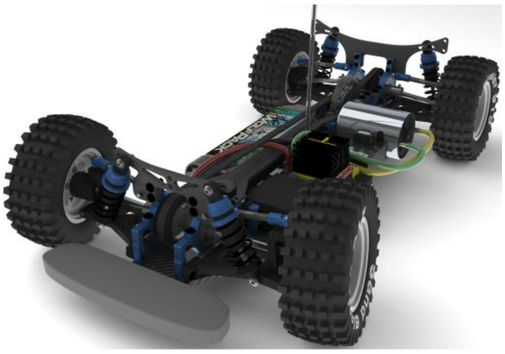
Grupa	Numele și prenumele	Semnătură student	Notă evaluare

Data: ____/____/_________

CS-I dr.ing.
Lucian Ștefăniță GRIGORE

Conf.dr.ing.
Iustin PRIESCU

Ș.L.dr.ing.
Dan-Laurențiu GRECU



Cuprins

1. INTRODUCERE

Pentru a controla de la distanță mașina, se folosește o pereche de module nRF24L01.

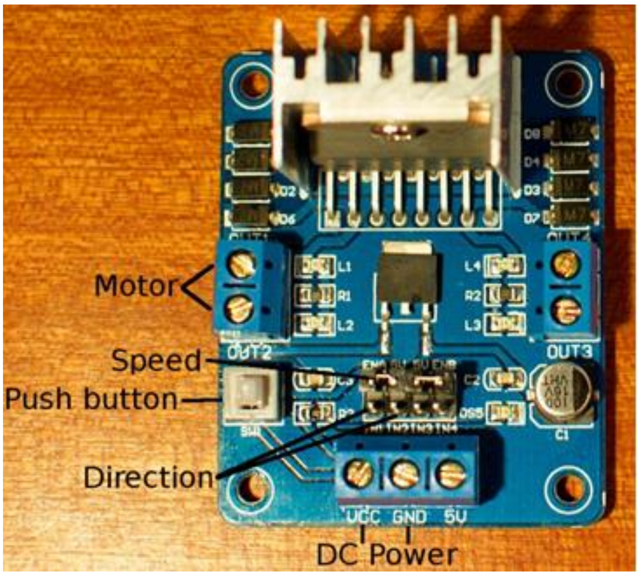
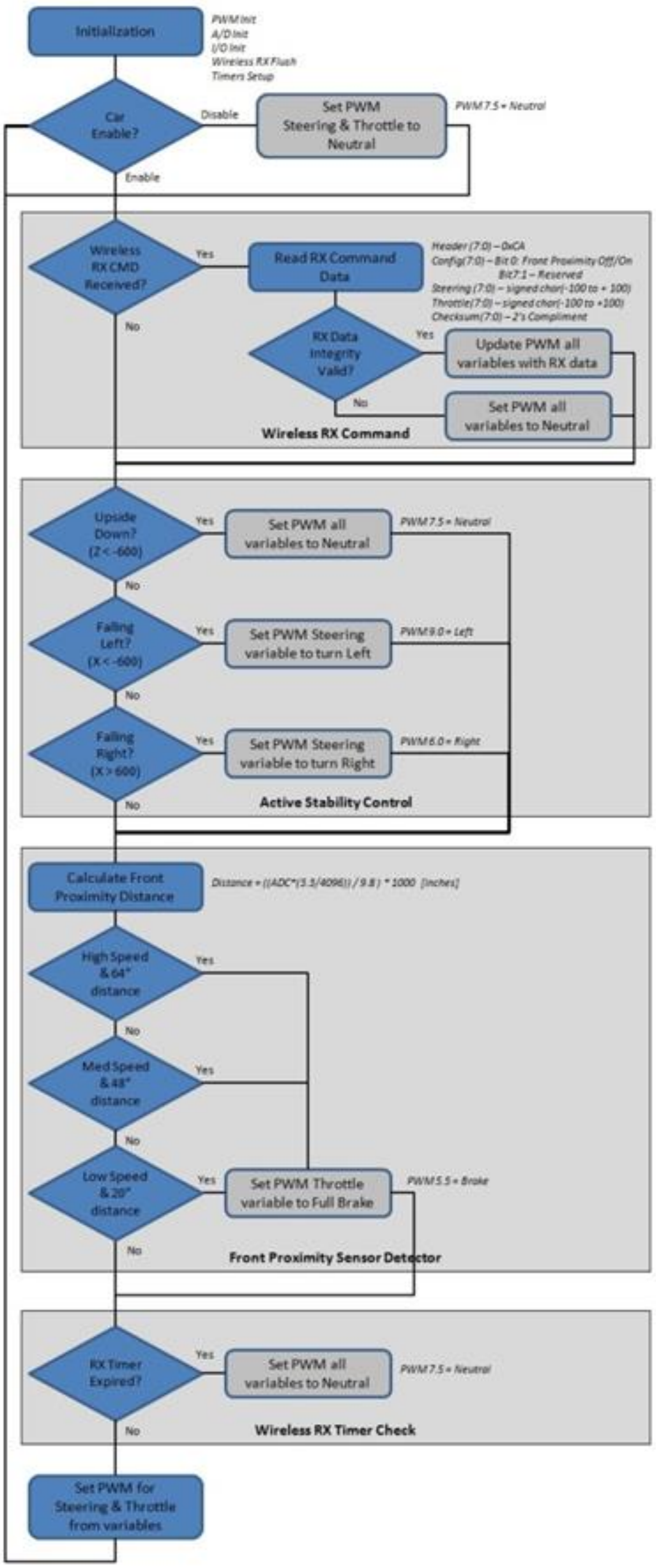


Fig. 1-1 Controler Arduino UNO
http://waihung.net/arduino-visual-basic-servo-control/

```
// Potentiometer controlling small dc motor through standalone L298n board

const int in1 = 2;    // direction pin 1
const int in2 = 4;    // direction pin 2
const int ena = 3;    // PWM pin to change speed

int pot;              // integer for potentiometer
int fspeed;             // forward speed
int bspeed;             // backward speed

void setup() {
  pinMode(in1, OUTPUT);
  pinMode(in2, OUTPUT);
  pinMode(ena, OUTPUT);
}

void loop() {
  pot = analogRead(A0);

  if (pot >= 480 && pot <= 540)
  {
    stop();
  }
  if (pot < 500)
  {
    fspeed = map(pot, 479, 0, 70, 250);
    forward(fspeed);
  }
  if (pot > 520)
  {
    bspeed = map(pot, 541, 1023, 70, 250);
    backward(bspeed);
  }
}

void stop()
{
  analogWrite(ena, 0);
}

void forward(int fspeed)
{
  digitalWrite(in1, HIGH);
  digitalWrite(in2, LOW);
  analogWrite(ena, fspeed);
}

void backward(int bspeed)
{
  digitalWrite(in1, LOW);
  digitalWrite(in2, HIGH);
  analogWrite(ena, bspeed);
}
```

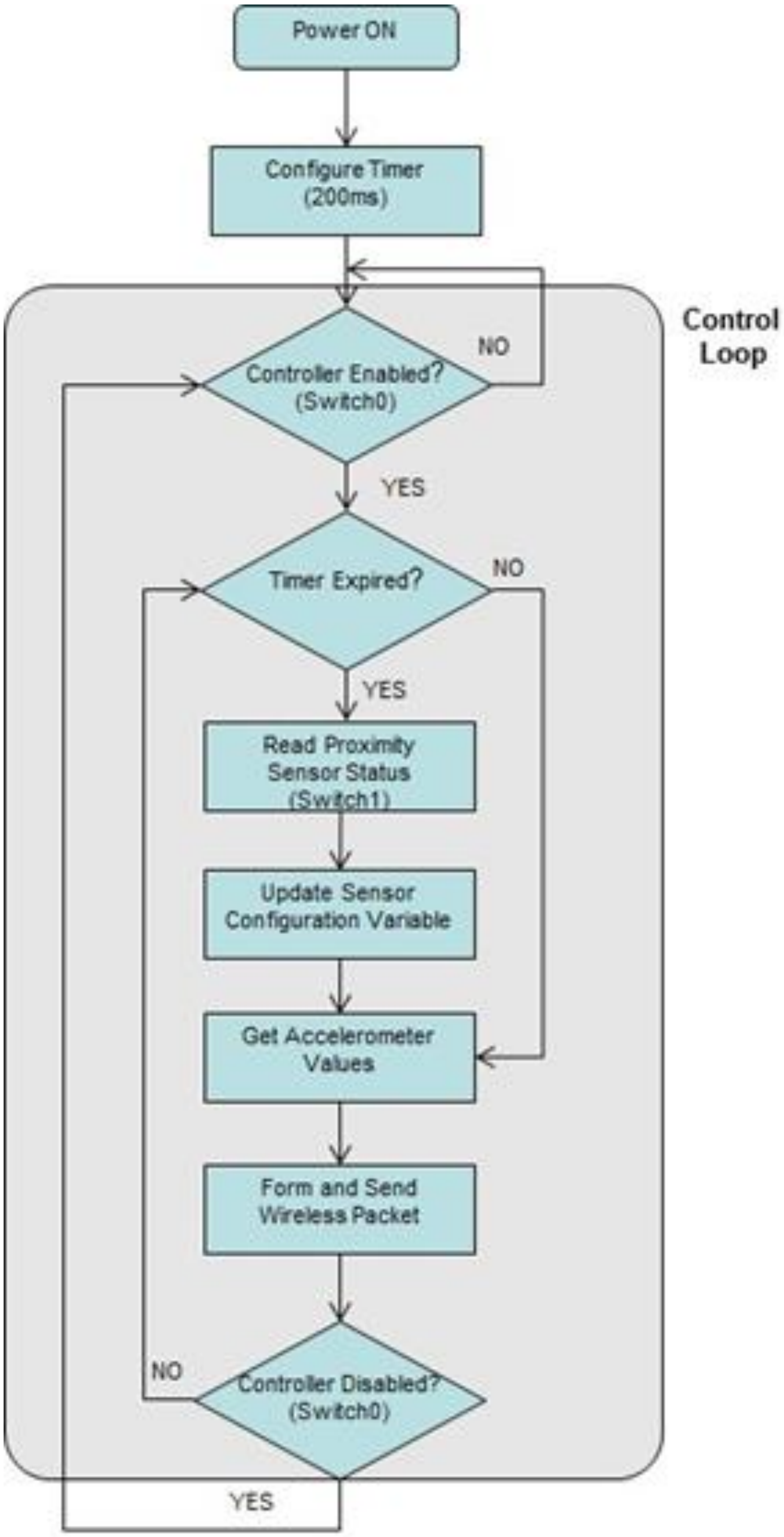
2. PROIECTARE SOFTWARE

RX Board Software Diagram[1]


Initialization
PWM Init
A/D Init
I/O Init
Wireless RX Flush
Timers Setup
Car Enable?
Disable
Set PWM Steering & Throttle to Neutral
PWM 7.5 = Neutral
Enable
Wireless RX CMD Received?
Yes
Read RX Command Data
Header (7:0) – 0xCA
Config(7:0) – Bit 0: Front Proximity Off/On
Bit7:1 – Reserved
Steering (7:0) – signed char(-100 to + 100)
Throttle(7:0) – signed char(-100 to +100)
Checksum(7:0) – 2's Compliment
RX Data Integrity Valid?
Yes
Update PWM all variables with RX data
No
Set PWM all variables to Neutral
No
Wireless RX Command
Upside Down? (Z < -600)
Yes
Set PWM all variables to Neutral
PWM 7.5 = Neutral
No
Falling Left? (X < -600)
Yes
Set PWM Steering variable to turn Left
PWM 9.0 = Left
No
Falling Right? (X > 600)
Yes
Set PWM Steering variable to turn Right
PWM 6.0 = Right
No
Active Stability Control
Calculate Front Proximity Distance
Distance = ((ADC*(3.3/4096)) / 9.8) * 1000 [inches]
High Speed & 64" distance
Yes
No
Med Speed & 48" distance
Yes
No
Low Speed & 20" distance
Yes
Set PWM Throttle variable to Full Brake
PWM 5.5 = Brake
No
Front Proximity Sensor Detector
RX Timer Expired?
Yes
Set PWM all variables to Neutral
PWM 7.5 = Neutral
No
Wireless RX Timer Check
Set PWM for Steering & Throttle from variables

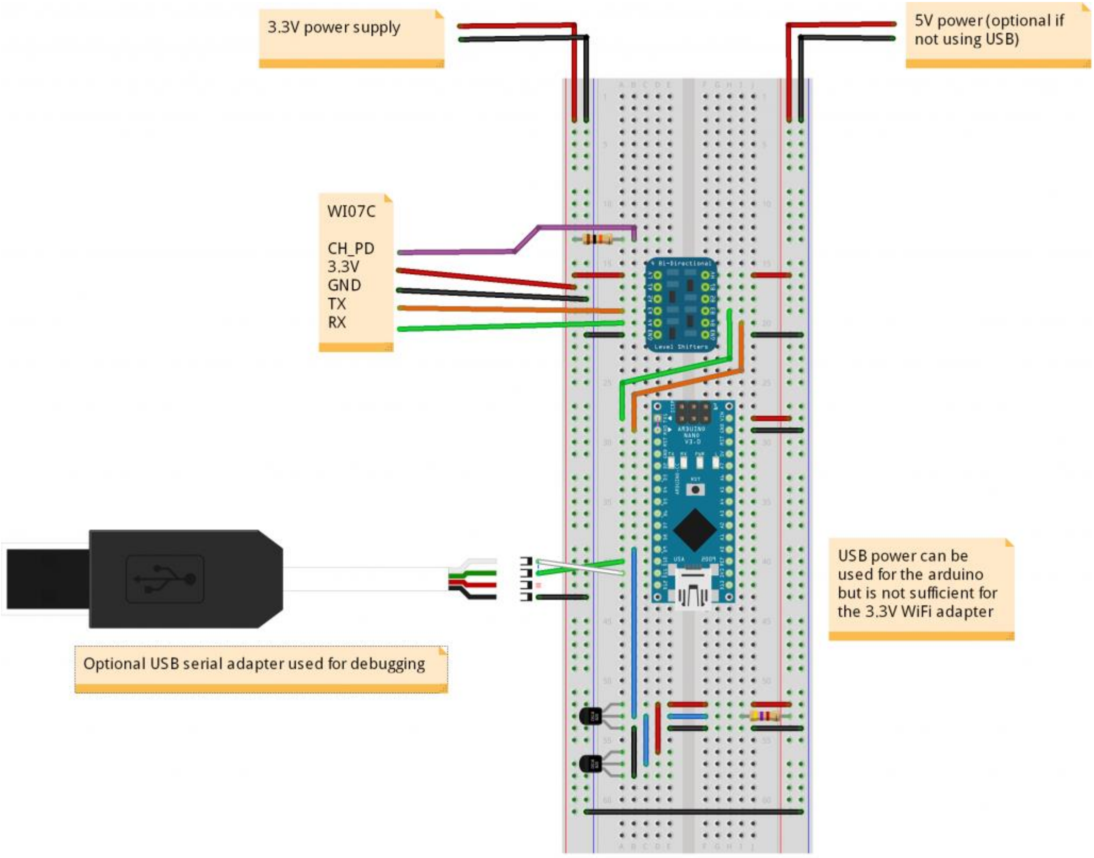
[1] http://www.socialledge.com/sjsu/index.php?title=F13:_Remote_Control_Car

TX Board Software Diagram


Power ON
Configure Timer (200ms)
Control Loop
Controller Enabled? (Switch0)
NO
YES
Timer Expired?
NO
YES
Read Proximity Sensor Status (Switch1)
Update Sensor Configuration Variable
Get Accelerometer Values
Form and Send Wireless Packet
Controller Disabled? (Switch0)
NO
YES

3. MODUL WiFi

Modulul Wi-Fi WI07C bazat pe cipul ESP8266. Folosește următoarea configurație simplă pe un Arduino nano pentru a citi temperaturile de la senzori 18BS20: Se formatează datele ca JSON și apoi se trimite pe WiFi la un server din rețeaua de lucru.


3.3V power supply
5V power (optional if not using USB)
WI07C
CH_PD
3.3V
GND
TX
RX
USB power can be used for the arduino but is not sufficient for the 3.3V WiFi adapter
Optional USB serial adapter used for debugging

Este posibil să se observe pe placa de schimbare a nivelului, deoarece pentru că IO digital de la arduino este de 5V, dar modulul WiFi are nevoie de 3.3V. Modulul Adafruit este o modalitate ușoară de a se alătura celor două. Senzorii de temperatură folosesc un protocol cu trei fire, care permite conectarea mai multor senzori în paralel și care să aibă adrese proprii. Există o bibliotecă de software care se ocupă de acest lucru.

Portul serial hardware este necesar pentru modulul WiFi, care are nevoie de 115200 baud. Aceasta înseamnă că nu se poate încărca o schiță nouă pe Arduino în timp ce aceasta este conectată la modulul WiFi. Dacă pinii TXD și RXD sunt conectați în mod corect la Arduino totul este bine.

În schița de mai sus observăm că WiFi încearcă să stabilească o conexiune TCP simplă către o adresă IP de server și un port la alegere. Odată ce conexiunea este stabilită, aceasta verifică senzorii de temperatură la fiecare 10 secunde și trimite temperaturile către server în format JSON, astfel: {"temp": [22.63,22.81]}. Este nevoie de un server TCP care să citească adresa IP și de asemenea și portul ales, desigur.

```
#include <SoftwareSerial.h>
#include <OneWire.h>
#include <DallasTemperature.h>

#define SSID       "MyHomeSSID"
#define PASS        "MyPassword"
#define TARGET_IP   "192.168.1.xx"
```

```
#define TARGET_PORT 5000
#define TEMPERATURE_PIN 9

SoftwareSerial dbgSerial(10,11); // RX,TX
OneWire wire(TEMPERATURE_PIN);
DallasTemperature sensors(&wire);

void setup()
{
   // WiFi module needs fast serial, so must use the hardware port which is also used for
   // uploading sketches
   Serial.begin(115200);
   Serial.setTimeout(5000);

   // For debugging, we therefore need a software serial port.  This can be much slower.
   dbgSerial.begin(9600);
   dbgSerial.println("Starting");

  delay(1000);

  // Connect to the wirelsess network
  dbgSerial.println("Joining network...");
  Serial.print("AT+CWJAP="");
  Serial.print(SSID);
  Serial.print("","");
  Serial.print(PASS);
  Serial.println(""");
  receive();

  // Just check that the WiFi module is joined to the network
  dbgSerial.println("Check connection...");
  Serial.println("AT+CWJAP?");
  receive();

  dbgSerial.println("Initialising sensors...");
  sensors.begin();


  dbgSerial.println("Connecting to server...");
  Serial.print("AT+CIPSTART="TCP","");
  Serial.print(TARGET_IP);
  Serial.print("",");
  Serial.print(TARGET_PORT);
  receive();
  delay(5000);

  dbgSerial.println("Ready to rumble!");
}

int incomingByte=0;
bool echoLocal = true;

// Get the data from the WiFi module and send it to the debug serial port
void receive(){
  delay(500);
  while (Serial.available() >0) {
    incomingByte = Serial.read();
    dbgSerial.write(incomingByte);
  }
  dbgSerial.println();
}

char temp1[10];
char temp2[10];

void loop()
{

  // call sensors.requestTemperatures() to issue a global temperature
  // request to all devices on the bus
  dbgSerial.print("Requesting temperatures...");
  sensors.requestTemperatures(); // Send the command to get temperatures
  dbgSerial.println("DONE");

  dtostrf(sensors.getTempCByIndex(0),1,2,temp1);
  dtostrf(sensors.getTempCByIndex(1),1,2,temp2);

  String json="{"temp":[" + String(temp1) + "," + String(temp2) + "]}";
  dbgSerial.print("Sending ");
  dbgSerial.println(json);

  // Send the data to the WiFi module
  Serial.print("AT+CIPSEND=");
  Serial.println(json.length());
  delay(500);
  Serial.print(json);
  receive();


  delay(10000);
}
```

4. ARDUINO SOFTWARE RC car

Processing Code

```
import processing.serial.*;

Serial myPort;  // Create object from Serial class
int[] KeyArray = new int[4];
int PrtChose = 6; //number of the serial port, starts at 0
PFont f; //initailize a font

void setup() {
  size(200, 300);
  noStroke();
  background(0);

  f = loadFont("ComicSansMS-48.vlw"); //Load Font
  textFont(f, 15);                  //Specify font type
  print(Serial.list());
  String[] Avail = Serial.list();
  for(int i =0; i < Avail.length; i++)
  {
    text(Avail[i], 10, 20*i+80); //print text of avaliable serial ports to app
  }

  fill(0,255,0);
  text(Avail[PrtChose], 10, 20*PrtChose+80); //highlite the port in use

  String portName = Serial.list()[PrtChose];
  myPort = new Serial(this, portName, 115200);
}

void draw()
{
  //chnage colors of 4 ellipses according to what keys are pressed
  if (KeyArray[0] == 1)
  {
    fill(0, 255, 0);
  }
  Else
  {
    fill(255, 0, 0);
  }
  ellipse(100, 20, 5, 5);

  if (KeyArray[1] == 1)
  {
    fill(0, 255, 0);
  }
  Else
  {
    fill(255, 0, 0);
  }
  ellipse(100, 40, 5, 5);
  if (KeyArray[2] == 1)
  {
    fill(0, 255, 0);
  }
  Else
  {
    fill(255, 0, 0);
  }
  ellipse(120, 40, 5, 5);

  if (KeyArray[3] == 1)
  {
    fill(0, 255, 0);
  }
  Else
  {
    fill(255, 0, 0);
  }
  ellipse(80, 40, 5, 5);
}

void keyReleased() //key released
{
  if (key == 'w') //fwd, fwd has been let up, so set stop fwd message
  {
    KeyArray[0] = 0; //no longer pressed
  }
  if (key == 's') //down
  {
    KeyArray[1] = 0;
  }
  if (key == 'd') //right
  {
    KeyArray[2] = 0;
  }
  if (key == 'a') //left
  {
    KeyArray[3] = 0;
  }

  writeMsg(); //send updated state over serial
}

void keyPressed() //key pressed
{
  if (key == 'w') //fwd
  {
    KeyArray[0] = 1; //fwd key pressed down, set forward command
  }
  if (key == 's') //down
  {
    KeyArray[1] = 1;
  }
  if (key == 'd') //right
  {
    KeyArray[2] = 1;
  }
  if (key == 'a') //left
  {
    KeyArray[3] = 1;
  }

  writeMsg();
}

void writeMsg()
{
  //so both fwd/back, lft/rght dont activate at same time
  if(KeyArray[0] == 1 && KeyArray[1] == 1)
  {
    KeyArray[0] = 0;
    KeyArray[1] = 0;
  }
  if(KeyArray[2] == 1 && KeyArray[3] == 1)
  {
    KeyArray[2] = 0;
    KeyArray[3] = 0;
  }
  //<0,0,0,0>   <fwd,bck,rght,Lft>
  myPort.write("<" + KeyArray[0] + "," + KeyArray[1] + "," + KeyArray[2] + "," + KeyArray[3] + ">"); //write to serial
  println("<" + KeyArray[0] + "," + KeyArray[1] + "," + KeyArray[2] + "," + KeyArray[3] + ">"); //display in console
}
```

ARDUINO CODE

```
int started = 0;
char inData[10];
int ended = 0;
char index = 0;
int final = 0;

boolean Fwd = 0;
boolean Bck = 0;
boolean Rght = 0;
boolean Lft = 0;

void setup()
{
  Serial.begin(115200);
  pinMode(13,OUTPUT);
}
void loop()
{
  GetBluData(); //input like: <0,0,0,0> then splits and writes values to F,B,R,L
  TriggerBtn(2,Fwd); //set digital pin 2 to value of Fwd ( HIGH or LOW)
  TriggerBtn(3,Bck);
  TriggerBtn(4,Rght);
  TriggerBtn(5,Lft);
 /*
  Serial.print(Fwd);
  Serial.print(",");
  Serial.print(Bck);
  Serial.print(",");sss
  Serial.print(Rght);
  Serial.print(",");
  Serial.println(Lft);
  */
}

void TriggerBtn(int PinNum, boolean ButtonState)
{
  if(ButtonState)
  {
    //turns on the button connected to the pin
    digitalWrite(13,HIGH); //some key has been pressed, LED to show it

    digitalWrite(PinNum, LOW); // PinNum is the number of the digital pin
    pinMode(PinNum, OUTPUT);  // Pull the signal low to activate button
  }
  Else
  {
     //releases button connected to the pin
    digitalWrite(13, LOW); //no key is being pressed, LED to show it

    pinMode(PinNum, INPUT);  // Release the button.
  }
}

void GetBluData()
{
  while(Serial.available() )
  {
  //finds < and >, the beginning and end of command
    char aChar = Serial.read();
    if(aChar == '<')
    {
      started = true;
      index = 0;
      inData[index] = '\0';
    }
    else if(aChar == '>')
    {
      ended = true;
    }

    else if(started)
    {
      inData[index] = aChar;
      index++;
      inData[index] = '\0';
    }

    else if (aChar =='*')
    {
      final = true;
    }
  }
  if(started && ended)
  {
    const char*  strDelimiter = ",";

    char* p;
    //<0,0,0,0>

 //splits to individual ints
    if ( p = strtok(inData, strDelimiter) )
    {
      Fwd = atoi(p);
    }
    if ( p = strtok(NULL, strDelimiter) )
    {
      Bck = atoi(p);
    }
    if ( p = strtok(NULL, strDelimiter) )
    {
      Rght = atoi(p);
    }
    if ( p = strtok(NULL, strDelimiter) )
    {
      Lft = atoi(p);
    }

    // Get ready for the next time
    started = false;
    ended = false;
    index = 0;
    inData[index] = '\0';
  }

}
```

```
# UTF-8 supported.

# The name of your library as you want it formatted
name = Arduino (Firmata)

# List of authors. Links can be provided using the syntax [author name](url)
authors = [David A. Mellis](http://dam.mellis.org/)

# A web page for your library, NOT a direct link to where to download it
url = http://arduino.cc/playground/Interfacing/Processing

# The category of your library, must be one (or many) of the following:
#   "3D"            "Animation"     "Compilations"      "Data"
#   "Fabrication"   "Geometry"      "GUI"               "Hardware"
#   "I/O"           "Language"      "Math"              "Simulation"
#   "Sound"         "Utilities"     "Typography"        "Video & Vision"
#
# If a value other than those listed is used, your library will listed as "Other."
categories = Hardware

# A short sentence (or fragment) to summarize the library's function. This will be
# shown from inside the PDE when the library is being installed. Avoid repeating
# the name of your library here. Also, avoid saying anything redundant like
# mentioning that its a library. This should start with a capitalized letter, and
# end with a period.
sentence = Controls Arduino boards running the Firmata firmware.

# Additional information suitable for the Processing website. The value of
# 'sentence' always will be prepended, so you should start by writing the
# second sentence here. If your library only works on certain operating systems,
# mention it here.
paragraph =  Works with the StandardFirmata example included in the Arduino software distribution.  To use
Firmata with other software, see [the Firmata github repository](https://github.com/firmata/arduino)

# Links in the 'sentence' and 'paragraph' attributes can be inserted using the
# same syntax as for authors. That is, [here is a link to Processing](http://processing.org/)


# A version number that increments once with each release. This
# is used to compare different versions of the same library, and
# check if an update is available. You should think of it as a
# counter, counting the total number of releases you've had.
version = 9  # This must be parsable as an int

# The version as the user will see it. If blank, the version attribute will be used here
prettyVersion = 9 # This is treated as a String


/*__________________________________________________________________________*/
/*  KLL-engineering                                      */
/*__________________________________________________________________________*/

// DIMMER_V0.1
// DIMMER_V0.2

// change memory for power on off status
// because from OFF state with long tip dimmer is working but then
// need 2 short tip to switch off again

// DIMMER V0.3   for MEGA

char prev[5]="V0.3";
                                      // wiring


const int ledPin    = 13;                       // arduino board LED "L"
                                      // ATmega328P PWM outputs: 3,5,6,9,10,11
                                      // serial communication
long ser0bd = 115200;                      // 300, 1200, 2400, 9600, 14400, 19200, 28800, 38400, 57600, 115200

// for main loop timecheck
 unsigned long Ltime;                         // milliseconds   works for about 50 days
 unsigned long Ltime_old;                       // milliseconds memory last scan
 unsigned long Ltime_delta;                      // milliseconds delta from last scan

                                      // L___ for calc actual values ( from millis )
                                      // I___ for user set timer, I1__ for ON, I0__ for OFF,
 int Ldays, I1days, I0days;
 int Lhours, I1hours, I0hours;
 int Lminutes, l1minutes, l0minutes;
 int Lseconds, l1seconds, l0seconds;
                                   // loop operating system memory
  unsigned long loopcount =0;                    // cycles
  unsigned long loopreport=1000000;              // reset and time diag

  int exec1count, exec2count, exec3count, exec4count, exec5count, exec6count, exec7count, exec8count,
exec9count; // now per job, only INT
  int exectim = 100;                        // default executer, if 1000000 loop need about 10sec, every 100 loops
would mean 1msec jobtimer
                                   // but even all 9 loops are enabled and all 9 run with this timer, they not are
executed
                                   // all in one loop and then exectim loops only counting.
                                   // each job has a loopcounterlimit of exectim + jobnumber
                                   // so we will have rotating time slices for each job,
                                   // !! jobs should not depend on each other in a time critical way !!

  boolean exec1conti = true;                   //   Ch1
  boolean exec2conti = true;                   //   Ch2
  boolean exec3conti = true;                   //   Ch3
  boolean exec4conti = true;                   //   Ch4
  boolean exec5conti = true;                   //   Ch5
  boolean exec6conti = true;                   //   Ch6
  boolean exec7conti = true;                   //   sinus ramp
  boolean exec8conti = false;                  //
  boolean exec9conti = false;                  //

// diagnostic print to USB
  boolean debugb = true;                      // push button diag

// variables for hardware pwm output to LT 3060 LED power repeater or MOSFET
 boolean pwminv = false;                      // invert pwm to 255 - signal
                                   // TRUE for LT3060 control input, FALSE for MOSFET gate 100R

 int PBexecuter = 500;                       // check on push button ( 5msec )
 int PBtipmin = 50, PBtipmax =1500;               // millis pressed, a tip must be between 50millis and 1.5 sec
pressed
 int PBdtipmin = 100, PBdtipmax =1000;            // valid pause time between double click ( double click speed )

// sinus ramp
 int rampcount;
 const float rad = 0.0245;                     // 2 * PI / 256
 byte rampsin;                             // byte format for RGB
 int sinexecuter = 1000;                      // same timing as ramp jobs


// channel (_A)
  const byte pwm_A = 2;                       // Dout PWM pin
  const byte buttonpin_A = 22;                   // D in for field push button TMPB DIMMER 1

  byte hw_A = 0;                            // variables to hold color values  start with OFF
  byte hw_Amem = 0;
  int moody_A = LOW, moody_Amem = LOW;             // variables for button state
  boolean PBtipA = false, PBtipAmem = false;        // internal procedure variables
  boolean onetipA = false, twotipA = false, longtipAstart = false, longtipAend = false; // handover for prog control,
  boolean powerstateA = false;                   // for switch light ON OFF by double click
  unsigned long PBLtimeA;                      // milliseconds   works for about 50 days
  unsigned long PBLtimeA_old;                    // milliseconds memory last scan
  unsigned long PBLtimeA_delta;                   // milliseconds delta from last scan

// channel (_B)
  const byte pwm_B = 3;                          // Dout PWM pin
  const byte buttonpin_B = 24;                     // D in for field push button TMPB DIMMER 1

  byte hw_B = 0;                          // variables to hold color values  start with HALF WHITE
  byte hw_Bmem = 0;
  int moody_B = LOW, moody_Bmem = LOW;             // variables for button state
  boolean PBtipB = false, PBtipBmem = false;       // internal procedure variables
  boolean onetipB = false, twotipB = false, longtipBstart = false, longtipBend = false; // handover for prog control,
  boolean powerstateB = false;                    // for switch light ON OFF by double click
  unsigned long PBLtimeB;                        // milliseconds   works for about 50 days
  unsigned long PBLtimeB_old;                    // milliseconds memory last scan
  unsigned long PBLtimeB_delta;                   // milliseconds delta from last scan

// channel (_C)
  const byte pwm_C = 4;                          // Dout PWM pin
  const byte buttonpin_C = 26;                     // D in for field push button TMPB DIMMER 1

  byte hw_C = 0;                          // variables to hold color values  start with HALF WHITE
  byte hw_Cmem = 0;
  int moody_C = LOW, moody_Cmem = LOW;             // variables for button state
  boolean PBtipC = false, PBtipCmem = false;       // internal procedure variables
  boolean onetipC = false, twotipC = false, longtipCstart = false, longtipCend = false; // handover for prog control,
  boolean powerstateC = false;                    // for switch light ON OFF by double click
  unsigned long PBLtimeC;                        // milliseconds   works for about 50 days
  unsigned long PBLtimeC_old;                    // milliseconds memory last scan
  unsigned long PBLtimeC_delta;                   // milliseconds delta from last scan

// channel (_D)
  const byte pwm_D = 5;                          // Dout PWM pin
  const byte buttonpin_D = 28;                     // D in for field push button TMPB DIMMER 1

  byte hw_D = 0;                          // variables to hold color values  start with HALF WHITE
  byte hw_Dmem = 0;
  int moody_D = LOW, moody_Dmem = LOW;             // variables for button state
  boolean PBtipD = false, PBtipDmem = false;       // internal procedure variables
  boolean onetipD = false, twotipD = false, longtipDstart = false, longtipDend = false; // handover for prog control,
  boolean powerstateD = false;                    // for switch light ON OFF by double click
  unsigned long PBLtimeD;                        // milliseconds   works for about 50 days
  unsigned long PBLtimeD_old;                    // milliseconds memory last scan
  unsigned long PBLtimeD_delta;                   // milliseconds delta from last scan


 // channel (_E)
  const byte pwm_E = 6;                          // Dout PWM pin
  const byte buttonpin_E = 30;                     // D in for field push button TMPB DIMMER 1
// use A0 as D in

  byte hw_E = 0;                          // variables to hold color values  start with HALF WHITE
  byte hw_Emem = 0;
  int moody_E = LOW, moody_Emem = LOW;             // variables for button state
  boolean PBtipE = false, PBtipEmem = false;       // internal procedure variables
  boolean onetipE = false, twotipE = false, longtipEstart = false, longtipEend = false; // handover for prog control,
  boolean powerstateE = false;                    // for switch light ON OFF by double click
  unsigned long PBLtimeE;                          // milliseconds   works for about 50 days
  unsigned long PBLtimeE_old;                        // milliseconds memory last scan
  unsigned long PBLtimeE_delta;                       // milliseconds delta from last scan

// channel (_F)
  const byte pwm_F = 8;                          // Dout PWM pin
  const byte buttonpin_F = 32;                      // D in for field push button TMPB DIMMER 1

  byte hw_F = 0;                          // variables to hold color values  start with HALF WHITE
  byte hw_Fmem = 0;
  int moody_F = LOW, moody_Fmem = LOW;              // variables for button state
  boolean PBtipF = false, PBtipFmem = false;        // internal procedure variables
  boolean onetipF = false, twotipF = false, longtipFstart = false, longtipFend = false; // handover for prog control,
  boolean powerstateF = false;                      // for switch light ON OFF by double click
  unsigned long PBLtimeF;                          // milliseconds   works for about 50 days
  unsigned long PBLtimeF_old;                        // milliseconds memory last scan
  unsigned long PBLtimeF_delta;                       // milliseconds delta from last scan

// MAGA has more PWM

// channel (_G)
  const byte pwm_G = 9;                          // Dout PWM pin
  const byte buttonpin_G = 34;                      // D in for field push button TMPB DIMMER 1

  byte hw_G = 0;                          // variables to hold color values  start with HALF WHITE
  byte hw_Gmem = 0;
  int moody_G = LOW, moody_Gmem = LOW;              // variables for button state
  boolean PBtipG = false, PBtipGmem = false;        // internal procedure variables
  boolean onetipG = false, twotipG = false, longtipGstart = false, longtipGend = false; // handover for prog control,
  boolean powerstateG = false;                      // for switch light ON OFF by double click
  unsigned long PBLtimeG;                          // milliseconds   works for about 50 days
  unsigned long PBLtimeG_old;                        // milliseconds memory last scan
  unsigned long PBLtimeG_delta;                       // milliseconds delta from last scan

// channel (_H)
  const byte pwm_H = 10;                          // Dout PWM pin
  const byte buttonpin_H = 36;                      // D in for field push button TMPB DIMMER 1

  byte hw_H = 0;                          // variables to hold color values  start with HALF WHITE
  byte hw_Hmem = 0;
  int moody_H = LOW, moody_Hmem = LOW;              // variables for button state
  boolean PBtipH = false, PBtipHmem = false;        // internal procedure variables
  boolean onetipH = false, twotipH = false, longtipHstart = false, longtipHend = false; // handover for prog control,
  boolean powerstateH = false;                      // for switch light ON OFF by double click
  unsigned long PBLtimeH;                          // milliseconds   works for about 50 days
  unsigned long PBLtimeH_old;                        // milliseconds memory last scan
  unsigned long PBLtimeH_delta;                       // milliseconds delta from last scan

// channel (_I)
  const byte pwm_I = 11;                          // Dout PWM pin
  const byte buttonpin_I = 38;                      // D in for field push button TMPB DIMMER 1

  byte hw_I = 0;                          // variables to hold color values  start with HALF WHITE
  byte hw_Imem = 0;
  int moody_I = LOW, moody_Imem = LOW;              // variables for button state
  boolean PBtipI = false, PBtipImem = false;        // internal procedure variables
  boolean onetipI = false, twotipI = false, longtipIstart = false, longtipIend = false; // handover for prog control,
  boolean powerstateI = false;                    // for switch light ON OFF by double click
  unsigned long PBLtimeI;                        // milliseconds   works for about 50 days
  unsigned long PBLtimeI_old;                     // milliseconds memory last scan
  unsigned long PBLtimeI_delta;                    // milliseconds delta from last scan

// channel (_J)
  const byte pwm_J = 12;                         // Dout PWM pin
  const byte buttonpin_J = 40;                    // D in for field push button TMPB DIMMER 1

  byte hw_J = 0;                            // variables to hold color values  start with HALF WHITE
  byte hw_Jmem = 0;
  int moody_J = LOW, moody_Jmem = LOW;              // variables for button state
  boolean PBtipJ = false, PBtipJmem = false;        // internal procedure variables
  boolean onetipJ = false, twotipJ = false, longtipJstart = false, longtipJend = false; // handover for prog control,
  boolean powerstateJ = false;                    // for switch light ON OFF by double click
  unsigned long PBLtimeJ;                        // milliseconds   works for about 50 days
  unsigned long PBLtimeJ_old;                     // milliseconds memory last scan
  unsigned long PBLtimeJ_delta;                    // milliseconds delta from last scan

// channel (_K)
  const byte pwm_K = 13;                         // Dout PWM pin
  const byte buttonpin_K = 42;                    // D in for field push button TMPB DIMMER 1

  byte hw_K = 0;                            // variables to hold color values  start with HALF WHITE
  byte hw_Kmem = 0;
  int moody_K = LOW, moody_Kmem = LOW;              // variables for button state
  boolean PBtipK = false, PBtipKmem = false;        // internal procedure variables
  boolean onetipK = false, twotipK = false, longtipKstart = false, longtipKend = false; // handover for prog control,
  boolean powerstateK = false;                    // for switch light ON OFF by double click
  unsigned long PBLtimeK;                        // milliseconds   works for about 50 days
  unsigned long PBLtimeK_old;                     // milliseconds memory last scan
  unsigned long PBLtimeK_delta;                    // milliseconds delta from last scan

// channel (_L)
  const byte pwm_L = 7;                         // Dout PWM pin
  const byte buttonpin_L = 44;                    // D in for field push button TMPB DIMMER 1

  byte hw_L = 0;                            // variables to hold color values  start with HALF WHITE
  byte hw_Lmem = 0;
  int moody_L = LOW, moody_Lmem = LOW;              // variables for button state
  boolean PBtipL = false, PBtipLmem = false;        // internal procedure variables
  boolean onetipL = false, twotipL = false, longtipLstart = false, longtipLend = false; // handover for prog control,
  boolean powerstateL = false;                    // for switch light ON OFF by double click
  unsigned long PBLtimeL;                        // milliseconds   works for about 50 days
  unsigned long PBLtimeL_old;                     // milliseconds memory last scan
  unsigned long PBLtimeL_delta;                    // milliseconds delta from last scan


                                  // setup
void setup() {
                                  // initialize the digital pin as an output.
                                  // Pin 13 has an LED connected on most Arduino boards:
  pinMode(ledPin, OUTPUT);                       // mainboard LED
  digitalWrite(ledPin, HIGH);                  // set the LED on, later this LED will show if one pushbutton is pressed

// CHANNEL A
  pinMode(pwm_A, OUTPUT);                         // PWM Dout pin to LED
  hwset(pwm_A,0);                    // set OFF
  // MOODY BUTTON
  pinMode(buttonpin_A, INPUT);              // D in push button to GND
  digitalWrite(buttonpin_A, HIGH);          // enable internal pullup resistor

// CHANNEL B
  pinMode(pwm_B, OUTPUT);                   // PWM Dout pin to LED
  hwset(pwm_B,0);                    // set OFF
  // MOODY BUTTON
  pinMode(buttonpin_B, INPUT);              // D in push button to GND
  digitalWrite(buttonpin_B, HIGH);          // enable internal pullup resistor

// CHANNEL C
  pinMode(pwm_C, OUTPUT);                   // PWM Dout pin to LED
  hwset(pwm_C,0);                    // set OFF
  // MOODY BUTTON
  pinMode(buttonpin_C, INPUT);              // D in push button to GND
  digitalWrite(buttonpin_C, HIGH);          // enable internal pullup resistor

// CHANNEL D
  pinMode(pwm_D, OUTPUT);                   // PWM Dout pin to LED
  hwset(pwm_D,0);                    // set OFF
  // MOODY BUTTON
  pinMode(buttonpin_D, INPUT);              // D in push button to GND
  digitalWrite(buttonpin_D, HIGH);          // enable internal pullup resistor

// CHANNEL E
  pinMode(pwm_E, OUTPUT);                   // PWM Dout pin to LED
  hwset(pwm_E,0);                    // set OFF
  // MOODY BUTTON
  pinMode(buttonpin_E, INPUT);              // D in push button to GND
  digitalWrite(buttonpin_E, HIGH);          // enable internal pullup resistor
//  pinMode(A0, INPUT);              // D in push button to GND
//  digitalWrite(A0, HIGH);          // enable internal pullup resistor

// CHANNEL F
  pinMode(pwm_F, OUTPUT);                   // PWM Dout pin to LED
  hwset(pwm_F,0);                    // set OFF
  // MOODY BUTTON
  pinMode(buttonpin_F, INPUT);              // D in push button to GND
  digitalWrite(buttonpin_F, HIGH);          // enable internal pullup resistor

// CHANNEL G
  pinMode(pwm_G, OUTPUT);                   // PWM Dout pin to LED
  hwset(pwm_G,0);                    // set OFF
  // MOODY BUTTON
  pinMode(buttonpin_G, INPUT);              // D in push button to GND
  digitalWrite(buttonpin_G, HIGH);          // enable internal pullup resistor

// CHANNEL H
  pinMode(pwm_H, OUTPUT);                   // PWM Dout pin to LED
  hwset(pwm_H,0);                    // set OFF
  // MOODY BUTTON
  pinMode(buttonpin_H, INPUT);              // D in push button to GND
  digitalWrite(buttonpin_H, HIGH);          // enable internal pullup resistor

// CHANNEL I
```

```
  pinMode(pwm_I, OUTPUT);                 // PWM Dout pin to LED
  hwset(pwm_I,0);                  // set OFF
  // MOODY BUTTON
  pinMode(buttonpin_I, INPUT);           // D in push button to GND
  digitalWrite(buttonpin_I, HIGH);      // enable internal pullup resistor

// CHANNEL J
  pinMode(pwm_J, OUTPUT);                 // PWM Dout pin to LED
  hwset(pwm_J,0);                  // set OFF
  // MOODY BUTTON
  pinMode(buttonpin_J, INPUT);           // D in push button to GND
  digitalWrite(buttonpin_J, HIGH);      // enable internal pullup resistor

// CHANNEL K
  pinMode(pwm_K, OUTPUT);                 // PWM Dout pin to LED
  hwset(pwm_K,0);                  // set OFF
  // MOODY BUTTON
  pinMode(buttonpin_K, INPUT);           // D in push button to GND
  digitalWrite(buttonpin_K, HIGH);      // enable internal pullup resistor

// CHANNEL L
  pinMode(pwm_L, OUTPUT);                 // PWM Dout pin to LED
  hwset(pwm_L,0);                  // set OFF
  // MOODY BUTTON
  pinMode(buttonpin_L, INPUT);           // D in push button to GND
  digitalWrite(buttonpin_L, HIGH);      // enable internal pullup resistor

// serial interface  only for diagnostic prints to terminal
  Serial.begin(ser0bd);
  Serial.println("kll-engineering on ARDUINO ");
  Serial.print(" 12 * (DIMMER 1 pwm channel, 1 PB ) ");
  Serial.println(prev);
}


// run
void loop() {

      // main RUN of CONTI JOBS ( semi parallel )

       if ( exec1conti )                          // JOB1
       {
         if (exec1count == 0)
         { // 1

         PB_read_A();
         PB_check_A();
         PB_prog_A();
         PB_ramp_A();

         PB_read_G();
         PB_check_G();
         PB_prog_G();
         PB_ramp_G();
         }
         exec1count = exec1count +1;
         if (exec1count == ( PBexecuter + 1 )) { exec1count = 0; }
```

```
}                                             // end if conti true

if ( exec2conti )                             // JOB2
{
  if (exec2count == 0)
  { // 2

  PB_read_B();
  PB_check_B();
  PB_prog_B();
  PB_ramp_B();

  PB_read_H();
  PB_check_H();
  PB_prog_H();
  PB_ramp_H();
  }
  exec2count = exec2count +1;
  if (exec2count == ( PBexecuter + 2 )) { exec2count = 0; }                  // job timer
}                                             // end if conti true

if ( exec3conti )                             // JOB3
{
  if (exec3count == 0)
  { // 3

  PB_read_C();
  PB_check_C();
  PB_prog_C();
  PB_ramp_C();

  PB_read_I();
  PB_check_I();
  PB_prog_I();
  PB_ramp_I();
  }
  exec3count = exec3count +1;
  if (exec3count == ( PBexecuter + 3 )) { exec3count = 0; }                  // job timer
}                                             // end if conti true

if ( exec4conti )                             // JOB4
{
  if (exec4count == 0)
  { // 4

  PB_read_D();
  PB_check_D();
  PB_prog_D();
  PB_ramp_D();

  PB_read_J();
  PB_check_J();
  PB_prog_J();
  PB_ramp_J();
  }
  exec4count = exec4count +1;
  if (exec4count == ( PBexecuter + 4 )) { exec4count = 0; }                  // job timer
```

```
}                                           // end if conti true

if ( exec5conti )                           // JOB5
{
  if (exec5count == 0)
  { //5

  PB_read_E();
  PB_check_E();
  PB_prog_E();
  PB_ramp_E();

  PB_read_K();
  PB_check_K();
  PB_prog_K();
  PB_ramp_K();
  }
  exec5count = exec5count +1;
  if (exec5count == (  PBexecuter + 5 )) { exec5count = 0; }               // job timer
}                                           // end if conti true

if ( exec6conti )                           // JOB6
{
  if (exec6count == 0)
  { // 6

  PB_read_F();
  PB_check_F();
  PB_prog_F();
  PB_ramp_F();

  PB_read_L();
  PB_check_L();
  PB_prog_L();
  PB_ramp_L();
  }
  exec6count = exec6count +1;
  if (exec6count == ( PBexecuter + 6 )) { exec6count = 0; }               // job timer
}                                           // end if conti true

if ( exec7conti )                           // JOB7
{
  if (exec7count == 0)
  { // 7
    // sinus ramp is actually a (1 - cos())*128 => 0 .. 255 .. 0
    rampcount = rampcount + 2;  // speed up
    if (rampcount >= 256) { rampcount = 0;}
    rampsin = byte( (1.0 - cos(float(rampcount) * rad)) * 128 );
    //if (debugp) {
    //                  Serial.print("rampcount: ");
    //                  Serial.print(rampcount);
    //                  Serial.print("rampsin: ");
    //                  Serial.println(rampsin,DEC);
    //          }  // if debug

  }
  exec7count = exec7count +1;
```

```
     if (exec7count == ( sinexecuter + 7 )) { exec7count = 0; }              // job timer

     }                                                  // end if conti true


   if ( exec8conti )                                  // JOB8
   {
     if (exec8count == 0)
     { //8

     }
     exec8count = exec8count +1;
     if (exec8count == ( exectim + 8 )) { exec8count = 0; }               // job timer
   }                                                  // end if conti true

   if ( exec9conti )                                  // JOB9
   {

     if (exec9count == 0)
     { // 9

     }
     exec9count = exec9count +1;
     if (exec9count == ( exectim + 9 )) { exec9count = 0; }                 // job timer about 1 msec
   }                                                  // end if conti true


      loopcount = loopcount + 1;                                 // for check cycle time

      if ( loopcount == loopreport ) {                           // of 4 294 967 295 unsigned long
                          loopcount = 0;                // reset
                          if (debugb) {
                                    print_time();
                                    print_variables();
                                  }  // if debug

                         } // end if loopcount

               digitalWrite(ledPin, ( moody_A || moody_B || moody_C || moody_D || moody_E || moody_F ||
moody_G || moody_H || moody_I || moody_J || moody_K || moody_L ) );               // show on board LED ON if a
button is pressed, for loopcheck

   } // end loop
```