Report 5

Subject: Embedded Systems

Topic: Operating systems - sequential systems

Student: Marusic Diana, FAF-171

Teacher: Bragarenco Andrei

Content

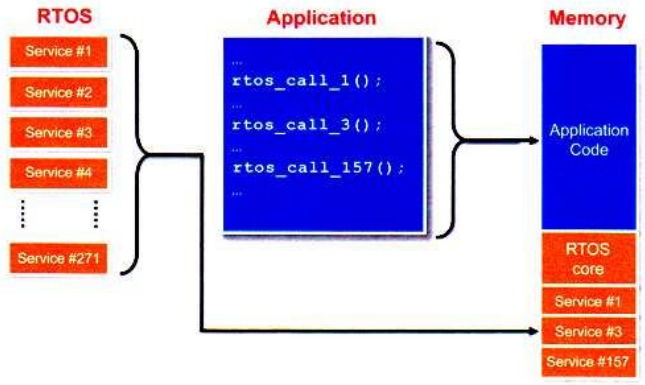
Domain

Operating System

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

RTOS - Real Time Operating System

A Real Time Operating System, commonly known as an RTOS, is a software component that rapidly switches between tasks, giving the impression that multiple programs are being executed at the same time on a single processing core.



Difference between OS and RTOS

The difference between an OS (Operating System) such as Windows or Unix and an RTOS (Real Time Operating System) found in embedded systems, is the response time to external events. OS's typically provide a non-deterministic, soft real time response, where there are no guarantees as to when each task will complete, but they will try to stay responsive to the user. An RTOS differs in that it typically provides a hard real time response, providing a fast, highly deterministic reaction to external events.

Task

In computing, a task is a unit of execution or a unit of work

Tasks and priorities

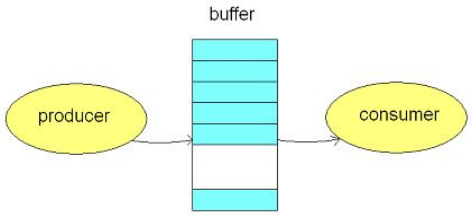
In a pre-emptive system each Task is given an individual priority value. The faster the required response, the higher the priority level assigned. When working in pre-emptive mode, the task chosen to execute is the highest priority task that is able to execute. This results in a highly responsive system

Multitasking

In computing terminology, multitasking is a concept of executing multiple tasks or processes by a computer over a period of time.

Provider-Consumer (Producer-Consumer)

The Provider-Consumer/ Producer-consumer design pattern is a pre-designed solution to separate the two main components by placing a queue in the middle.

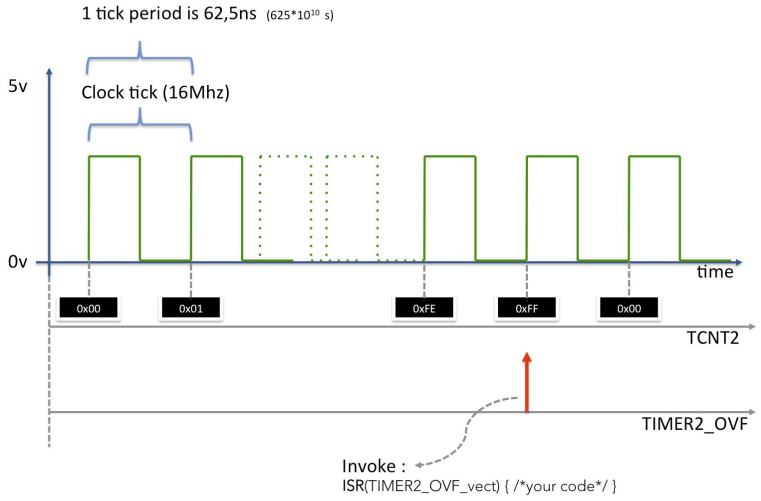


Interrupts

Interrupt is a process of letting the computer know that a different task is in need for its service. Interrupts play an important role. When a task interrupts the computer, it puts the present task on hold, executes the new task and returns back to the original task.

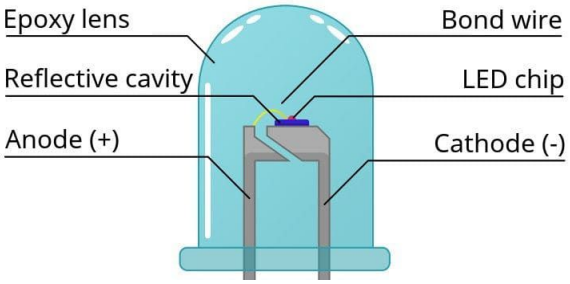
Timer interrupt

Perhaps the most important interrupt for operating system design is the "timer interrupt", which is emitted at regular intervals by a timer chip. A software interrupt, also called a processor generated interrupt, is generated by the processor executing a specific instruction.



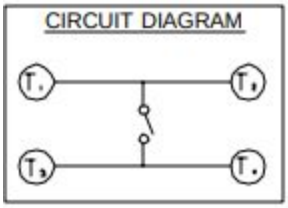
Component description

LED (Light-Emitting Diode)



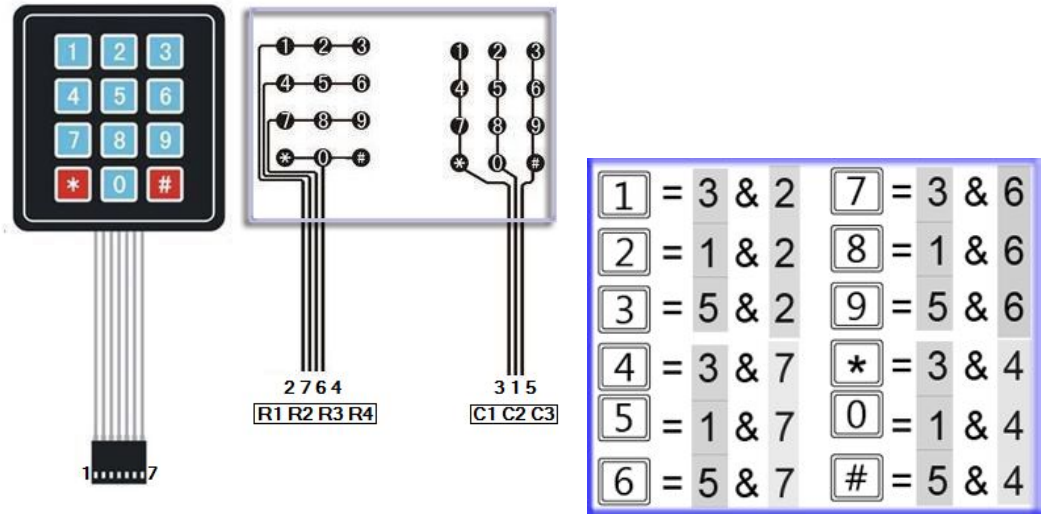
LED is a semiconductor light source that emits light when current flows through it. Electrons in the semiconductor recombine with electron holes, releasing energy in the form of photons. The color of the light (corresponding to the energy of the photons) is determined by the energy required for electrons to cross the band gap of the semiconductor. White light is obtained by using multiple semiconductors or a layer of light-emitting phosphor on the semiconductor device.
There are some advantages of LEDs over incandescent light bulbs. Unlike incandescent bulbs, LEDs don't have filaments that burn out, they use less electricity, and they don't get very hot. LEDs last just as long as a standard transistor. The lifespan of an LED surpasses the short life of an incandescent bulb by thousands of hours.

Button (push button)



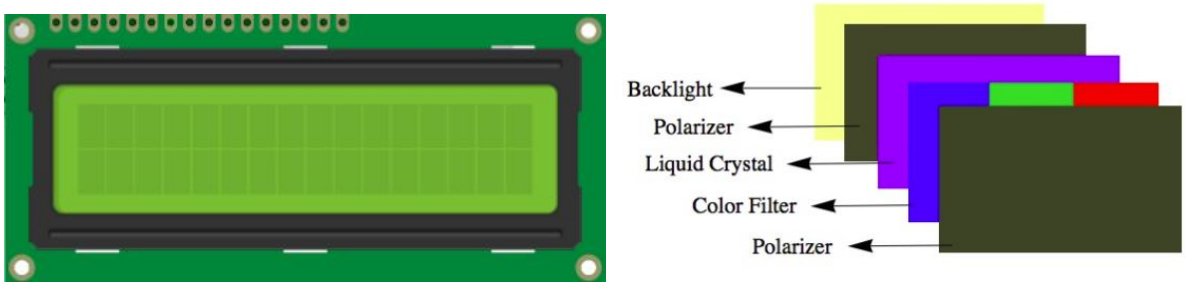
A push-button or simply button is a simple switch mechanism to control some aspect of a machine or a process. Buttons are typically made out of hard material, usually plastic or metal. These buttons usually contain a spring to return to their un-pushed state.

Keypad matrix (3x4)



A keypad is one of the most commonly used input devices in microprocessor applications. In a standard keypad wired as an X-Y switch matrix, open switches connect a row to a column when pressed. A keypad with 12 keys (3x4) is wired as 3 columns by 4 rows.

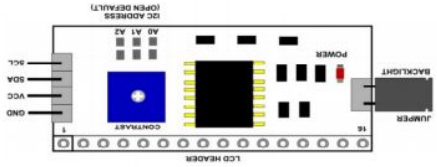
LCD (Liquid Crystal Display)



LCD (Liquid Crystal Display) is a type of flat panel display which uses liquid crystals in its primary form of operation. They work on the principle of blocking light.

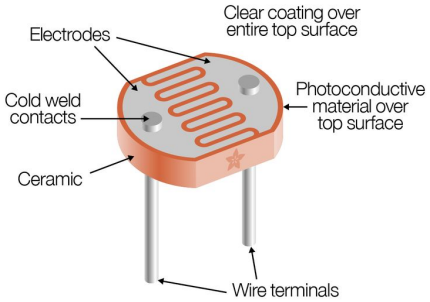
Specifically, an LCD is made of two pieces of polarized glass (also called substrate) that contain a liquid crystal material between them. A backlight creates light that passes through the first substrate. At the same time, electrical currents cause the liquid crystal

I2C serial adaptor for LCD display



The LCD screen module for Arduino needs 16 pins to connect to Arduino board. This leaves less pins available for other components to connect to the development board. The **Converter Board I2C Serial Interface Module for LCD Display** makes it possible to control LCD using only 4 pins and allowing I2C interface communication.

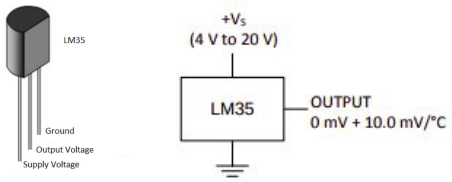
Photoresistor (LDR - Light Dependant Resistor)



A **photoresistor** is an active component that decreases resistance with respect to receiving luminosity on the component's sensitive surface. The resistance of a photoresistor decreases with increase in incident light intensity; in other words, it exhibits photoconductivity.
In the dark, a photoresistor can have a resistance as high as several megaohms (MΩ), while in the light, a photoresistor can have a resistance as low as a few hundred ohms. If incident light on a photoresistor exceeds a certain frequency, photons absorbed by the semiconductor give bound electrons enough energy to jump into the conduction band. The resulting free electrons (and their hole partners) conduct electricity, thereby lowering resistance.

LM35 temperature sensor

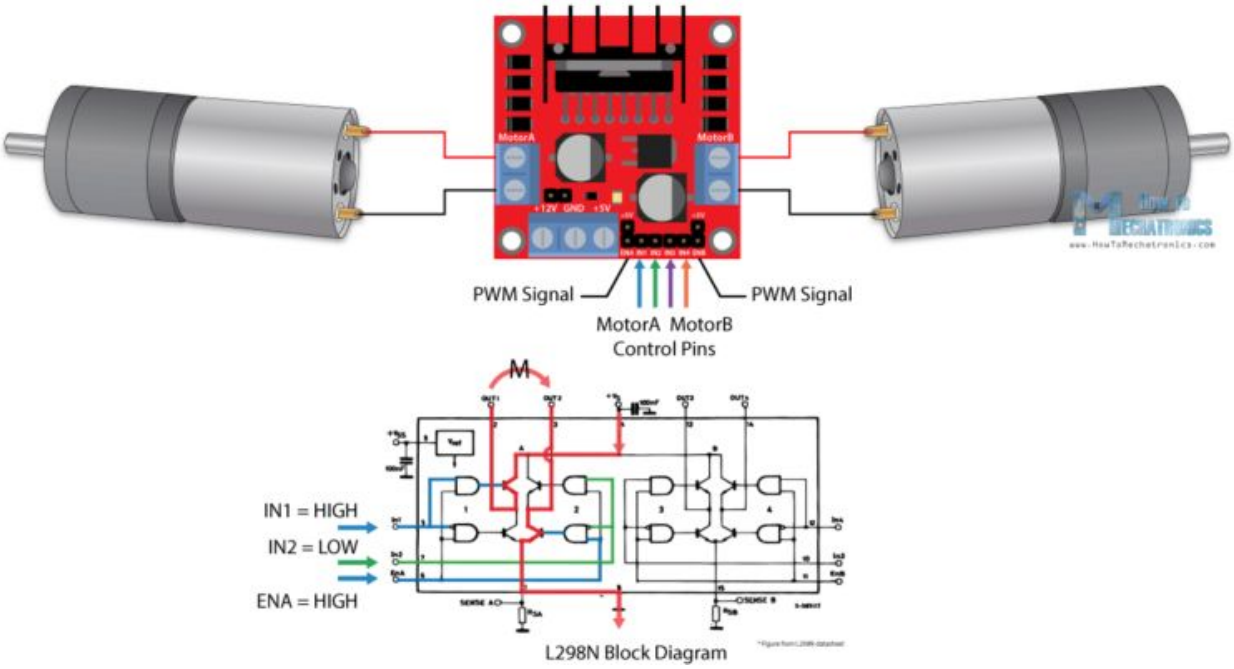
The LM35 series are precision integrated-circuit temperature devices with an output voltage linearly proportional to the Centigrade temperature.The LM35 device does not require any external calibration or trimming to provide typical accuracies of ±¼°C at room temperature and ±¾°C over a full −55°C to 150°C temperature range.



DC Motor

A DC(direct current) motor is any of a class of rotary electrical motors that converts direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current in part of the motor.
A DC motor consists of an stator, an armature, a rotor and a commutator with brushes. Opposite polarity between the two magnetic fields inside the motor cause it to turn. DC motors are the simplest type of motor and are used in household appliances, such as electric razors, and in electric windows in cars.

L298n motor driver



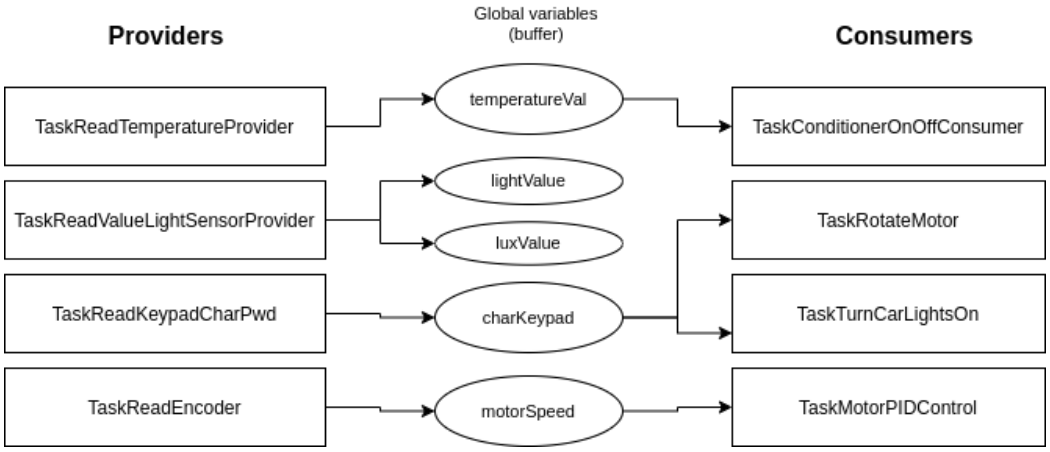
The L298N is a dual H-Bridge motor driver which allows speed and direction control of two DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A.
The L298N module has two screw terminal blocks for the motor A and B, and another screw terminal block for the Ground pin, the VCC for motor and a 5V pin which can either be an input or output. This depends on the voltage used at the motors VCC. The module has an onboard 5V regulator which is either enabled or disabled using a jumper. If the motor supply voltage is up to 12V we can enable the 5V regulator and the 5V pin can be used as output, for example for powering our Arduino board. But if the motor voltage is greater than 12V we must disconnect the jumper because those voltages will cause damage to the onboard 5V regulator. In this case the 5V pin will be used as input as we need to connect it to a 5V power supply in order for the IC to work properly.
Next are the logic control inputs. The Enable A and Enable B pins are used for enabling and controlling the speed of the motor. If a jumper is present on this pin, the motor will be enabled and work at maximum speed, and if we remove the jumper we can connect a PWM input to this pin and in that way control the speed of the motor. If we connect this pin to a Ground the motor will be disabled.

Arduino Mega

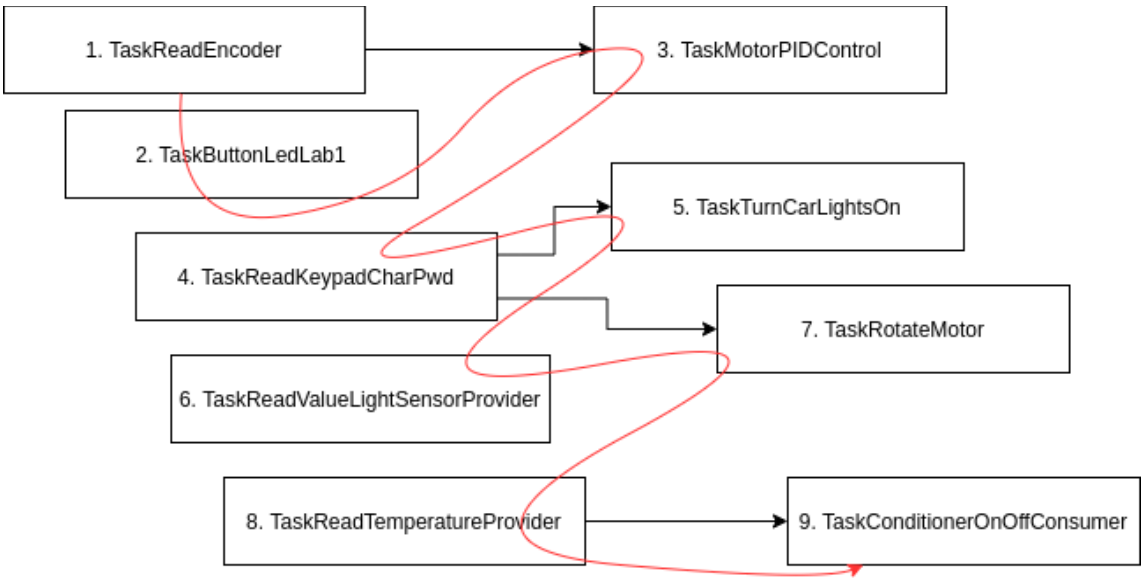
The Arduino Mega 2560 is a microcontroller board based on the ATmega2560. It has 54 digital input/output pins (of which 15 can be used as PWM outputs), 16 analog inputs, 4 UARTs (hardware serial ports), a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller. The Mega 2560 board is compatible with most shields designed for the Uno.

Implementation

The implementation of this laboratory work consisted of connecting the components from all previous laboratories to the same Arduino and providing the concept of multitasking by using the timper-api library that implements timer interrupts on Arduino. The tasks were structured by using the Provider-Consumer pattern, and using global variables as buffers, or intermediaries between tasks. Below is the Providers-Consumers diagram:



According to the provider-consumer diagram, the ordering of tasks was decided and offsets and recurrences for each task were determined. Below are presented all the tasks, the tasks order according to the red arrow as well as an explanation for the reasoning of why each offset was used.



Offsets and recurrences - reasoning why

The recurrences and offsets established for each task are presented below:

Nr.	Task	Code	Recurrence(ms)	Offset (ms)
1	TaskReadEncoder	READ_ENCODER	1	0
2	TaskButtonLedLab1	BTN_LED_1	1000	300
3	TaskMotorPIDControl	MOTOR_PID	1000	500
4	TaskReadKeypadCharPwd	GET_KPD	130	1000
5	TaskTurnCarLightsOn	TURN_LIGHTS_ON	1000	1000
6	TaskReadValueLightSensorProvider	READ_LIGHT_SENSOR	3000	1000
7	TaskRotateMotor	ROTATE_MOTOR	300	1100
8	TaskReadTemperatureProvider	TP	1000	1500
9	TaskConditionerOnOffConsumer	CONDIT_ON_OFF	10 000	2000

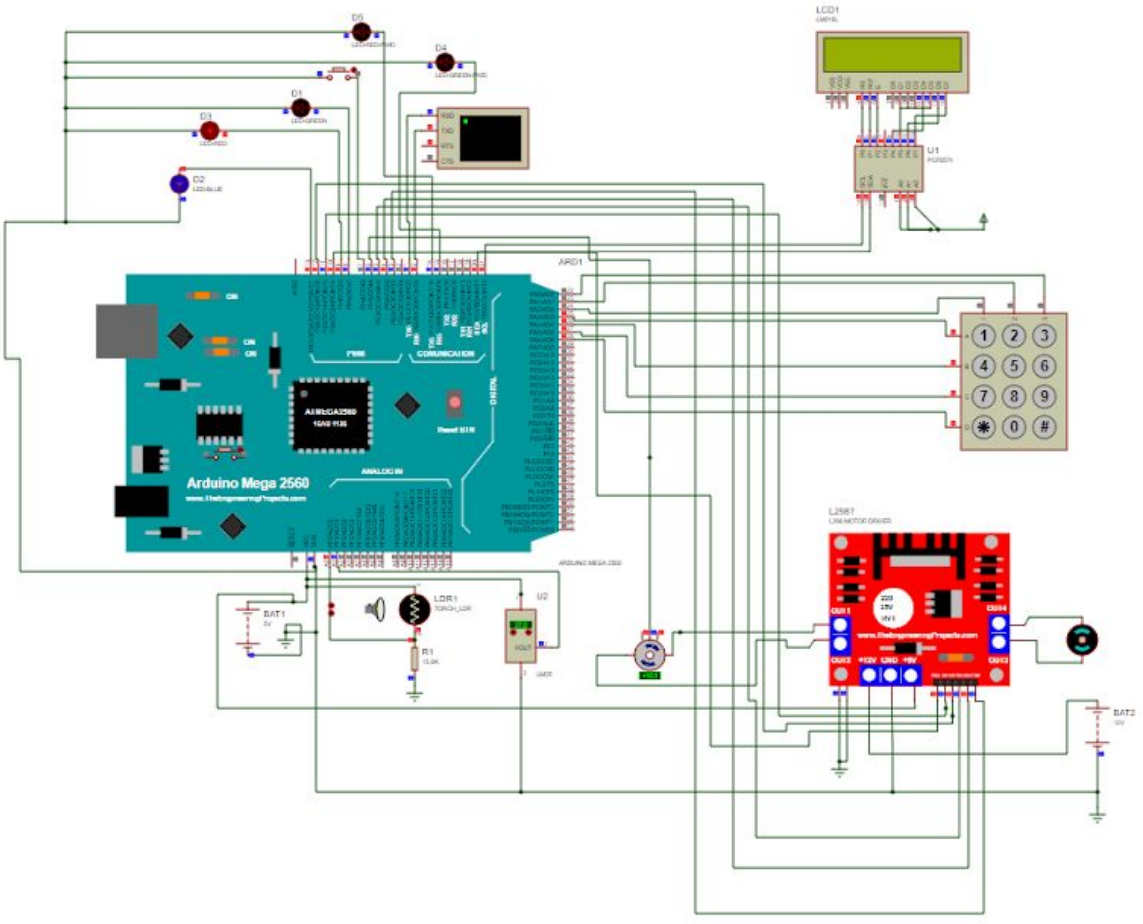
In the tasks order scheme presented above, it can be observed that the first rule for selecting offsets of the tasks, is to always put Providers with an offset smaller than the Consumers, because providers need to write in global variables (buffers) values the the consumers need to use.

The recurrence of each task was selected according to how often the task should execute. The task of reading encoder should be the most frequent, having the smallest offset, because the encoder returns values of 0 and 1 only when a rotation is complete, so the system should be able to detect these changes very fast. The task of reading the button and turning the led on/off has the recurrence of 1 second, because this is the approximate time needed to press a button. The task of reading the keypad char has a small offset of 130 ms because the keypad is very sensitive to changes and this time ensures that the press on the keypad will be recorded. Also because there are 2 tasks that use the keypad char, this task should be performed more often than Turn Car Lights On and Rotate Motor. The task of reading the value of light sensor is 3 seconds,

because we use this value only for reading the sensor and showing it, it doesn't require fast actions. The Temperature Provider executes once in a second, because usually temperature doesn't change very fast and 1 second should be enough to detect a significant change in temperature. Finally, the Conditioner Turn On/Off task executes once in 10 seconds, because 10 seconds usually doesn't mean a very big change in temperature and also because we need to avoid the very often turn on/off of the conditioner because it can damage the device and also because it uses the hysteresis control.

Circuit scheme

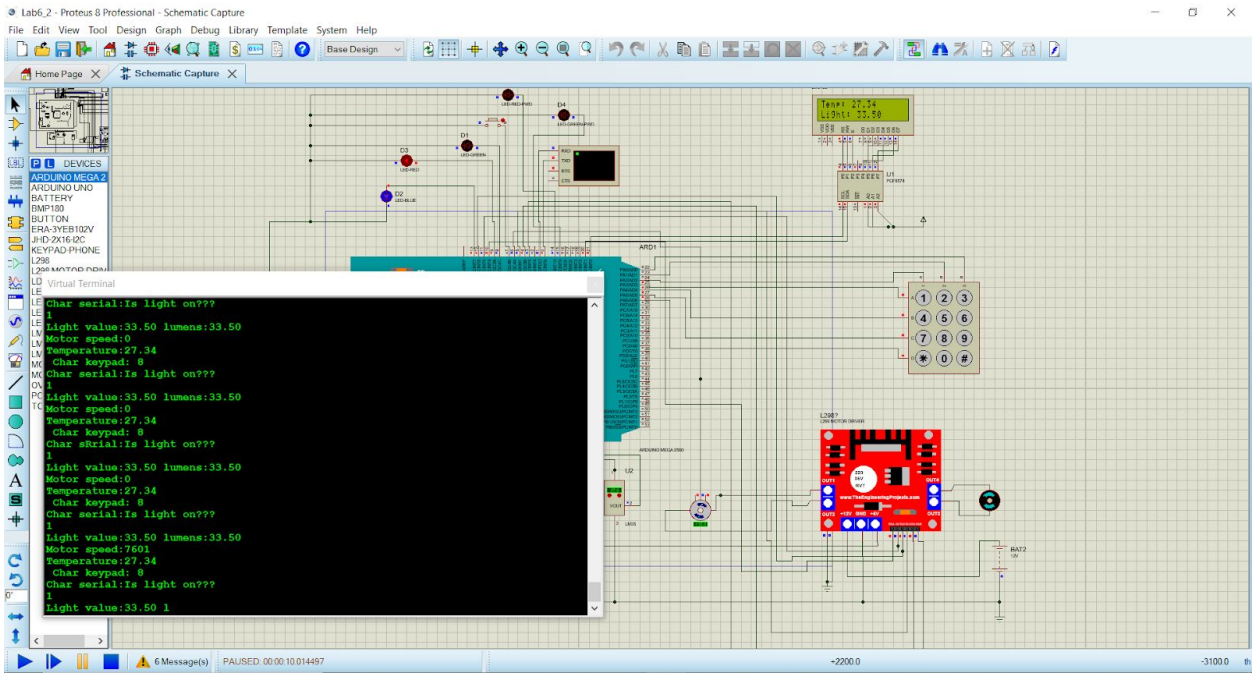
Below is presented the circuit scheme of the project - where all the components from previous laboratories are connected to the same MCU running multiple tasks. There are more LEDs because in different laboratories LEDs have different functions. The circuit was simulated using Proteus.



Proof

Video:

https://youtu.be/tosGpgGgvjY



Annex - source code

```
-----------------------
Main file
-----------------------

#include "timer-api.h"
#include "tasks.h"

// components
#include "led.h"
#include "myserial.h"
#include "light.h"
#include "car.h"
#include "motor.h"
#include "encoder.h"
#include "analoglibrary.h"
#include "light.h"
#include "lightsensor.h"
#include "mystdio.h"
#include "conditioner.h"
#include "password.h"
#include "lcd.h"
#include "mypid.h"

int rec_cnt_TP = OFFS_TP;
int rec_cnt_CONDIT_ON_OFF = OFFS_CONDIT_ON_OFF;


#define NR_OF_TASKS 9
int rec_cnts[NR_OF_TASKS] = {
                            OFFS_TP,
                            OFFS_CONDIT_ON_OFF,
                            OFFS_GET_KPD,
                            OFFS_TURN_LIGHTS_ON,
                            OFFS_BTN_LED_1,
                            OFFS_READ_LIGHT_SENSOR,
                            OFFS_ROTATE_MOTOR,
                            OFFS_READ_ENCODER,
                            OFFS_MOTOR_PID
                           };
int rec_values[NR_OF_TASKS] = {
```

```
                        REC_TP,
                        REC_CONDIT_ON_OFF,
                        REC_GET_KPD,
                        REC_TURN_LIGHTS_ON,
                        REC_BTN_LED_1,
                        REC_READ_LIGHT_SENSOR,
                        REC_ROTATE_MOTOR,
                        REC_READ_ENCODER,
                        REC_MOTOR_PID
                      };
void (*fp[NR_OF_TASKS])() = {
                TaskReadTemperatureProvider,
                TaskConditionerOnOffConsumer,
                TaskReadKeypadCharPwd,
                // TaskReadSerialChar,
                // TaskChechPassword,
                TaskTurnCarLightsOn,
                TaskButtonLedLab1,
                TaskReadValueLightSensorProvider,
                // TaskShowTempLCDConsumer,
                TaskRotateMotor,
                TaskReadEncoder,
                TaskMotorPIDControl
              };

void timer_handle_interrupts(int timer) {
  for(int i=0; i<NR_OF_TASKS; i++) {
    if(--rec_cnts[i] <=0) {
     (*fp[i])();
      rec_cnts[i] = rec_values[i];
    }
  }
}

void setup() {
  SerialInit();
  MystdioInit();
  LEDs_Init();
  InitLight();
  ConditionerInit();
  PasswordInit();
  //  EncoderInit();
  InitCar();
```

```
  LightSensorInit();
  LightSensorFiltersInit();
  LcdInit();
  PID_Init();
  // PasswordInit();
  printf("Initializing timer");
  delay(1000);
  timer_init_ISR_1KHz(TIMER_DEFAULT);
}


void loop() {
  printf("\n\rTemperature:");
  Serial.print(temperatureVal);
  printf("\n\r Char keypad: %c", charKeypad);

  printf("\n\rChar serial:%c", charSerial);
  Serial.println("Is light on???");
  Serial.println(IsLightOn());
  printf("Light value:");
  Serial.print(lightValue);
  printf(" lumens:");
  Serial.print(luxValue);
  printf("\n\rMotor speed:");
  Serial.print(motorSpeed);

  LCD_SetCursor(0,0);
  WriteLCD("Temp: ");
  WriteLCD(temperatureVal);
  LCD_SetCursor(0,1);
  WriteLCD("Light: ");
  WriteLCD(lightValue);

}


-----------------------
analoglibrary.cpp
-----------------------
/*
 * analoglibrary.cpp
 *
 *  Created on: Feb 22, 2020
```

```
 *      Author: mdiannna
 */
#include "analoglibrary.h"
#include <Arduino.h>
#include <Wire.h>
#include <stdio.h>
#include "mystdio.h"


/**
Convert the raw data value (0 - 1023) to voltage (0.0V - 5.0V):
**/
float ADCtoVoltage(float value) {
	float voltage = value * (5.0 / 1024.0);
	return voltage;
}

void sortBubble(float * arr, int n) {
	float temp;
	for(int i=0; i<n; i++) {

		for(int k=i+1; k<n; k++) {
			if(arr[k] < arr[i]) {
				temp = arr[k];
				arr[k] = arr[i];
				arr[i] = temp;
			}
		}
	}
}

void swap(float * arr, int ind1, int ind2) {
	float temp = 0.0;
	temp = arr[ind1];
	arr[ind1] = arr[ind2];
	arr[ind2] = temp;
}

void sort(float * arr) {
	float min = arr[0];
	float temp;
	char minInd = 0;
	char medInd = 1;
```

```
	if(arr[1] < min) {
		min = arr[1];
		swap(arr, 0, 1);
	}

	if(arr[2] < min) {
		min = arr[2];
		swap(arr, 0, 2);
	}


	float max = arr[2];
	char maxInd = 2;

	if(arr[1] > max) {
		max = arr[1];
		swap(arr, 2, 1);
	}
}


/**
Salt and pepper filter
**/
float SaltAndPepperFilter(float * buff) {
	// Serial.println("Salt and pepper filter");
	sort(buff);
	float median = buff[1];
	// printf("Median:");
	// Serial.println(median);
	return median;
}


/**
Filtru medie ponderata
**/
float WeightedAverageFilter(float * buff, int n) {
	float weights[3] = {0.25, 0.5, 0.75};
	float sum = 0;

	for(int i=0; i<n; i++) {
```

```
                sum += weights[i] * buff[i];
        }
        return sum/3.0;
}

void PushQueue(float * buff, float value, int size) {
        for(int i=size-1; i>0; i--) {
                buff[i] = buff[i-1];
        }
        buff[0] = value;
}
```

---

analoglibrary.h

```
/*
 * analoglibrary.h
 *
 *  Created on: Feb 22, 2020
 *      Author: mdiannna
 */

#ifndef ANALOGLIBRARY_H_
#define ANALOGLIBRARY_H_

float ADCtoVoltage(float value);
void SaltAndPepperFilter();
//Filtru medie ponderata
float WeightedAverageFilter(float *, int);

float SaltAndPepperFilter(float *);
void PushQueue(float *, float, int);

#endif /* ANALOGLIBRARY_H_ */
```

---

button.cpp

```
#include "button.h"
#include <Arduino.h>

void ButtonInit() {
  pinMode(BUTTON_PIN, INPUT_PULLUP);
```

```
}

int IsButtonPressed() {
  int buttonState = digitalRead(BUTTON_PIN);
  if(buttonState==BUTTON_PRESSED) {
    return 1;
  }
//else
  return 0;
}

int IsButtonReleased() {
  int buttonState = digitalRead(BUTTON_PIN);
  if(buttonState==BUTTON_RELEASED) {
    return 1;
  }
//else
  return 0;
}
```

---

button.h

```
#ifndef BUTTON_H_
#define BUTTON_H_

#define BUTTON_PIN 7
#define BUTTON_PRESSED 1
#define BUTTON_RELEASED 0


void ButtonInit();
int IsButtonPressed();
int isButtonReleased();


#endif
```

---
car.cpp

```
#include "car.h"
#include "motor.h"
#include <Arduino.h>
#include <avr/io.h>

int motorPower = 50;

// s- stop
// f- forward
// b- backward
// l - left
// r - right
char direction = 's';

void InitMotors() {
	InitMotorA();
	InitMotorB();
}

void InitCar() {
	InitMotors();
	// InitEncoder();

	// Set initial rotation direction
	digitalWrite(in1, LOW);
	digitalWrite(in2, HIGH);
	analogWrite(enA,0);
}

int PowerToSpeed(int power) {
	// map 0-100% power to 0-255 speed
	return map(power, 0, 100, 0 , 255);
}

void MoveForward(int power) {
	direction = 'f';
	MotorForward('A', PowerToSpeed(power));
	MotorForward('B', PowerToSpeed(power));
}
```

```
void MoveForwardA(int power) {
	direction = 'f';
	MotorForward('A', PowerToSpeed(power));
}

void MoveForwardB(int power) {
	direction = 'f';
	MotorForward('B', PowerToSpeed(power));
}

void MoveBackward(int power) {
	direction = 'b';
	MotorBackward('A', PowerToSpeed(power));
	MotorBackward('B', PowerToSpeed(power));
}


void MoveForward() {
	direction = 'f';
	MotorForward('A', PowerToSpeed(motorPower));
	MotorForward('B', PowerToSpeed(motorPower));
}

void MoveBackward() {
	direction = 'b';
	MotorBackward('A', PowerToSpeed(motorPower));
	MotorBackward('B', PowerToSpeed(motorPower));
}

void StopCar() {
	MotorsStop();
	direction = 's';
}

void TurnLeft() {
	direction = 'l';
	int power = TURN_POWER;
	MotorBackward('A', PowerToSpeed(power));
	MotorForward('B', PowerToSpeed(power));
}

void TurnRight() {
	direction = 'r';
```

```
	int power = TURN_POWER;
	MotorForward('A', PowerToSpeed(power));
	MotorBackward('B', PowerToSpeed(power));
}

char GetCarDirection() {
	return direction;
}

void ResetCarDirectionSpeed() {
	switch(direction) {
		case 's':
			StopCar();
		 break;
		case 'f':
			MoveForward(motorPower);
		 break;
		case 'b':
			MoveBackward(motorPower);
		 break;
		case 'l':
			TurnLeft();
		 break;
		case 'r':
			TurnRight();
		 break;
	// s- stop
	// f- forward
	// b- backward
	// l - left
	// r - right
	}
}

void SetCarSpeed(int powerSpeed) {
	motorPower = powerSpeed;
	ResetCarDirectionSpeed();
}
```

---

car.h

```
#ifndef CAR_H_
#define CAR_H_

#define TURN_POWER 40


void InitCar();


void MoveForward(int power);
void MoveForwardA(int power); // motor A
void MoveForwardB(int power); // motor B
void MoveBackward(int power);

void MoveForward();
void MoveBackward();

void TurnLeft();
void TurnRight();
void StopCar();

void SetCarSpeed(int);

#endif
```

---

conditioner.cpp

```
#include "led.h"
#include "conditioner.h"

void ConditionerInit() {
  LEDs_Init();
}

void TurnConditionerOn() {
  Turn_LED_Off(GREEN_LED_PIN);
  Turn_LED_On(RED_LED_PIN);
}
```

```
void TurnConditionerOff() {
  Turn_LED_Off(RED_LED_PIN);
  Turn_LED_On(GREEN_LED_PIN);
}


int IsConditionerTurnedOn() {
  if(Is_RED_LED_On()==1 && Is_GREEN_LED_On()==0) {
    return 1;
  }
  return 0;
}

int IsConditionerTurnedOff() {
  if(Is_RED_LED_On()==0) {
    return 1;
  }
  return 0;
}


int TemperatureExceedsMaxTemp(double t) {
  if(t>=MAX_TEMP) {
    return 1;
  }
  return 0;
}

int TemperatureBelowMinTemp(double t) {
  if(t<=MIN_TEMP) {
    return 1;
  }
  return 0;
}
```

---

conditioner.h

```
#ifndef CONDITIONER_H_
#define CONDITIONER_H_
```

```
#define MAX_TEMP 26
#define MIN_TEMP 24

void ConditionerInit();
void TurnConditionerOn();
void TurnConditionerOff();
int IsConditionerTurnedOn();
int IsConditionerTurnedOff();

int TemperatureExceedsMaxTemp(double);
int TemperatureBelowMinTemp(double);


#endif
```

---

encoder.cpp

```
#include "encoder.h"
#include <Arduino.h>
#include "motor.h"
#include "mystdio.h"

int maxTimeOneRotation;

void EncoderInit() {
  printf("Encoder is initializing... please wait...");
  pinMode(ENCODER_PIN,INPUT);
  MotorForward(165);

  delay(1000);

  MotorForward(255);
  delay(3000);

  while(ReadEncoderValue()!=1);
  unsigned long startTime = millis();
  delay(50);
  while(ReadEncoderValue()!=1);
  unsigned long elapsedTime = millis()-startTime;
  Serial.println("max time 1 rotation:");
  Serial.println(elapsedTime);
```

```
  maxTimeOneRotation = elapsedTime;
  delay(3000);
}

double ReadEncoderValue() {
   int value = digitalRead(ENCODER_PIN);
   return value;
}
```

---
encoder.h

```
#ifndef ENCODER_H_
#define ENCODER_H_

#define ENCODER_PIN 6

void EncoderInit();
double ReadEncoderValue();

#endif
```

---
lcd.cpp

```
#include <Wire.h>
#include <stdio.h>
#include <Arduino.h>
#include "lcd.h"
#include <LiquidCrystal_I2C.h>

//initialize the liquid crystal library
//the first parameter is the I2C address
//the second parameter is how many rows are on your screen
//the third parameter is how many columns are on your screen

// For real hardware:
LiquidCrystal_I2C lcd(0x27, 16, 2);

void lcdOn() {
```

```
  lcd.backlight();
}

void lcdOff() {
  lcd.noBacklight();
}


void LcdInit() {
  //initialize lcd screen
  lcd.init();
  // turn on the screen
  lcdOn();
}

void PutCharLCD(char c) {
  lcd.print(c);
}

void WriteLCD(char * s) {
        lcd.print(s);
}


void WriteLCD(int s) {
        lcd.print(s);
}


void WriteLCD(float s) {
        lcd.print(s);
}

void LCD_SetCursor(int i, int j) {
        lcd.setCursor(i,j);
}
```

---

lcd.h

```
#ifndef LCD_H_
#define LCD_H_
```

```
#include <Wire.h>
void lcdOn();
void lcdOff();
void LcdInit();
void PutCharLCD(char);
void WriteLCD(char *);
void WriteLCD(int);
void WriteLCD(float);
void LCD_SetCursor(int i, int j);
#endif
```

---

led.cpp

```
#include "led.h"
#include <Arduino.h>
#include <Wire.h>


void LED_Init(int led_pin) {
  pinMode(led_pin, OUTPUT);//Change to output my pins
  digitalWrite(led_pin,LED_OFF);//Turn off LED
}

void LEDs_Init() {
  LED_Init(RED_LED_PIN);
  LED_Init(GREEN_LED_PIN);
  LED_Init(GREEN_LED_PASSWORD_PIN);
  LED_Init(BLUE_LED_PIN);
  LED_Init(RED_LED_PASSWORD_PIN);
}

int Is_RED_LED_On() {
  int LEDState = digitalRead(RED_LED_PIN);
  if(LEDState==LED_ON) {
    return 1;
  }
//else
  return 0;
}
```

```
int Is_GREEN_LED_On() {
  int LEDState = digitalRead(GREEN_LED_PIN);
  if(LEDState==LED_ON) {
    return 1;
  }
//else
  return 0;
}


int Is_RED_LED_Password_On() {
  int LEDState = digitalRead(RED_LED_PASSWORD_PIN);
  if(LEDState==LED_ON) {
    return 1;
  }
//else
  return 0;
}

int Is_GREEN_LED_Password_On() {
  int LEDState = digitalRead(GREEN_LED_PASSWORD_PIN);
  if(LEDState==LED_ON) {
    return 1;
  }
//else
  return 0;
}

int Is_BLUE_LED_On() {
  int LEDState = digitalRead(BLUE_LED_PIN);
  if(LEDState==LED_ON) {
    return 1;
  }
//else
  return 0;
}

void Turn_LED_On(int led_pin) {
  digitalWrite(led_pin,LED_ON);
}

void Turn_LED_Off(int led_pin) {
  digitalWrite(led_pin,LED_OFF);
```

```
}


/************************
 Functions for blue LED - default
 ************************/
int Is_LED_On() {
  return Is_BLUE_LED_On();
}

void LED_On() {
  Turn_LED_On(BLUE_LED_PIN);
}

void LED_Off() {
  Turn_LED_Off(BLUE_LED_PIN);
}
```

---

led.h

```
#ifndef LED_H_
#define LED_H_

// builtin pin is 13 - LED_BUILTIN
#define RED_LED_PIN 9
#define GREEN_LED_PIN 8
#define BLUE_LED_PIN 13
#define RED_LED_PASSWORD_PIN 14
#define GREEN_LED_PASSWORD_PIN 15


#define LED_ON 1
#define LED_OFF 0

// Blue led init
void LED_Init();
void LED_Init(int);
void LEDs_Init();
```

```
int Is_RED_LED_On();
int Is_GREEN_LED_On();
int Is_LED_On(); // equivalent to Is_Blue_LED_On()
int Is_BLUE_LED_On();
int Is_RED_LED_Password_On();
int Is_GREEN_LED_Password_On();

void Turn_LED_On(int led_pin);
void Turn_LED_Off(int led_pin);

// Functions for blue LED:
void LED_On();
void LED_Off();


#endif /* LED_H_ */
```

---

light.cpp

```
#include "light.h"
#include "led.h"
#include <Arduino.h>

void InitLight() {
	LEDs_Init();
}

void LightOn() {
	LED_On();
}

void LightOff() {
	LED_Off();
}


int IsLightOn() {
	return Is_LED_On();
}
```

---
light.h

```
#ifndef LIGHT_H_
#define LIGHT_H_

void InitLight();
void LightOn();
void LightOff();
int IsLightOn();

#endif
```

---
lightsensor.cpp

```
/*
 * photoresistor.c
 *
 *  Created on: Feb 22, 2020
 *      Author: mdiannna
 */


#include "lightsensor.h"
#include "analoglibrary.h"
#include <stdio.h>
#include <avr/io.h>
#include <Arduino.h>


float * buffLight;
float * buffLight2;
int cntLight = 0;
float median = 0;


void LightSensorInit() {
```

```
	printf("Light sensor init");
}

int ReadLightLevel() {
	 int lightLevel = analogRead(PHOTORESISTOR_PIN);
	 return lightLevel;
}

float ResistanceToLumen(int resistance) {
	//return 500/(resistance/1000); // Conversion resistance to lumen
	float result = 500.00/resistance;
	return result;
}

float VoltageToResistance( float Vout) {
	//	Alternative formula:
	//int Rl = (R*Vin)/Vout - R;
	float Rl = (R * (Vin - Vout))/Vout;
	return Rl;
}


void LightSensorFiltersInit() {
	buffLight = (float *) malloc(3*sizeof(float));
	buffLight2 = (float *) malloc(3*sizeof(float));
	buffLight[0] = 0.0;
	buffLight[1] = 0.0;
	buffLight[2] = 0.0;

	buffLight2[0] = 0.0;
	buffLight2[1] = 0.0;
	buffLight2[2] = 0.0;
}


void showLightResults(float lightLevel) {
	printf("\n-------\n");
	printf("\n");
	float Vout = ADCtoVoltage(lightLevel);
	float resistance = VoltageToResistance(Vout);

	//printf("LIGHT LEVEL: %d\n\r", lightLevel);
	printf("LIGHT LEVEL:\n\r");
```

```
	Serial.println(lightLevel);

	//printf("Voltage: %.2f\n\r", Vout);
	Serial.println(Vout);

	printf("Resistance: %d\n\r", resistance);
	Serial.println(resistance);



	float lux = ResistanceToLumen(resistance);
	printf("Light intensity(lumen) - physical parameter: %f\n\r");
	Serial.print(lux);
	printf("\n-------\n");
}

float calcLux(float lightLevel) {
	float Vout = ADCtoVoltage(lightLevel);
	float resistance = VoltageToResistance(Vout);
	float lux = ResistanceToLumen(resistance);

	return lightLevel;
}

float MyFilter(float lightLevel) {
	if(cntLight==3){
		cntLight = 0;
	}

	buffLight[cntLight] = lightLevel;

	median = SaltAndPepperFilter(buffLight);
	// printf("median:");
	// Serial.println(median);
	PushQueue(buffLight2, median, 3);
	cntLight++;

	return WeightedAverageFilter(buffLight2, 3);
}
```

---
lightsensor.h

```
/*
 * photoresistor.h
 *
 *  Created on: Feb 22, 2020
 *      Author: mdiannna
 */

#ifndef LIGHTSENSOR_H_
#define LIGHTSENSOR_H_

#include "analoglibrary.h"
#define PHOTORESISTOR_PIN 0 //A0

// #define R 10000 //ohm resistance value
#define R 15000 //ohm resistance value
// float R = 15000;

#define Vin 5.0 // input voltage

// extern float * buffLight;
// extern float * buffLight2;
// extern int cntLight;
// extern float median;

void LightSensorInit();
void LightSensorFiltersInit();
int ReadLightLevel();

float VoltageToResistance(float Vout);
float ResistanceToLumen(int resistance);

void showLightResults(float lightLevel);
float calcLux(float lightLevel);
float MyFilter(float lightLevel) ;
#endif /* LIGHTSENSOR_H_ */
```

---

motor.cpp

```
#include <Arduino.h>
#include <avr/io.h>
```

```
#include "motor.h"

void InitMotor() {
   pinMode(enA, OUTPUT);
   pinMode(in1, OUTPUT);
   pinMode(in2, OUTPUT);
}

void InitMotorA() {
   pinMode(enA, OUTPUT);
   pinMode(in1, OUTPUT);
   pinMode(in2, OUTPUT);
}

void InitMotorB() {
   pinMode(enB, OUTPUT);
   pinMode(in3, OUTPUT);
   pinMode(in4, OUTPUT);
}

void MotorsInit() {
   InitMotorA();
   InitMotorB();
}

// Motor A
void MotorForward(int speed) {
   digitalWrite(in1, LOW);
   digitalWrite(in2, HIGH);
   analogWrite(enA,speed);
}

void MotorForward(char motor, int speed) {
   if(motor=='A') {
      digitalWrite(in1, LOW);
      digitalWrite(in2, HIGH);
      analogWrite(enA,speed);
   }
   if(motor=='B') {
      digitalWrite(in3, LOW);
      digitalWrite(in4, HIGH);
      analogWrite(enB,speed);
```

```
    }

}


// Motor A
void MotorBackward(int speed) {
    digitalWrite(in1, HIGH);
    digitalWrite(in2, LOW);
    // analogWrite(enA,PowerToSpeed(power));
    analogWrite(enA,speed);
}


void MotorBackward(char motor, int speed) {
    if(motor=='A') {
        digitalWrite(in1, HIGH);
        digitalWrite(in2, LOW);
        // analogWrite(enA,PowerToSpeed(power));
        analogWrite(enA,speed);
    }
    if(motor=='B') {
        digitalWrite(in3, HIGH);
        digitalWrite(in4, LOW);
        // analogWrite(enB,PowerToSpeed(power));
        analogWrite(enB,speed);
    }

}


// Motor A
void MotorStop() {
    digitalWrite(in1, LOW);
    digitalWrite(in2, LOW);
    analogWrite(enA,0);
}

void MotorStop(char motor) {
    if(motor=='A') {
        digitalWrite(in1, LOW);
        digitalWrite(in2, LOW);
        analogWrite(enA,0);
```

```
    }

    if(motor=='B') {
        digitalWrite(in3, LOW);
        digitalWrite(in4, LOW);
        analogWrite(enA,0);
    }

}

void MotorsStop() {
    MotorStop('A');
    MotorStop('B');
}
```

---

motor.h

```
#ifndef MOTOR_H_
#define MOTOR_H_

// Motor A
#define enA 10
#define in1 11
#define in2 12

// Motor B
// #define in3 22
// #define in4 24
// #define enB 13

#define in3 5
#define in4 4
#define enB 3

#define MAX_MOTOR_SPEED 120

void InitMotor(); //motor A
void InitMotorA(); //motor A
void InitMotorB(); //motor B

void MotorForward(int speed); //motor A
```

```
void MotorForward(char motor, int speed);

void MotorBackward(int speed); //motor A
void MotorBackward(char motor, int speed);


void MotorStop(); //motor A
void MotorStop(char motor);
void MotorsStop(); // both motors
void MotorsInit(); // both motors



#endif
```

---

mykeypad.cpp

```
#include "mykeypad.h"


const byte Rows= 4;
const byte Cols= 3;

// key map as on the key pad:
char keymap[Rows][Cols]=
{
  {'1', '2', '3'},
  {'4', '5', '6'},
  {'7', '8', '9'},
  {'*', '0', '#'}
};

byte rPins[Rows]= {25,26,27,28}; //Rows 0 to 3
byte cPins[Cols]= {24,23,22}; //Columns 0 to 2

Keypad kpd = Keypad(makeKeymap(keymap), rPins, cPins, Rows, Cols);

char GetCharKeypad() {
   char c = '+';
    while(c=='+') {
      char keypressed = kpd.getKey();
       if (keypressed != NO_KEY)
```

```
      {    c = keypressed;
           return c;
      }
    }
}
```

---
mykeypad.h

```
#ifndef MYKEYPAD_H_
#define MYKEYPAD_H_

#include <Arduino.h>
#include <Keypad.h>

extern Keypad kpd;

char GetCharKeypad();

#endif
```

---
mypid.cpp

```
#include "mypid.h"
#include <PID_v1.h>
#include <Arduino.h>
#include "motor.h"

double Setpoint, Input, Output;

//Specify the links and initial tuning parameters
// Kp, Ki, Kd
PID myPID(&Input, &Output, &Setpoint,600,100,10, DIRECT);


void PID_Init() {
  //turn the PID on
  myPID.SetMode(AUTOMATIC);
```

```
  //kp = aTune.GetKp();
 // ki = aTune.GetKi();
 // kd = aTune.GetKd();
 // myPID.SetTunings(kp,ki,kd);
}

void PID_SetInput(double input) {
        Input = map(Input, 0, MAX_MOTOR_SPEED, 0, 255);  //Scales 0-1023 to 0-255 (or
whatever you want)
        myPID.Compute();
}

double PID_GetOutput() {
        return Output;
}
```

---
mypid.h

```
#ifndef MYPID_H_
#define MYPID_H_


void PID_Init();
void PID_SetInput(double input);
double PID_GetOutput();

#endif
```

---
myserial.cpp

```
#include <Arduino.h>

void SerialInit() {
   Serial.begin(9600);
}
```

---
myserial.h

```
#ifndef MYSERIAL_H_
#define MYSERIAL_H_

void SerialInit();

#endif
```

---
mystdio.cpp

```
#include <Arduino.h>
#include <stdio.h>
#include "lcd.h"
#include "mykeypad.h"


int MyPutChar( char c, FILE *t) {
	Serial.write( c );
	// PutCharLCD(c);
}

char MyGetChar( FILE *stream) {
	return GetCharKeypad();
}


void MystdioInit() {
	// Varianta 1
	fdevopen( &MyPutChar, 0);
	fdevopen( &MyGetChar, 1);
	// Varianta 2
	// fdevopen( &MyPutCh, &MyGetCh);
}

// Varianta 2

// int MyPutCh(char ch, FILE *t) {
//   Serial.print(ch);
// }
```

```
// int MyGetCh(FILE *t) {
//   while(!Serial.available());
//   char ch = Serial.read();
//   return ch;
// }
```

---
mystdio.h

```
#ifndef MYSTDIO_H_
#define MYSTDIO_H_

void MystdioInit();

#endif
```

---
password.cpp

```
#include "password.h"
#include "mykeypad.h"
#include <stdlib.h>
#include <stdio.h>
#include <Arduino.h>

char * CORRECT_PASSWORD = "1234#";


int PasswordIsCorrect(char * pwd) {
  if(strcmp(CORRECT_PASSWORD, pwd)==0) {
          return 1;
  }
  return 0;
}

void PasswordInit() {
          password = (char *) malloc (5*sizeof(char));
          password = "";
          passwordIndex = 0;
}
```

```
//char* ReadPassword() {
//  char * buf = (char *) malloc (32);
//  int i = 0;
//
//  char c = '+';
//  while(c!='#') {
//    c = '+';
//    while(c=='+') {
//      char keypressed = kpd.getKey();
//      if (keypressed != NO_KEY) {
//        c = keypressed;
//        printf("%c", c);
//        buf[i]= c;
//        i++;
//      }
//    }
//
//  }
//  return buf;
//}
//
```

---
password.h

```
#ifndef PASSWORD_H_
#define PASSWORD_H_

extern char* password;
extern int passwordIndex;

void PasswordInit();
int PasswordIsCorrect(char * password);
void ResetPassword();

#endif
```

---
temperature.cpp

```
#define TEMPERATURE_SENSOR_PIN 1

#include <Arduino.h>
#include <Wire.h>

int val;

float ReadTemperature() {
  val = analogRead(TEMPERATURE_SENSOR_PIN);
  float mv = ( val/1024.0)*5000;
  float cel = mv/10;
  float farh = (cel*9)/5 + 32;
  return cel;
}

void ExampleTemperature()
{
  float temp = ReadTemperature();
  Serial.print("TEMPERATURE = ");
  Serial.print(temp);
  Serial.print("*C");
  Serial.println();
  // delay(1000);
}
```

---
temperature.h

```
#ifndef TEMPERATURE_h
#define TEMPERATURE_h

// Change to respective sensor
#define TEMPERATURE_SENSOR_PIN 1

float ReadTemperature();
void ExampleTemperature();
```

```
#endif

-----------------------
tasks.cpp
-----------------------
#include "tasks.h"
#include "conditioner.h"
#include "temperature.h"
#include <Arduino.h>
#include "mykeypad.h"
#include "password.h"
#include "mystdio.h"
#include "car.h"
#include "light.h"
#include "button.h"
#include "led.h"
#include "lightsensor.h"
#include "lcd.h"
#include "motor.h"
#include "encoder.h"
#include "mypid.h"

float temperatureVal = 0.0;
char charKeypad = '-';
int passwordIsCorrect = 0;
char charSerial;
char * password;
float lightValue = 0;
float luxValue = 0;
int motorSpeed = 0;

int passwordIndex;

// TP task code
void TaskReadTemperatureProvider() {
        Serial.println("Read temperature");
        temperatureVal = ReadTemperature();
}

// ConditOnOff code
void TaskConditionerOnOffConsumer() {
        if(TemperatureExceedsMaxTemp(temperatureVal) && IsConditionerTurnedOff())
   {
```

```
    TurnConditionerOn();
  }
  else if(TemperatureBelowMinTemp(temperatureVal)&&IsConditionerTurnedOn())
  {
    TurnConditionerOff();
  }
}

void TaskReadSerialChar() {
        Serial.println("Write char:");
        if (Serial.available() > 0) {
        charSerial = Serial.read();
  }
}


void TaskReadKeypadCharPwd() {
        char keypressed = kpd.getKey();
        if (keypressed != NO_KEY)
        {
                charKeypad = keypressed;
                // password[passwordIndex] = charKeypad;
                // passwordIndex++;

                // if(passwordIndex>=3) {
                //      password = "";
                //      passwordIndex = 0;
                // }
        }
}

// // TASK_CHK_PWD
// void TaskChechPassword() {
//      if(charKeypad=='#') {
//              passwordIsCorrect = 0;
//              if(PasswordIsCorrect(password)==1  ){
//                      passwordIsCorrect = 1;
//              }
//              password = "";
//              // password[0] = '*';
//              // password[1] = '*';
//              // password[2] = '*';
//              // password[3] = '*';
```

```
//                  passwordIndex = 0;
//          charKeypad='-';
//          }
// }


// TURN_LIGHTS_ON
void TaskTurnCarLightsOn() {
          if(charKeypad=='#' && IsLightOn() ){
                    LightOff();
          } else
                    if(charKeypad=='*' && IsLightOn()==0 ){
                    {
                              LightOn();
                    }
          }
}

// BTN_LED_1
void TaskButtonLedLab1() {
          if(IsButtonPressed()==1) {
                    if(Is_LED_On()) {
                              LED_Off();
                    } else {
                              LED_On();
                    }
          }
}


// READ_LIGHT_SENSOR
void TaskReadValueLightSensorProvider() {
          float lightLevel = ReadLightLevel();
          float filteredValue = MyFilter(lightLevel);

          lightValue = filteredValue;
          luxValue = calcLux(filteredValue);
}

//SHOW_TEMP_LCD
void TaskShowTempLCDConsumer(){
          LCD_SetCursor(0,0);
```

```
	WriteLCD("Temp: ");
	WriteLCD(temperatureVal);
}

// ROTATE_MOTOR - lab 3 Actuators
void TaskRotateMotor() {
	if( charKeypad=='0') {
		// StopCar();
		MotorStop('A');
	} else if(charKeypad=='1') {
		MoveForwardA(20);
	} else if(charKeypad=='2') {
		MoveForwardA(30);
	} else if(charKeypad=='3') {
		MoveForwardA(40);
	} else if(charKeypad=='4') {
		MoveForwardA(50);
	} else if(charKeypad=='5') {
		MoveForwardA(60);
	} else if(charKeypad=='6') {
		MoveForwardA(70);
	} else if(charKeypad=='7') {
		MoveForwardA(80);
	} else if(charKeypad=='8') {
		MoveForwardA(90);
	} else if(charKeypad=='9') {
		MoveForwardA(100);
	}
}


int oldEncState =0;

// READ_ENCODER
void TaskReadEncoder() {
	static int encoderSpeedLocal = 0;
	int newEncState = digitalRead(ENCODER_PIN);

	if(newEncState!=oldEncState) {
		motorSpeed = encoderSpeedLocal;
		encoderSpeedLocal = 0;
	} else {
		encoderSpeedLocal++;
```

```
        }
        oldEncState = newEncState
;}

// MOTOR_PID
// doesn't work properly
void TaskMotorPIDControl() {
        // global motorSpeed
        PID_SetInput(motorSpeed);
        double output = PID_GetOutput();
        // MotorForward('B', output);
        MotorForward('B', 60);
}
```

---

tasks.h

```
#define REC_TP 1000
#define OFFS_TP 1500

#define REC_CONDIT_ON_OFF 10000 //10 sec
#define OFFS_CONDIT_ON_OFF 2000

#define REC_GET_KPD 130
#define OFFS_GET_KPD 1000


#define REC_READ_SCHAR 500
#define OFFS_READ_SCHAR 550

#define REC_CHK_PWD 1000
#define OFFS_CHK_PWD 600


#define REC_TURN_LIGHTS_ON 1000
#define OFFS_TURN_LIGHTS_ON 1000

#define REC_BTN_LED_1 1000
#define OFFS_BTN_LED_1 300

#define REC_READ_LIGHT_SENSOR 3000
```

```
#define OFFS_READ_LIGHT_SENSOR 1000

#define REC_SHOW_TEMP_LCD 1000
#define OFFS_SHOW_TEMP_LCD 2000

#define REC_ROTATE_MOTOR 300
#define OFFS_ROTATE_MOTOR 1100

// READ_ENCODER
#define REC_READ_ENCODER 1 //1ms
#define OFFS_READ_ENCODER 0

#define REC_MOTOR_PID 1000 //1sec
#define OFFS_MOTOR_PID 500

extern float temperatureVal;
extern double pressureVal;
extern double relPressureVal;
extern char charKeypad;
extern int passwordIsCorrect;
extern char charSerial;
extern float lightValue;
extern float luxValue;
extern int motorSpeed;

// TP task code
void TaskReadTemperatureProvider();

// ConditOnOff code
void TaskConditionerOnOffConsumer();

// GET_KPD code
void TaskReadKeypadCharPwd();

// READ_SCHAR code
// void TaskReadSerialChar();

// TASK_CHK_PWD
void TaskChechPassword();

// TURN_LIGHTS_ON
void TaskTurnCarLightsOn();
```

```
// BTN_LED_1
void TaskButtonLedLab1();

// READ_LIGHT_SENSOR
void TaskReadValueLightSensorProvider();

//SHOW_TEMP_LCD
void TaskShowTempLCDConsumer();

// ROTATE_MOTOR
void TaskRotateMotor();

// READ_ENCODER
void TaskReadEncoder();

// MOTOR_PID
void TaskMotorPIDControl();
```