Light Assistance System

Author: Pascu Mihail-Eugen

2021

Contents

1. Project description

The purpose of the application is to control the light system of a car. The system is using a light sensor for detecting the environmental light intensity and two infrared sensors for detecting the obstacles from the left of the car, right of the car or ahead (both infrared sensor detecting an object in the same time). The light system of the example car is composed of two L.E.D. (one for the left side and one for the right side).

2. Hardware description

The microcontroller that is used in this project is Intel Quark SoC x1000. It have all the needed components to run an operating system (in our case, a Linux kernel designed for the embedded devices). The internal components of this microcontroller that will be used in this project are: GPIOs (for digital inputs and PWM outputs), ADC (for analog input), Timer and Interrupts (for configuring the PWM outputs in a timely manner).

All the components that were used in this project are:

- **Intel Galileo Gen 2 development board**, which contains the microcontroller Intel Quark SoC x1000. This is the programmable part of the system and this is the part that handles all the logic needed for the system to function properly;
- **Two KeyesIR (KY-032) infrared obstacle detection sensors**, which are used to detect if an obstacle is in front of the car. The basic working principle is the following, it emits a 38kHz signal using an IR LED and the reflected light from the object is received by an IR Receiver. When no light is received this is interpreted as no object detected, if light is received, then the object is detected. The detection range for this sensor, according to the datasheet, is between 2-40 cm.
- **Two white LEDs**, which simulates the two headlights of a car. These two LEDs come in the company of who 270Ohm resistor for limiting the drawn current.
- **One Photoresistor**, use to detect the environmental light intensity. To measure the light intensity, a voltage divider is build with the help of an 10kOhm resistor.
- **Wires and breadboard,** for connecting the used components.

The system logic is focused on the establishment of the output PWM signal that is feeded to the two LEDs (the car headlights). The duty cycle is inversely proportional to the light intensity of the external environment (which is provided by the photoresistor as an analog voltage level, that is processed by the ADC to a value between 0 and 4095. The Intel Galileo Gen 2's ADC have a 12 bit representation of the analog value so the maximal output value is 2^12=4096). If an object is detected (an IR sensor output is 1) the corresponding LED's duty cycle is decreased even further to lower the LEDs brightness.

Bellow is presented the schematic of the implemented circuit (Fig. 1):

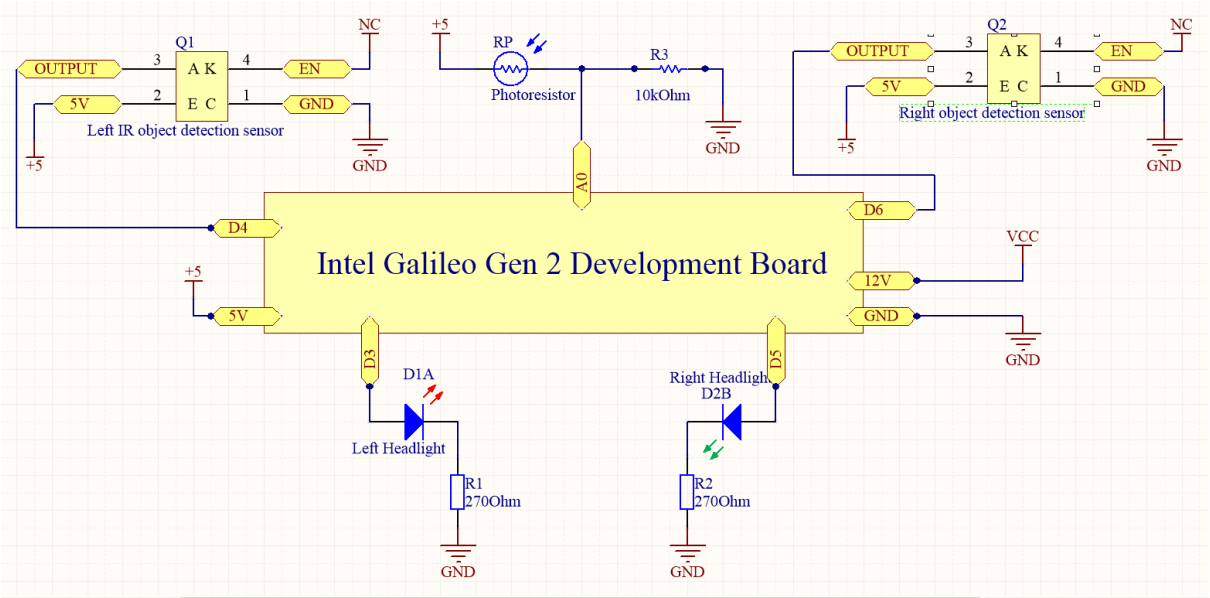

Fig.1 The implemented electronic circuit

3. Software description

The software was developed in the Arduino IDE, an easy IDE in which can be developed software for many development boards (including the one from the current project – Intel Galileo Gen 2) not only Arduinos. With the help of this IDE the application can be written, compiled and sent to the development board without any overhead of configuring the communication between the PC and the board. This IDE was chose because of the ease of which the application can be delivered to the hardware system that will implement the solution.

The software make use of the operating system that is present on the Intel Galileo Gen 2, a kernel of Linux developed for the embedded devices. To configure and set the pins and all the peripherical devices that are used in this project, the drivers for these, from the Linux kernel are used. For example, for writing the 1 to output for the pin D5, we are using the pca953x kernel GPIO driver. A problem that was encountered was that of the multiplexed pins of the Intel Galileo Gen 2. For example, to be able to set the D3 pin as pwm output, the GPIO76 must be set to output LOW and GPIO64 must be set to output HIGH (as can be seen in Fig2 – took from https://emutex.com/educational/71-getting-started-with-intel-galileo-gen-2).

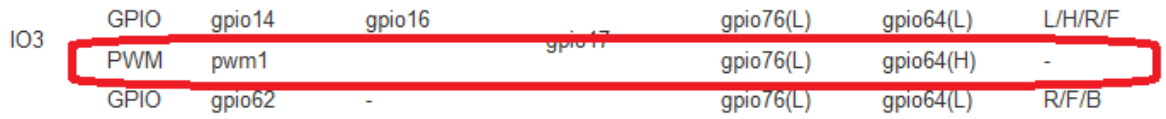

Fig2. D3 (IO3) pin multiplexing

The software is divided in two parts, one for configuring the board (can be seen in Fig.3) and the other one for running the program that implements the logic of the system (can be seen in Fig.4).

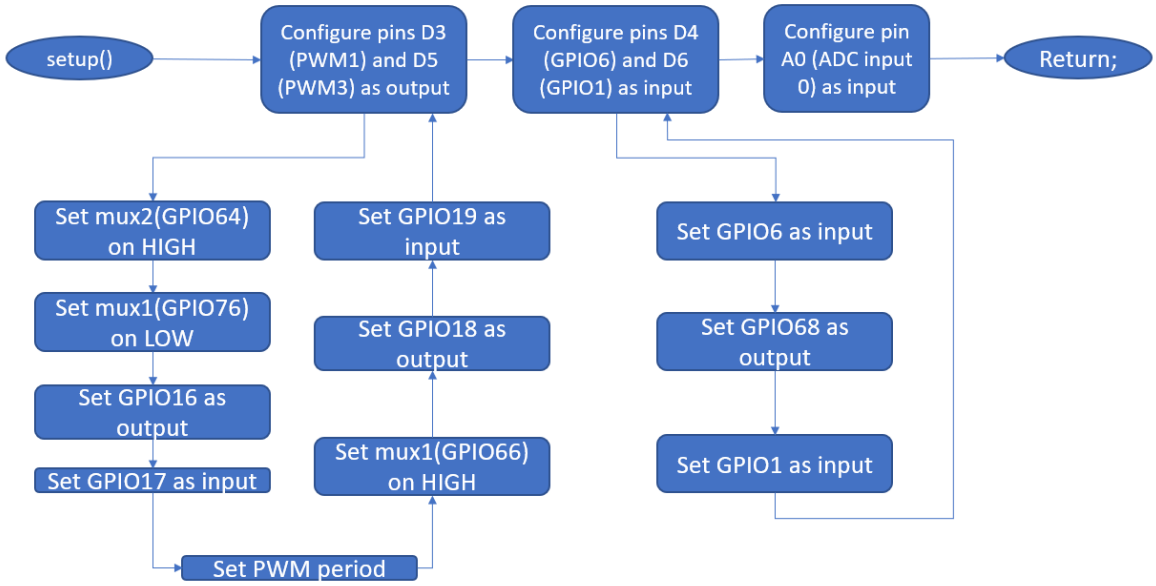

Fig.3 The flow of **setup** function

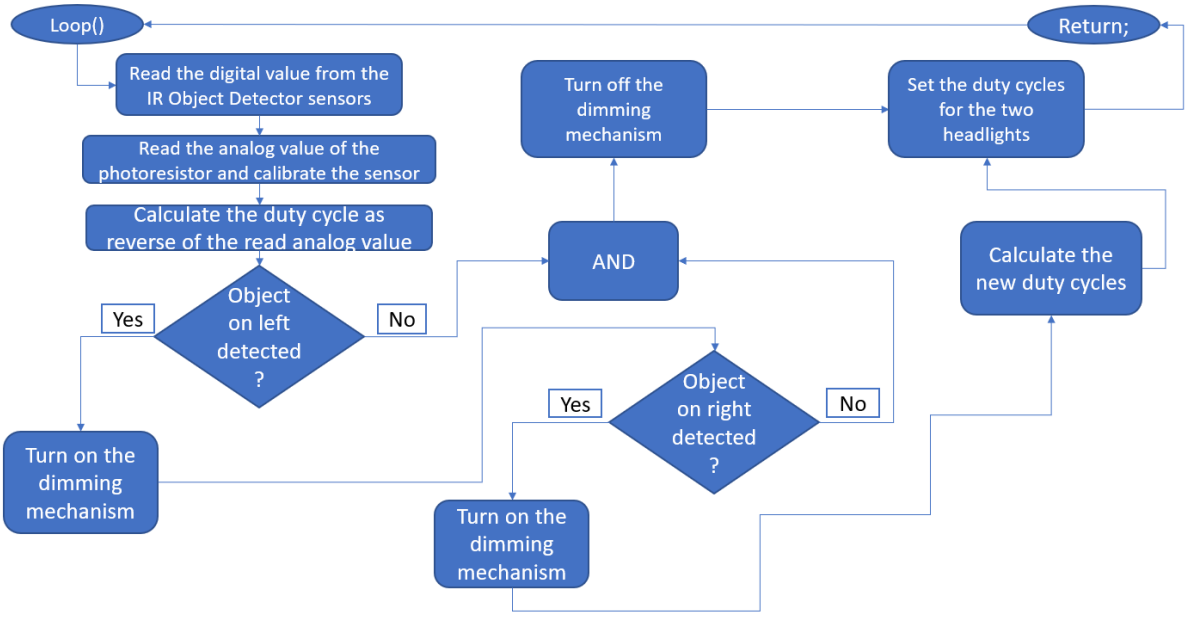

Fig.4 The flow of the **loop** function

4. Problems and solutions

The problems encountered in making this project were both hardware and software.

- Firstly, the KeyesIR sensors were not calibrated properly so a physical modification to them had to be made. To fix this, the IR LED was separated from the IR Receiver by placing a piece of cardboard between the two, but this must be placed in a way that it does not interrupt the

incoming IR light coming from the object that reflected it. Another modification that was needed to be made, was the fine adjustment of the emitted light signal. In order for this sensor to work properly, a signal of 38kHz must be emitted. This adjustment was made with the help of the two potentiometers found on the KeyesIR sensor and an oscilloscope (as can be seen in the Fig.5)

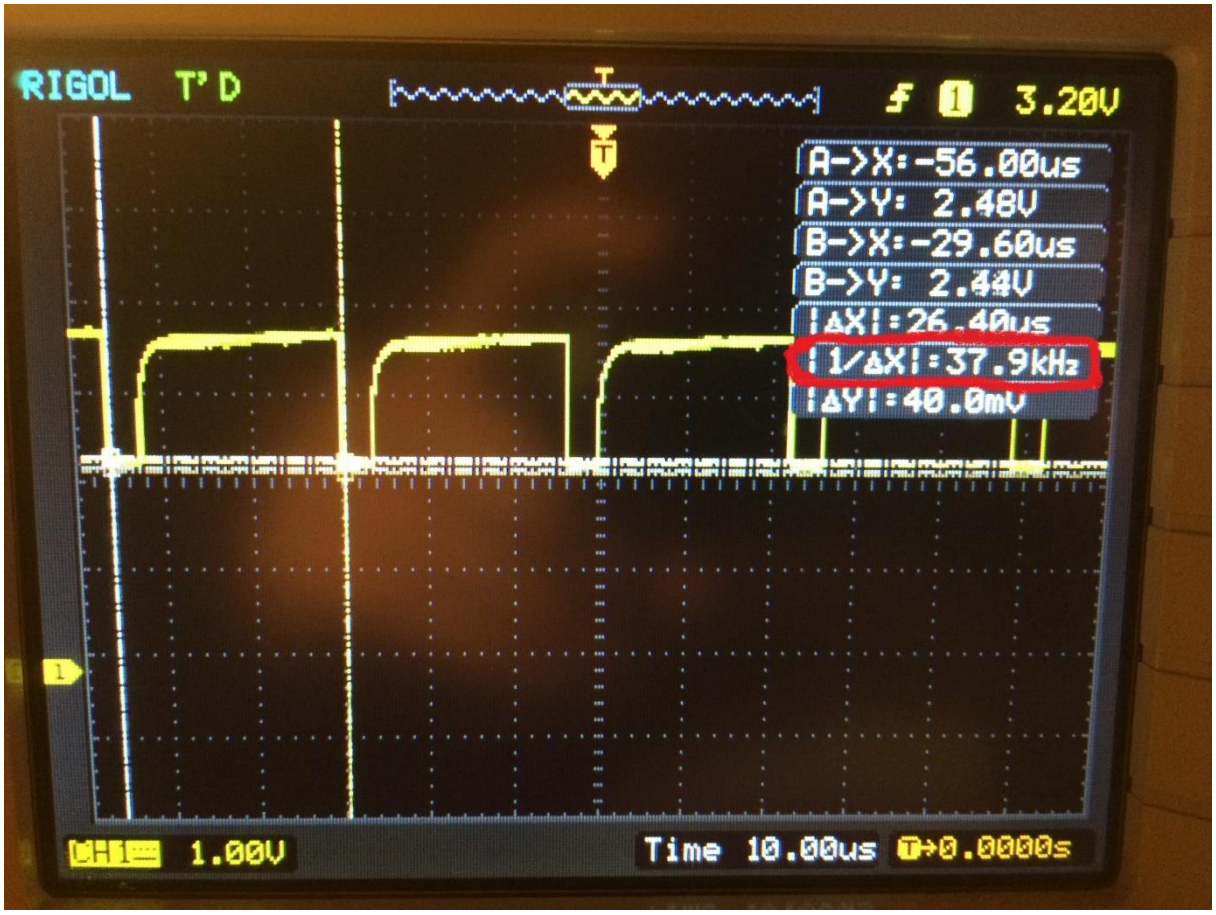

Fig.5 Oscilloscope showing that the emitted signal is 38kHz.

- The datasheet for Intel Quark SoC x1000 is not to explicit at what addresses are located the needed registers in order to configure the board (the GPIO, ADC, Timer, PWM and others). The solution for this problem was to search for explanations from the community of embedded developers that had worked with Intel Galileo Gen 2. Another solution was to make the most of the Linux kernel that was running on this board so the kernel modules that control the peripherals of this microcontroller were used.
- Lastly, the pin multiplexing for the Intel Galileo Gen 2 was not clearly explained in any of the official documentation. Luckily, the community of developers had a table in which is explained how the multiplexing works on this development board;

5. Conclusions

The purpose of this project was to demonstrate that the conventional headlights can be improved by some simple yet very useful ideas, like avoiding blinding the drivers from the opposite direction or headlight brightness decrease as the environmental light increases. This exact project is

not made to be applicable on an actual car. For a project to be applied on the car, the photoresistor must be replaced with a much more stable light sensor (like the one exemplified here https://www.hella.com/techworld/us/Technical/Car-electronics-and-electrics/Check-change-rain-sensor-42078/), the object detection must be much improved (the object should be detected from a far range – at least 50meters, and this should be done with LiDARs or AI powered cameras) and also the topic of energy efficiency must be covered.

After realizing this project, the following skills have been acquired and improved:

- Understanding the architecture of the Intel Quark SoC x1000;
- Learned how to use the Linux kernel for embedded devices;
- How some development boards are adapted to the Arduino Code style (how many microcontrollers can be programmed with the same syntax as the board Arduino Uno);
- Always the non-professional sensors must be calibrated and carefully adjusted in order to function properly (as expected according to their datasheet);
- Beside the official documentation, a good source for learning and getting better at understanding a microcontroller is the community of developers (from sites like https://hackaday.com/ , https://embeddedbits.org/ , and others).

6. Source code

```
#include <stdio.h>

//define used pins
#define LEFT_HEADLIGHT "pwm1"
#define RIGHT_HEADLIGHT "pwm3"
#define LEFT_DISTANCE_SENSOR "gpio6"
#define RIGHT_DISTANCE_SENSOR "gpio1"
#define PHOTORESISTOR 0

//define the constant values used
#define PWM_PERIOD_NS 10000000
#define MAX_ANALOG_IN ((float)4095.00))
#define reverseDutyCycle(x) (1-(x/MAX_ANALOG_IN)
#define DIMMING_STEP (dutyCycle/10)

uint32_t dutyCycleDecrementValueLeft=0;
uint32_t dutyCycleDecrementValueRight=0;
uint32_t dutyCycleValueLeft=0;
uint32_t dutyCycleValueRight=0;

uint32_t minBrightness=1024;
uint32_t maxBrightness=3072;

bool IRSensorLeft=false;
bool IRSensorRight=false;

uint32_t dutyCycle=0;
uint32_t obstacleDutyCycle=0;

// the setup function runs once when you press reset or power the board
void setup() {
```

```
 //pinMode(LEFT_HEADLIGHT, OUTPUT);
    //set pin mux2 GPIO on HIGH
    configureGPIO(64,"out","hiz",1);
    //set pin mux1 GPIO on LOW
    configureGPIO(76,"out","hiz",0);
    //set level shifter GPIO to out
    configureGPIO(16,"out","hiz");
    //set 22k pull-up GPIO to in
    configureGPIO(17,"in","pullup");
    //configure PIN3 as pwm
    configurePWM(1,PWM_PERIOD_NS);

 //pinMode(RIGHT_HEADLIGHT, OUTPUT);
    //set pin mux1 GPIO on HIGH
    configureGPIO(66,"out","hiz",1);
    //set level shifter GPIO to out
    configureGPIO(18,"out","hiz");
    //set 22k pull-up GPIO to in
    configureGPIO(19,"in","pullup");
    //configure PIN5 as pwm
    configurePWM(3,PWM_PERIOD_NS);

 //pinMode(LEFT_DISTANCE_SENSOR, INPUT);
    //configure pin 4 as INPUT
    configureGPIO(6,"in","hiz");

 //pinMode(RIGHT_DISTANCE_SENSOR, INPUT);
    //set pin mux1 GPIO on LOW
    configureGPIO(68,"out","pulldown",0);
    //configure pin 6 as INPUT
    configureGPIO(1,"in","hiz");

 //pinMode(PHOTORESISTOR, INPUT);
    configureGPIO(49,"in","pullup");
}

// the loop function runs over and over again forever
void loop() {
  dutyCycle=0;
  obstacleDutyCycle=0;
  IRSensorLeft=readIRSensorValue(LEFT_DISTANCE_SENSOR);
  IRSensorRight=readIRSensorValue(RIGHT_DISTANCE_SENSOR);

  //dutyCycle=analogRead(PHOTORESISTOR);
  dutyCycle= calibratePhotoResistor(readAnalogPin(PHOTORESISTOR));

  //calculate the duty cycle as inverse of the value read from the photoresistor
  dutyCycle=(uint32_t)(reverseDutyCycle(dutyCycle)*PWM_PERIOD_NS); // percentage *
pwm_period
  obstacleDutyCycle=dutyCycle - DIMMING_STEP;

  //run the Dimming mechanism
  checkAndRunDimming();

  //analogWrite(LEFT_HEADLIGHT,dutyCycleValueLeft);
  writeDutyCycle(LEFT_HEADLIGHT,dutyCycleValueLeft);

  //analogWrite(RIGHT_HEADLIGHT,dutyCycleValueRight);
  writeDutyCycle(RIGHT_HEADLIGHT,dutyCycleValueRight);
```

```
}

void checkAndRunDimming(){
    if (true == IRSensorLeft || true == IRSensorRight ){
      delay(1);
    }
    if (true == IRSensorLeft ){
      dutyCycleDecrementValueLeft+=DIMMING_STEP;
      if( dutyCycleDecrementValueLeft >= obstacleDutyCycle )
dutyCycleDecrementValueLeft = obstacleDutyCycle;
    }else{
      dutyCycleDecrementValueLeft=0;
    }
    if (true == IRSensorRight ){
      dutyCycleDecrementValueRight+=DIMMING_STEP;
      if( dutyCycleDecrementValueRight >= obstacleDutyCycle )
dutyCycleDecrementValueRight = obstacleDutyCycle;
    }else{
      dutyCycleDecrementValueRight=0;
    }
  dutyCycleValueLeft=dutyCycle-dutyCycleDecrementValueLeft;
  dutyCycleValueRight=dutyCycle-dutyCycleDecrementValueRight;
}

void configureGPIO(int gpio,String gpioDirection,String drive){
  configureGPIO(gpio,gpioDirection,drive,0);
  return;
}

void configureGPIO(int gpio,String gpioDirection,String drive,int value){
  String sgpio=String(gpio);
  String stringCommand;
  char charArrayCommand[300];

  //reserve the wished gpio
  memset((void *)charArrayCommand, sizeof(charArrayCommand), 0);
  stringCommand="echo "+sgpio+" > /sys/class/gpio/export";
  stringCommand.toCharArray(charArrayCommand, sizeof(charArrayCommand), 0);
  system(charArrayCommand);

  //set the direction of the gpio
  memset((void *)charArrayCommand, sizeof(charArrayCommand), 0);
  stringCommand="echo "+gpioDirection+" > /sys/class/gpio/gpio"+sgpio+"/direction";
  stringCommand.toCharArray(charArrayCommand, sizeof(charArrayCommand), 0);
  system(charArrayCommand);

  //set the drive of the gpio
  memset((void *)charArrayCommand, sizeof(charArrayCommand), 0);
  stringCommand="echo "+drive+" > /sys/class/gpio/gpio"+sgpio+"/drive";
  stringCommand.toCharArray(charArrayCommand, sizeof(charArrayCommand), 0);
  system(charArrayCommand);

  //set the value of the gpio (usefull only when set to direction output
  memset((void *)charArrayCommand, sizeof(charArrayCommand), 0);
  stringCommand="echo "+String(value)+" > /sys/class/gpio/gpio"+sgpio+"/value";
  stringCommand.toCharArray(charArrayCommand, sizeof(charArrayCommand), 0);
  system(charArrayCommand);
  return;
```

```
}

void configurePWM(int pwm_pin,int pwm_period){
  String sgpio=String(pwm_pin);
  String stringCommand;
  char charArrayCommand[300];

  //reserve the wished pwm pin
  memset((void *)charArrayCommand, sizeof(charArrayCommand), 0);
  stringCommand="echo "+sgpio+" > /sys/class/pwm/pwmchip0/export";
  stringCommand.toCharArray(charArrayCommand, sizeof(charArrayCommand), 0);
  system(charArrayCommand);

  //set the pwm pin to enable
  memset((void *)charArrayCommand, sizeof(charArrayCommand), 0);
  stringCommand="echo 1 > /sys/class/pwm/pwmchip0/pwm"+sgpio+"/enable";
  stringCommand.toCharArray(charArrayCommand, sizeof(charArrayCommand), 0);
  system(charArrayCommand);

  //set the pwm period ( this is set for all the pwm outputs )
  memset((void *)charArrayCommand, sizeof(charArrayCommand), 0);
  stringCommand="echo "+String(pwm_period)+" >
/sys/class/pwm/pwmchip0/device/pwm_period";
  stringCommand.toCharArray(charArrayCommand, sizeof(charArrayCommand), 0);
  system(charArrayCommand);

  return;
}

bool readIRSensorValue(String gpio){
  if(readDigitalPin(gpio)==0){
    delayMicroseconds(395);              // wait for 15 pulses at 38kHz.
    if(readDigitalPin(gpio)==0)          // check if it was a false read
      return true;
  }
  return false;
}

bool readDigitalPin(String gpio){
   FILE *fpipe;
   String command = "cat /sys/class/gpio/"+gpio+"/value";

   // This buffer will containg the script response
   char cmd_rsp[4];
   // buffer to be used with popen
   char cmd_char[300];
   // clear message buffer
   memset((void *)cmd_char, sizeof(cmd_char), 0);
   // convert the message to char array
   command.toCharArray(cmd_char, sizeof(cmd_char), 0);
   if ( (fpipe = (FILE*)popen((char *)cmd_char,"r")) ){
      while ( fgets( cmd_rsp, sizeof(cmd_rsp), fpipe)) {}
      pclose(fpipe);
      return atoi(cmd_rsp);
   }
   return -1;
}

int readAnalogPin(int pin){
```

```
   FILE *fpipe;
   String command = "cat
/sys/bus/iio/devices/iio:device0/in_voltage"+String(pin)+"_raw";
   // This buffer will containg the script response
   char cmd_rsp[12];
   // buffer to be used with popen
   char cmd_char[300];
   // clear message buffer
   memset((void *)cmd_char, sizeof(cmd_char), 0);
   // convert the message to char array
   command.toCharArray(cmd_char, sizeof(cmd_char), 0);
   if ( (fpipe = (FILE*)popen((char *)cmd_char,"r")) ){
      while ( fgets( cmd_rsp, sizeof(cmd_rsp), fpipe)) {}
      pclose(fpipe);
      return atoi(cmd_rsp);
   }
   return -1;
}

void writeDutyCycle(String pin, uint32_t value){
   String command = "echo "+String(value)+" >
/sys/class/pwm/pwmchip0/"+pin+"/duty_cycle";
   // buffer to be used to send the command
   char cmd_char[300];
   // clear message buffer
   memset((void *)cmd_char, sizeof(cmd_char), 0);
   // convert the message to char array
   command.toCharArray(cmd_char, sizeof(cmd_char), 0);
   system(cmd_char);
   return;
}

uint32_t calibratePhotoResistor(uint32_t analogValue){
   uint32_t brightness=0;

   //find if a new min or max value have been found
   if(minBrightness > analogValue)  minBrightness= analogValue;
   if(maxBrightness < analogValue)  maxBrightness= analogValue;
   //Adjust the brightness level to produce a result between 0 and 4096 (2^12).
   brightness=map(analogValue, minBrightness, maxBrightness, 0, 4095);
    // in case the sensor value is outside the range seen during calibration
   brightness = constrain(brightness, 0, 4095);

   if(brightness < 200)  brightness = 0;
   if(brightness > 3900) brightness = 4095;

   return brightness;
}
```

7. Bibliography

1. Intel Galileo Gen 1, useful for understanding how pin multiplexing on Intel Galileo development boards is done : http://www.malinov.com/Home/sergey-s-blog/intelgalileo-programminggpiofromlinux
2. Intel Galileo Gen 2 pin mapping : https://emutex.com/educational/71-getting-started-with-intel-galileo-gen-2

3. Intel Galileo Gen 2 pwm muxing: https://www.intel.com/content/www/us/en/support/articles/000006120/boards-and-kits/intel-galileo-boards.html
4. Intel Galileo Gen 2 connect serial to the Linux kernel: https://www.intel.com/content/www/us/en/support/articles/000006142/boards-and-kits/intel-galileo-boards.html
5. Embedded Linux kernel tips for understanding kernel modules : https://stackoverflow.com/questions/16033748/how-do-can-i-find-out-which-linux-driver-is-hogging-my-gpio; https://forum.odroid.com/viewtopic.php?f=95&t=22436&p=150877#p150823
6. A short introduction in photoresistors: https://www.instructables.com/Photoresistors/