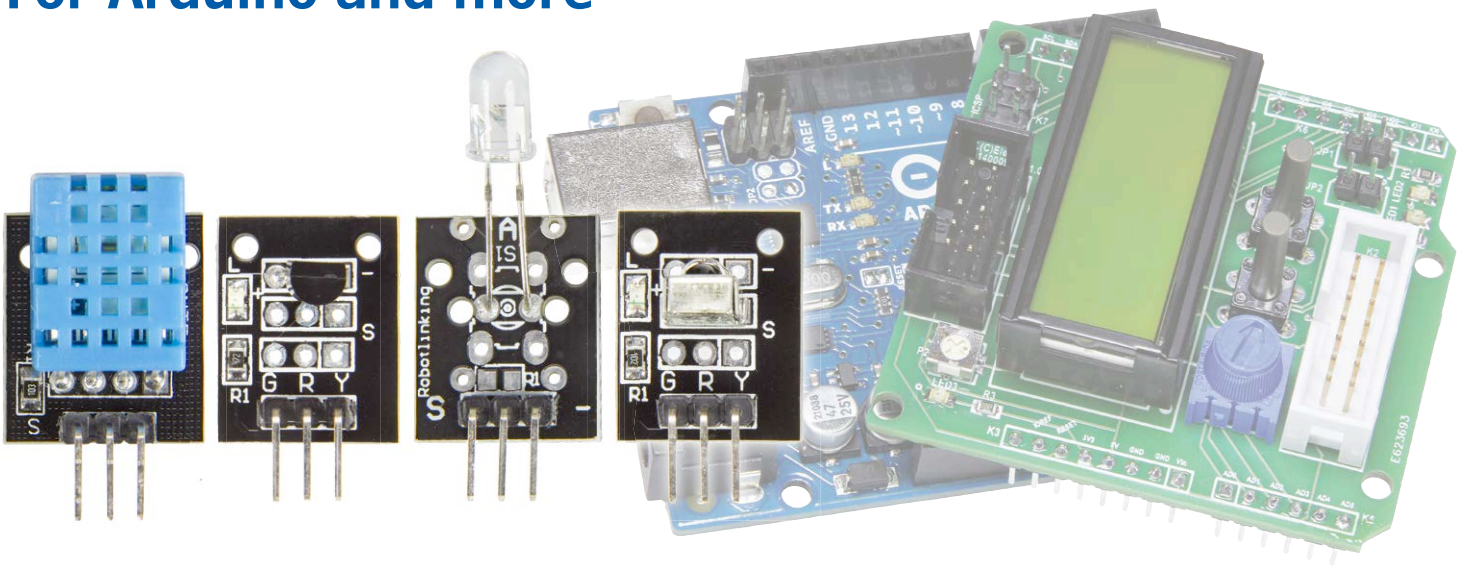
Sensors Make Sense (3)

For Arduino and more



By **Burkhard Kainka** (Germany)

Ever since the earliest days of telegraphy, electrical communication has been possible over a single wire. Today the 1-Wire Bus and similar protocols function using just one conductor, without the need for an additional clock line. Infrared (IR) remote controls operate on similar principles. In this session we get to grips with temperature and humidity sensors among other things, and in particular with the *IR Transmitter* and *Receiver* in Elektor's 35 Sensors Kit.

We commence with the temperature sensor *18B20 Temp* in the kit, which is available from the Elektor Store [1]. The DS18B20 under discussion may look like a regular transistor in a TO92 package but it's actually a complex IC embracing a temperature sensor and a special interface. The 1-Wire Bus was developed by the Dallas Semiconductor Corporation. One or more sensors can be handled using a single wire, if you discount the GND line. As this wire carries only data, it can also be used to power the IC. In the main, however, people use the third (V_{DD}) connection of the DS18B20 for supplying power, which then adds up to three wires in total. On the sensor PCB there is also an LED plus its dropper resistor, connected to the data line.

Arduino software for the 18B20

There are two Libraries that we need to load into the Arduino IDE: *OneWire* and *DallasTemperature*. Both are supplied on

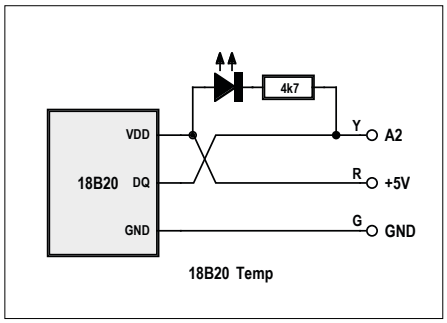


Figure 1. 18B20 temperature sensor connections.

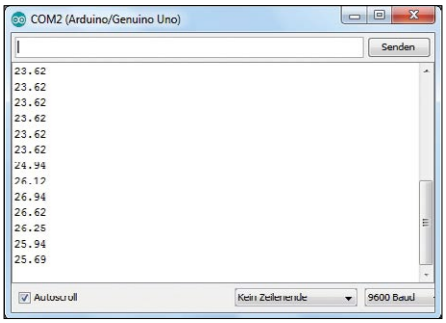


Figure 2. Temperature output in the serial monitor.

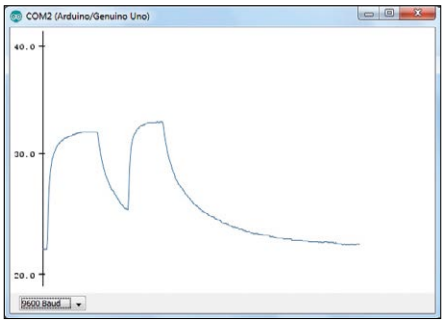


Figure 3. Temperature curve.

Listing 1. Temperature measurement using the DS18B20.

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <LiquidCrystal.h>
#define ONE_WIRE_BUS A2
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);

float temp;
int minTemp;
int maxTemp;
LiquidCrystal lcd(2,3,4,5,6,7);

void setup(void)
{
  Serial.begin(9600);
  sensors.begin();
  lcd.begin(16, 2);
  minTemp = 100;
  maxTemp = -100;
}

void loop(void)
{
  sensors.requestTemperatures();
  temp = sensors.getTempCByIndex(0);
  Serial.println(temp);
  lcd.setCursor(0, 0);
  lcd.print(temp);
  lcd.print (" C   ");
  if (temp < minTemp) minTemp = temp;
  if (temp > maxTemp) maxTemp = temp;
  lcd.setCursor(0, 1);
  lcd.print(minTemp);
  lcd.setCursor(5, 1);
  lcd.print(maxTemp);
  delay (500);
}
```

the Sensor Kit CD and they have to be copied into the *Libraries* folder in the *Arduino Sketchbook* directory. In each case you will find a header file **.h* and a C++ file **.cpp*. It's interesting to look closer at these files. If you manage to get all the way to the end, leaving nothing at all unread, that's pretty good going! After that, all you need do is integrate the header files into your own program and then call up a few functions. Thanks to Arduino, all this is straightforward. The program in **Listing 1** shows how to interrogate and retrieve a temperature pure and simple, repeated every 500 ms. The data line is connected to A2 (= PC2) (**Figure 1**). All the code mentioned in this article can of course be downloaded from the Elektor website [4].
Not only does the thermometer have a serial output but it can also indicate the temperature on the LCD screen, if you are using the Elektor Extension Shield [2] [5]. And because there is space on the second line of the display, we have additionally programmed a minimum/maximum thermometer.

The temperature is indicated as a real number to two decimal places. The actual resolution amounts to 0.06 degrees C per step. The absolute accuracy is given within 0.5 degrees. On the serial monitor (**Figure 2**) you can read the data. A touch on the sensor will demonstrate a change in temperature.
The serial plotter indicates changes in temperature over time (**Figure 3**). In the plot shown the sensor was touched (warmed) twice by finger. This shows clearly the differing time constants for heating and cooling. Also interesting is how the finger was visibly warmer on the second touch. Something must have caused the rise in temperature during the intervening time. Thanks to the high resolution of the sensor even small variations can be detected.

18B20 in Bascom

Bascom supports the 1-Wire Bus, although the Bus alone does not provide a complete solution for using the 18B20. So you need to dive a little deeper (into the data sheet) and do some programming of your own. Of course this takes some time but it does also open up some interesting opportunities. You can then make use of some special features of the sensor, for instance for altering the resolution or for reading out the unique ID number included in every IC.

The Bascom sample code (**Listing 2**) shows how we read out temperatures. Two commands need to be sent, (*Skip ROM, &HCC* and *Convert T, &H44*). Following a delay while the measurement itself is taken, we send the command *Read Scratch-*

The 1-Wire protocol

With the 1-Wire Bus everything is channeled down the single wire DQ. However, we also have GND and V_{CC} wires. In standby mode the data line DQ is taken High with a pull-up resistor. The Master (Controller) can now send a Reset pulse for initiating communication with Slaves, in order to then send commands or receive data. Both partners can load data onto the Bus. A 0 Bit is represented by a 15 µs long Low pulse and a subsequent, 45 µs long High state. In contrast a 1 Bit is symbolized by a 60 µs long Low pulse. Between individual Bits there is a resting state, during which the data line is taken High by the pull-up.
The Master always sends a Reset followed by one or more commands. A sensor chip then responds with the wanted data. If you study the data sheet for the chips in detail, you will find information there not only on the general Bus protocol but also countless commands and the makeup of the data structure that is repeated back. Matters become even more complex, because multiple Slaves can be attached to the same Bus. You can hardly imagine how much work it would take to program all of this yourself. Thank goodness you don't have to reinvent the wheel every time and can refer to ready-made code almost always.

Listing 2. Temperature measurement in Bascom (excerpt).

```
'DS18B20LCD  AD2, PORTC.2
...
Do
  1wreset
  1wwrite &HCC
  1wwrite &H44
  Waitms 800
  1wreset
  1wwrite &HCC
  1wwrite &HBE
  Dat(1) = 1wread(2)
  1wreset
  Temp = 256 * Dat(2)
  Temp = Temp + Dat(1)
  Temp = Temp * 0.0625
  Print Temp
  Tempint = Round(temp)
  If Tempint > Maxtemp Then Maxtemp = Tempint
  If Tempint < Mintemp Then Mintemp = Tempint
  Locate 1 , 1
  Lcd Temp ; " C   "
  Locate 2 , 1
  Lcd Mintemp
  Locate 2 , 5
  Lcd Maxtemp
  Waitms 200
Loop

End
```

pad (&HBE) and then read out the two Bytes. From these the software calculates a 16-Bit number and the temperature (in steps of .0625 degrees). That's pretty clever and even better, the program is only slightly longer than the Arduino version. Once again we have the bonus of a minimum/maximum thermometer with LCD readout.

Temperature and humidity using the DHT11

At first glance the combined humidity and temperature sensor DHT11 looks like a straightforward resistive humidity sensor. This impression is reinforced when you see that an analog Pin is recommended (**Figure 4**). In reality, however, the insignificant-looking exterior conceals a complex sensor with a digital interface.

The Chinese company Aosong has come up here with a scheme that (only at first sight) brings to mind the 1-Wire Bus of Dallas Semiconductor. Each measurement is initiated by a Low pulse from the Master (controlling device) lasting at least 18 ms. After this a total of 40 Bits are read out, which the Master requests each time by means of an 80 µs long Low pulse. The sensor responds with High pulses, in which a period lasting 28 µs maximum stands for a zero and a period of 70 µs for a one. The 40 Bits then contain one High Byte and one Low Byte for the humidity and the temperature plus an additional parity Byte for checking correct transfer of the data. With the DHT11 the Low Bytes are always set to zero, meaning that no post-decimal point figures are transferred. However, there is also a DHT22 device using the same protocol, which does indeed handle the decimal places in addition. Consequently both types of sensor can be interrogated using the same software Library. Once again we are fortunate that someone has already taken the trouble to re-format the complicated protocol into an Arduino Library. Using this is quite simple, once you have copied the Library directory *DHT* from the CD into the Arduino Library folder. The sample code (**Listing 3**) indicates how the two

Listing 3. DHT11 in Arduino-C.

```
//DHT11LCD, pin AD2

#include <dht.h>
#define dht_apin A2
#include <LiquidCrystal.h>
LiquidCrystal lcd(2,3,4,5,6,7);
float temperature;
float humidity;

dht DHT;

void setup(){
  Serial.begin(9600);
  delay(500);
  delay(1000);
  lcd.begin(16, 2);
}

void loop(){
   DHT.read11(dht_apin);
   humidity = DHT.humidity;
   temperature = DHT.temperature;
   Serial.print("Humidity = ");
   Serial.print(DHT.humidity);
   Serial.println(" %  ");
   Serial.print("Temperature = ");
   Serial.print(temperature);
   Serial.println(" C  ");
   lcd.setCursor(0, 0);
   lcd.print(temperature);
   lcd.print (" C   ");
   lcd.setCursor(0, 1);
   lcd.print(humidity);
   lcd.print (" %   ");
   delay(2000);
}
```

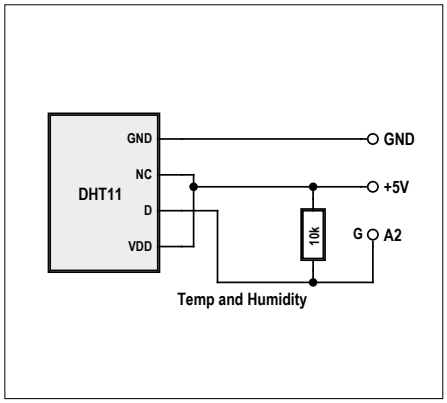


Figure 4. Temperature and humidity sensor DHT11 connections.

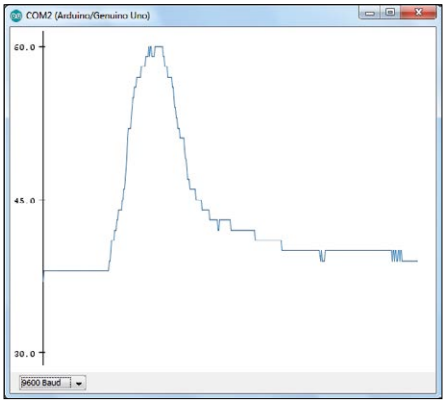


Figure 5. Measuring ambient humidity by finger touch.

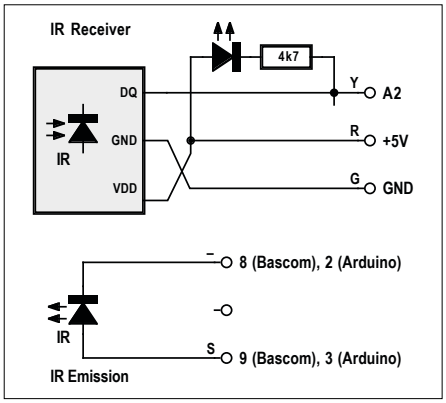


Figure 6. IR receiver and transmitter.

measured values are interrogated and can be displayed on the LCD screen. Once again we select Pin AD2 for connecting the data line.
On the datasheet the absolute accuracy for temperature measurements is stated modestly as within 2 degrees, that for air humidity as within 5 %. Ambient humidity is difficult to judge because all run-of-the-mill moisture meters are fairly imprecise. However, for temperature at least, the results measured on our sample device seemed really good. A digital thermometer that was to hand indicated 23.8 °C, the DHT11 showed 24 °C, and the DS18B20 read 23.37 °C, in each case leaving enough time for the sensor to settle properly.

DHT11 and Bascom

Many ready-to-use commands and functions exist in Bascom of course but the DHT11 is not supported directly as such. Consequently for most topics your most profitable approach is to search the Net to see if someone else has tackled them already. On this occasion our search delivered positive results in the Bascom Forum, where a user by the name of Grütze had written a program called *DHT11LCD.bas* that does exactly what we want. The code is easy to read and reflects more or less exactly what is said in the data sheet. Only minor adjustments are needed for using it with the Extension Shield and enabling the sensor to work using Pin AD2. Beyond this, we also incorporated a serial output, initially only for the humidity data, as it was intended that these would be mapped out using the serial plotter from the Arduino IDE (**Figure 5**).
The significant activity takes place in the function *Get_dht11()*. This is where we carry out exactly the same processes that exist in the corresponding Library for the Arduino. In this way we can create a complete miniature weather station equally well using Bascom (**Listing 4**).

Infrared remote control

We are all familiar with infrared remote controls or 'zappers' for TVs and other home entertainment equipment. Numerous different manufacturers and protocols exist that are not compatible with one another, meaning that a remote control handset must be a correct match for the equipment it commands. One feature common to all designs is that the signal sent by the infrared transmit diode is modulated with a frequency between 30 and 40 kHz. This signal is then pulsed (in one of a number of methods) so as to transmit individual packets of data. An integrated infrared receiver detects the signals with a photo diode, amplifies and filters them and demodulates them back into a digital signal. By design the internal filters are dimensioned for specific frequencies in the range 30 to 40 kHz, although the bandwidth is sufficient to also receive 'off-frequency' signals at shorter ranges.

You can also use an IR zapper for other, more general, remote control or switching tasks. The Sensor Kit includes an IR transmit diode or emitter (*IR emission* in the diagram) and an integrated IR Receiver (**Figure 6**). In conjunction with an Arduino you have the choice of receiving or transmitting IR signals (or both!). Here we will demonstrate a program that can do both. Two buttons are used to send commands that are evaluated in the receiver in order to switch an output. The same assignment is performed both in Bascom and in Arduino-C. Both programs are sufficiently compatible to the extent that that the Bascom controller can tell the Arduino-programmed Uno what is to be switched and vice versa.

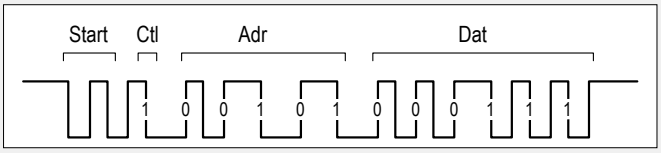
Listing 4. Using the DHT11 in Bascom (excerpt).

```
'DHT11LCD an AD2, PORTC.2
...
Do
   If Get_dht11() = 1 Then
      Print Humidity
      Locate 1 , 1
      Lcd "H: " ; Humidity ; " %  "
      Locate 2 , 1
      Lcd "T: " ; Temperature ; " C  "
    End If
    Waitms 1000
Loop

End
```

The RC-5 protocol

Infrared remote controls for TV receivers, video recorders and other home entertainment devices operate in part using the RC-5 standard defined by Philips. This employs modulated optical signals in the range 30 kHz to around 40 kHz. The remote control sends individual bursts (packets of data pulses) 0.888 ms or 1.776 ms in length. At a modulation



frequency of 36 kHz a short burst contains 32 individual pulses and a long one 64. The complete data packet lasts about 25 ms and is repeated every 100 ms for as long as a button is pressed.
The protocol employs a bi-phase signal. A Bit has a length of 1.776 ms. If the 36 kHz pulse lies in the first half of this time period, it represents a logical Zero; a logical One is signaled by a pulse in the second half. The signal is introduced every time with a start sequence that never changes. There then follow three data fields:

- The Control or Check Bit (Ctl) alternates between 0 and 1 with every key press. In this way the receiver can differentiate whether a key has been pressed once for a long time or several times briefly.
- The Device or System Address Bits (Adr) comprise 5 Bits, in which the high-value Bits are transferred first. Common device addresses are 0 for TV sets and 5 for video recorders. In this way several remote controls can be deployed in the same room.
- The Data or Command Bits (Dat) comprise 6 Bits for up to 64 differing keys (pressbuttons). The number keys 0 to 9 generate codes from 0 to 9. Here too the highest value Bits are sent first.

A standard frequently used for IR remote controls is RC-5, developed by Philips (see boxout panel). Simple commands for this are provided in Bascom, making decoding zappers a simple task. **Listing 5** provides a simple RC-5 receiver and transmitter in Bascom. All received data is displayed on the LCD screen. The command is also sent over the serial interface. At the same time a check is made for any output to switch to Port B. Given that LED2 is available at B.2 on the Extension Shield, we have designed the code to illuminate this with button 2 on the zappers and switch it off with button 0.

The RC-5 receiver can be connected to any input Pin you choose. So we have again deployed input PC3 (AD3) for this. For the command *Getrc5* Bascom uses the *Timer0* Interrupt in the background, which you must enable globally. In addition we have switched in the internal pull-up resistor for input PC3. It is then possible to use the controller for transmitting (only) without an IR receiver connected. Had we not done this, an open high-impedance input without a pull-up might otherwise assume a Low state and cause the program to hang.

For the transmit output (command *Sendrc5* in Bascom) we normally use PB1 (Arduino Pin 9), because the output OC1A of Timer 1 is connected to this Pin, used for generating the 36 kHz transmit signal (warning: in the Arduino-C++ software a different output is used for this task, meaning that on this occasion we cannot keep the same hook-up allocations). In Bascom the standby state of this Portpin must be defined in advance using a Port command; here it needs to be Low during inactive intervals. For each pulse packet the software then switches over from the Port to the timer output. Because the IR diode has no series resistor, it makes sense to switch this to a different output Port, in order to use the two internal resistors for current limiting. Here we selected PB0, which is also switched Low as an output. In that way we still have four outputs on Port B as potential switching outputs for received data. In transmit mode two commands are supported. If you press button S1 on the Extension Shield, then the code for button 2 on the remote control is transmitted. This switches on the output at the receiver, making LED2 come on. With button S2 you send a zero and switch off the output at the receiver once more.

Arduino and IR

For programming in Arduino-C we need to install the *IRremote* Library (**Listing 6)**. This makes use of Timer2 and its output OC2B (PD3, Arduino Pin 3) for generating pulses applied to the IR LED. This is also the E wire of the LCD screen on the Extension Shield. That unfortunately means the program cannot interact simultaneously with the LCD. However, that's not totally bad news, as we still have the serial outputs for viewing the received data.
The input is a matter of choice and is assigned here to A2 once more. Programming an input pull-up is overwritten by the Library, rendering this ineffective. But it's not even necessary, as reception does not block the program even when an open input is in the Low state. Our goal remains achievable, to use a program without modification or reconfiguration for a choice of receiving only, transmitting only or handling both tasks.

The crucial advantage of the *IRremote* Library is that it can 'speak' not only RC-5 but also a multiplicity of other standards. Most of us have a whole collection of disparate remote controls at home. If so, it makes good sense to point each of these once at the IR Receiver. You will then get a report of the standard employed and the data received. RC-5 is shown as Type 3. If you press several times on button 2 of an RC-5-compatible remote control, the following messages appear:

```
3
382
3
B82
```

The data is output as 12-Bit numbers. The lower 5 Bits denote the key code. Next comes the 5-Bit device address, in this case the address 14 for a DVBT receiver. The highest value Bit is the

Toggle Bit, which changes with every key press and does not need to be evaluated. It's worth noting down this information, so you can resend it identically. You can ignore the Toggle Bit when doing this. If you transmit 382_{hex} via the IR diode, this corresponds to button *2* in RC-5 code.
Because in this situation the Arduino has to get by without the Extension Shield, the switching output is assigned to connector 13. Doing this enables us to control the LED on the Arduino. The relay can now be connected to Portpin 13 and straightaway a load can be switched either with an RC-5 remote control or else by a second Arduino with an IR diode.
Here too the pressbutton commands 2 and 0 can be transmitted back in the reverse direction. As in the Bascom version, the relevant buttons are assigned to A0 and A1, in which the internal pull-ups have been enabled. Here you can hook up plenty of the things that the sensor kit has in store. Of course these do not have to be touch switches every time. Why not use a magnetic sensor, a position sensor or an optical sensor? You could turn an infinitely large number of ideas into reality. The TV could be turned off by sunrise at the latest, or make the position sensor on the door handle recognize when somebody enters the room and then switch on the lights and the radio to welcome them!

(160210)

Web Links

[1] www.elektor.com/arduino-sensor-kit
[2] www.elektormagazine.com/160152
[3] www.elektormagazine.com/160173
[4] www.elektormagazine.com/160210
[5] www.elektormagazine.com/140009

Listing 5. RC-5 transmitter and receiver in Bascom.

```
'RC5LCD  In AD2, PORTC.2, Out OC1A, PORTB1
...
Do
  If S2 = 0 Then
    Togbit = 0 : Address = 0 : Command = 2
    Do
      Rc5send Togbit , Address , Command
      Waitms 100
    Loop Until S2 = 1
  End If
  If S1 = 0 Then
     Togbit = 0 : Address = 0 : Command = 0
     Do
      Rc5send Togbit , Address , Command
      Waitms 100
    Loop Until S1 = 1
  End If
  Getrc5(address , Command)
  If Address < 255 Then
    Locate 1 , 1
    Lcd Address ; "   "
    Locate 2 , 1
    Togbit = Command / 128
    Lcd Togbit ; " "
    Locate 2 , 5
    Command = Command And &B01111111
    Lcd Command ; "   "
    Print Command
    If Command = 2 Then Portb.2 = 1
    If Command = 0 Then Portb.2 = 0
  End If
Loop
```

Listing 6. IR control in Arduino-C.

```
#include <IRremote.h>
int RECV_PIN = A2;
IRrecv irrecv(RECV_PIN);
IRsend irsend;
decode_results results;
int d;
int S1 = A0;
int S2 = A1;
int LED = 13;
int kathode =2;
void setup()
{
  Serial.begin(9600);
  irrecv.enableIRIn();
  pinMode(S1, INPUT_PULLUP);
  pinMode(S2, INPUT_PULLUP);
  pinMode(RECV_PIN, INPUT_PULLUP);
  pinMode(LED, OUTPUT);
  pinMode(kathode, OUTPUT);
}

void loop() {
  if (irrecv.decode(&results)) {
    Serial.println(results.decode_type);
    Serial.println(results.value, HEX);
    d = results.value & 15;
    Serial.println(d);
    if (d==2) digitalWrite(LED,1);
    if (d==0) digitalWrite(LED,0);
  }
  if (digitalRead(S1) == 0) irsend.sendRC5(0x382,
32);
  if (digitalRead(S2) == 0) irsend.sendRC5(0x380,
32);
  irrecv.enableIRIn();
  delay(100);
}
```