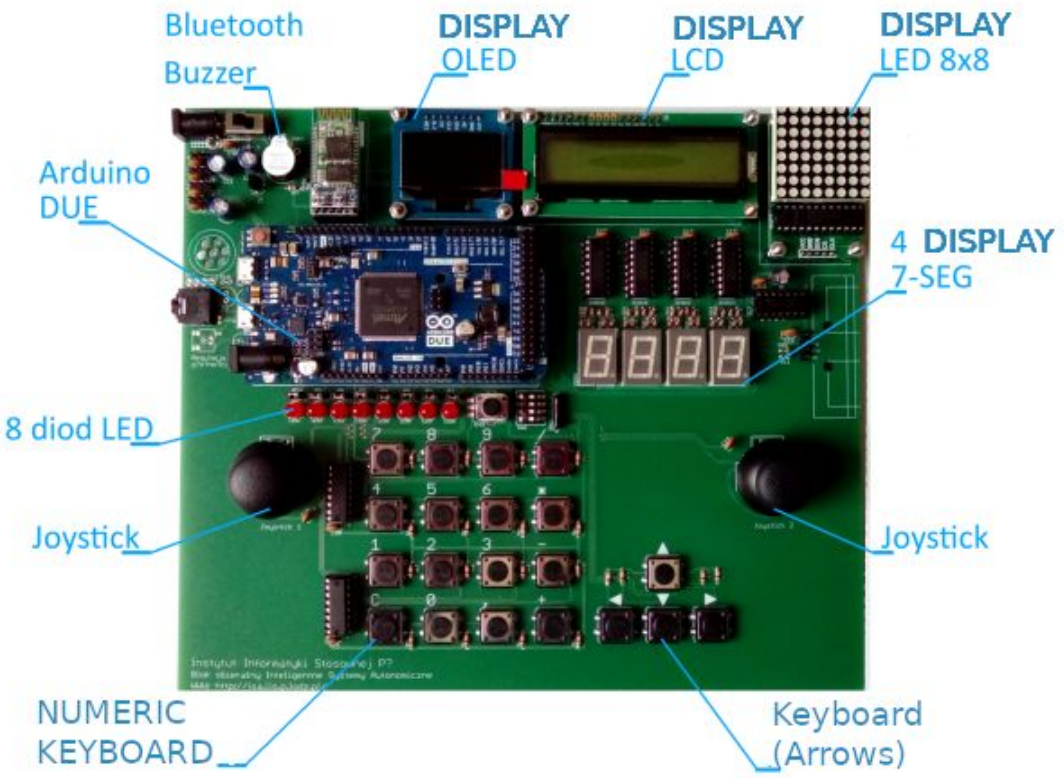
Introduction to programming Arduino based Educational Board

Authors: **Piotr Duch, Tomasz Jaworski**
Institute of Applied Computer Science
Łódź University of Technology

Contact: pduch@iis.p.lodz.pl, tjaworski@iis.p.lodz.pl

Educational Board



LCD Display

Functions:

LCD display is an alphanumeric display, which has 2 rows and 16 columns. Library **ISALiquidCrystal** facilitates use of the LCD display mounted on educational development board. Class **ISALiquidCrystal** is responsible for handling LCD display which is available after including **LiquidCrystal.h** library to the program. In your program you can have only one object of the **ISALiquidCrystal** class.

Methods available in class **ISALiquidCrystal:**

- *lcd.begin()* - initialization of LCD display driver. It should be called in *setup()* function. *lcd* - a variable of type **ISALiquidCrystal.**
- *lcd.print(data)* - displays the message passed in the parameter *data*, on LCD display. The data to be displayed can be one of the following types: **char**, **byte**, **int**, **long** or **String**. *data* - data to be displayed on the LCD display.
- *lcd.setCursor(col, row)* - sets cursor in a given position. *col* - the column at which to position the cursor, *row* - the row at which to position the cursor. **Rows and columns are numbered from 0.**
- *lcd.clear()* - clears LCD display and sets the cursor in the left upper corner (0, 0).

delay(ms) - pause the currently running program for a specified amount of time. *ms* - a number of milliseconds to pause the program.

Example - program displaying the words “hello, world!” on the LCD screen:

```
#include <ISALiquidCrystal.h>
ISALiquidCrystal lcd;
void setup()
{
  lcd.begin();
  lcd.print("hello, world!");
}
void loop() {}
```

Task 1.

Write a program that will display the time elapsed since the program was started in the format mm::ss. Remember to complete the number with a non-significant zeros before displaying. (Use the delay() function).

LEDs

Functions and symbols:

- *pinMode(pin, mode)* - determines the operating mode for each pin. *pin* - pin number, which mode of operation is to be set, *mode* - pin operation mode, can take one of two values: **OUTPUT** or **INPUT**.
    - **INPUT** means that pin will work in the input mode, which allows the program to read information from the device connected to the pin (eg the up arrow button).
    - **OUTPUT** means that pin will work in the output mode, which allows the program to control the device connected to the pin. eg turn on/off LEDs.
- *digitalWrite(pin, value)* - write a *value* to the *pin*. *pin* - the pin number, *value* - can take one of two values: **HIGH** or **LOW**
    - **HIGH** corresponds to the logical true value (1) in C (eg LED on).
    - **LOW** corresponds to the logical false value (0) in C (eg LED off).
- Educational development board have 8 LEDs, marked with symbols **LED1**, **LED2**, …, **LED7**, **LED8**. There is also available array **LEDS**[], allowing to refer to the appropriate diode thanks to its index, eg code *int i* = 4; *LEDS*[i] is equivalent to **LED5**. All these symbols are available after including header file **ISADefinitions.h.**

Example - program to turn on LED1:

```
#include <ISADefinitions.h>

void setup(){
  pinMode(LED1, OUTPUT);
  digitalWrite(LED1, HIGH);
}
void loop(){}
```

Task 2.

Write a program that will turn on the first LED, then turn on the next LED in the same time turning off the previous one etc. The turned on LED should first move from the left to the right and then in opposite direction. Only one LED can be turned on at a time.

Buttons

Function:

The **ISAButtons** library is responsible for buttons handling (there are 16 buttons available on numeric keypad, buttons are numbered from 0 to 15). Library is available after including header file **ISAButtons.h** to the program. In your program you can have only one object of the **ISAButtons** class.

Methods available in class ISAButtons:

- *buttons.init()* - initialization method for buttons. It should be called in a function *setup()*. ***buttons*** - a variable of type **ISAButtons**.
- *buttons.buttonPressed(buttonId)* - checks if the button with the given index has been pressed. Returns **true** if the button has been pressed otherwise returns **false**. *buttons* - a variable of type **ISAButtons**, *buttonId* - button identifier, which state is to be checked. The button state is refreshed every 25 milliseconds.
- *buttons.buttonReleased(buttonId)* - checks if the button with the given index has been released. Returns **true** if the button has been released otherwise returns **false**. *buttons* - a variable of type **ISAButtons**, *buttonId* - button identifier, which state is to be checked. The button state is refreshed every 25 milliseconds.
- *buttons.buttonState(buttonId)* - This method checks state of the button with the given index. Returns true if the button has been pressed otherwise returns false. *buttons* - class object **ISAButtons**, *buttonId* - button identifier, whose condition is to be checked. The button state is refreshed every 25 milliseconds.

Arrow Keys

Additionally, the four arrow keys are available. However, they are not supported by the class **ISAButtons** (which is used to handle numeric keypad). To read state from these keys, you should use function *pinMode* and *digitalRead*. Keys are available under the following symbols **KEY_UP**, **KEY_DOWN**, **KEY_RIGHT** and **KEY_LEFT**. There also exists array of arrow - **KEY_ARROWS**[]. These symbols are available in header file **ISADefinitions.h**.

Warning! Arrow keys work in reverse logic - function *digitalRead* for pressed key return **false** otherwise **true**. You should use negation (!).

Example - checking button state with library ISAButtons:

```
#include <ISADefinitions.h>
#include <ISAButtons.h>
ISAButtons button;
bool state = false;
void setup(){
  pinMode(LED1, OUTPUT);
  button.init();
}
void loop(){
```

```
  if (button.buttonPressed(0)) {
     state = !state;
     digitalWrite(LED1, state);
  }
  delay(30);
}
```

Example- checking state of “up” key:

```
#include <ISADefinitions.h>

void setup(){
  pinMode(KEY_UP, INPUT);
  pinMode(LED1, OUTPUT);
}
void loop(){
  bool up = !digitalRead(KEY_UP);
  digitalWrite(LED1, up);
}
```

Task 3.

Using the combination of two buttons, move the diode to the left or right. Only one LED can be turned on at a time.

Potentiometer and Joysticks

Functions:

- *analogRead(pin)* - reads the value from the pin. Returns integer value in range 0 - 1023. *pin* - pin number of the analog input pin to read from.
- *analogWrite(pin, value)* - writes the analog value to the pin. It can be used to set intensity of light emitted by diode or to set velocity of an engine. *pin* - pin number of the analog output pin to write to, *value* - a value from range 0 – 255, where 0 is always off, and 255 always on.
- Potentiometer and joysticks are available through the symbols from header file **ISADefinitions.h**. These symbols are: **POT** (potentiometer), **JOY1X**, **JOY1Y** (joystick 1, X and Y axis), **JOY2X**, **JOY2Y** (joystick 2, X and Y axis).

Example - program reads the current value of the potentiometer and displays it on the LCD display:

```
#include <ISADefinitions.h>
#include <ISALiquidCrystal.h>
ISALiquidCrystal lcd;
void setup(){
     lcd.begin();
}
void loop()
{
     lcd.clear();
     int pot = analogRead(POT);
     lcd.print(pot);
     delay(250);
}
```

Task 4.

Using the potentiometer, move the diode to the left or right (note that the analogRead function returns values between 0 and 1023).

Task 5.

Write a program that will display the current Joystick tilt on the LCD display.

Seven-segment display

Functions:

Library **ISA7SegmentDisplay** is responsible for handling seven-segment display. It is available after including header file **ISA7SegmentDisplay.h**.

Methods available in class ISA7SegmentDisplay:

- *seg.init()* - initialization of seven-segment display. *seg* - a variable of type **ISA7SegmentDisplay**. The method should be called in the function *setup*().
- *seg.displayDigit(digit, dispID, dot = false)* - displays the digit on the display. *seg* - a variable of type **ISA7SegmentDisplay**, *digit* - the digit to be displayed, must be in the range <0, 9>, in other case function does nothing, *dispID* - identification number of display, on which the number should be displayed, must be a value in the range <0, 4), in other case functions does nothing, *dot* - pass information whether to display a dot or not, by default is set to false.
- *seg.setLed(values, dispID)* - sets state of LED on a given display. *seg* - a variable of type **ISA7SegmentDisplay**, *values* - 8-bit value, in which every bit set to 1 turn on the given diode, and set to 0 turns it off, *dispID* - id of the display on which the LEDs should be switched on, must be a value in the range <0, 4), otherwise the function will not do anything.

Example - program displays digits on a seven-segment display:

```
#include <ISADefinitions.h>
#include <ISA7SegmentDisplay.h>
ISA7SegmentDisplay sseg;
void setup(){
  sseg.init();
}
void loop(){
  for (int i = 0; i < 10; ++i) {
     sseg.displayDigit(i, 0);
     delay(250);
  }
}
```

Task 6.

Write a program that will display on the seven-segment display the time that has elapsed since the program was started, in the format mm: ss. Use a dot to display a colon (parameter dot).

8x8 LED matrix

Functions:

Library **ISALedControl** is responsible for handling 8x8 LED matrix, it is available after including header file **ISALedControl.h**.

Methods available in class ISALedControl:

- *lc.init()* - initializes 8x8 LED matrix. *lc* - a variable of type **ISALedControl**. Method should be called in *setup*() function.
- *lc.clearDisplay()* - turns off all diodes on the display. *lc* - a variable of type **ISALedControl**.
- *lc.setRow(row, value)* - changes the state of diodes in one row. *lc* - a variable of type **ISALedControl**, *row* - row id, *value* - eight bit value, in which each bit set on 1 turns on corresponding diode, and each bit set on 0 turns off diode.
- *lc.setColumn(col, value)* - changes the state of diodes in the given column. *lc* - a variable of type **ISALedControl**, *col* - column id, *value* - eight bit value, in which each bit set on 1 turns on corresponding diode, and each bit set on 0 turns off diode.
- *lc.setLed(row, col, state)* - changes the state of the diode. *lc* - a variable of type **LedControl**, *row* - row id (from range <0; 7>), *col* - column id (from range <0; 7>), *state* - state of the diode (**true** - turned on, **false** - turned off).

Example - program turns on diode in upper left corner of the 8x8 LED matrix display:

```
#include <ISALedControl.h>
ISALedControl lc;
void setup()
{
  lc.init();
  lc.setLed(0, 0, true);
}
void loop() {}
```

Task 7.

Write a program, that will turn on one by one each diode (in snake's zigzag pattern) on 8x8 LED matrix display.

OLED display

Functions:

Library **ISAOLED** is responsible for handling OLED. It is available after including header file **ISAOLED.h**. Object of class **ISAOLED** stores a frame of image (of size 128 columns and 64 rows) and allows execution of simple graphical operations, eg drawing points. After performing graphical operations one should execute method *renderAll(),* which will send the frame containing the image from objects (Arduino memory) to the display. In effect created earlier image will be displayed.

Methods available in class ISAOLED:

- *oled.begin()* - initializes OLED display of resolution 128x64. *oled* - a variable of type **ISAOLED**. The method should be called in the function *setup*().
- *lc.clear(bool render = true)* - turns off all pixels on the display. *oled* - a variable of type **ISAOLED**, *render* - logical value resulting in immediate removal of image from display.
- *oled.write(data)* - displays a character based on ASCII code. *oled* - a variable of type **ISAOLED**, *data* - ASCII code of a symbol, that is going to be displayed, non-ASCII symbols will not be displayed.
- *oled.*gotoXY*(cx, cy)* - sets a cursor for displaying text in position *cx*, *cy*. *oled* - object of type **ISAOLED**, *cx* - coordinate on X axis, on which the cursor is set (accepts values from range <0; 127>), *cy* - coordinate on Y axis, on which the cursor is set (accepts values from range<0; 7>).
- *oled.setPixel(x, y, v)* - changes state of the given pixel. *oled* - a variable of type **ISAOLED**, *x, y* - coordinates of the pixel, *v* - state of the pixel (**true** - turned on, **false** - turned off).
- *oled.writeLine(x1, y1, x2, y2)* - draws only vertical or horizontal lines. *oled* - a variable of type **ISAOLED**, *x1, y1* - coordinates of beginning of the segment, *x2, y2* - coordinates of end of the segment.
- *oled.writeRect(x, y, w, h, fill)* - draws a rectangle. *oled* - a variable of type **ISAOLED**, *x, y* - coordinates of upper left corner of the rectangle, *w* - width of the rectangle, *h* - height of the rectangle, *fill* - flag indicates whether the rectangle will be filled or not (**true** - filled, **false** - only borders).
- *oled.print(text)* - displays text on the display. *oled* - a variable of type **ISAOLED**, *text* - text, that is going to be displayed (may include also digits).
- *oled.renderAll()* - sends built frame of image to the display and displays it immediately. *oled* - a variable of type **ISAOLED**.

Example - program displays words “Hello world” on OLED display:

```
#include <ISAOLED.h>
ISAOLED oled;
void setup() {
  oled.begin();
```

```
  oled.gotoXY(20, 2);
  oled.print("Hello world");
  oled.renderAll();
  delay(1000);
}
void loop() {}
```

Task 8.

Write a program, that will draw on OLED display a chessboard of size 8x8.

Thank you for taking part in Arduino classes :)