

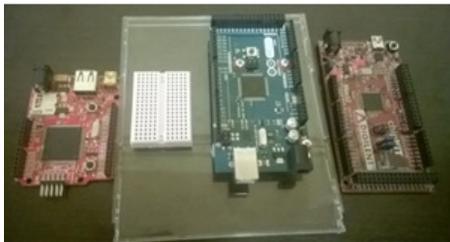
Laboratory 3 – Working with the LCD shield and the interrupt system

1. Working with the LCD shield

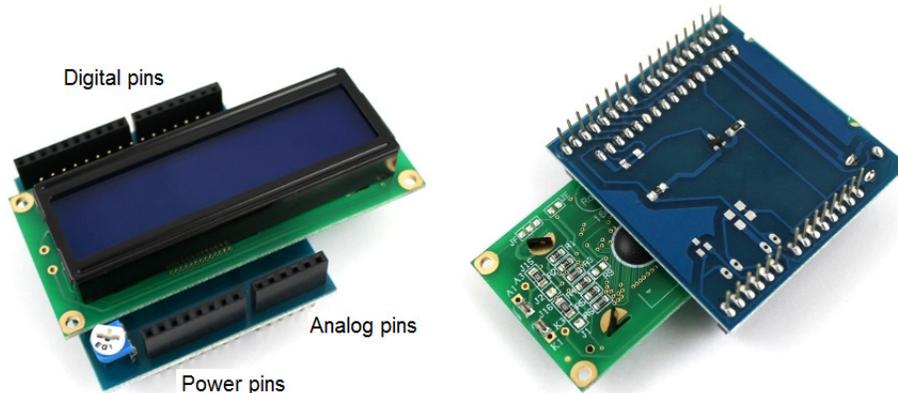
The „shields” are PCBs (Printed Circuit Boards) that can be placed over the Arduino boards, extending their capabilities. The mapping between the Arduino pins and the devices present on the shields should be carefully taken into account in when programming for proper mapping and conflicts avoidance (the pins provided/used by the shield are specified in the product datasheet). In the figure bellow such shield examples are shown: an LCD shield, an SD Card shield and a voice recognition EasyVR shield.



Since the Arduino boards are very popular, many competitors are using the same design layout (see the figure bellow: left - a Fez Domino board with ARM processor; right- a CipKit board with PIC processor) being also able to use the Arduino compatible shields.



In this laboratory an LCD (Liquid Crystal Display) shield will be used. It contains the display and a potentiometer for tuning its brightness.

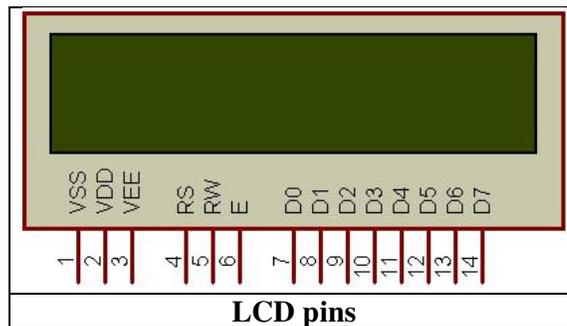


The shield is placed over the Arduino board in such a way that the longer pin bars (10 + 8) correspond to the digital pins (see the figure below). **DO NOT remove the shield from the Arduino board (because you will damage its pins)!!!**



The LCD uses the digital pins 2 ... 7: 7 – RS, 6 – En, 5 – DB4, 4 – Db5, 3 – DB6, 2 – DB7 (their mining is in the table bellow).

The shield uses the classic LCD controller Hitachi HD44780. One such controller can be used for LCDs with max. 80 characters and 4 lines. The pinout of the shield is shown bellow:



Pin No.	Name	Description
1	VSS	Power supply (GND)
2	VCC	Power supply (+5V)
3	VEE	Contrast adjust
4	RS	Register Select: 0 = Instruction input; 1 = Data input
5	R/W	0 = Write to LCD module; 1 = Read from LCD module
6	EN	Enable signal
7	D0	Data bus line 0 (LSB)
8	D1	Data bus line 1
9	D2	Data bus line 2
10	D3	Data bus line 3
11	D4	Data bus line 4
12	D5	Data bus line 5
13	D6	Data bus line 6
14	D7	Data bus line 7(MSB)
LCD pins description		

The HD44780 contains 2x 8 bits registers: a Data register and an Instruction register. The instruction register is used to send command to the LCD (i.e. shift, clear etc.). The data register is used to buffer the displayed data. When the EN signal is activated, the Data present on the data pins are transferred firstly in the Data register and then moved in the DDRAM (Display Data RAM) and displayed on the LCD. The Data register is used also to transfer data to the CGRAM (Character Generator RAM) – the memory used to store the user defined characters.

The DDRAM (Display Data RAM) stores the data to be displayed, represented as 8 bit characters. Its total capacity is 80x8 bits (80 characters). The remaining free DDRAM memory can be used as a generic RAM (our LCD uses only 2x16=32 characters).

The graphical shape of the characters is stored in the CGROM (Character Generator Read Only Memory). This memory contains the matrices of points for every character (5x8 or 5x10 – depending on the device).

Upper 4 Bits Lower 4 Bits	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
xxxx0000	CG RAM (1)		0	1	2	3	4	5	6	7	8	9	A	B	C	D
xxxx0001	(2)	!	1	A	Q	a	q				.	7	4	ä	q	
xxxx0010	(3)	"	2	B	R	b	r				"	イ	ツ	×	β	θ
xxxx0011	(4)	#	3	C	S	c	s				」	ウ	テ	モ	ε	∞
xxxx0100	(5)	\$	4	D	T	d	t				、	イ	ト	ハ	μ	Ω
xxxx0101	(6)	%	5	E	U	e	u				・	オ	ナ	1	ε	Ü
xxxx0110	(7)	&	6	F	V	f	v				ヲ	カ	ニ	ヨ	ρ	Σ
xxxx0111	(8)	'	7	G	W	g	w				ア	キ	ヌ	ラ	g	π
xxxx1000	(1)	<	8	H	X	h	x				イ	ク	ネ	リ	ジ	×
xxxx1001	(2)	>	9	I	Y	i	y				ウ	ケ	ル	ル	リ	ユ
xxxx1010	(3)	*	:	J	Z	j	z				エ	コ	ハ	レ	ジ	キ
xxxx1011	(4)	+	;	K	[k	[オ	サ	ヒ	ロ	*	斤
xxxx1100	(5)	,	<	L	¥	l	l				ハ	シ	フ	ワ	¢	円
xxxx1101	(6)	-	=	M]	m]				ユ	ズ	ハ	ン	モ	÷
xxxx1110	(7)	.	>	N	^	n	^				ヨ	セ	ホ	°	ん	
xxxx1111	(8)	/	?	O	_	o	_				ツ	ソ	マ	°	ö	■

The character codes for 5 x 8 points chars

The shape of the characters with codes between 0x00 and 0x07 can be defined by the user. For each character an 8 byte matrix will be specified (one byte for each row). The least significant 5 bits from each row will specify which pixels will be turned on or off (see the example below).

Custom Pattern	Decimal	Hex
	Row 1: 4	0x04
	Row 2: 14	0x0E
	Row 3: 14	0x0E
	Row 4: 14	0x0E
	Row 5: 31	0x1F
	Row 6: 0	0x00
	Row 7: 4	0x04

Example of a user defined character

The LCD displays can communicate with the microcontroller on 8 or 4 bits. Usually the 8 bit connection is preferred (to spare the number of pins). The LCD shield used in the lab is configured (by default) with a 4 bit data interface (and uses only 7 pins from the Arduino board).

Example: display character strings and numerical values on the LCD

In this first example a string of characters will be displayed on the first line of the LCD while on the second line a number representing the elapsed time (seconds) from the program start will be displayed.

```
//includes the LCD library
#include <LiquidCrystal.h>
/* LCD RS - pin 7
 * LCD Enable - pin 6
 * LCD D4 - pin 5
 * LCD D5 - pin 4
 * LCD D6 - pin 3
 * LCD D7 - pin 2
```

The 7-th pin of the LCD is connected to the display brightness control potentiometer! */

```
// Init the LCD with the stated pin numbers
```

```
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
unsigned long time;
```

```
void setup()
```

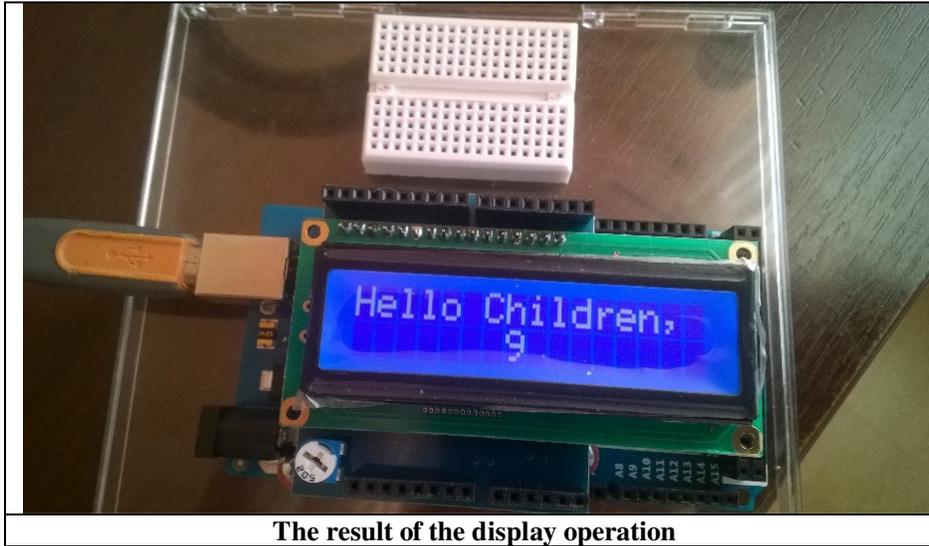
```
{
  // Sets the no. of rows and columns of the LCD
  lcd.begin(16, 2);
}
```

```
void loop()
```

```
{
  // Read the number of elapsed seconds from the program start
  time = millis() / 1000;
  // Set the cursor on col 0, row 0 (first row)
  lcd.setCursor(0, 0);
  // Write a string of characters
  lcd.print("Hello Children");
  // Move the cursor in the middle of the second row (row 1)
  lcd.setCursor(7, 1);
  // Display the elapsed time
  lcd.print(time);
}
```

}

In the figure bellow the result is shown:



The result of the display operation

Example: generation of user defined characters

This example shows the user defined character generation and usage procedure

```
#include <LiquidCrystal.h>
```

```
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
```

```
// Character matrix for the first character: every line is a row of pixels of the character
```

```
byte happy[8] = {
```

```
  B00000,
```

```
  B11011,
```

```
  B11011,
```

```
  B00000,
```

```
  B00000,
```

```
  B10001,
```

```
  B01110,
```

```
};
```

```
// Matrix for the second character
```

```
byte sad[8] = {
```

```
  B00000,
```

```
  B11011,
```

```
  B11011,
```

```
  B00000,
```

```
  B00000,
```

```
  B01110,
```

```
  B10001,
```

```
};
```

```
void setup() {
```

```
  lcd.begin(16, 2);
```

```
  // The 2 character are stored in the CGROM, user defined area, pos. 0 and 1
```

```
lcd.createChar(0, happy);
lcd.createChar(1, sad);

// Display the first line: a string followed by the 1-stuser defined char
lcd.setCursor(0, 0);
lcd.print("Happy ");
lcd.write(byte(0)); // See the difference between print and write
/* When you are referring the "0" user defined char you must write a cast to the "byte" type,
otherwise the compiler throws an error. Exception is the case when you are referring a
variable:
    byte zero = 0;
    lcd.write(zero);
*/

// Display the second line
lcd.setCursor(0, 1);
lcd.print("Sad ");
lcd.write(1); // when you are referring other characters then "0" it is not necessary to cast
}

void loop()
{ }
```

2. Working with the interrupt system

Interrupts are events that require immediate attention from the microcontroller. As a response to such an event, the microcontroller stops its normal instruction execution sequence and “serves” the interrupt (it executes an Interrupt Service Routine (ISR) – a routine attached to that specific interrupt event). In order to respond to interrupt events, the microcontroller should have the Global Interrupt Enable bit and the Interrupt specific bit enabled. The following issues are essential in order to have proper interrupt system usage:

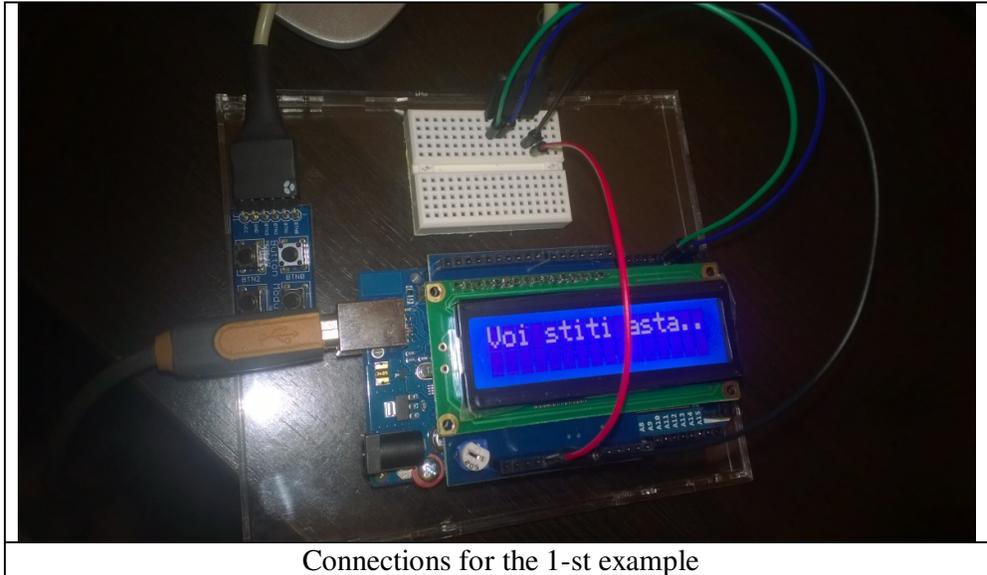
- The interrupt should be activated on its specific bit
- The Global Interrupt Enable bit (bit I in the Status Register – SREG) should be set
- The stack should be initialized (Arduino IDE does it by default)
- Every ISR ends with an RETI instruction (which resumes the normal program execution flow) – added by default by the Arduino IDE compiler

Interrupts can be of 2 types: internal and external. The internal ones are associated with events related to the microcontroller peripherals (ex. Timer/Counter, Analog Comparator etc.). External interrupts are triggered on external pins (ex. Buttons connected to these pins).

Each AVR MCU has an interrupt list which includes the type of the event that triggers the interrupt. When the interrupts are enabled and such an event occurs, the microprocessor will jump to a specific address in the program memory (referred by the entry in the Interrupt Vector Table (IVT) corresponding to the interrupt number). By writing an ISR (Interrupt Service Routine) and attaching its address in the IVT (Interrupt Vector Table) the system will be determined to execute a specific action as a response to a particular interrupt event.

In the first example external interrupts triggered by buttons will be used and corresponding messages will be displayed on the LCD. To run this example you need the ATmega 2560 board with the LCD shield and a PModBtn peripheral module. The ATmega 2560 has 6 external interrupt pins (the pin correspondence is available here: <http://arduino.cc/en/Hacking/PinMapping2560>).

The buttons will be connected to the pins corresponding to the interrupts INT0 and INT1 (digital pins 20 and 21). In the figure below are shown the necessary connections are shown followed by the corresponding code:



```
// Include the header for the avr interrupt system
#include "avr/interrupt.h"
// Include the LCD library
#include <LiquidCrystal.h>

// Init the LCD
LiquidCrystal lcd(7, 6, 5, 4, 3, 2);
volatile int buttonVariable; //public variable that can be modified by the ISR

void setup(void)
{
  buttonVariable = 0; // Init the variable shared between the ISR and the main program

  // Set the LCD row and col number
  lcd.begin(16, 2);
  lcd.print("Incepe experimental");
  delay(1000); // perform a 1 sec delay to display this message on the LCD

  // Set pin 21 as input (the pin corresponding to INT0)
  pinMode(21, INPUT);
  // Set pin 20 as input (the pin corresponding to INT1)
  pinMode(20, INPUT);

  pinMode(13, OUTPUT); // Set pin 13 as output
  digitalWrite(13, HIGH); // Lit up the onboard LED

  delay(1000);

  EIMSK |= (1 << INT0); // Activate INT0
  EIMSK |= (1 << INT1); // Activate INT1
```

```
EICRA |= (1 << ISC01); // Specify INT0 triggering behavior: falling edge of the
EICRA |= (1 << ISC11); // Same behavior for INT1
sei(); // Global interrupt system activation

digitalWrite(13, LOW); // Turn off the onboard LED

lcd.clear(); // Erase the LCD screen
}

void loop()
{
// If an interrupt was triggered/executed the LCD has to be erased and the main message displayed
if(buttonVariable == 1)
{
  lcd.clear(); // Erase the LCD
  buttonVariable = 0; // Global variable re-initialized
}

delay(1000);
lcd.setCursor(0,0); // Set the LCD cursor
lcd.print("Liniste..."); // Display a message
}

// ISR for INT0 ("INT0_vect" is a predefined name (address) for INT0 ISR
ISR(INT0_vect)
{
  digitalWrite(13, !digitalRead(13)); // Change the status of pin 13
  lcd.setCursor(0,0); // Move the LCD cursor in the top-left corner
  lcd.print("Intreupem");// Display message
  lcd.setCursor(0,1);
  lcd.print("ptr stirea zilei");
  buttonVariable = 1;
}

// ISR for INT1
ISR(INT1_vect)
{
  digitalWrite(13, !digitalRead(13));
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Stirea Doi");
  buttonVariable = 1;
}
```

Delay() and millis() function does not work during an ISR (they are based on interrupts but the interrupt system is deactivated during the execution of an ISR). In generally, during the execution of an ISR the system does not respond to any other interrupt request. Therefore it is recommended that the execution time of an ISR to be as short as possible.

If a variable is modified inside an ISR and if you want to access its modified value globally, declare the variable as volatile. Hence the compiler knows that the **variable** can be modified any time, discarding any code optimization and placing the variable in the RAM.

Arduino allows the usage of the interrupt system without microcontroller specific knowledge by using generic functions. The first one is **attachInterrupt()**. It attaches an ISR to an external interrupt, replacing any previous attachment:

attachInterrupt(interrupt, ISR, mode)

The first parameter is the interrupt number but does not match neither the external interrupt number of the Atmega2560 nor the digital pin number of the board. It is a simple index. The table below shows the difference between these numbers:

attachInterrupt	Name	Pin on chip (TQFP)	Pin on board
0	INT4	6	D2
1	INT5	7	D3
2	INT0	43	D21
3	INT1	44	D20
4	INT2	45	D19
5	INT3	46	D18

Therefore the use of the **digitalPinToInterrupt(pin)** function is recommended – it will return the interrupt number attached to the digital pin (if exists):

The second parameter is the name (address) of the ISR.

The third parameter is the trigger response behavior: **RISING** edge, **FALLING** edge, **LOW** level or **HIGH** level or level toggle (**CHANGE**).

Function **noInterrupts()** disables the interrupts. They can be reactivated using: **interrupts()**.

The **detachInterrupt(interrupt)** function disables the interrupt with the number specified in the parameter.

The previous example was implemented using the AVR configuration registers, being dependent on the microcontroller specifications. In the following example the same functionality can be achieved using Arduino specific functions:

```
#include <LiquidCrystal.h>
```

```
LiquidCrystal lcd(7,6,5,4,3,2);
volatile int buttonVariable;
```

```
void setup()
{
  buttonVariable = 0;
  lcd.begin(16,2);
  lcd.print("A inceput");
  lcd.setCursor(0,1);
  lcd.print("din nou");
  delay(1000);
```

```
// The 2 interrupt pins 21 and 20 declared as inputs with pull-up resistors activated
```

```
pinMode(20 ,INPUT);
pinMode(21 ,INPUT);
digitalWrite(20, HIGH);
digitalWrite(21, HIGH);

// Atach ISRs to the interrupts corresponding to pins 21 and 20 (INT0 and INT1)
attachInterrupt(digitalPinToInterrupt(20), functieUnu, RISING);
attachInterrupt(digitalPinToInterrupt(21), functieDoi, CHANGE);
}

void loop()
{
// Insert here task for normal program flow ....
  lcd.print("Programul principal");
  delay(1000);
}

// First ISR function
void functieUnu()
{
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Functia Unu");
}

// 2-nd ISR function
void functieDoi()
{
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("Functia Doi");
}
```

Individual Work

1. Run the presented examples.
2. Using the character generator, create an animation that indicates the time elapsed (a clock with a rotating arm, an hourglass etc.).
3. Create a timer using the interrupts. One button starts/stops the timer while the other resets the timer.
4. Combine 2 + 3: the animation runs as long the timer is running and is topped when the timer is stopped.
5. Bonus: use an Arduino board for generating signals of different frequencies and duty cycles (amount of time the signal is 1), by changing the state of a pin (using digitalWrite) at regular time intervals, controlled by delay() or millis(). Use the external interrupts of another board to measure the parameters of this signal, and display the parameters on the LCD.

References

<https://www.arduino.cc/en/Reference/LiquidCrystal>
<https://www.arduino.cc/en/Reference/AttachInterrupt>