Design with Microprocessors

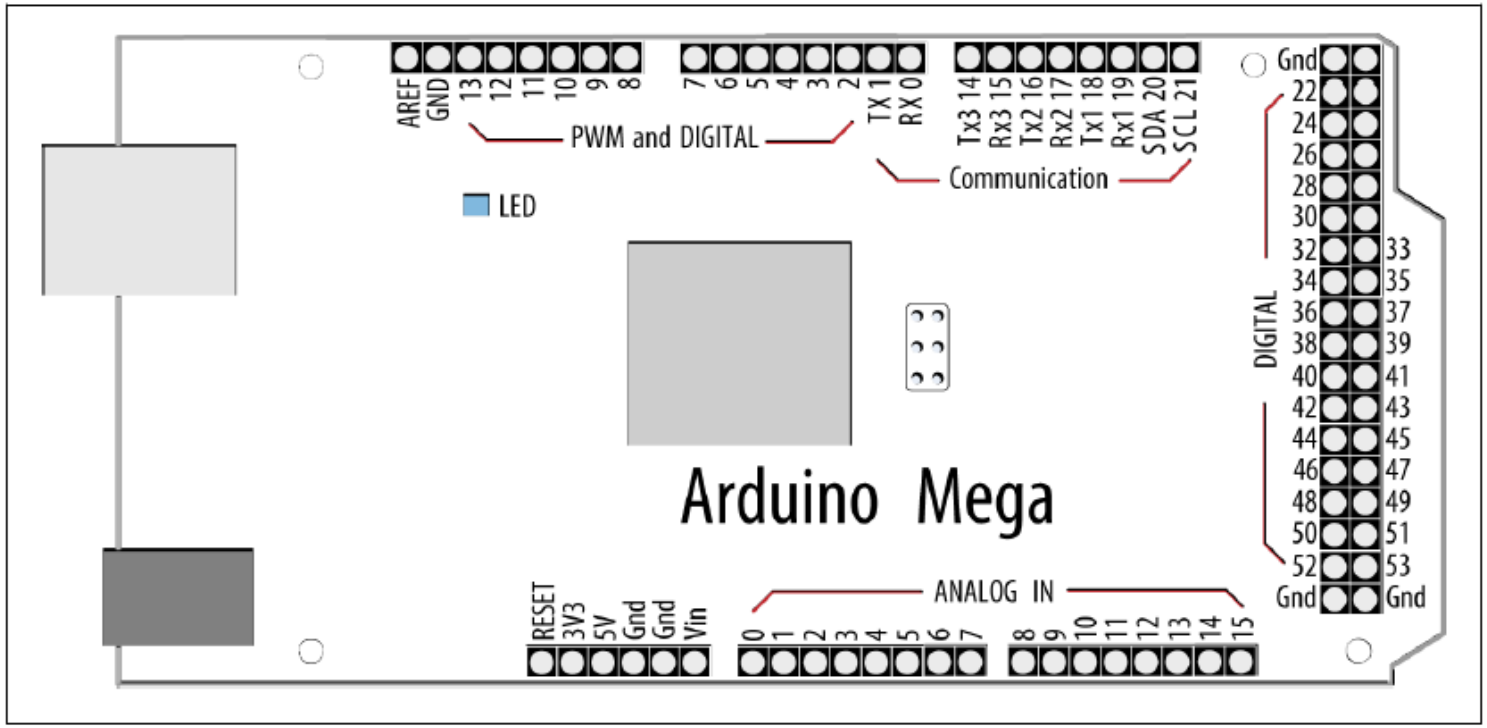
Lecture 3

Year 3 CS
Academic year 2017/2018
1st Semester

Lecturer: Radu Dănescu

Input / Output with Arduino

- Digital input/output pins, connected to the AVR microcontroller's ports
- The IDE will handle the correspondence between digital pins and port bits
- The programming logic is pin oriented
- Some digital pins have special functions (UART or I²C serial communication, wave generation, or analog input)
- The pins RX0 and TX0 must be avoided! They are reserved for serial communication via USB, which includes programming the board
- Usually there is a LED on the board, connected to pin 13



Input / Output with Arduino

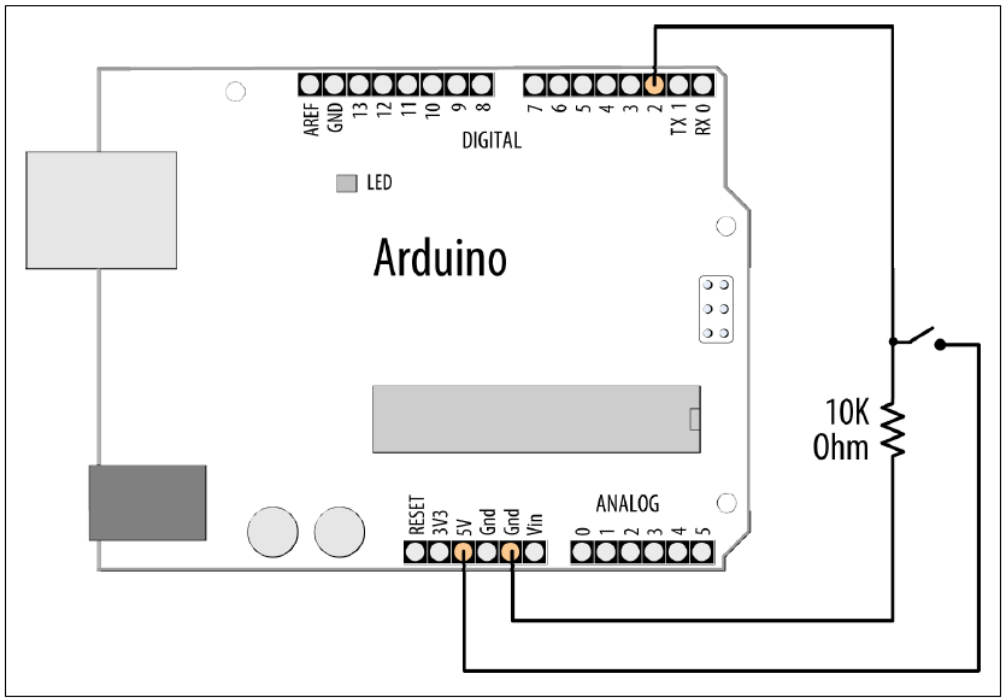
- The correspondence between the microcontroller pins of ATMega2560 and the digital pins of the Arduino Mega board http://arduino.cc/en/Hacking/PinMapping2560
- Selection:

43	PD0 (SCL/INT0)	Digital pin 21 (SCL)
44	PD1 (SDA/INT1)	Digital pin 20 (SDA)
45	PD2 (RXD1/INT2)	Digital pin 19 (RX1)
46	PD3 (TXD1/INT3)	Digital pin 18 (TX1)
47	PD4 (ICP1)	
48	PD5 (XCK1)	
49	PD6 (T1)	
50	PD7 (T0)	Digital pin 38

71	PA7 (AD7)	Digital pin 29
72	PA6 (AD6)	Digital pin 28
73	PA5 (AD5)	Digital pin 27
74	PA4 (AD4)	Digital pin 26
75	PA3 (AD3)	Digital pin 25
76	PA2 (AD2)	Digital pin 24
77	PA1 (AD1)	Digital pin 23
78	PA0 (AD0)	Digital pin 22

Input / Output with Arduino

- Basic signal source: a button connected to a digital input pin
- One can use a pull down resistor, so that when the button is released a logic ‘0’ is generated
- Use the on-board LED for output



Input / Output with Arduino

- Example code:

```
const int ledPin = 13;                  // Constants for pin numbers
const int inputPin = 2;                 // Numbers can be used directly, but this adds flexibility

void setup() {                          // Set up the pin directions
  pinMode(ledPin, OUTPUT);              // Declare LED pin as output
  pinMode(inputPin, INPUT);             // Declare button pin as input
}

void loop(){                            // Read button state
  int val = digitalRead(inputPin);      // If pressed, write '1' on the LED pin
  if (val == HIGH)
  {
    digitalWrite(ledPin, HIGH);
  }
  else                                  // otherwise write '0'
  {
    digitalWrite(ledPin, LOW);          // Obviously, you can write the button state directly to the LED:

  }
}
```

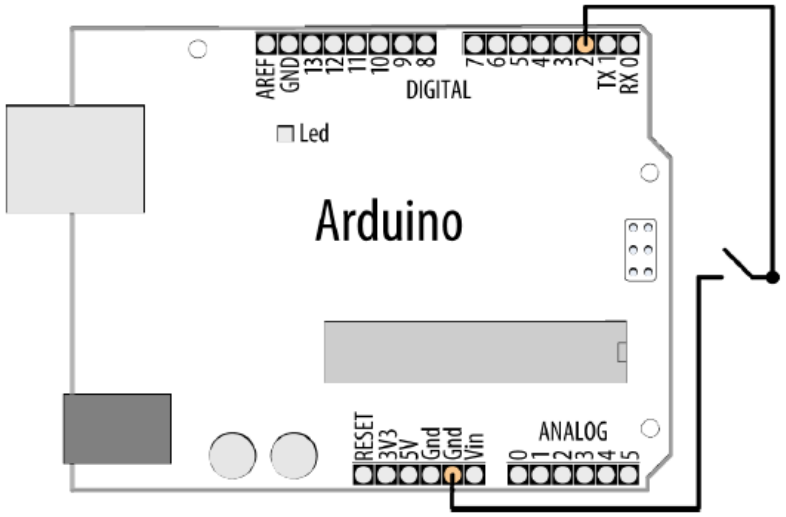
```
void loop()
{
    digitalWrite(ledPin, digitalRead(inputPin));

}
```

Input / Output with Arduino

- Using a button without external resistors
- You can use the internal 'Pull Up' resistors attached to each pin

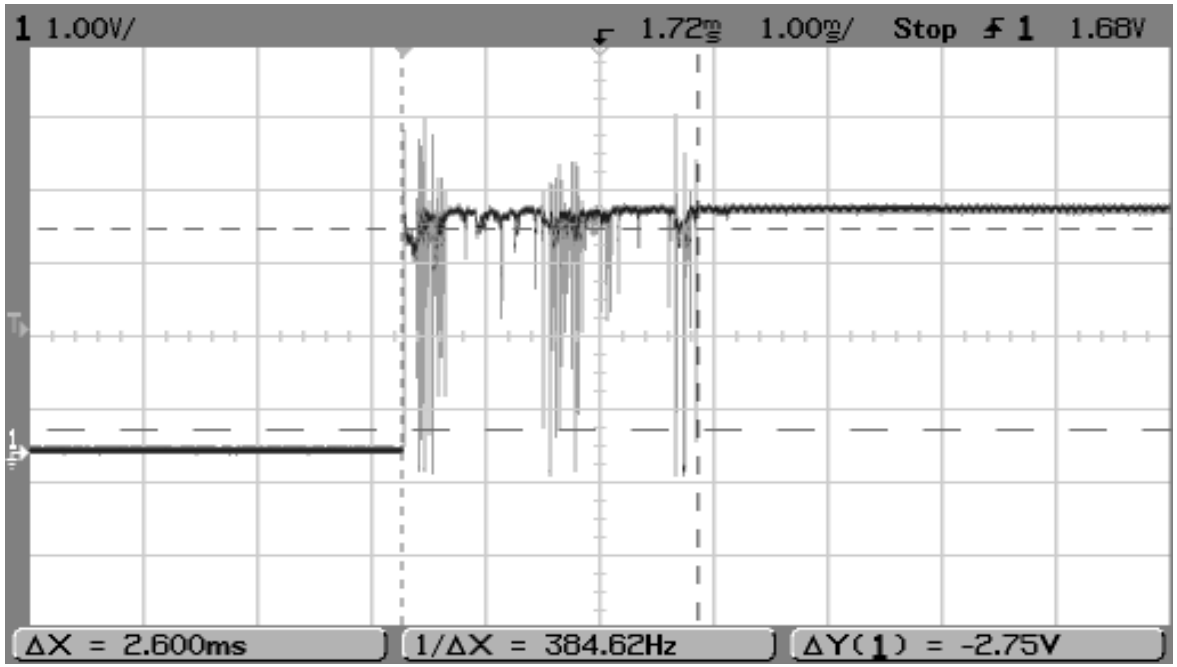
```
const int ledPin = 13;
const int inputPin = 2;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(inputPin, INPUT);
  digitalWrite(inputPin,HIGH);
}
void loop(){
  int val = digitalRead(inputPin);
  if (val == HIGH)
  {
    digitalWrite(ledPin, HIGH);
  }
  else
  {
    digitalWrite(ledPin, LOW);
  }
}
```

```
// Same constants, same pins

// setting the pin directions

// Activate the pull up resistor by writing a high value
// on the input pin!

// Same code as before
```



Input / Output with Arduino

- **Reading unstable input data**
  - A mechanical contact can oscillate between ‘closed’ and ‘open’ many times before setting to a stable position.
  - A microcontroller may be fast enough to detect some of these oscillations, and interpret them as multiple button presses.
  - Some button devices, such as Pmod BTN, have circuits to filter out these oscillations.
  - If such circuits do not exist, the problem must be solved by software.



Input / Output with Arduino

- **Reading unstable input data**
  - The principle of software based filtering: check the state of the pin multiple times, until it is stable.
  - Effect: ignoring the unstable period, validating the input only when stable.
  - Example source code:

```
const int inputPin = 2;
const int ledPin = 13;
const int debounceDelay = 10;    // The time interval (ms) in which the signal must be stable

boolean debounce(int pin)   // This function returns the stable state of the pin
{
  boolean state;            // Current state, previous state
  boolean previousState;

  previousState = digitalRead(pin);    // The first (initial) state
  for(int counter=0; counter < debounceDelay; counter++)  // For the whole time interval
  {
      delay(1);                        // Wait 1 ms
      state = digitalRead(pin);        // Read current state
      if( state != previousState)      // If the states are different, restart counting
      {
         counter = 0;   // Set counter back to zero
         previousState = state;  // Set current state as initial state for a new cycle
      }
  }
  // // If we have reached this point, the signal is stable
  return state;   // Return the stable, current state
}
```

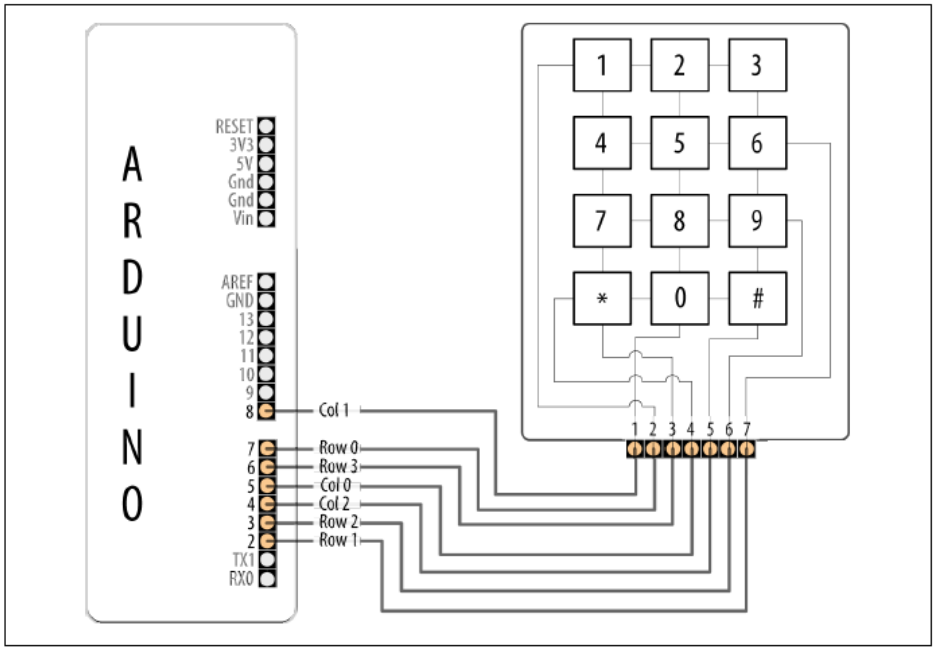
Input / Output with Arduino

- **Reading unstable input data**
  - Example source code (continued):

```
void setup()
{
  pinMode(inputPin, INPUT);
  pinMode(ledPin, OUTPUT);
}

void loop()
{
  if (debounce(inputPin))          // Use the debounce() function instead of digitalRead()
  {
    digitalWrite(ledPin, HIGH);
  }
}
```

Input / Output with Arduino

- **I/O with multiple pins. Using a KeyPad**
  - Pressing a key makes a contact between a row and a column
  - The default state of the rows is ‘1’, by using pull up resistors
  - If the pressed key’s column is zero, the key’s row becomes ‘0’. If the column is ‘1’, the row does not change its state.
  - Working principle: activating one column at the time (setting them one by one to ‘0’), and reading the state of the rows
  - The columns must be connected to output pins, and the rows to input pins



Arduino pin	Keypad connector	Keypad row/column
2	7	Row 1
3	6	Row 2
4	5	Column 2
5	4	Column 0
6	3	Row 3
7	2	Row 0
8	1	Column 1

Input / Output with Arduino

- **I/O with multiple pins. Using a KeyPad**
  - Example code:

```
const int numRows = 4;          // Number of rows
const int numCols = 3;          // Number of columns
const int debounceTime = 20;    // Number of milliseconds of delay

// Define the pins attached to columns and rows, in their logical order
const int rowPins[numRows] = { 7, 2, 3, 6 }; // row pins
const int colPins[numCols] = { 5, 8, 4 };    // column pins

// LUT for identifying the key at the intersection of a row with a column
const char keymap[numRows][numCols] = {
  { '1', '2', '3'  } ,
  { '4', '5', '6'  } ,
  { '7', '8', '9'  } ,
  { '*', '0', '#'  }
};

void setup()   // system setup
{
  Serial.begin(9600);  // Setting up the USB serial interface, for communication with the PC
  for (int row = 0; row < numRows; row++)
  {
    pinMode(rowPins[row],INPUT);          // Set row pins as input
    digitalWrite(rowPins[row],HIGH);      // Activate the pull up resistors
  }
  for (int column = 0; column < numCols; column++)
  {
    pinMode(colPins[column],OUTPUT);      // Set column pins as output

    digitalWrite(colPins[column],HIGH);   // Set all columns to '1' - inactive
  }
}
```

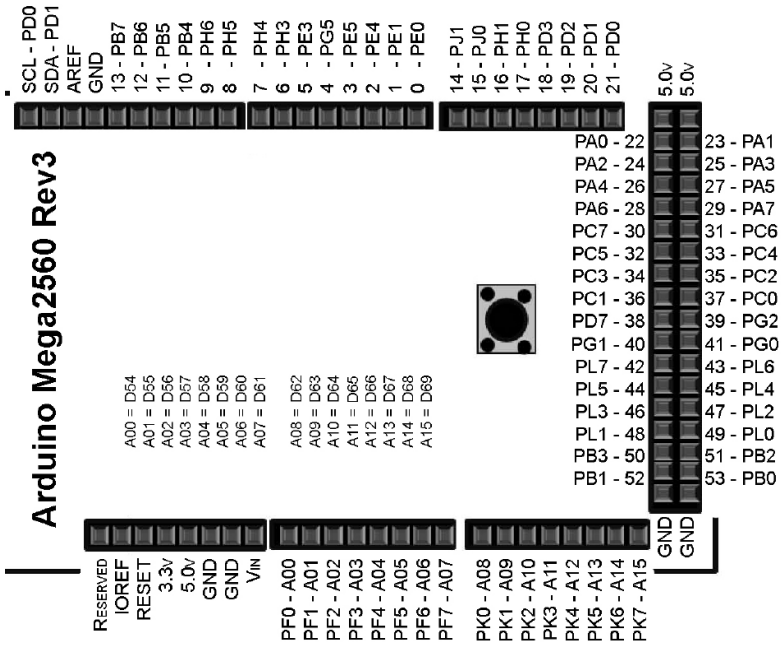
Input / Output with Arduino

- **I/O with multiple pins. Using a KeyPad**
  - Example code (continuation):

```
void loop()
{
  char key = getKey();      // Call the key reading function (below)
  if( key != 0) {           // If the function returns 0, no key is pressed
                            // If the result is not zero, a key is pressed, and the function returns its associated character
    Serial.print("Got key ")  // Use the serial interface to display that the key is pressed
    Serial.println(key);      // and its associated character
  }
}

 // The KeyPad scanning function, returns 0 if no key is pressed, or the key character otherwise.
char getKey()
{
  char key = 0;                                   // default return code is zero, no key pressed

  for(int column = 0; column < numCols; column++)   // scanning the columns
  {
    digitalWrite(colPins[column],LOW);            // activate current column
    for(int row = 0; row < numRows; row++)        // check the rows one by one

    {
      if(digitalRead(rowPins[row]) == LOW)        // if row is '0', a key is pressed on this row
      {
        delay(debounceTime);                      // delay for input filtering
        while(digitalRead(rowPins[row]) == LOW)
            ;                                     // wait for key releaseasteptare eliberare tasta


        key = keymap[row][column];                // the column and the row of the key are known
                                                  // Use the LUT to get the ASCII code of the key's character
      }
    }
    digitalWrite(colPins[column],HIGH);           // de-activate the column
  }
  return key;   // Return key character code, or 0
}
```

Input / Output with Arduino

- **I/O using the microcontroller's ports**
- **Disadvantages**
  - Hardware-dependent approach, the code may not work on other boards
  - You must know the correspondence between the pin and the port bit
  - Some ports are reserved, and changing their state is not recommended
- **Advantages**
  - High speed. Reading and writing a port about 10x faster than using digitalWrite() and digitalRead()
  - Multiple pins can be read or written simultaneously (digitalRead and digitalWrite only work with one pin at the time)



Input / Output with Arduino

- **Example:** connect 8 LEDs to pins 22…29 of Arduino Mega (connected to PortA). We want to light alternatively the odd and even LEDs, with 1 second delay between changes
- **Source code, classical Arduino approach**:

```
const  int PortAPins[8]={22, 23, 24, 25, 26, 27, 28, 29}

void setup()
{
    for (int b=0; b<8; b++)
            pinMode(PortAPins[b], OUTPUT);          // Set all pins to output
}

void loop()
{
    for (int b=0; b<8; b+=2)                        // b=0, 2, 4, 6
            {
            digitalWrite(PortAPins[b], HIGH);       // write '1' on the even pins
            digitalWrite(PortAPins[b+1], LOW);      // write '0' on the odd pins
            }
    delay(1000);                                    // 1 second delay (1000 ms)
    for (int b=0; b<8; b+=2)
            {
            digitalWrite(PortAPins[b], LOW);        // write '0' on the even pins
            digitalWrite(PortAPins[b+1], HIGH);     // write '1' on the odd pins
            }
    delay(1000);                                    // 1 second delay (1000 ms)
}
```

Input / Output with Arduino

- **Example:** connect 8 LEDs to pins 22…29 of Arduino Mega (connected to PortA). We want to light alternatively the odd and even LEDs, with 1 second delay between changes
- **Source code, using Port A of ATMega2560**:

```
void setup()
{
    DDRA = B11111111;                    // all pins of Port A are configured as output
}

void loop()
{
     PORTA = B01010101;                  // 1 on the even pins, 0 on the odd pins
     delay(1000);                        // 1 second delay (1000 ms)
     PORTA = B10101010;                  // 0 on the even pins, 1 on the odd pins
     delay(1000);                        // 1 second delay (1000 ms)
}
```

Handling the external interrupts

- Detecting events on the pins, without permanently checking their state by digitalRead
- Depending on the Arduino board, the number of external interrupts is variable:

Board	int.0	int.1	int.2	int.3	int.4	int.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	

- For handling an interrupt, an Interrupt Service Routine (ISR) must be attached. This is done by using the function **attachInterrupt()**, with the syntax:

  attachInterrupt(interrupt, ISR, mode)

  interrupt – number of the external interrupt (0, 1, 2, …)
  ISR – name of the Interrupt Service Routine (a function of your program)
  mode – triggering mode:
  LOW – trigger on level ‘0’
  CHANGE – trigger on pin level change
  RISING – trigger on rising edge of the input signal
  FALLING – trigger on falling edge of the input signal

Handling the external interrupts

- De-activating the interrupt handling process is done by calling the function **detachInterrupt()**, with the syntax:

  detachInterrupt(interrupt)
  interrupt - interrupt number

- If a temporary de-activation of all interrupts is desired, call the function **noInterrupts()**, without parameters. For re-activating the interrupts, call the function **interrupts()**.
- The interrupt system is implicitly active! Deactivation must be done for short periods of time only, otherwise the Arduino functions may be impaired.

Handling the external interrupts

- **Example:** measuring the width of pulses of a signal (for example, if the signal is from an IR remote receiver, the width of a signal signals whether the pulse is a ‘0’ or a ‘1’).

```
const int irReceiverPin = 2;          // Pin 2, connected to external interrupt 0
const int numberOfEntries = 64;       // Number of transitions that we'll analyze

volatile unsigned long microseconds; // Variable for keeping the number of microseconds since the program started
volatile byte index = 0;              // Position in the transition array
volatile unsigned long results[numberOfEntries];  // Interval time array – the result

void setup()
{
  pinMode(irReceiverPin, INPUT);          // Set the interrupt pin as input
  Serial.begin(9600);                     // USB Serial communication for result display
  attachInterrupt(0, analyze, CHANGE);    // Attach the ISR to interrupt 0, triggered when the signal changes levels
  results[0]=0;
}

void loop()
{
  if(index >= numberOfEntries)            // Check if the maximum number of transitions has been reached
  {
    Serial.println("Durations in Microseconds are:") ; // If yes, display the measured intervals
    for( byte i=0; i < numberOfEntries; i++)
    {
      Serial.println(results[i]);
    }
    index = 0; // After display, re-set the transitions counter and start again
  }
  delay(1000);
}
```

Handling the external interrupts

- **Example:** measuring the width of pulses of a signal (for example, if the signal is from an IR remote receiver, the width of a signal signals whether the pulse is a ‘0’ or a ‘1’).
- Continued:

```
void analyze()            // The interrupt service routine
{
  if(index < numberOfEntries  )     // If we have not reached the end of the array
  {
    if(index > 0)                   // But is also not the first detected transition
    {
      results[index] = micros() - microseconds;   // Measure the time passed since the last transition
    }
    index = index + 1;              // Increment the index in the transition array
  }
  microseconds = micros();          // Keep the current time, to be used as reference for the next transition
}
```

- The function **micros()** returns the number of microseconds since the program was started.
- For measuring bigger intervals, but with lower precision, you can use **millis()**, which returns the number of milliseconds since the program was started.

Handling the external interrupts

Attention:

- All global variables that can be modified inside an ISR function must be declared as "**volatile**". This way, the compiler will know they can change at any moment, and will not try to optimize them by assigning them to registers, or by assuming them constant. They will always be mapped as a location in the RAM.
- Only one ISR function can run at any given time. All other interrupts are, during this time, disabled.
- Since **delay()** and **millis()** rely on the interrupt system, they will not work properly during the execution of an ISR.
- For short delays inside an ISR, one can use the function **delayMicroseconds()**, which does not use interrupts.
- It is not recommended to use the Serial interface inside an ISR.

The interrupt number confusion

- The number specified as parameter to attachInterrupt() is not the same number as the external interrupt number of the AVR microcontroller:
- It is also not the digital pin number.

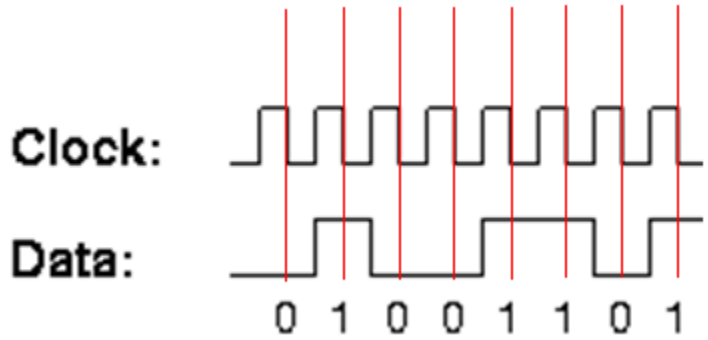
attachInterrupt	Name	Pin on chip	Pin on board
		(TQFP)	
0	INT4	6	D2
1	INT5	7	D3
2	INT0	43	D21
3	INT1	44	D20
4	INT2	45	D19
5	INT3	46	D18

- Solution: use of the **digitalPinToInterrupt(pin)** function
  - Example:

  attachInterrupt(digitalPinToInterrupt(21) , isr, FALLING) will attach to **Arduino interrupt 2**, which is the **AVR interrupt INT0**, connected to **digital pin 21**, the service routine isr, which will be triggered when the pin's logic level will fall from '1' to '0'.
- If a digital pin has no interrupt attached to it, the function **digitalPinToInterrupt** will return the value **-1** .

Exercises

- Display, using the serial interface, the number of the pins that can be used with external interrupts.
- Write a program capable of receiving serial synchronous data, as shown in the figure below:



- Change the program of the previous exercise, to use an additional signal which marks the beginning and the end of the byte: