Design with Microprocessors

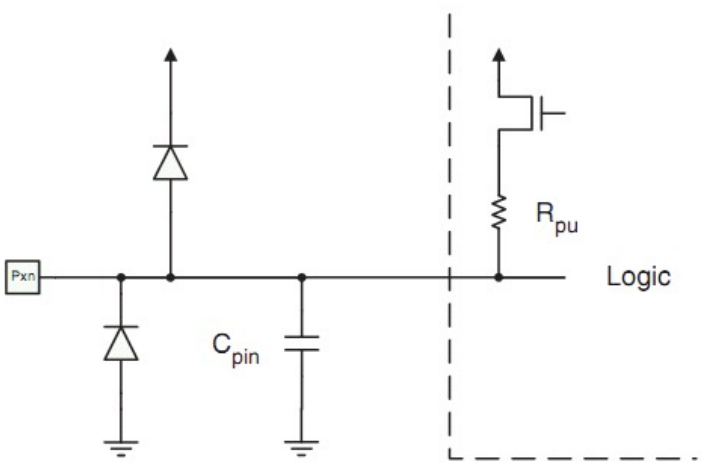
Lecture 2

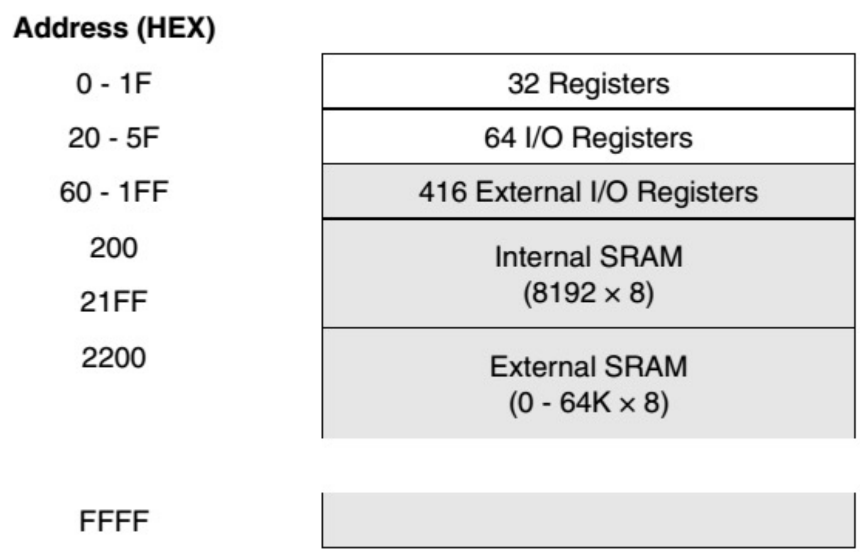
Year 3 CS
Academic year 2023/2024
1st Semester

Lecturer: Radu Dănescu

Input / Output

- Input/Output ports:
  - ATmega 328P (UNO): **PORT B, C, D**
  - ATmega 2560 (MEGA): **PORT A, B, C, D, E, F, G, H, J, K, L**
- PORTA... PORTE can be accessed by dedicated I/O instructions **in**, **out**
- PORTF … PORTL can be accessed only by **ld**, **st** – extended I/O space
- Each bit of each port can be configured as either **input** or **output**, using the register DDRx
- The port can be written using register PORTx
- The state of the input port's pins can be written from PINx
- Each pin has static electricity protection diodes
- Each pin has a "pull up" resistor that can be activated or deactivated by logic (program)





Input / Output ports

Three I/O memory address locations are allocated for each port x (A… L):

- Data Register – PORTx,
- Data Direction Register – DDRx
- Port Input Pins – PINx

Example (PORTA):

PORTA – Port A Data Register

Bit	7	6	5	4	3	2	1	0	
0x02 (0x22)	**PORTA7**	**PORTA6**	**PORTA5**	**PORTA4**	**PORTA3**	**PORTA2**	**PORTA1**	**PORTA0**	**PORTA**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRA – Port A Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x01 (0x21)	**DDA7**	**DDA6**	**DDA5**	**DDA4**	**DDA3**	**DDA2**	**DDA1**	**DDA0**	**DDRA**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

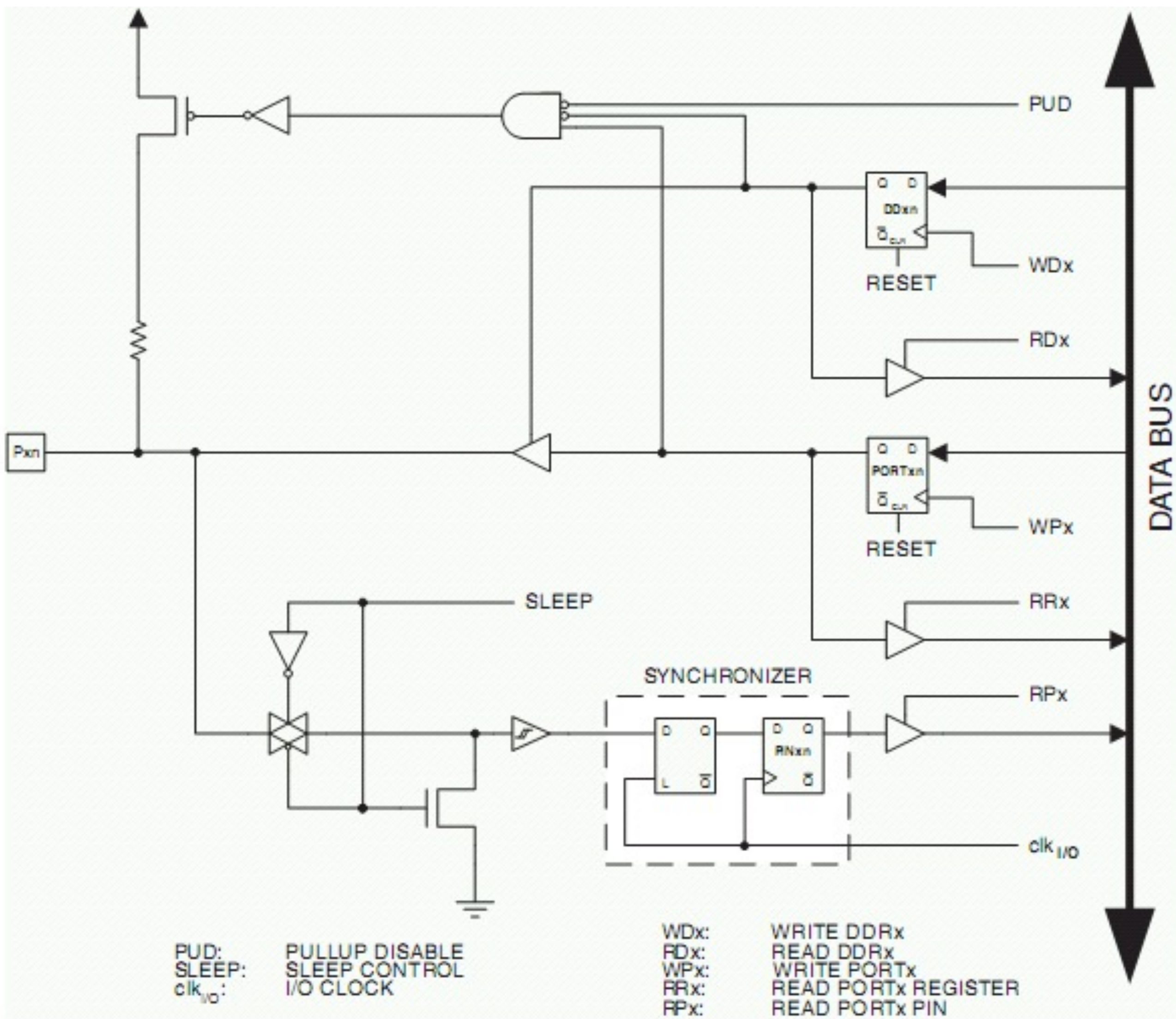
PINA – Port A Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x00 (0x20)	**PINA7**	**PINA6**	**PINA5**	**PINA4**	**PINA3**	**PINA2**	**PINA1**	**PINA0**	**PINA**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Notation: PORTxn = pin n of PORTx (ex: PORTB3 - bit no. 3 in Port B).

An I/O pin

- Internal structure of one I/O pin (one bit of an I/O port)



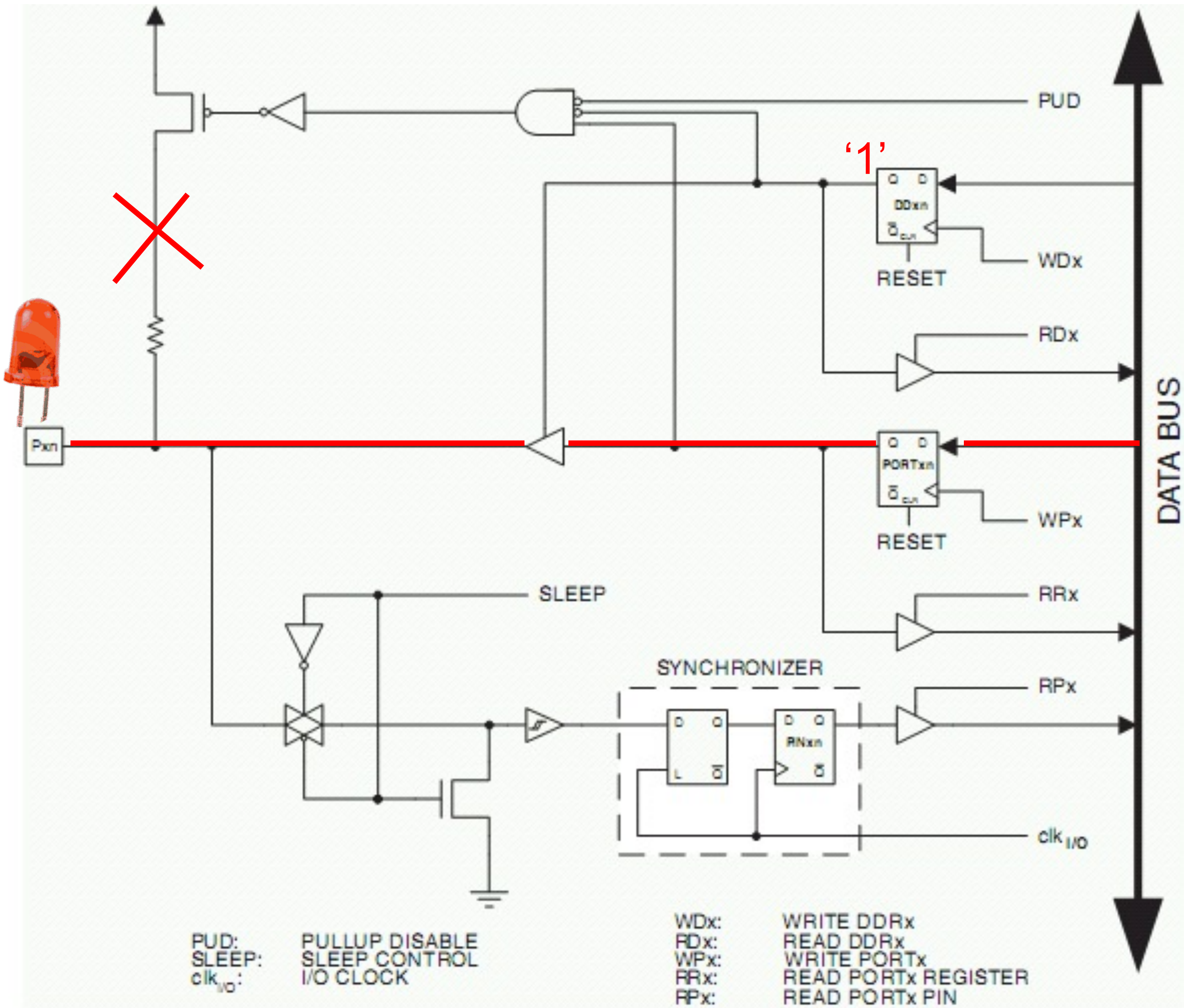
Direction control

Output data

Input data

An I/O pin

- Output configuration

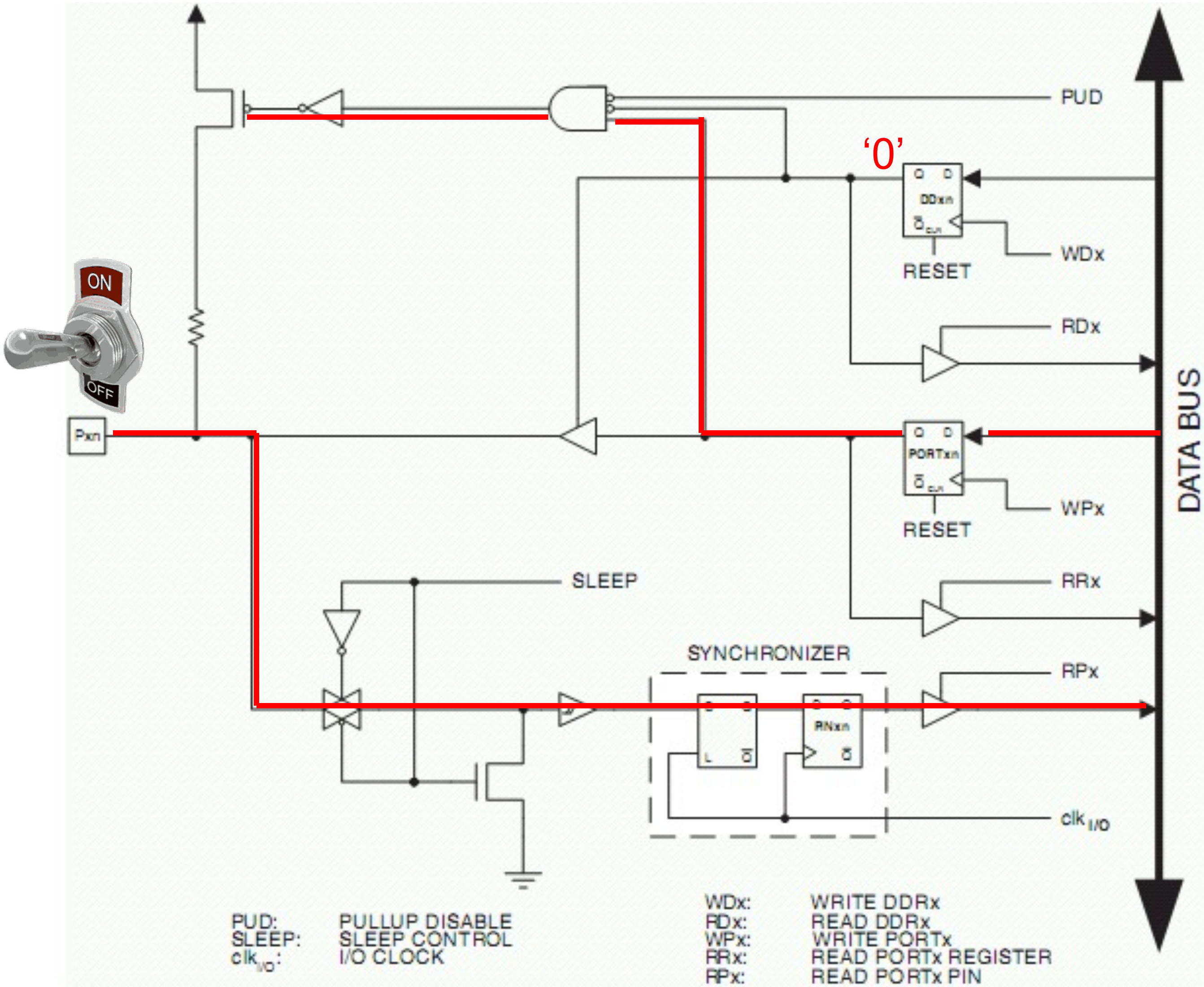


Direction bit = 1

Data written on PORTx are sent to the external pin

An I/O pin

- Input configuration



Direction bit = 0

A '1' written to PORTx activates the pull up resistor

The data on the external pin can be read as PINx

Input / Output

- Possible states of the I/O pins

DDxn	PORTxn	PUD (MCUCR)	I/O	Pull-up	Comment
0	0	X	Input	No	Tri-state (Hi-Z)
0	1	0	Input	Yes	Pxn will source current if ext. pulled low.
0	1	1	Input	No	Tri-state (Hi-Z)
1	0	X	Output	No	Output Low (Sink)
1	1	X	Output	No	Output High (Source)

- PUD – Pull Up Disable – GLOBAL
  - Setting bit 4 of MCUCR to '1' disables all PU resistors

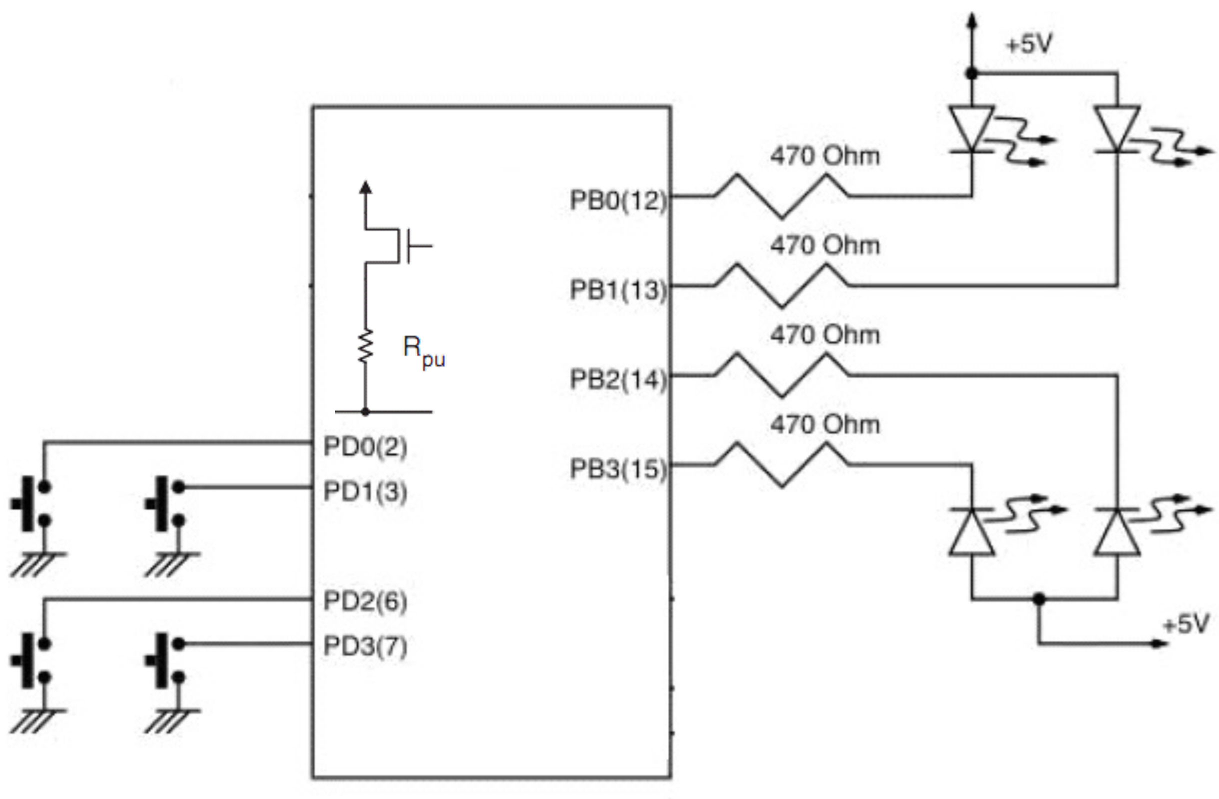
Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R/W	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

```
in r17, MCUCR
ori r17, 0b00010000
out MCUCR, r17
```

```
sbi MCUCR, 4
```

Input / Output

- Example – Buttons and LEDs
- The pull-up resistors hold the input pin to logic ‘1’ when the button is released
- When the button is pressed, the pin level becomes ‘0’, connected to GND
- A logic ‘0’ on the output pins (B) cause a voltage difference that lights the LEDs
- A logic ‘1’ on the output pins will turn the light off.



Input / Output

- Example – Buttons and LEDs – The program

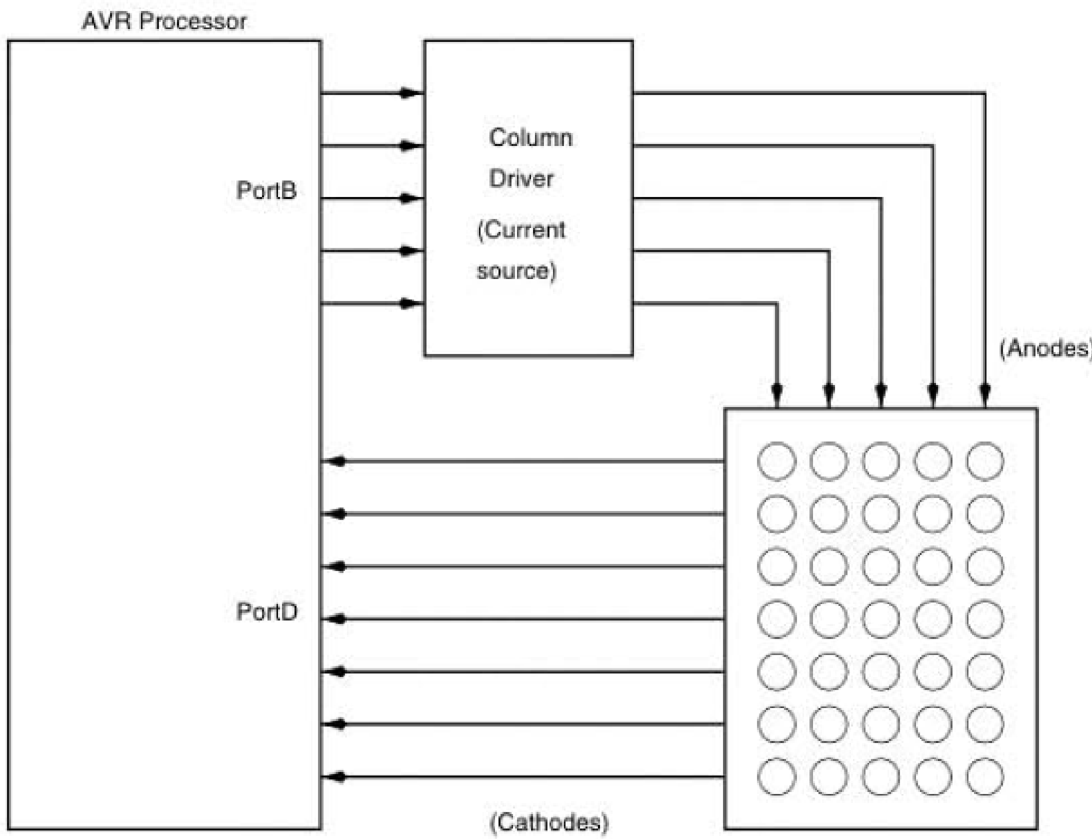
ldi r16, 0x00	
out DDRD, r16	Direction of port D - input
ldi r16, 0xFF	
out PORTD, r16	‘1’ in PORTD – pull up resistors activated
ldi r16, 0xFF	
out DDRB, r16	Direction of port B - output
loop:	
in r16, PIND	Read port D
out PORTB, r16	Write port B
rjmp loop	

- Attention!!

in r16, PIND	Reading the state of external pins, changed by external activity
in r16, PORTD	Reading the state of the PORTD register, written from inside, by program

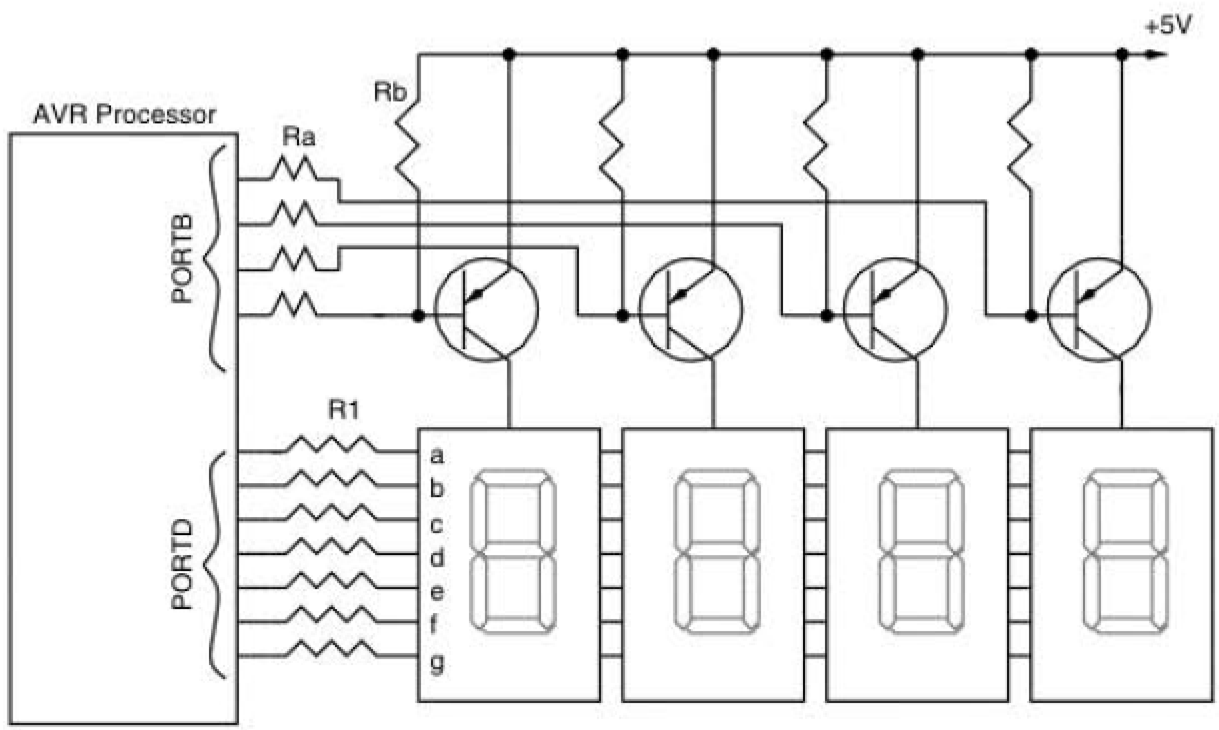
Input / Output

- Example – LED matrix
  - Both ports (D and B) are output
  - To light a LED, the anode must be ‘1’ and the cathode must be ‘0’
  - Only one line (or column) is active at once
  - For using the whole matrix – **sweeping**



Input / Output

- Example – 4x7 segments display
  - Each digit has 7 led-s, common anode configuration
  - A logic ‘1’ on the anode activates the digit – a single digit is active at one given time!
  - Selective values of ‘0’ on each cathode draws the digit pattern
  - Sweeping (refreshing) is needed to make all digits appear lit



Current limiting resistors for LEDs

- Each LED type has a typical forward voltage drop V_f
- The voltage difference between the digital output voltage V_{CC} and V_f is the voltage drop on the limiting resistor
- The current intensity through the output pin is:

$$I = \frac{V_R}{R} = \frac{V_{CC} - V_F}{R}$$

- The higher the intensity, the brighter the LED.
- *I* must be limited below 20 mA, to protect the microcontroller (and the LED).

Typical LED Characteristics			
Semiconductor Material	Wavelength	Colour	V_F @ 20mA
GaAs	850-940nm	Infra-Red	1.2v
GaAsP	630-660nm	Red	1.8v
GaAsP	605-620nm	Amber	2.0v
GaAsP:N	585-595nm	Yellow	2.2v
AlGaP	550-570nm	Green	3.5v
SiC	430-505nm	Blue	3.6v
GaInN	450nm	White	4.0v

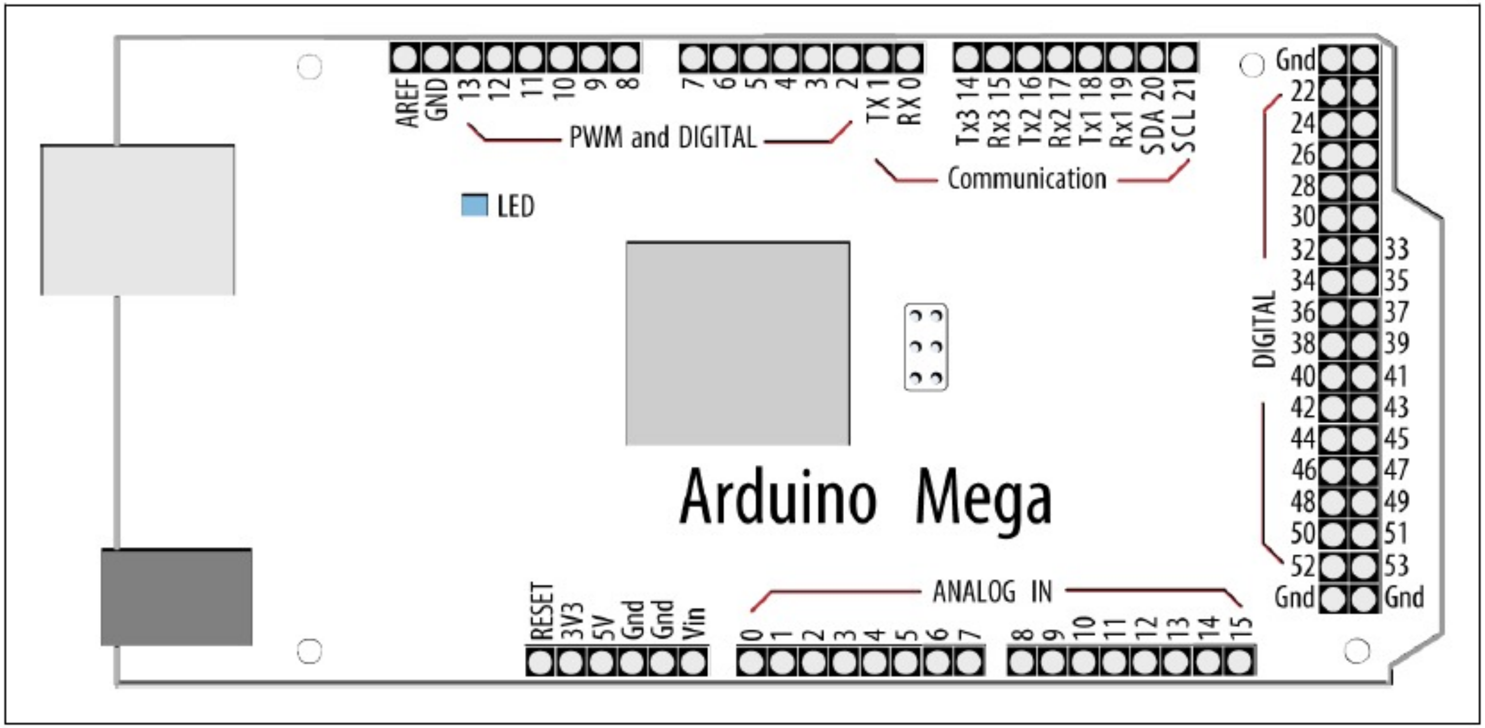
- Example: 1 Red LED
- V_F = 1.8 V , V_{CC} = 5 V
- **For I = 10 mA**
- R = (5 V – 1.8 V) / 0.01A = 320 Ohm

- **For I = 20mA**
- R = (5 V – 1.8 V) / 0.02A = 160 Ohm

http://www.electronics-tutorials.ws/diode/diode_8.html

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 I^2C 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

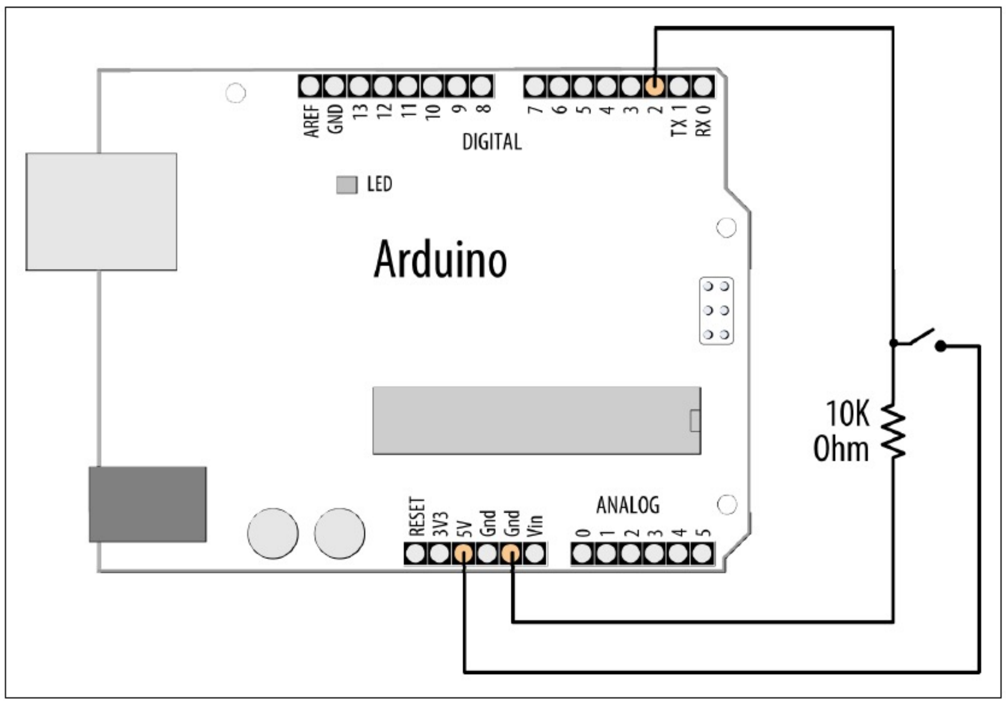
- The correspondence between the microcontroller pins of ATMega2560 and the digital pins of the Arduino Mega board http://arduino.cc/en/Hacking/PinMapping2560
- Selection:

43	PD0 (SCL/INT0)	Digital pin 21 (SCL)
44	PD1 (SDA/INT1)	Digital pin 20 (SDA)
45	PD2 (RXDI/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

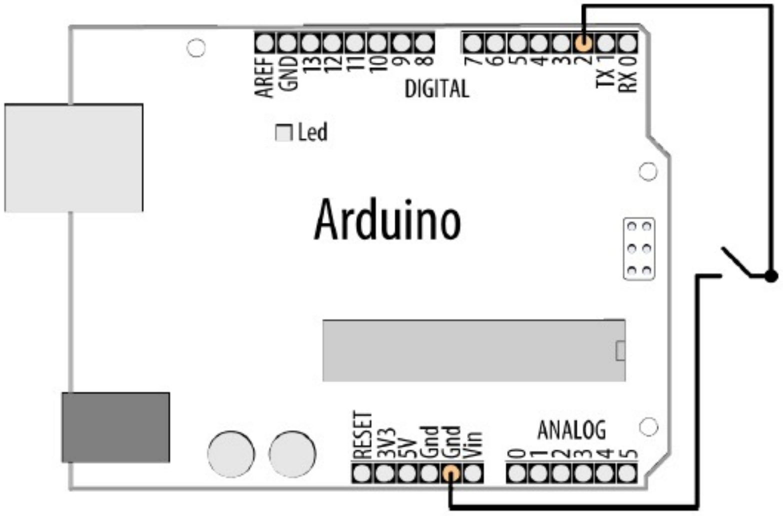
```
const int ledPin = 13;
const int inputPin = 2;

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

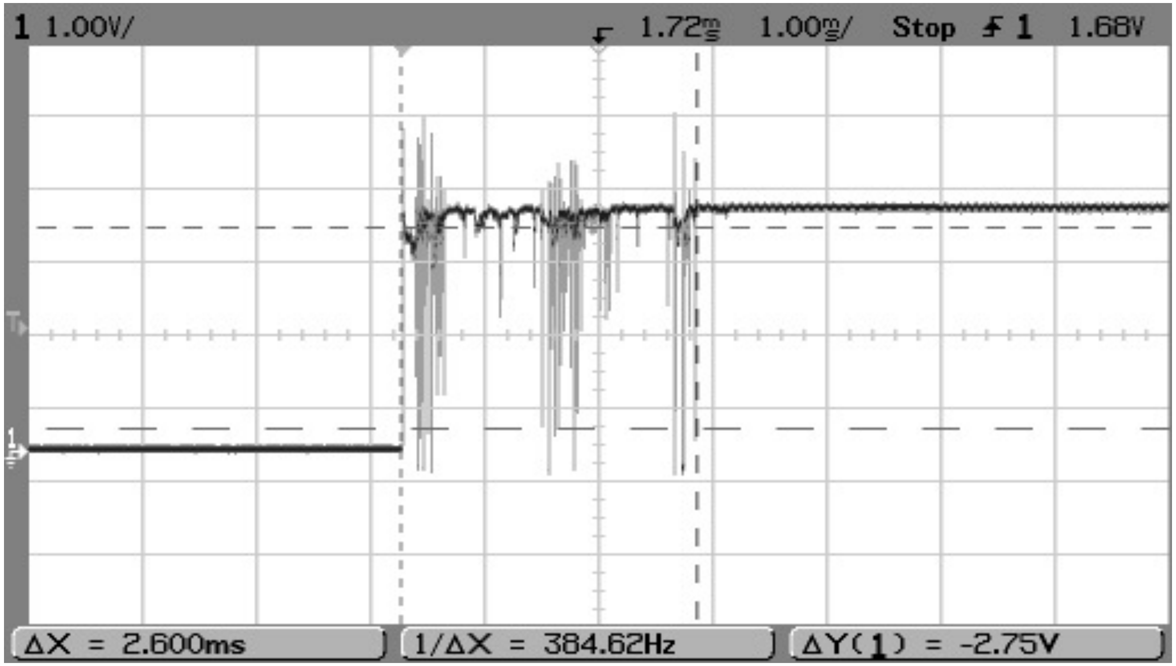
// Activate the pull up resistor by writing a high value
// on the input pin!

// 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
}
```

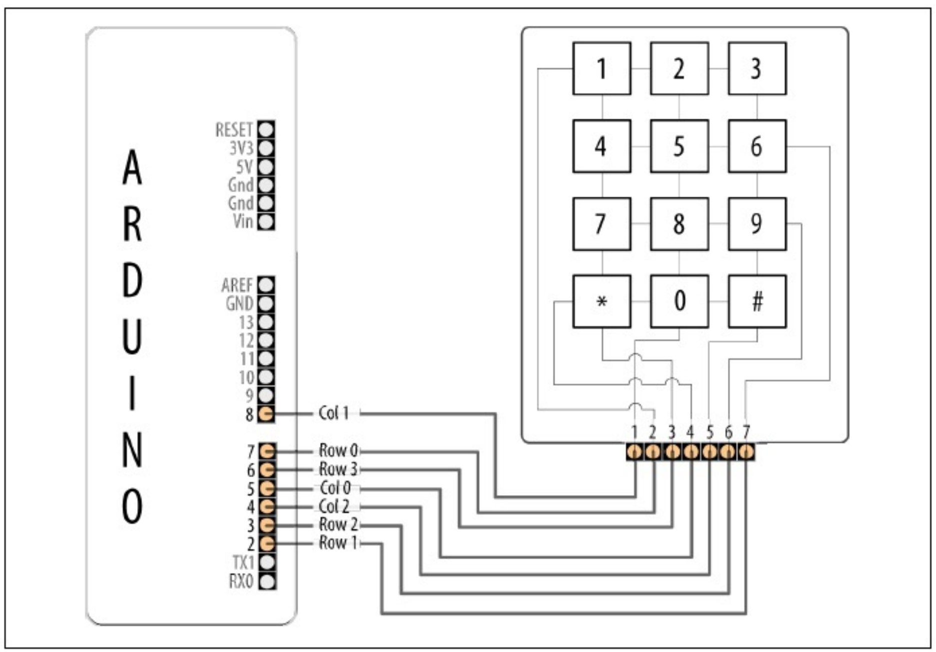
Input / Output with Arduino

- **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
  }
}
```

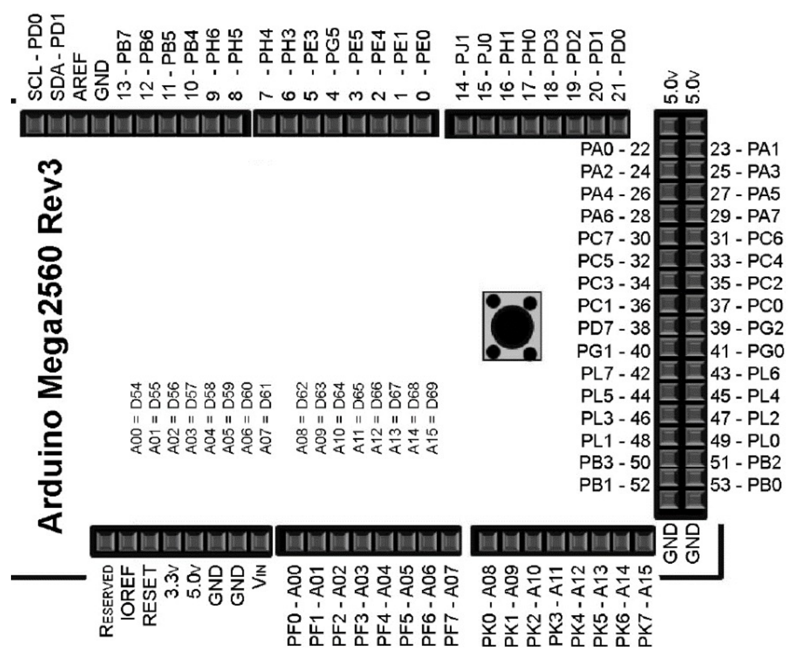
Input / Output with Arduino

- **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 release

        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)
}
```