Sensors & Microsystem Electronics: microcontrollers

PART 1: GETTING STARTED

Getting started:

READ: UCONTROLLERS_GETSTARTED.PDF

Task 1

TURN ON THE UPPER LED WHEN THE JOYSTICK IS PRESSED

Program structure

Source code (.asm)

- Includes & Definitions
- Boot code & Interrupt vectors
- Init function
  - Setup pins, timers & peripherals
- Main loop
  - Program business logic
- Other functions & data
  - Subroutines
  - Helper functions
  - Interrupt functions
  - tables

→ **Compiled into binary (.hex)**

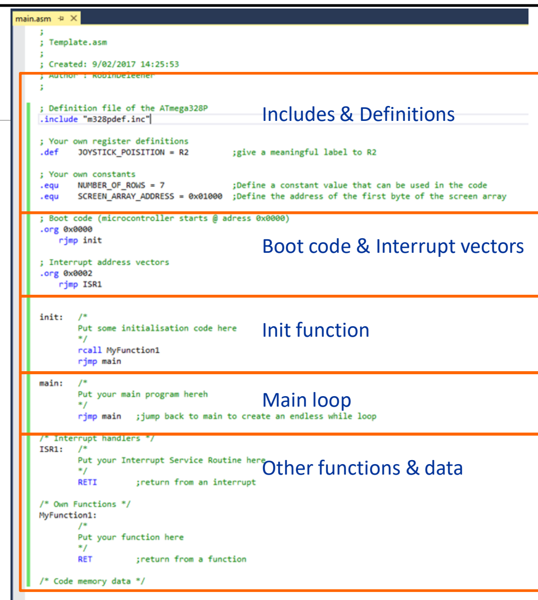
↓

Uploaded to program memory

↓

Executed on boot/reset

Example template



Consider the goal and the logic of the task

Turn ON the upper LED when the JOYSTICK is pressed

- Input → Pushbutton of the joystick
- Output → LED
- Init
  - Configure pushbutton pin
  - Configure LED pin
- Main loop
  - Get pushbutton state
  - Make decision
    - Pressed → TURN ON LED
    - Not Pressed → TURN OFF LED
  - Go back to beginning of main loop

Microcontrollers execute code sequentially

Microcontrollers only do EXACTLY what you tell them to do.

Nothing more, Nothing less!

Implementation of start-up

- uC starts executing at address 0 (0x000) of the program memory
- Where is the first useful instruction? → 'Init function'

```
; BOOT
.ORG 0x0000
RJMP init
```

Table 12-1. Reset and Interrupt Vectors in ATmega48A and ATmega48PA

Vector No.	Program Address	Source	Interrupt Definition
1	0x000	RESET	External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset
2	0x001	INT0	External Interrupt Request 0
3	0x002	INT1	External Interrupt Request 1
4	0x003	PCINT0	Pin Change Interrupt Request 0
5	0x004	PCINT1	Pin Change Interrupt Request 1
6	0x005	PCINT2	Pin Change Interrupt Request 2
7	0x006	WDT	Watchdog Time-out Interrupt
8	0x007	TIMER2 COMPA	Timer/Counter2 Compare Match A
9	0x008	TIMER2 COMPB	Timer/Counter2 Compare Match B
10	0x009	TIMER2 OVF	Timer/Counter2 Overflow
11	0x00A	TIMER1 CAPT	Timer/Counter1 Capture Event
12	0x00B	TIMER1 COMPA	Timer/Counter1 Compare Match A
13	0x00C	TIMER1 COMPB	Timer/Coutner1 Compare Match B
14	0x00D	TIMER1 OVF	Timer/Counter1 Overflow
15	0x00E	TIMER0 COMPA	Timer/Counter0 Compare Match A
16	0x00F	TIMER0 COMPB	Timer/Counter0 Compare Match B
17	0x010	TIMER0 OVF	Timer/Counter0 Overflow
18	0x011	SPI, STC	SPI Serial Transfer Complete
19	0x012	USART, RX	USART Rx Complete
20	0x013	USART, UDRE	USART, Data Register Empty
21	0x014	USART, TX	USART, Tx Complete
22	0x015	ADC	ADC Conversion Complete
23	0x016	EE READY	EEPROM Ready

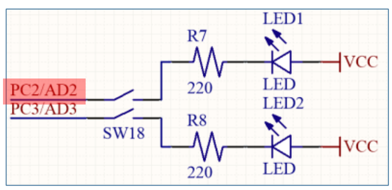
The **.ORG** directive forces the compiler to put the next chunk of code at a specific address, in this case, it puts "RJMP init" at address 0x0000. This address can be found int the vector table, and corresponds to the address loaded when the microcontroller is reset or boots up. The other addresses have different meanings and applications (see presentation about interrupts)

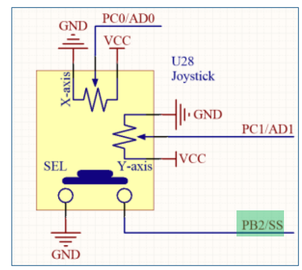
These two lines have the following function: When the microcontroller is reset, it executes address 0x0000 of the program code. With these two lines we have forced a Jump to Init at this address. So on reset, the microcontroller will immediately jump to the initialization function.

Pin mapping

The LED is connected to pin **PC**2, which is bit 2 of the 8-bit register **PORTC.**

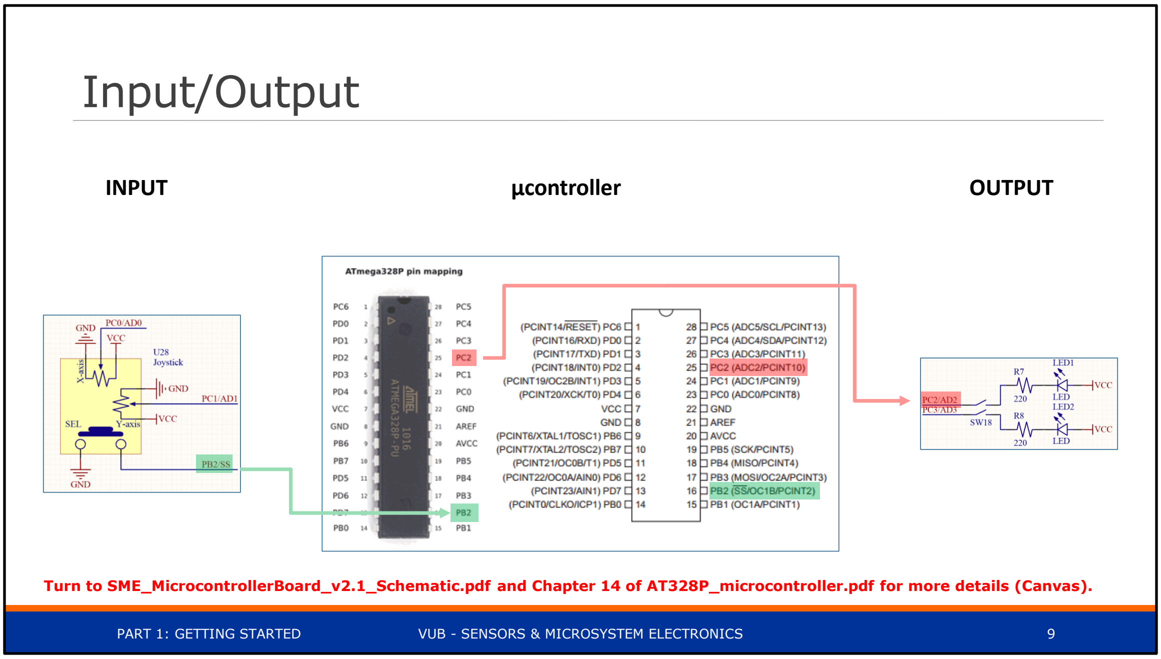
The Joystick switch is connected to pin **PB**2, which is bit 2 of the 8-bit register **PORTB**.





The microcontroller is used to interact with the outside world trough electrical signals applied or read from its Input/Output – pins. The signals can be either digital (1/0) or analog (which is in turn digitalized)

Digital pins can be either an input or an output, This is always viewed from the perspective of the microcontroller:
If the microcontroller needs to get the value that is externally applied to a pin (e.g. A button, a signal coming from another chip) it is an **IN**PUT.
If the microcontroller applies a value on a pin, setting it either to a high or a low potential (“sending a signal to outside of the microcontroller) it is an **OUT**PUT

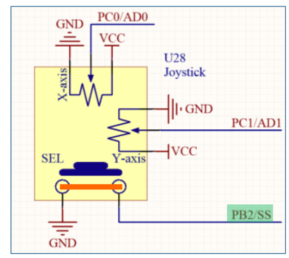


The pins of the microcontroller are separated in BANKS of 8 pins (Remember: its an 8bit microcontroller, everything is in groups of 8 bits)

In these banks each pin can be set individually to either INPUT or OUTPUT, or their alternative function.

Electrical connections

- JOYSTICK PUSH/SELECT
  - Press the joystick to connect GND (the ground) to pin PB2
  - Inputs have high impedance
    - Pin PB2 floats when the joystick is released
  - How to prevent floating pins?



Looking at the schematic one can see that when the button is pressed, PB2 is connected to ground. Yet when it is open it is not connected to anything.

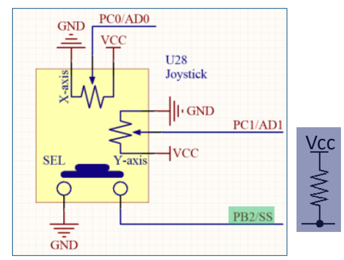
The button is connected to an input. In general, an input has a high impedance, which means you do not need a lot of current to change its value. But this also means, if there is nothing to force it to a certain value, it will float. A floating pin will have an undefined value because it can be either high or low due to electromagnetic and capacitive effects.

As such, any input pin we use, should not be left floating at any time.

Electrical connections

- JOYSTICK PUSH/SELECT
  - Press the joystick to connect GND (the ground) to pin PB2
  - Pin PB2 floats when the joystick is released
  - How to prevent floating pins?

- → pull-up resistor

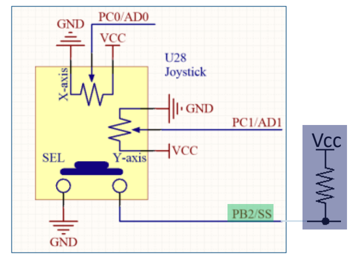


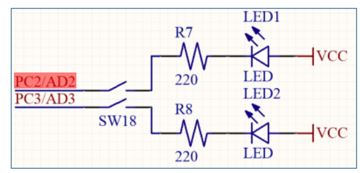
To prevent the floating of PB2 we need to get it to a known value even when the button is open. Since the button shorts to ground when pressed, we should get the pin to VCC when not pressed otherwise there is no distinction between a closed and open button. To do this we need to connect it in an “overridable” way to VCC. We do this with a pull-up resistor.

When the switch is open the resistor will “Softly” pull it to VCC. Yet when we push the button, we connect the pin to ground with a resistance close to 0 Ohm. As such the pin will be held at low level.

Electrical connections

- JOYSTICK PUSH/SELECT
  - Press the joystick to connect GND (the ground) to pin PB2
  - Pin PB2 floats when the joystick is released
  - How to prevent floating pins? → Pull-up Resistor
- LED
  - Close switch SW18 by moving the switch towards "on"
  - How to turn ON LED1?
  - → **connect GND to pin PC2**
  - How to turn OFF LED1?
  - → **connect VCC to pin PC2**



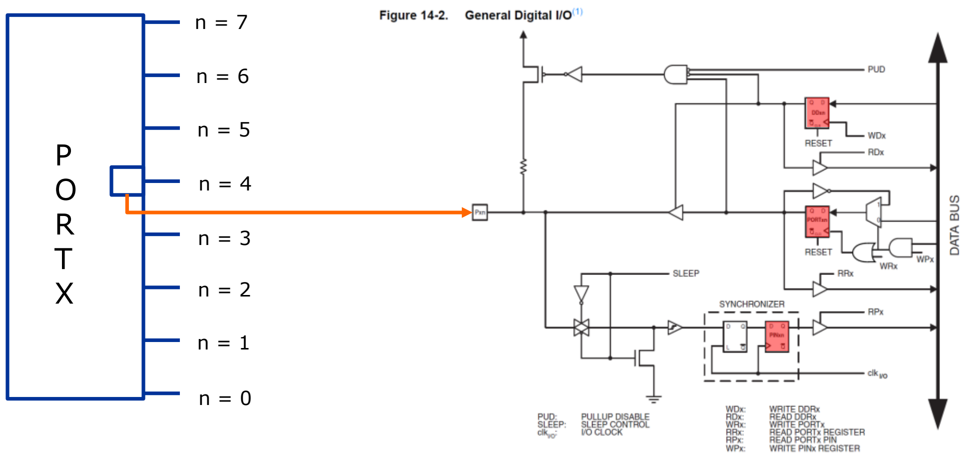


The LED is an output, outputs can be either HIGH or LOW (1 or 0) this corresponds to VCC and Ground, in this case this is 5V and 0V respectively.

In this schematic, the anode of the LED is connected to VCC. To get current flowing trough the LED, the cathode side needs to be connected to a lower potential.

This results in setting the pin to LOW to light up the led, and setting it to HIGH to turn it off.

Logical pin interface



The image on the right shows the internal structure of a single I/O pin. In red three registers are indicated. We have a PIN, PORT and DDR register.

Pin configuration

14.2.1 Configuring the Pin

Each port pin consists of three register bits: DDxn, PORTxn, and PINxn. As shown in "Register Description" on page 91, the DDxn bits are accessed at the DDRx I/O address, the PORTxn bits at the PORTx I/O address, and the PINxn bits at the PINx I/O address.

The DDxn bit in the DDRx Register selects the direction of this pin. If DDxn is written logic one, Pxn is configured as an output pin. If DDxn is written logic zero, Pxn is configured as an input pin.

If PORTxn is written logic one when the pin is configured as an input pin, the pull-up resistor is activated. To switch the pull-up resistor off, PORTxn has to be written logic zero or the pin has to be configured as an output pin. The port pins are tri-stated when reset condition becomes active, even if no clocks are running.

If PORTxn is written logic one when the pin is configured as an output pin, the port pin is driven high (one). If PORTxn is written logic zero when the pin is configured as an output pin, the port pin is driven low (zero).

14.2.2 Toggling the Pin

Writing a logic one to PINxn toggles the value of PORTxn, independent on the value of DDRxn. Note that the SBI instruction can be used to toggle one single bit in a port.

Table 14-1. Port Pin Configurations

DDxn	PORTxn	PUD (in 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)

Each pin has four different possible configurations:

When DDR = 0, the pin is an input. In this case we have two possibilities:

PORT = 0, the pin is a normal High impedance input

PORT = 1, the pin is an input with an internal (on-chip) PULL-UP resistor enabled

When DDR = 1, the pin is an output. In this case we have two possibilities:

PORT = 0, the pin is driven with a LOW value (GND)

PORT = 1, the pin is driven with a HIGH value (VCC)

In all cases will the PIN bit contain the value of the pin, regardless whether it is an input or an output. And is used to get the value of this specific pin.

As an extra feature, when the pin is an OUTPUT, writing a 1 to the PIN register, will result in inverting the value (1→ 0 or 0→ 1) this can simplify your code. The alternative is reading the value of the PORT, inverting it an writing it back. **NOTE:** this write operation will not overwrite the value in the pin register, it will still mirror the value of the pin.

Configuration of the input pin

```
init:
; Configure input pin PB2
CBI DDRB,2 ; Pin PB2 is an input
SBI PORTB,2 ; Enable the pull-up resistor
```

Mnemonic	Operands	Description	Operation	Flags	Cycles
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2

Clear **B**it in **I**/O register (**CBI**)

Set **B**it in **I**/O register (**SBI**)

14.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
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	1	0	0	

14.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
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	

14.4.4 PINB – The Port B Input Pins Address[1]

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
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	sense	N/A	N/A	

14.4.5 PORTC – The Port C Data Register

Bit	7	6	5	4	3	2	1	0	
0x08 (0x28)	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.6 DDRC – The Port C Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x07 (0x27)	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

14.4.7 PINC – The Port C Input Pins Address[1]

Bit	7	6	5	4	3	2	1	0	
0x06 (0x26)	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Configuration of the output pin

```
; Configure output pin PC2
SBI DDRC,2 ; Pin PC2 is an output
SBI PORTC,2 ; Output Vcc => LED1 is turned off!
```

Mnemonic	Operands	Description	Operation	Flags	Cycles
SBI	P,b	Set bit in I/O register	I/O(P,b) = 1	None	2
CBI	P,b	Clear bit in I/O register	I/O(P,b) = 0	None	2

Clear **B**it in **I**/O register (**CBI**)

Set **B**it in **I**/O register (**SBI**)

14.4.2 PORTB – The Port B Data Register

Bit	7	6	5	4	3	2	1	0	
0x05 (0x25)	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	PORTB
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	1	0	0	

14.4.3 DDRB – The Port B Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	DDRB
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	

14.4.4 PINB – The Port B Input Pins Address[1]

Bit	7	6	5	4	3	2	1	0	
0x03 (0x23)	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	PINB
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	sense	N/A	N/A	

14.4.5 PORTC – The Port C Data Register

Bit	7	6	5	4	3	2	1	0	
0x08 (0x28)	–	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	PORTC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	0	0	

14.4.6 DDRC – The Port C Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x07 (0x27)	–	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	DDRC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	1	0	0	

14.4.7 PINC – The Port C Input Pins Address[1]

Bit	7	6	5	4	3	2	1	0	
0x06 (0x26)	–	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	PINC
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	N/A	N/A	N/A	N/A	1	N/A	N/A	

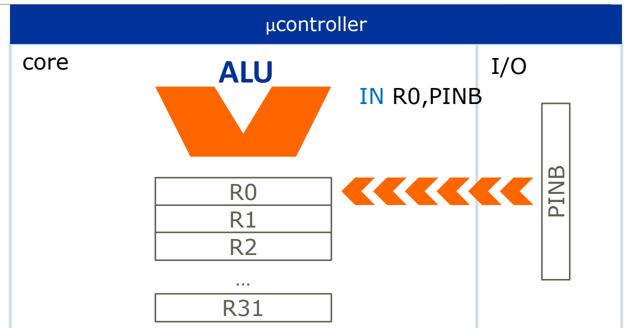
Get value of PINB

```
main:
; Get value of PINB
IN R0,PINB
```

INPUT (**IN**)

OUTPUT (**OUT**)

Mnemonic	Operands	Description	Operation	Flags	Cycles
IN	Rd,P	In Port	Rd = P	None	1
OUT	P,Rr	Out Port	P = Rr	None	1



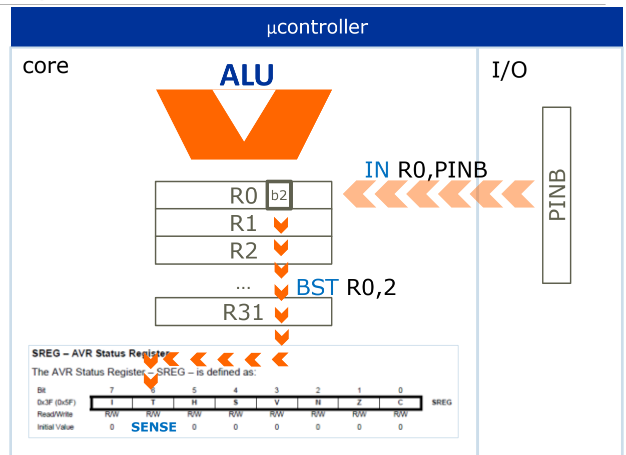
SREG – AVR Status Register

The AVR Status Register – SREG – is defined as:

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
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	

Copy PB2 to the T flag

```
; Copy PB2 (bit 2 of PINB)
; to the T flag
BST R0,2
```



Bit Store from register to T flag (BST)

- **Bit 6 – T: Bit Copy Storage**

The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.

Make a decision

Mnemonic	Operands	Description	Operation	Flags	Cycles
BRTS	k	Branch if T flag set	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Branch if T flag cleared	if(T==0) PC = PC + k + 1	None	1/2

```
; Copy PB2 (bit 2 of PINB) to the T flag
BST R0,2

; The joystick is pressed of the T flag is cleared
BRTC JoyPressed

JoyNotPressed:
    SBI PORTC,2 ; Turn off LED1
    RJMP main

JoyPressed:
    CBI PORTC,2 ; Turn on LED1
    RJMP main
```

BRanch if **T** flag is **C**leared (**BRTC**)

BRanch if **T** flag is **S**et (**BRTS**)

Relative **J**u**MP** to (**RJMP**)

Repeat forever

```
main: ; Loops always begin with a label

    ; Your logic goes here

    RJMP main ; Go back to the beginning
```

Solution of the first task

```
.INCLUDE "m328pdef.inc"          ; Load addresses of (I/O) registers

.ORG 0x0000
RJMP init                        ; First instruction that is executed by the microcontroller

init:
; Configure input pin PB2
CBI DDRB,2                 ; Pin PB2 is an input
SBI PORTB,2                ; Enable the pull-up resistor

; Configure output pin PC2
SBI DDRC,2                 ; Pin PC2 is an output
SBI PORTC,2                ; Output Vcc => LED1 is turned off!

main:
IN R0,PINB                 ; Get value of PINB
BST R0,2                   ; Copy PB2 (bit 2 of PINB) to the T flag

; The joystick is pressed of the T flag is cleared
BRTC JoyPressed                  ; Branch if the T flag is cleared

JoyNotPressed:
   SBI PORTC,2             ; Turn off LED1
   RJMP main               ; Create an infinite loop

JoyPressed:
   CBI PORTC,2             ; Turn on LED1
   RJMP main               ; Create an infinite loop
```

Verify the functional behaviour

- Task 1:
  - Create a new project
  - Implement Task 1
  - Build & upload to the board.
    - See ucontroller_GetStarted.pdf for instructions.
- Verify that you get the expected functional behaviour
- Task 2: turn on the lower LED when the switch is in the HIGH state
- Task 3: blink an LED at a visible speed
- Task 4: sound the buzzer at an audible frequency

Sidenote: how does the buzzer work?

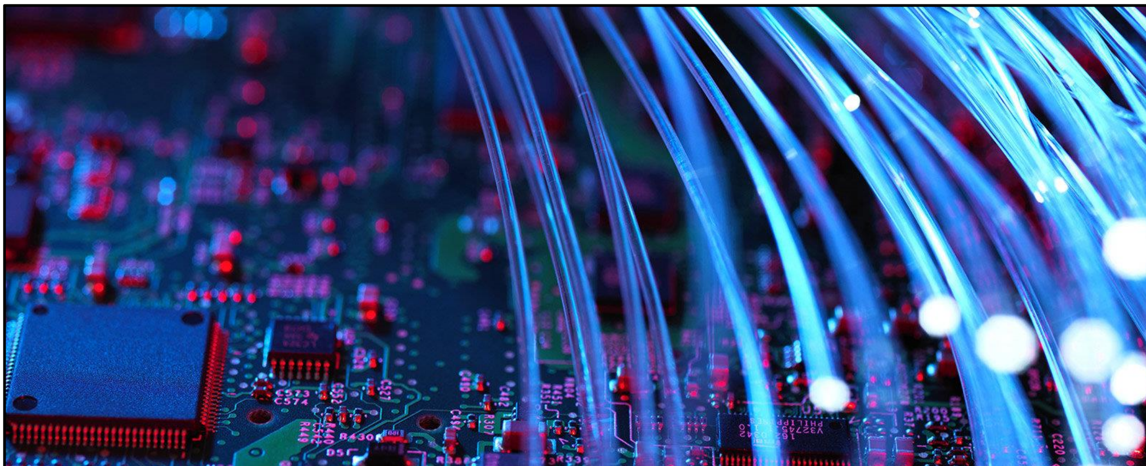
- The buzzer consists of a piezo element that contracts or expand when the applied signal changes from low to high or vice-versa
- This expansion/contraction causes vibrations in the air resulting in sound
- The frequency of the applied signal determines the created audio frequency.
- A square wave is close enough to a sine wave to create rudimentary sounds

To sound the buzzer the pin of connected to the buzzer (PB1) needs to be toggled. To get a square wave at 440Hz the pin needs to be set and cleared once every period, this comes to a frequency of 880Hz.

HINT: use SBI PINx,1 to toggle a pin (explanation see datasheet or slide about pin config in Part1)

What to do now?

- Download the **ucontrollers_gettingstarted.pdf** from canvas & follow the instructions
- Make and understand task 1 using the information and code snippets from this presentation
- Upload to your microcontroller board and see if it works.
- Continue working on task 2-4



End of Part 1: Getting started