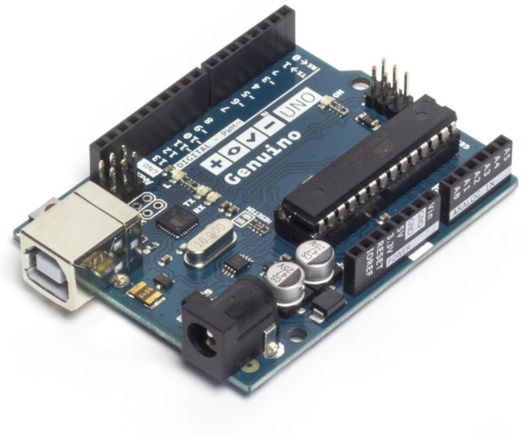


Sensors & Microsystem Electronics: Microcontrollers

INTRODUCTION

Course objectives

- Understand the basic and advanced concepts related to the use of microcontrollers
- Program in assembly language
- Understand electronic schematics
- Employ the microcontroller and schematics to sense and drive the environment
- Combine different functional units and peripherals to build a working system

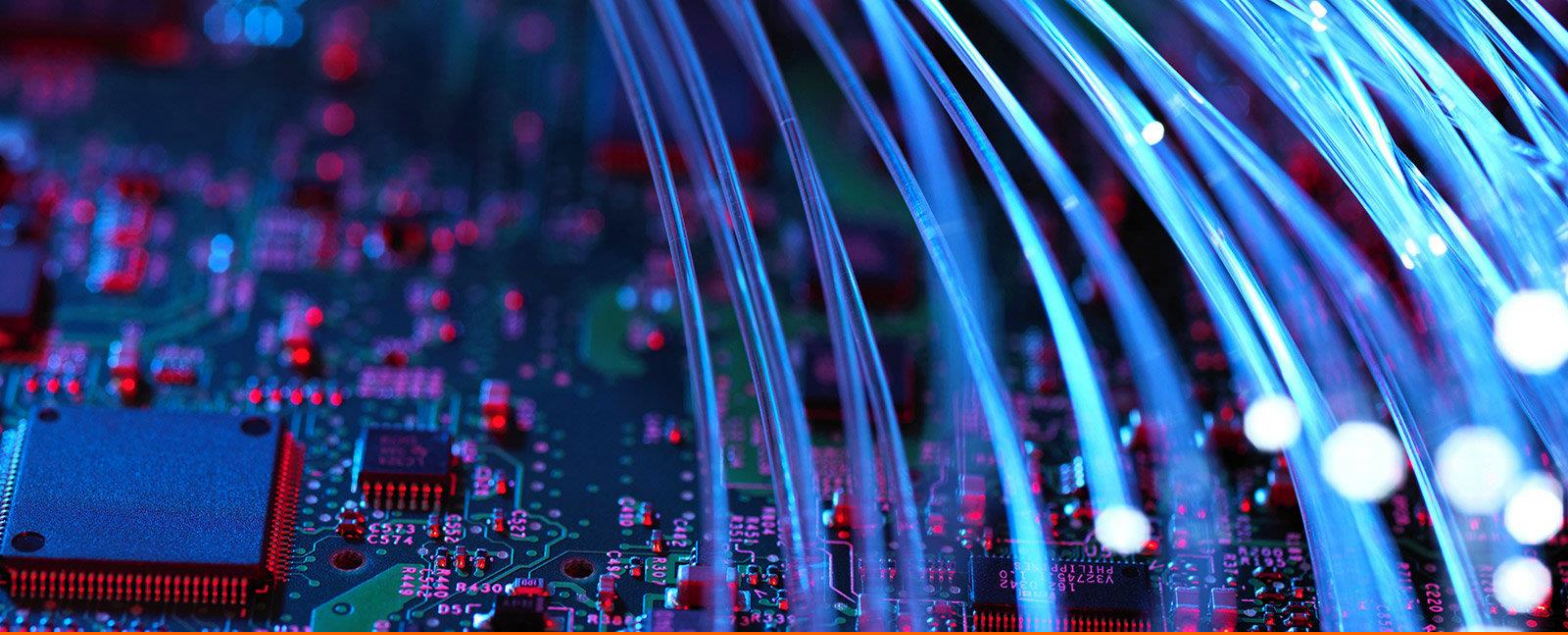
Approach

- Each video introduces some functional units & principles:
  - I/O ports, timers & interrupts, keyboard, display, …
- Write basic programs that configure these units so that we can observe their functionality
- Make the 10 basic programs that are listed on Canvas as tasks
- From easy to difficult
- In preparation of your project

Contact information

- Ayman Morsy
  - Office: K5.56
  - Ayman.morsy@vub.be

- Tuur Bruneel
  - Office: K5.04c
  - Tuur.Bruneel@vub.be



This is an ENGINEERING lab

What does that mean?

You will need to figure things out by yourself!

WE ARE HERE TO HELP YOU FIGURE THINGS OUT, NOT GIVE YOU THE ANSWER

For most problems there is no exact or single answer…

ANY WAY THAT WORKS IS VALID. ANYTHING ELSE DEPENDS ON HOW YOU DEFINE YOUR FIGURE OF MERIT

Start simple and build from there…

DEFINE, ANALYSE, COMPARTMENTALIZE TO SIMPLIFY YOUR PROBLEM.

REPEAT AT EACH LEVEL OF SIMPLIFICATION.

TEST INDIVIDUAL PARTS AND BRING THEM TOGETHER BIT BY BIT.

Be bold and dare to try!

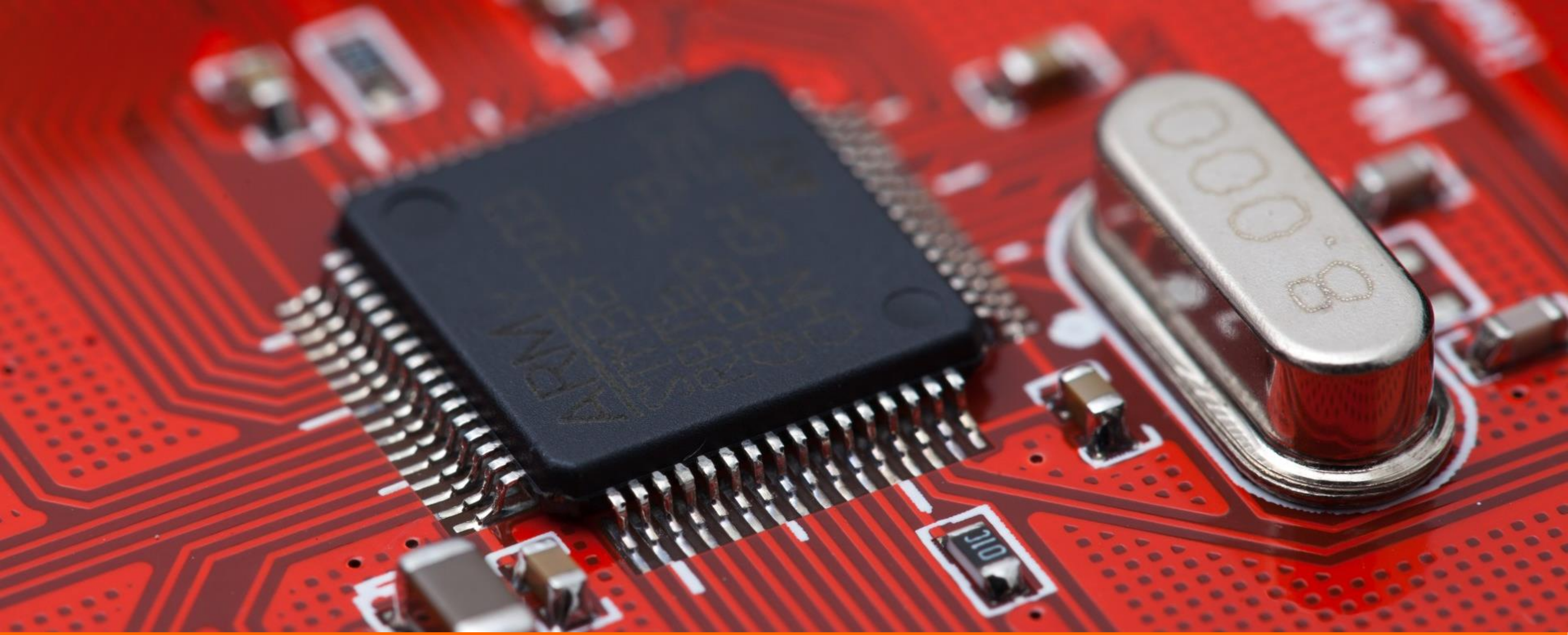
BUILDS THINGS, BREAK THINGS, CHANGE THINGS.

UNDERSTAND WHAT YOU ARE DOING, WHAT IS HAPPENING AND WHY.

RTFM!

READ THE … MANUAL

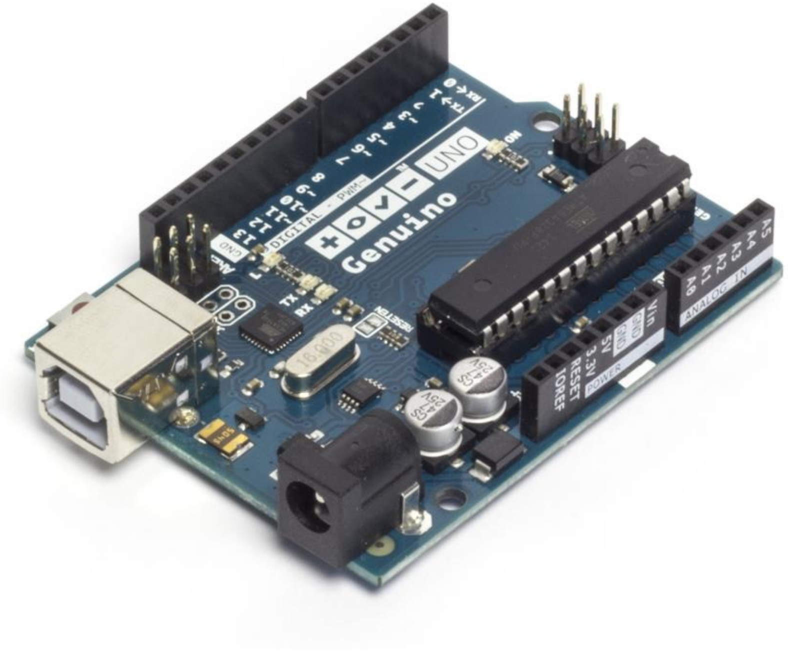
DATASHEETS, MANUALS, SCHEMATICS, DOCUMENTATION, …

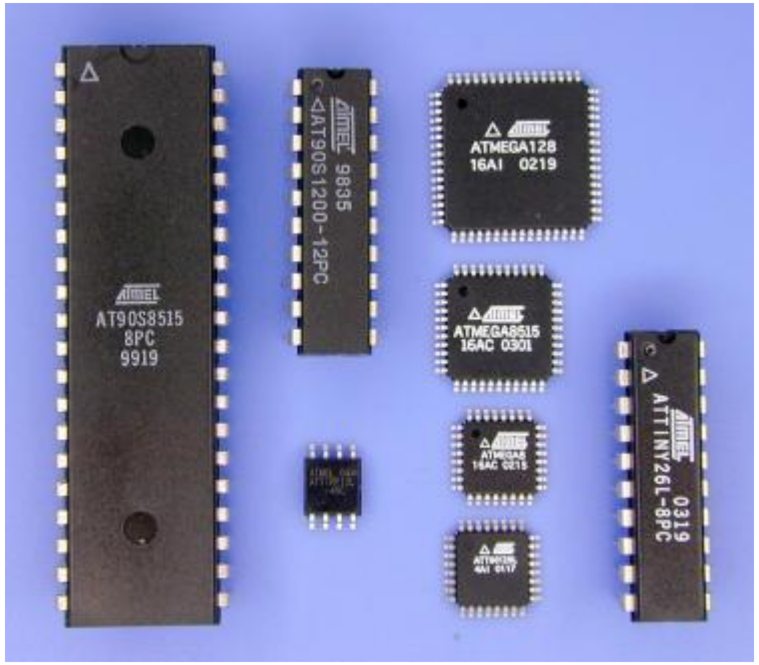


Introduction to microcontrollers

What is a microcontroller?

- System on Chip (SoC)
- I/O control
- Low power
- Applications
  - Clock radios
  - Telephones
  - Cars
  - …





Microprocessors & Microcontrollers

MICROPROCESSOR

- General purpose applications
  - E.g. digital signal processing, personal computers, servers and devices running operating systems
- Multiple processes
- Control unit
  - Instructions @ external memory
  - Very fast (GHz)
  - Instruction pipelining and multicore architectures
  - More expensive (5 euro – 2000 euro)
- External program memory & RAM

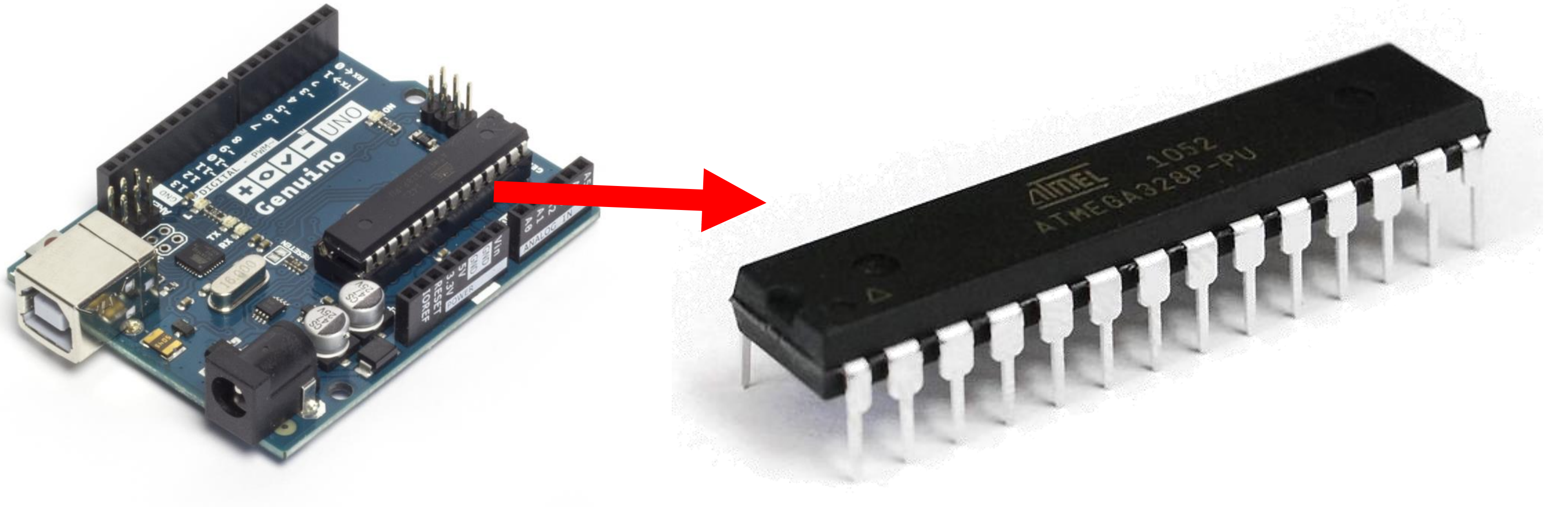
MICROCONTROLLER

- Embedded systems
  - E.g. Automatically controlled devices, human-machine interfaces,...
- 1 dedicated task
- Control unit + on chip functional units
- Instructions internally stored
  - Fast (kHz-MHz)
  - Sequential instruction execution
  - Very cheap (10 cent – 5 euro)

Hybrids possible!

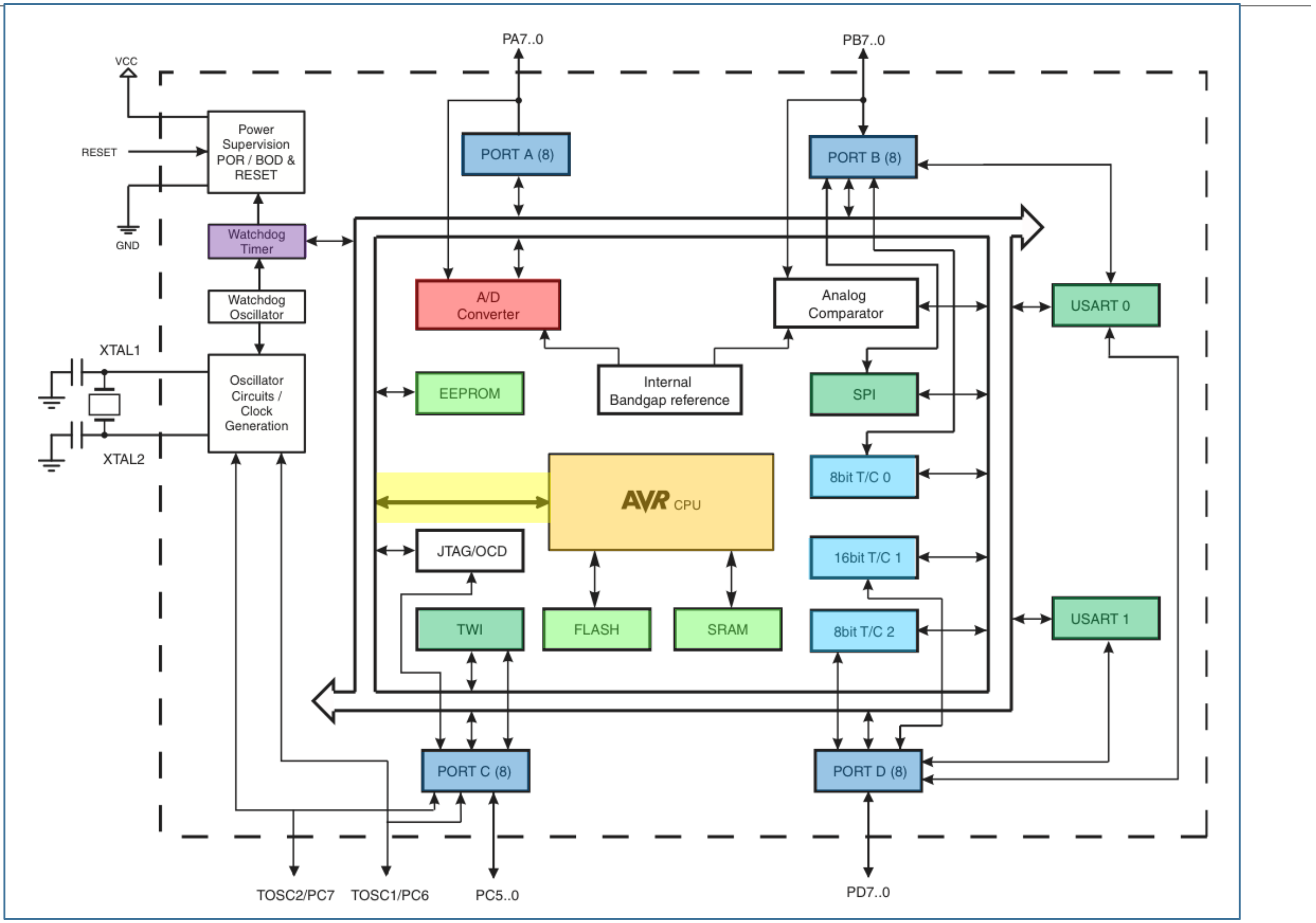
Hardware: ATMega328P (Arduino UNO)

- 8-bit AVR architecture
  - EVERYTHING is grouped in 8 bits or 1 byte
    - 1 bit → 1 or 0
    - 1 byte → 8-bit group



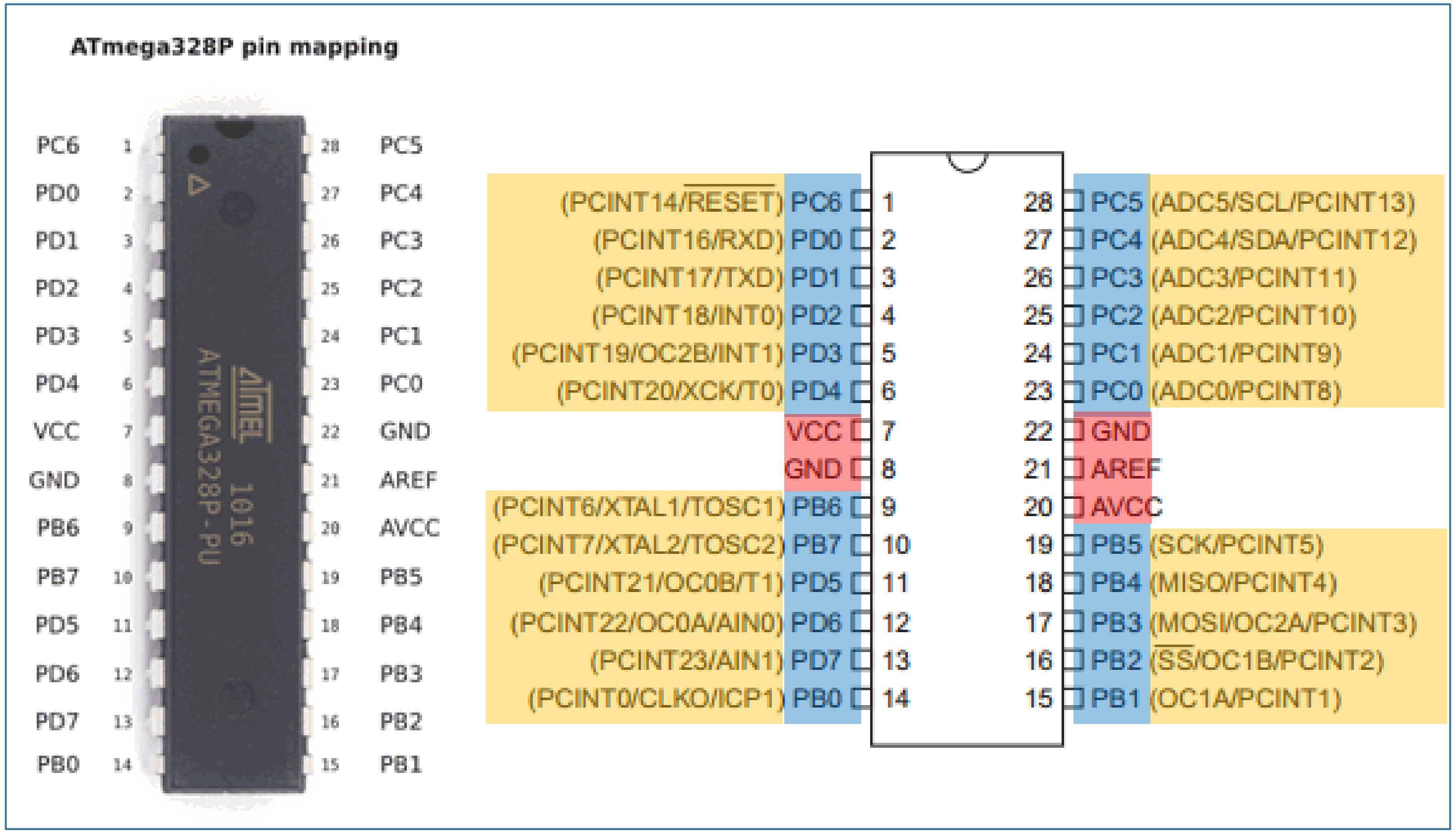
Hardware

- ATMega328P
  - Present on the Arduino Uno
  - Peripheral units
    - UART/SPI/TWI
    - ADC
    - 4 I/O ports
  - Internal units
    - 32 Registers
    - Watchdog timer
    - 3 Timers
    - Memory:
      - Flash/SRAM/EEPROM
    - Interrupt system



Hardware

- ATMega328P
  - 28 pin DIP
  - Power pins
  - GPIO pins
    - 4 pin banks
    - Max 8 pins / bank
  - Alternate functions
    - Examples
      - Communication
      - ADC
      - Timer/PWM
      - ...
    - See datasheet

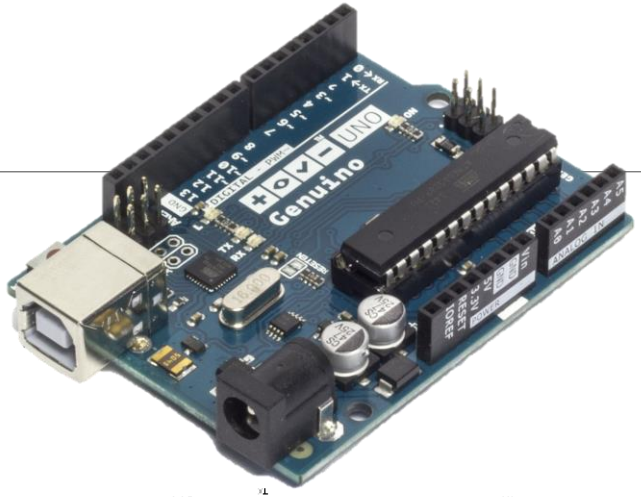


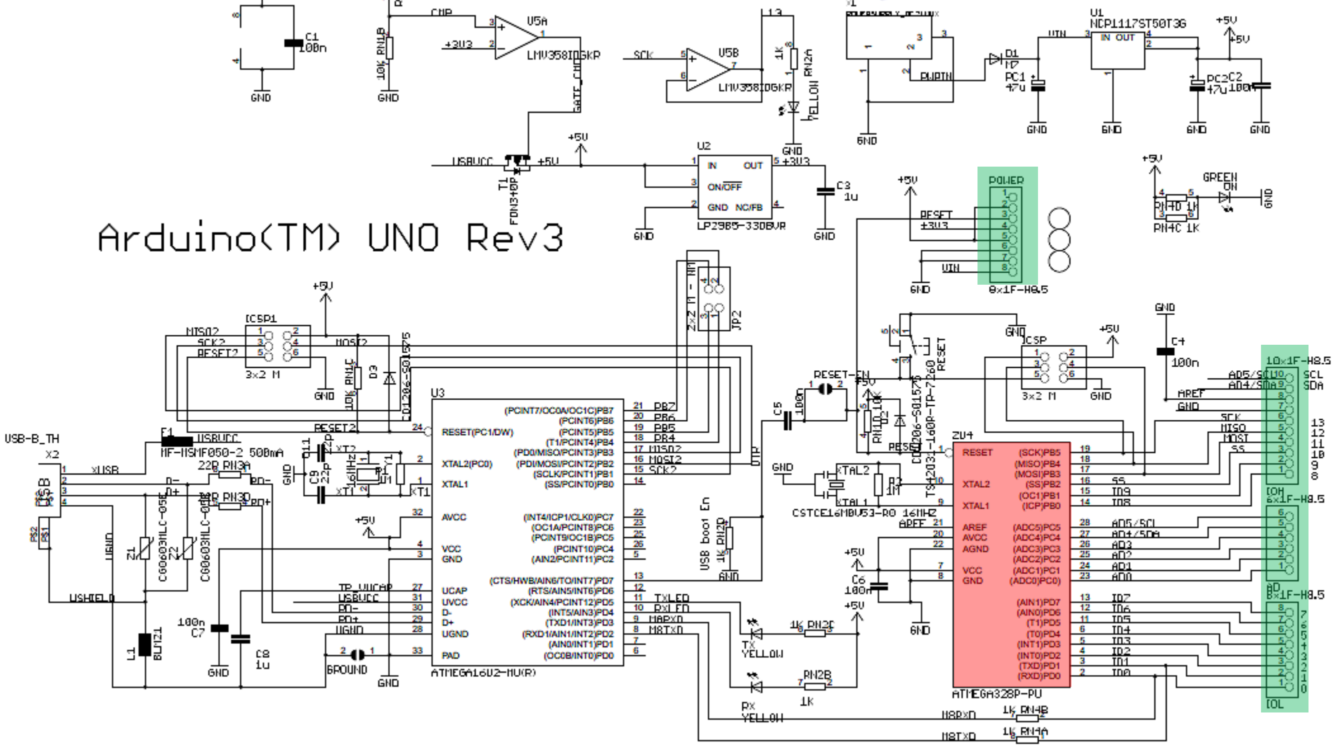
ATmega328P pin mapping

PC6 1 28 PC5
PD0 2 27 PC4
PD1 3 26 PC3
PD2 4 25 PC2
PD3 5 24 PC1
PD4 6 23 PC0
VCC 7 22 GND
GND 8 21 AREF
PB6 9 20 AVCC
PB7 10 19 PB5
PD5 11 18 PB4
PD6 12 17 PB3
PD7 13 16 PB2
PB0 14 15 PB1
ATMEL
ATMEGA328P-PU
1016
(PCINT14/RESET) PC6 1 28 PC5 (ADC5/SCL/PCINT13)
(PCINT16/RXD) PD0 2 27 PC4 (ADC4/SDA/PCINT12)
(PCINT17/TXD) PD1 3 26 PC3 (ADC3/PCINT11)
(PCINT18/INT0) PD2 4 25 PC2 (ADC2/PCINT10)
(PCINT19/OC2B/INT1) PD3 5 24 PC1 (ADC1/PCINT9)
(PCINT20/XCK/T0) PD4 6 23 PC0 (ADC0/PCINT8)
VCC 7 22 GND
GND 8 21 AREF
(PCINT6/XTAL1/TOSC1) PB6 9 20 AVCC
(PCINT7/XTAL2/TOSC2) PB7 10 19 PB5 (SCK/PCINT5)
(PCINT21/OC0B/T1) PD5 11 18 PB4 (MISO/PCINT4)
(PCINT22/OC0A/AIN0) PD6 12 17 PB3 (MOSI/OC2A/PCINT3)
(PCINT23/AIN1) PD7 13 16 PB2 (SS/OC1B/PCINT2)
(PCINT0/CLKO/ICP1) PB0 14 15 PB1 (OC1A/PCINT1)

Hardware

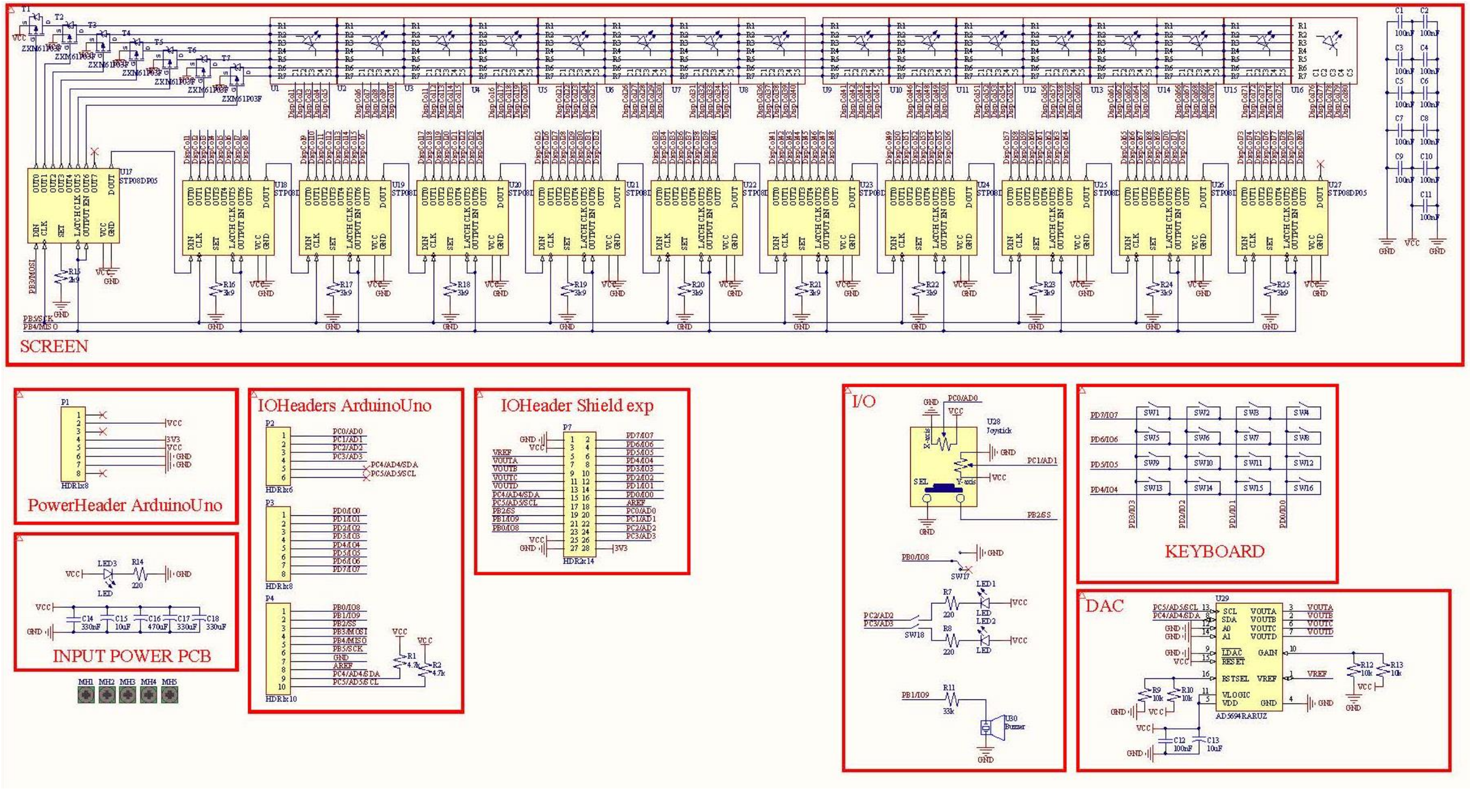
- Arduino Uno
  - ATMega328P with basic needs
    - Power supply
    - Clock generation
    - Programming circuitry
  - I/O ports & power to headers





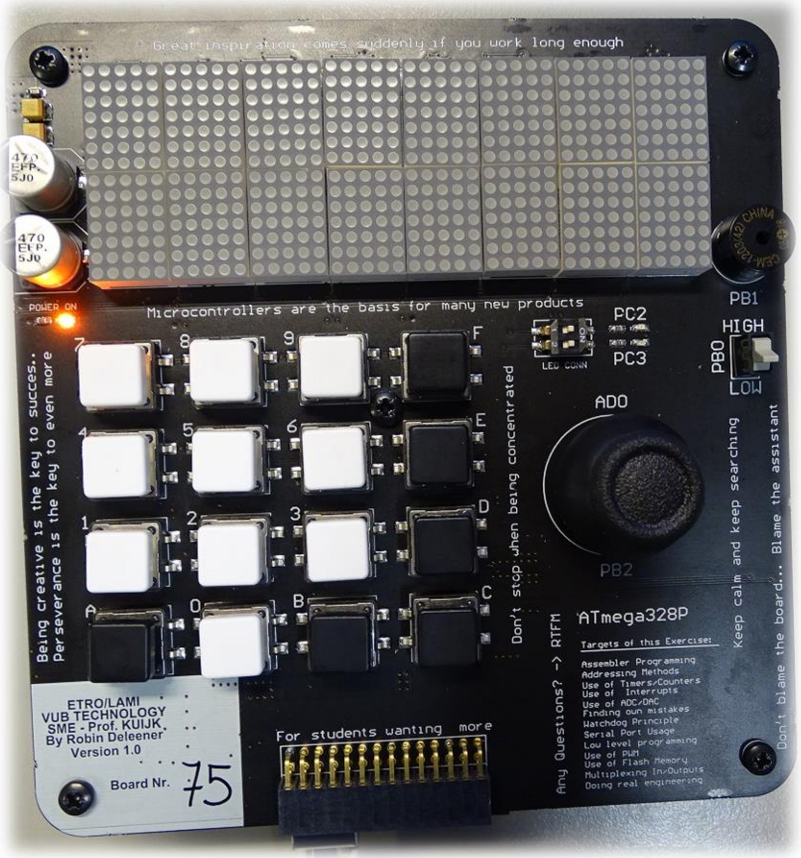
Hardware

- Demo board
  - Arduino Uno → Arduino shield
  - Headers connected to peripherals
    - LED-matrix (screen)
    - Keyboard
    - LEDs
    - Buzzer
    - Joystick
    - Switch
    - Digital to Analog Converter (DAC)
    - External interface



Hardware

- Demo board
  - Arduino Uno → "Arduino shield"
  - Headers connected to peripherals
    - LED-matrix (screen)
    - Keyboard
    - LEDs
    - Buzzer
    - Joystick
    - Switch
    - Digital to Analog Converter (DAC)
    - External interface


Great inspiration comes suddenly if you work long enough
Microcontrollers are the basis for many new products
POWER ON
PC2
PC3
PB1
HIGH
PB0
LOW
ADO
PB2
7
8
9
F
4
5
6
E
1
2
3
D
A
0
B
C
Being creative is the key to succes..
Perseverance is the key to even more
Don't stop when being concentrated
Any Questions? -> RTFM
ATmega328P
Keep calm and keep searching
Don't blame the board... Blame the assistant
ETRO/LAMI
VUB TECHNOLOGY
SME - Prof. KUIJK
By Robin Deleener
Version 1.0
Board Nr. 75
For students wanting more

Software

- Arduino Software IDE
  - Arduino programming language
    - Based on wiring & C/C++
    - Makes it easy to write code
    - Lots of libraries available
    - Hardware Abstraction Layer (HAL) → specific hardware operation is completely hidden
    - Source code is independent of the Instruction Set Architecture (ISA)
- Microchip Studio (formerly atmel studio)
  - C/C++
    - Makes it easy to write code
    - Lots of libraries available
    - HAL → minimal hardware transparency
    - Source code is independent of the ISA
- Assembly language
  - Hard to write
  - No libraries
  - No HAL → full hardware transparency
  - For 1 specific microcontroller (ISA)

Assembly language characteristics

- Low level code → full insight into hardware operation
- Optimized code → fast
- Register level programming
- Limited number of instructions

ADD, SUB, MUL, INC, AND, OR, …	(arithmetic and logic instructions)
RJMP, RCALL, BREQ, BRCS, …	(branch instructions)
SBI, LSL, ROL, …	(bit and bit-test instructions)
MOV, LD, ST, PUSH, POP, …	(data transfer instructions)

- Not verbose→ document your code!

Assembly language references

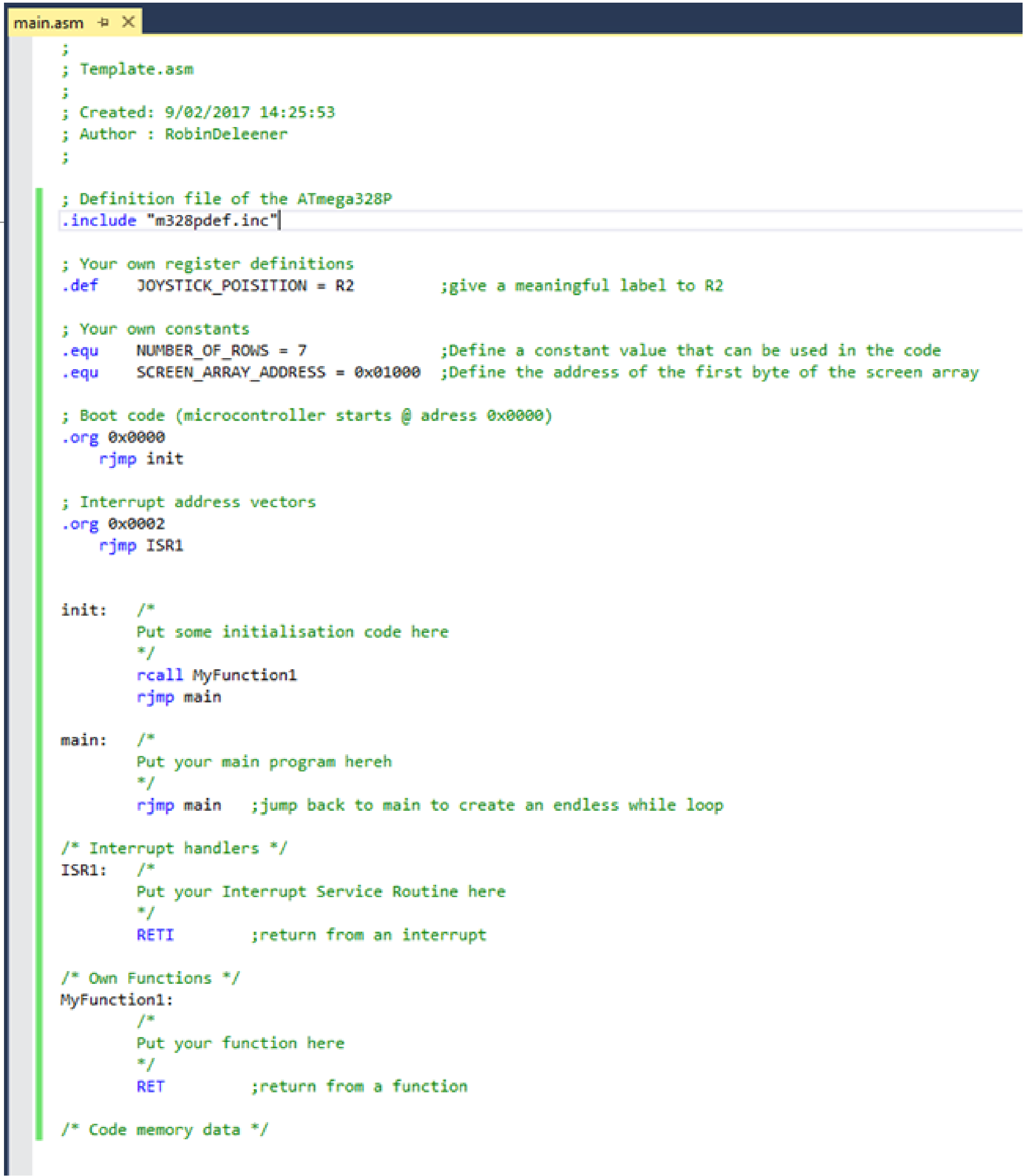
- ATmega328P instruction set

- ATmega328P datasheet

- Labels and directives
  - Assembler Directives List
  - Examples:
    - SomeLabel: defines an identifier that refers to the address in the program memory where the subsequent code is placed
    - .ORG specifies the address in the program memory where the subsequent code is placed
    - .DEF defines an identifier that refers to a register
    - .EQU defines an identifier that refers to a fixed value → constants

```
.DEF speed                = R16
.EQU SPEED_INCREMENT      = 1

.ORG 0x0000
        RJMP main

main:
        DEC  speed
        ADDI speed,SPEED_INCREMENT
        RJMP main
```

Example template



```
;
; Template.asm
;
; Created: 9/02/2017 14:25:53
; Author : RobinDeleener
;

; Definition file of the ATmega328P
.include "m328pdef.inc"

; Your own register definitions
.def    JOYSTICK_POISITION = R2         ;give a meaningful label to R2

; Your own constants
.equ    NUMBER_OF_ROWS = 7              ;Define a constant value that can be used in the code
.equ    SCREEN_ARRAY_ADDRESS = 0x01000  ;Define the address of the first byte of the screen array

; Boot code (microcontroller starts @ adress 0x0000)
.org 0x0000
    rjmp init

; Interrupt address vectors
.org 0x0002
    rjmp ISR1


init:   /*
        Put some initialisation code here
        */
        rcall MyFunction1
        rjmp main

main:   /*
        Put your main program hereh
        */
        rjmp main   ;jump back to main to create an endless while loop

/* Interrupt handlers */
ISR1:   /*
        Put your Interrupt Service Routine here
        */
        RETI        ;return from an interrupt

/* Own Functions */
MyFunction1:
        /*
        Put your function here
        */
        RET         ;return from a function

/* Code memory data */
```

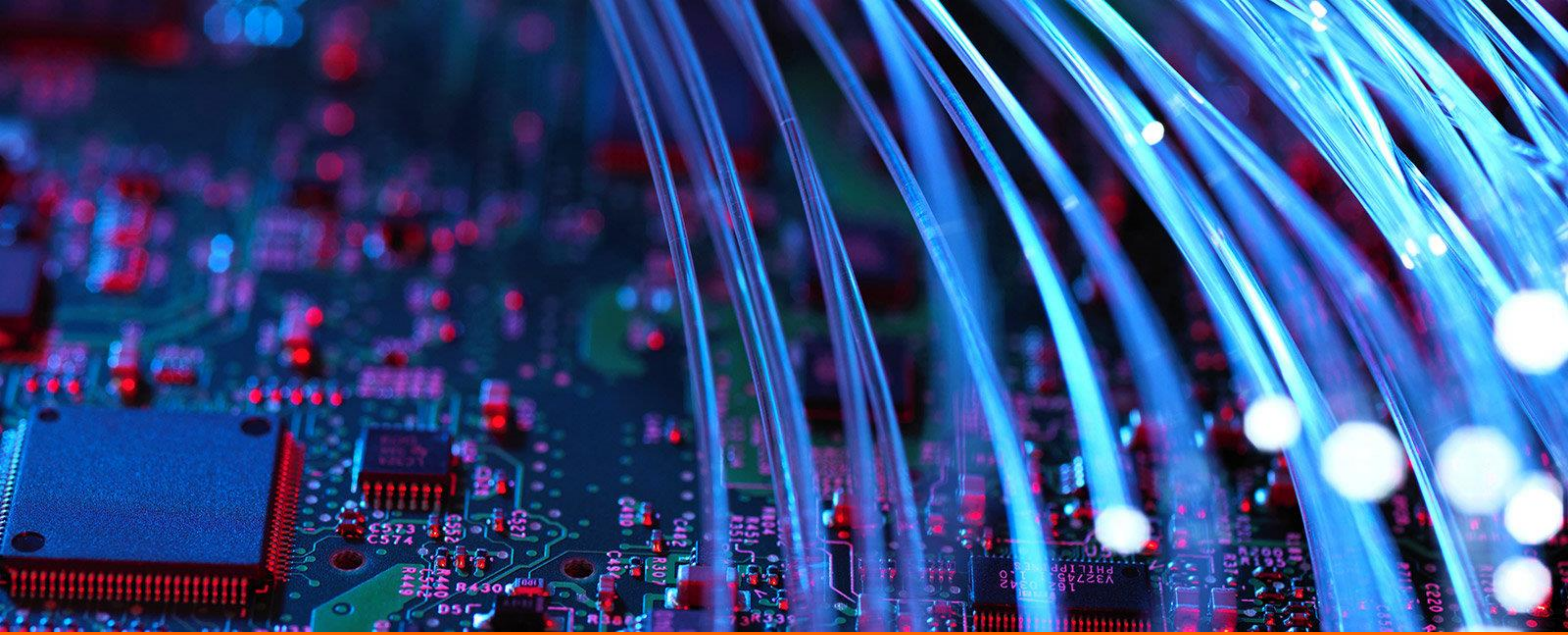
Example program

```
;include file
.include "m328pdef.inc"

; BOOT code
.org 0x0000
rjmp init

; INIT code
init:
     CBI DDRB,2;set PB2 (JOYSTICK) as input
     SBI PORTB,2;enable pull-up
     SBI DDRC, 2;set PC2 (LED1) as output
     SBI PORTC,2;write a logic "1" -> LED OFF


rjmp main;jump to endless loop
```

```
; MAIN code
main:
     IN R0,PINB;copy PB input to R0
     BST R0,2;copy PB2 into T
     BRTC PB2pressed ;jump if JOYSTICK is
     pressed
     SBI PORTC, 2;LED1 OFF
     rjmp main
PB2pressed:
     CBI PORTC, 2;LED1 ON
     rjmp main
```

Microcontrollers are dumb

THEY ONLY DO EXACTLY WHAT YOU TELL THEM TO. NOTING MORE, NOTHING LESS. SO COVER ALL YOUR BASES AND ALL SCENARIOS



End of the introduction