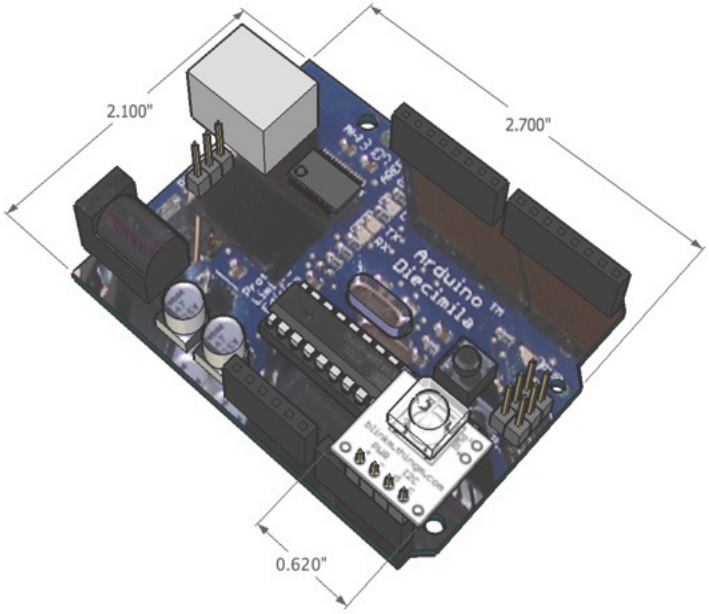
**Microcontrollers 3A SAGI**
**2019-2020 Polytech Angers**

# ARDUINO UNO
# Microcontroller ATMega328




*B. Cottenceau* B311 Polytech Angers Belle-Beille
bertrand.cottenceau@univ-angers.fr

# Table des matières

# 1 Introduction

The UNO model from ARDUINO is an electronic card whose heart is an ATMEL microcontroller of reference ATMega328. The ATMega328 is an 8-bit microcontroller of the AVR family which can be programmed in C/C++ language.

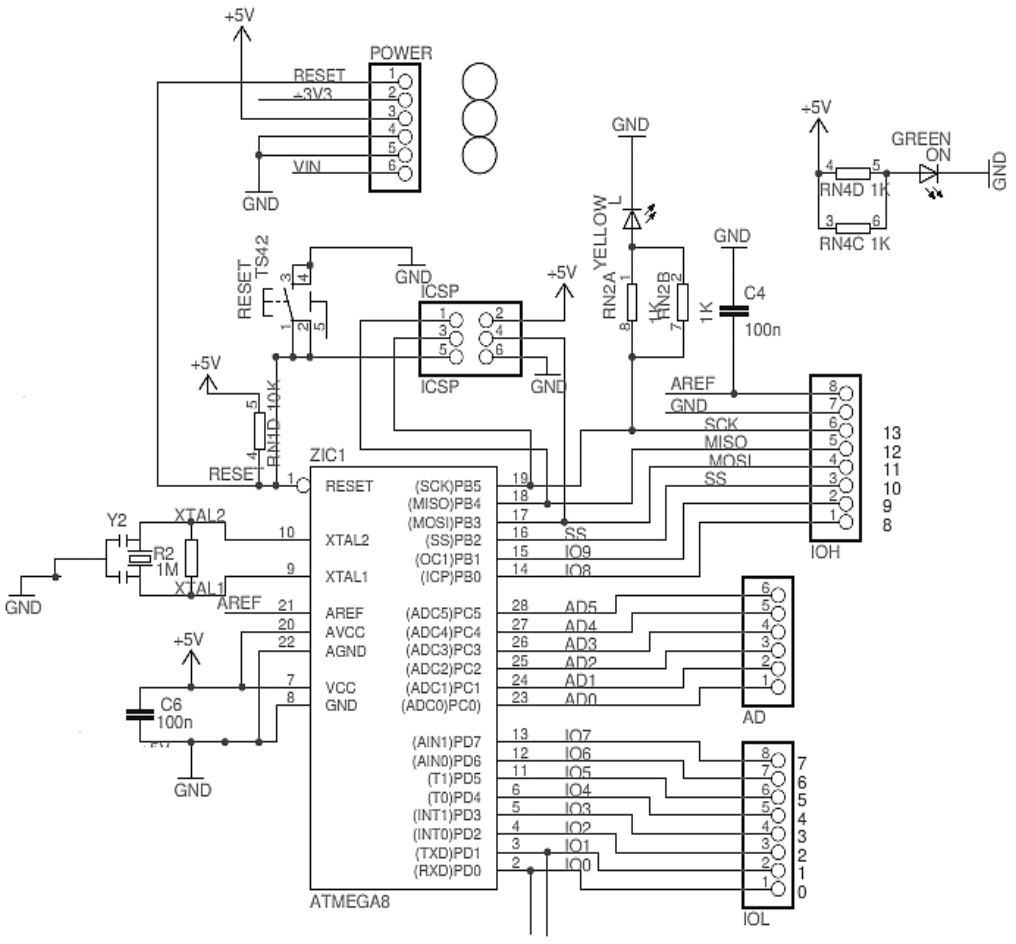
The main interest of the ARDUINO cards (other models exist : Mega, Nano...) is their ease of implementation. A development environment (IDE), based on open-source tools, is provided. In addition, loading the compiled program into the memory of the microcontroller is very simple (via USB port). Finally, a lot of function libraries are also provided for the use of common I/Os: logical I/O, ADC converters, generation of PWM signals, operation of TWI/I2C busses, LCD displays ...

The objective of the Microcontrollers course is not only to know how to use the Arduino UNO board. It is above all an opportunity to tackle low-level programming problems (the binary value of the manipulated variables is very important) and to learn how to use the C language for this low-level programming, in particular by knowing how to manage registers/variables "at bit level". So, when you're complicating the task, when an Arduino function exists, tell yourself that it's intentional.

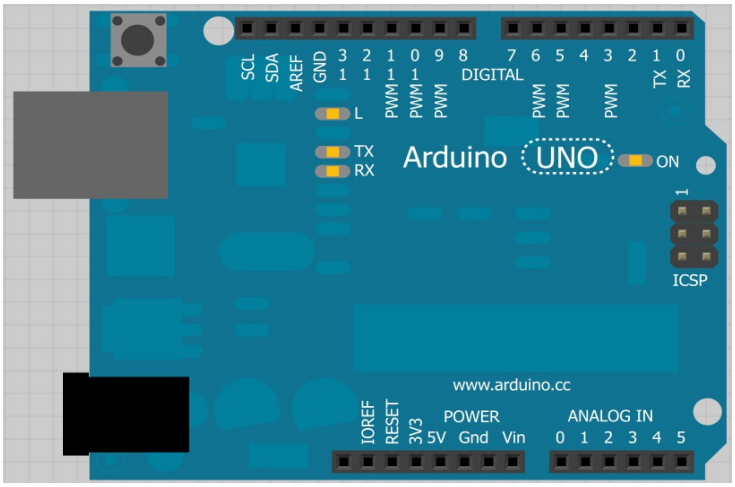
The purpose of this document is to highlight some technical information concerning the operation of integrated peripherals, especially when you don't use the "turnkey" functions of ARDUINO, in order to understand how it works!

# 2 Arduino UNO schematic

The pins of the microcontroller are connected to connectors according to the diagram below.



Viewed from above, the picture provides the following information:




(connector) Numbers 0 to 7 <-> Pins PD0 to PD7 (microcontroller)
(connector) Numbers 8 to 13 <-> Pins PB0 to PB5 (microcontroller)
(connector) ANALOG IN 0 to 5 <-> Pins PC0 to PC5 (microcontroller)

**WARNING**: with the Arduino functions (pinMode, digitalRead, digitalWrite ...), the signals are marked according to the numbering of the connectors (left part). When programming at low level, however, the names of the registers/pins of the microcontroller are used (right-hand side).

```
digitalWrite(10,HIGH);  //Arduino sets output PB2 of the micro. to HIGH.
analogRead(1);   //Arduino reads the analog input from PC1
```

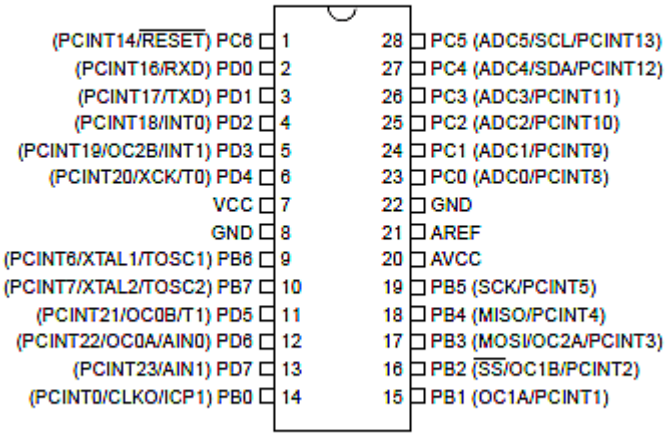
## 3 Microcontroller ATMEL ATMega328

The microcontroller of the Arduino UNO card is an **ATMega328**. It is an ATMEL microcontroller of the AVR 8bits family. The main features are :

**FLASH** = program memory 32Ko
**SRAM** = data 2Ko
**EEPROM** = data (persistent) 1Ko

**Digital I/O** = 3 ports PortB, PortC, PortD (23 I/O pins)

**Timers/Counters** : Timer0 et Timer2 (8 bits), Timer1 (16bits)
Each timer can be used to generate PWM signals. (6 pins OCxA/OCxB)



**Multi-function pins**: all pins have several different functions, chosen by programming. They therefore have several names on the pinout.

For example, pins PB1, PB2, PB3, PD3, PD5, PD6 can be used as PWM (Pulse Width Modulation) outputs, i.e. outputs that will act as analog outputs. They correspond to the pins of connectors 3,5,6,9,10 and 11. The other role is linked to the timers and these pins are then called OCxA or OcxB in the documentation. They are the same pins, but for a different function.

**PWM** = 6 pins **OC0A(PD6)**, **OC0B(PD5)**, **0C1A(PB1)**, **OC1B(PB2)**, **OC2A(PB3)**, **OC2B(PD3)**

Pins from PORTC can be converted with a ADC device (Analog to Digital Converter).

**Analog to Digital Converter** (10bits) = 6 multiplexed inputs **ADC0(PC0)** à **ADC5(PC5)**

**I2C bus** (aka TWI Two Wire Interface) = pins **SDA(PC5)/SCL(PC4).**

**Serial port (USART)** = transmit receive **TXD(PD1)/RXD(PD0)**

**Analog Comparator** = pins AIN0(PD6) and AIN1 (PD7) can trigger an interrupt

**Watchdog Timer.**

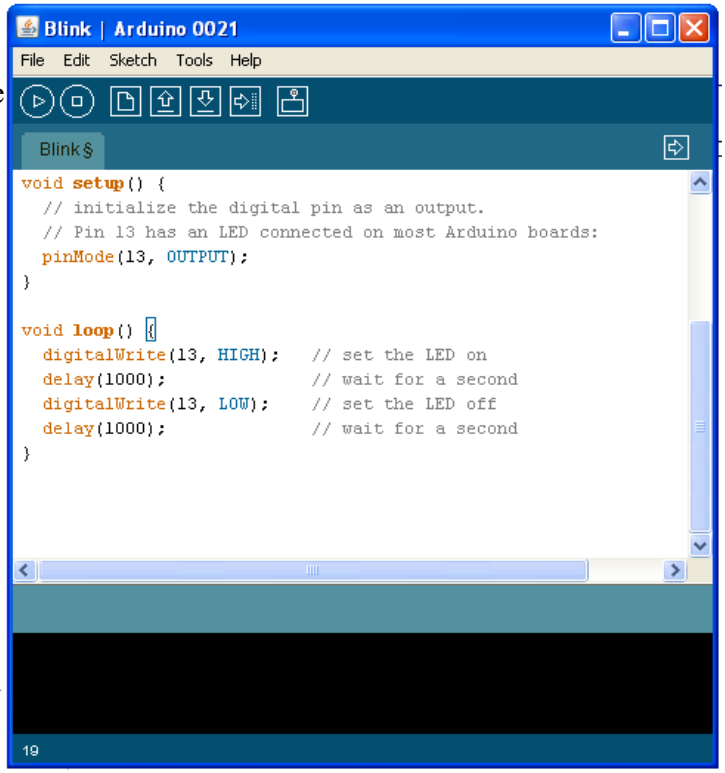
**INTERRUPTS (24 possible causes (**cf interrupt vectors)**) :** in summary

- Interrupts for inputs **INT0 (PD2) and INT1 (PD3)**
- Interrupts on Pin Change **PCINT0** to **PCINT23**
- Interrupts for Timers 0, 1 et 2 (different settings)
- Interrupt for the Analag Comparator
- Interrupt for **ADC**
- Interrupt for the serial port **USART**
- Interrupt for the **TWI** (**I2C**) bus

# 4 Programming in the Arduino's IDE

## *4.1 Introduction*

ARDUINO provides a development environment (IDE) with a source editor, where compiling and loading operations in the microcontroller memory are reduced to clicks on buttons in the IDE (very simple).




The communication between the PC and the card is done via the USB port, by installing a suitable driver on the PC (supplied by ARDUINO).

Structure of an ARDUINO project

The tool requires the application to be structured in a specific way. The compiler used is AVR GCC (C/C++ compiler for AVR processor).

The main program (main function) is imposed and described below.

The customized parts are

- **the setup() function:** initializations (timers, interrupts...)
- **the loop() function** : indefinitely re-executed

```
// THE ARDUINO MAIN PROG
#include <WProgram.h>

int main(void)
{
     init();   // initialisations for delays, PWM ...

     setup();
     for (;;)  loop();   // indefinitely repeated
     return 0;
}
```

---

**AN ARDUINO PROGRAM = 1 setup() function + 1 loop() function**

### *4.2 Langage C pour ARDUINO UNO*

Variables Types/size :

**boolean** : true/false (8 bits)
**char** = 8bits signed integer 8 [-128,+127]
**byte** / **unsigned char** : 8 bits unsigned integer [0,255]
**Note**: some constants are predefined and allow us to write `byte b = B10010;`
**int** : 16 bits signed integers [-32768,+32767]
**word** / **unsigned int** : 16 bits unsigned integers [0,65535]
**long** : 32 bits signed integers
**unsigned long** : 32 bits unsigned integers
**float /double** : floating-point IEEE 32 bits (float and double = same size for Arduino)

```
//Blink (Arduino)
void setup()
{
   pinMode(13, OUTPUT);  //pin PB5 as output
}

void loop()
{
  digitalWrite(13,HIGH);  // set PB5
  delay(200);      //  200 ms
  digitalWrite(13,LOW);      // clear PB5
  delay(1000);     //  1 s
}
```

The simplest example, provided by ARDUINO, consists of flashing the LED (on the UNO board) connected to pin PB5 of the microcontroller, pin no. 13 on the board connectors.

The setup() function configures pin PB5 (connection n°13 on the board) as an output, using the Arduino pinMode() function. The loop() function then describes what will be repeated indefinitely: set PB5 to 1 for 200ms then set PB5 to 0 for 1s, and so on.

**Some functions from the API ARDUINO.**

Constants : **HIGH, LOW, INPUT, OUTPUT, INPUT_PULLUP**

**Digital I/O**

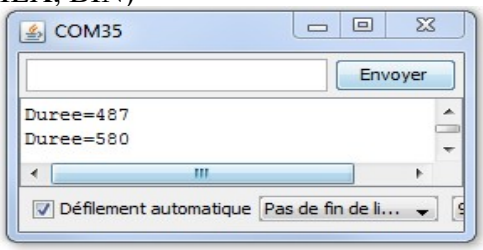
**pinMode(pin,mode) :** pin, mode = INPUT/OUTPUT/INPUT_PULLUP
**digitalWrite(pin,value) :** pin, value= HIGH/LOW
**int digitalRead(pin) :** pin, returns the value

**Timers**

**delay(unsigned long ms) :** ms milliseconds (ms encoded with 32 bits)
**delayMicroseconds(unsigned int ms)** : ms microseconds (ms 16 bits)
**unsigned long micros() :** time in microseconds from the startup. Reset every 70mn.
**unsigned long millis() :** time in milliseconds from the startup. Reset every 50 days.

**Serial communication : sends/receives** data : allows the Arduino card to communicate with the connected computer

**Serial.begin**(9600);  // setting : speed = 9600 bits/s
**Serial.println**(v,f);  // sends v with format f (DEC, HEX, BIN)




```
//Example :
// prints the duration (millis) when the signal
// on PC5 is equal to 0. This duration is sent
// to the computer via USB and can be displayed
// with the serial monitor

void setup()
{
   Serial.begin(9600);         // UART microC  9600 bauds
   pinMode(A5,INPUT_PULLUP);   // PC5 as input (with pull-up)
   pinMode(13,OUTPUT);         // PB5 (LED) as output
   digitalWrite(13,LOW);       // PB5 set to 0 (LED off)
}

void loop()
{
  unsigned long t1,t2,duree; // local variables
  while(digitalRead(A5)==1);
  // PC5 goes from 1->0  (falling edge)
  t1=millis();
  digitalWrite(13,HIGH);    // sets LED ON
  while(digitalRead(A5)==0);
  // PC5 0->1  (rising edge)
  t2=millis();
  digitalWrite(13,LOW);     // resets LED
  duree = t2-t1;
  Serial.print("Duree="); //duration=
  Serial.println(duree,DEC);  // sends the duration
}
```

**Use of ATMega328 registers in programs**

If you want to control the Arduino peripherals at the lowest level, i.e. without using the - yet very nice - Arduino functions, you have to read/write to internal registers of the microcontroller. These registers are detailed, for some devices, in the rest of this document. Note that the complete technical documentation of this microcontroller is several hundred pages long. This is therefore a very partial presentation.

For example, making the LED on the Arduino board (PB5) flash, without using the ARDUINO functions, requires access to the I/O port configuration registers (see section 5.2). For pin PB5, the registers involved are DDRB, PORTB and PINB. In the C program, the registers are referred to by their name in UPPERCASE.

**Low-level control of PB5.**

```
// Example BLINK: pin control without Arduino's functions

//    LED <- PB5 : LED ON when PB5=1

void setup() {
      // setting : PB5 as output (see section 5.2)
  DDRB |= 0x20;   // DDRB.5 <- 1
      // or DDRB|=B100000;  // B100000 = 0x20
      // ou DDRB|=32;        // 0x20 = 32
  PORTB &= 0xDF;  // PORTB.5 <- 0
      // or DDRB&= ~0x20;  // 0xDF is the complement to 0x20
}

void loop() {
  PORTB |= 0x20;  // PORTB.5 <- 1
  delay(200); // 200 ms
  PORTB &= 0xDF;; // PORTB.5 <- 0
  delay(1000); // 1s (only Arduino's function used here)
}
```

**Remarks** :

The C language examples found on the internet use sometimes confusing writing "styles". Especially when it comes to managing I/O ports, one often has to perform logical operations (&,|,~,^) to set bits to 0, 1 or invert bits, i.e. to modify or read one or more bits of an 8-bit register.

Confusing example:

```
PORTB &= ~(1<<PORTB5);  // resets bit 5 of PORTB
```

*Explanation* : PORTB5 is a constant equal to 5.

`1<<PORTB5` means "shift of (0000 0001)b (5 bits towards the left)"

Therefore, `(1<<PORTB5)` is equal to (0010 0000)b

`Then ~(1<<PORTB5)` is the complement say (1101 1111)b

Finally, `PORTB &= ~(1<<PORTB5)` means `PORTB = PORTB & ~(1<<PORTB5)`

The logical AND operation leaves all PORTB bits unchanged except bit 5 which is set to 0 (due to the bit at 0 of the binary value (1101 1111)b = ~(1<<PORTB5)).

In short, the following expressions achieve the same treatment

```
PORTB &= ~(1<<PORTB5);
PORTB &= ~0x20;            //  (20)h=(100000)b
PORTB &= 0xDF;             //  (DF)h is the complement of (20)h
PORTB &= ~B00100000;       //  B00100000 (constant) is equal to 0x20
PORTB &= B11011111;        //  B11011111 (constant)
```

# 5 Internal structure of ATMega328 (excerpts from ATMEL documentations)

The use of integrated peripherals (digital inputs and outputs, timers, ...) is based on the use (read/write) of internal registers. These registers, mainly 8 bits, are described by an UPPERCASE name in C programs. This section provides some important details about the internal registers of the ATMega328 microcontroller involved in the operation of peripherals. Some parts are excerpts from the Atmel documentation.

For the complete documentation (442p): search with keywords ATMega328 datasheet

---

**Notation :** thereafter, for a register named R, the notation R.n designates the nth bit of register R. Be careful, it is only a notation. The C compiler cannot exploit this notation.

Ex: PORTB.5 means "bit number 5 of the register called PORTB".

---

```
PORTB.5=1;        // KO, not a C language expression !
PORTB |= 0x20;    // OK: bit PORTB.5 set to 1
```

### *5.1 Status Register (SREG)*

The **SREG** register contains flags and the general interrupt authorization bit. The bits in this register are : Z (Zero), C (Carry), S (Sign) ... The general interrupt enable bit is bit I (SREG.7).

**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 | |

- **Bit 7 – I: Global Interrupt Enable**

The Global Interrupt Enable bit must be set for the interrupts to be enabled. The individual interrupt enable control is then performed in separate control registers. If the Global Interrupt Enable Register is cleared, none of the interrupts are enabled independent of the individual interrupt enable settings. The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts. The I-bit can also be set and cleared by the application with the SEI and CLI instructions, as described in the instruction set reference.

**Note** : in C language, bit I is modified with functions `sei()` (set IT) `cli()` (Clear IT)

### *5.2 Digital I/O*

The microcontrollers have logical input/output pins, just like on a PLC. To set the state of an output to 0 or 1, or to read the state of an input, internal registers described below must be used.

The inputs/outputs are divided into 3 groups of pins called ports. Port B groups the pins marked PBx, port C the PCx pins and port D the PDx pins (see pinout). Each port is operated by means of 3 registers.

Ex: PORTB, DDRB et PINB registers for controlling pins PB0 à PB7

**PORTx** = for WRITING output values
**DDRx** = Determines the DIRECTION of each pin of the port (1-Output 0-Input).
**PINx** = for READING the input value

**DDRx = Direction of port x**

Ex**:** if bit DDRB.6 is equal to 1 then pin PB6 is a logical output

```
// Example ports
void setup()
{
  DDRB |= 0x40;   // DDRB.6 <- 1  <-> PB6 as output
  DDRD &= ~0x08;  // DDRD.3 <- 0  <-> PD3 as input
}
```

**PORTx for writing digital outputs:** if a pin is configured as an output (DDRx.n=1) then writing the PORTx.n bit defines the state of the output (0 or 1).

```
// Example : writing an output
void setup()
{
  DDRB |= 0x40;    // PB6 as an output
  PORTB &= ~0x40;  // PORTB.6 <-0 (set PB6 to 0)
}
```

**PINx for reading logical inputs** :if a pin is configured as an input (DDRx.n=0) then reading the PINx.n bit allows knowing the status of the input.

```
// Example : reading an input
void setup()
{
  DDRD &= ~0x08;  // PD3 as input
}
void loop()
{
      if((PIND&0x08)!=0)  // if PD3 equals 1
      {
            ...
      }
}
```

In the documentation, the registers involved are described below (example for pins PB0 to PB7).

**MCUCR – MCU Control Register**

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

- **Bit 4 – PUD: Pull-up Disable**

When this bit is written to one, the pull-ups in the I/O ports are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-ups ({DDxn, PORTxn} = 0b01). See "Configuring the Pin" on page 76 for more details about this feature.

**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 | 0 | 0 | 0 | |

**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 | |

**PINB – The Port B Input Pins Address**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x03 (0x23) | PINB7 | PINB6 | PINB5 | PINB4 | PINB3 | PINB2 | PINB1 | PINB0 | PINB |
| Read/Write | R | R | R | R | R | R | R | R | |
| Initial Value | N/A | N/A | N/A | N/A | N/A | N/A | N/A | N/A | |

### Activation of internal pull-up resistors (important)

In MOS technology, a not-connected input has an undetermined state. Also, when you want to use push buttons/switches, you connect them in such a way as to bring the input back to 0 when you close the contact. Conversely, when the contact is open, the state of the input must be brought to 1 by pull-up resistors. These internal resistors are activated (or not) by programming:

**PORTx**.n=**1** AND **DDRx**.n=**0** ↔ internal **pull-up activated**
**PORTx**.n=0 OR **DDRx**.n=1 ↔ internal pull-up of Pxn not activated

```
// Input PIN with internal PULL-UP resistor
void setup()
{
  DDRD  &= ~0x02;   // DDRD.1 <-0  (PD1 as input)
  PORTD |= 0x02;    // PORTD.1<-1  (internal pull-up on PD1)
}
```

# 6 Interrupts for ATMega328 (Arduino UNO)

**Interrupt (IT)** = suspension of the program to carry out a particular treatment. But this is different from a function or a subprogram: it is not explicitly called by the program. You don't write the call of this specific task. It is the processor that, following the detection of a particular cause, triggers the interrupt processing. The function related to interrupt processing is called an interrupt function or interrupt service routine (ISR).

Peripherals can lead to interruptions. This mechanism ensures very short response times between the cause of the interrupt and its processing.

**Important**: ISR calls are inserted asychronically (it is not known in advance when they will be called) into the program execution. For example, an Arduino program is interrupted every millisecond by an ISR linked to Timer 0. This ISR Timer 0 routine updates the time variables used by the delay() millis() etc. functions. An Arduino program is therefore cyclically suspended (interrupted) for the ISR Timer 0, which slows down its execution by about 6%.

Below is the interrupt vector, i.e. all sources (possible causes) on ATMEGA238.

### 11.4 Interrupt Vectors in ATmega328P

**Table 11-6.** Reset and Interrupt Vectors in ATmega328P

| VectorNo. | Program Address(2) | Source | Interrupt Definition |
|---|---|---|---|
| 1 | 0x0000(1) | RESET | External Pin, Power-on Reset, Brown-out Reset and Watchdog System Reset |
| 2 | 0x0002 | INT0 | External Interrupt Request 0 |
| 3 | 0x0004 | INT1 | External Interrupt Request 1 |
| 4 | 0x0006 | PCINT0 | Pin Change Interrupt Request 0 |
| 5 | 0x0008 | PCINT1 | Pin Change Interrupt Request 1 |
| 6 | 0x000A | PCINT2 | Pin Change Interrupt Request 2 |
| 7 | 0x000C | WDT | Watchdog Time-out Interrupt |
| 8 | 0x000E | TIMER2 COMPA | Timer/Counter2 Compare Match A |
| 9 | 0x0010 | TIMER2 COMPB | Timer/Counter2 Compare Match B |
| 10 | 0x0012 | TIMER2 OVF | Timer/Counter2 Overflow |
| 11 | 0x0014 | TIMER1 CAPT | Timer/Counter1 Capture Event |
| 12 | 0x0016 | TIMER1 COMPA | Timer/Counter1 Compare Match A |
| 13 | 0x0018 | TIMER1 COMPB | Timer/Coutner1 Compare Match B |
| 14 | 0x001A | TIMER1 OVF | Timer/Counter1 Overflow |
| 15 | 0x001C | TIMER0 COMPA | Timer/Counter0 Compare Match A |
| 16 | 0x001E | TIMER0 COMPB | Timer/Counter0 Compare Match B |
| 17 | 0x0020 | TIMER0 OVF | Timer/Counter0 Overflow |
| 18 | 0x0022 | SPI, STC | SPI Serial Transfer Complete |
| 19 | 0x0024 | USART, RX | USART Rx Complete |
| 20 | 0x0026 | USART, UDRE | USART, Data Register Empty |
| 21 | 0x0028 | USART, TX | USART, Tx Complete |
| 22 | 0x002A | ADC | ADC Conversion Complete |

**Table 11-1.** Reset and Interrupt Vectors in ATmega48PA (Continued)

| Vector No. | Program Address | Source | Interrupt Definition |
|---|---|---|---|
| 24 | 0x017 | ANALOG COMP | Analog Comparator |
| 25 | 0x018 | TWI | 2-wire Serial Interface |
| 26 | 0x019 | SPM READY | Store Program Memory Ready |

### *6.1 External Interrupts (pins PD2 and PD3)*

These are interrupts where the causes are related to levels or changes of state of pins PD2 (INT0) or PD3 (INT1) of the microcontroller. Please note the name INT0/INT1 in reference to this alternative function of pins PD2/PD3. For this external interrupt role, the pins must be configured as inputs (see 5.2 DIGITAL I/O).

Pins INT0 (PD2)/INT1(PD3): Configurable to trigger interrupts (no. 2 and 3 in the vector or INT0_vect/INT1_vect). The possible causes (selected by programming) are

detection of a level 0 on the input (low level),
falling/rising edge
any logical change

**Choosing the cause of interruption:** what is written in the EICRA register

the EICRA.1-EICRA.0 bits for INT0 (see table 12.2)
EICRA.3-EICRA.2 bits for INT1 (same table for ISC11-ISC10)

**EICRA – External Interrupt Control Register A**

The External Interrupt Control Register A contains control bits for interrupt sense control.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x69) | – | – | – | – | ISC11 | ISC10 | ISC01 | ISC00 | EICRA |
| Read/Write | R | R | R | R | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Table 12-2.** Interrupt 0 Sense Control

| ISC01 | ISC00 | Description |
|---|---|---|
| 0 | 0 | The low level of INT0 generates an interrupt request. |
| 0 | 1 | Any logical change on INT0 generates an interrupt request. |
| 1 | 0 | The falling edge of INT0 generates an interrupt request. |
| 1 | 1 | The rising edge of INT0 generates an interrupt request. |

**Enable interrupts INT0/INT1 = bit SREG.7 to 1 and set EIMSK.0/EIMSK.1 to 1**

**EIMSK – External Interrupt Mask Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x1D (0x3D) | – | – | – | – | – | – | INT1 | INT0 | EIMSK |
| Read/Write | R | R | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

```
// INT0/INT1
void setup()
{
  cli();              // no IT
  EICRA &= 0xF0;      // reset bits EICRA.3-EICRA.0
  EICRA |=0x09;       // ISC11=1 ISC10=0  (falling edge INT1)
                      // ISC01=0 ISC00=1 (pin change on INT0)
  EIMSK |=0x03;       // INT0/INT1 enabled
  sei();              // IT allowed
}
```

**Detail : internal Flags** = when the ISR starts, a flag in EIFR is set

**EIFR – External Interrupt Flag Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x1C (0x3C) | – | – | – | – | – | – | INTF1 | INTF0 | EIFR |
| Read/Write | R | R | R | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

## Interrupts for Arduino (interrupt vector)

```
#define INT0_vect          _VECTOR(1)   /* External Interrupt Request 0 */
#define INT1_vect          _VECTOR(2)   /* External Interrupt Request 1 */
#define PCINT0_vect        _VECTOR(3)   /* Pin Change Interrupt Request 0 */
#define PCINT1_vect        _VECTOR(4)   /* Pin Change Interrupt Request 1 */
#define PCINT2_vect        _VECTOR(5)   /* Pin Change Interrupt Request 2 */
#define WDT_vect           _VECTOR(6)   /* Watchdog Time-out Interrupt */
#define TIMER2_COMPA_vect  _VECTOR(7)   /* Timer/Counter2 Compare Match A */
#define TIMER2_COMPB_vect  _VECTOR(8)   /* Timer/Counter2 Compare Match B */
#define TIMER2_OVF_vect    _VECTOR(9)   /* Timer/Counter2 Overflow */
#define TIMER1_CAPT_vect   _VECTOR(10)  /* Timer/Counter1 Capture Event */
#define TIMER1_COMPA_vect  _VECTOR(11)  /* Timer/Counter1 Compare Match A */
#define TIMER1_COMPB_vect  _VECTOR(12)  /* Timer/Counter1 Compare Match B */
#define TIMER1_OVF_vect    _VECTOR(13)  /* Timer/Counter1 Overflow */
#define TIMER0_COMPA_vect  _VECTOR(14)  /* TimerCounter0 Compare Match A */
#define TIMER0_COMPB_vect  _VECTOR(15)  /* TimerCounter0 Compare Match B */
#define TIMER0_OVF_vect    _VECTOR(16)  /* Timer/Couner0 Overflow */
#define SPI_STC_vect       _VECTOR(17)  /* SPI Serial Transfer Complete */
#define USART_RX_vect      _VECTOR(18)  /* USART Rx Complete */
#define USART_UDRE_vect    _VECTOR(19)  /* USART, Data Register Empty */
#define USART_TX_vect      _VECTOR(20)  /* USART Tx Complete */
#define ADC_vect           _VECTOR(21)  /* ADC Conversion Complete */
#define EE_READY_vect      _VECTOR(22)  /* EEPROM Ready */
#define ANALOG_COMP_vect   _VECTOR(23)  /* Analog Comparator */
#define TWI_vect           _VECTOR(24)  /* Two-wire Serial Interface */
```

## Example :

**`ISR(name) { ... }`** = **ISR for a source in this list**

```
// Example : External Interrupt 0
ISR(INT0_vect)  // ISR INT0
{
      PORTB ^= 0x20;  // toggle PORTB.5
}
void setup(){  //
      cli();
      Serial.begin(9600);
      DDRB |= 0x20;          // PB5 as an output
      PORTB &= ~0x20;        // PORTB.5 <-0
      DDRD &= ~0x04;         // PD2 as an input
      PORTD |= 0x04;         // PORTD.2=1 <-> pull-up
      EICRA = 0x02;    //  ↓on INT0 (table 12-2)
      EIMSK |= 1;      // INT0 enabled
      sei();
}
int cpt=0;  // global variable
void loop() { // this function can be suspended for the ISR
      Serial.println(cpt,DEC);
      cpt++;
      delay(1000);
}
```

For the previous program, a push-button is connected between PD2(/INT0) and ground GND. When the push button is pressed, the level of input PD2 is set to 0. Here, the internal pull-up resistor is used to set the level to 1 when the button is released.

**Principle**: Each time the pushbutton changes the level on input INT0(PD2) from 1 to 0, the interrupt function associated with INT0 is executed. This action has the effect of reversing the state of the LED and returning to the main program. It is important to understand that the interrupt only lasts a few microseconds. Apart from these interrupts, the main program (loop() function) sends the value of the cpt variable to the serial port every second.

**Note**: this way of decoupling the processing of the push button from that of the main program is similar to task parallelization.

**[Technical point]** Set EIFR register flag to 0: EIFR register bits indicate (flags) that an INT0/INT1 interrupt request is pending: a flag of 1 in this register means that the cause of IT has been detected but the ISR routine is not yet executed. If you want to cancel an int. request (before it is executed), you must reset these flags to 0. Strange: To cancel a request (clear flag), you must write 1 (not 0) in the EIFR register for the flag concerned.

```
EIFR|=1; // cancel IT request INT0
EIFR|=2; // cancel IT request INT1
```

### *6.2 "Pin Change" interrupts (possible for all logical pins)*

Pin Change interrupts (PCINT0_vect, PCINT1_vect, PCINT2_vect) are used to associate a task (ISR) with a cause which is the change of state of an input. Each edge, rising or falling, leads to an interruption. This mechanism can be used for any binary input. On the other hand, the same ISR is used for a set of pins. It is therefore sometimes difficult to detect which specific input is causing the interrupt, compared to INT0/INT1 where it is known exactly what the cause of the interrupt is.

**Pins PCINT0 to PCINT23** (alternative name for PBx, PCx and PDx): Configurable to trigger interrupts (#4, #5 and #6 or PCINT0_vect, PCINT1_vect, PCINT2_vect) following pin changes (configured as DDRx.n=1 input). The pins are separated into 3 subgroups, there is one interrupt source per subgroup, and for each pin the "Pin Change Interrupt" system can be activated or not.

Pin Change 3 groups of pins (linked to ports B, C and D)

PCINT0 - PCINT7 <-> pins PB0 to PB7 group linked to IT PCINT0_vect
PCINT8 - PCINT15 <-> pins PC0 to PC7 group related to IT PCINT1_vect
PCINT16 - PCINT23 <-> pins PD0 to PD7 group linked to IT PCINT2_vect

Registers PCMSK0, PCMSK1 and PCMSK2 control, for each of these groups (i.e. for each port B, C D), which pin(s) can lead (or not) to a "pin change" type interrupt.

Enable interrupts PCINT0 to PCINT23 if bit SREG.7=1 and set PCIEx to 1

PCICR Registry: IT Pin Change activation for a group

PCICR.0: Activation of IT Pin Change for the pins of port B (PB0 to PB7)
PCICR.1: Activation of IT Pin Change for the pins of port C (PC0 to PC6)
PCICR.2: Activation of IT Pin Change for the pins of port D (PD0 to PD7)

**PCICR – Pin Change Interrupt Control Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x68) | – | – | – | – | – | PCIE2 | PCIE1 | PCIE0 | PCICR |
| Read/Write | R | R | R | R | R | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

Activation within a group: the PCMSKx register determines which pins in the group are taken into account for the "pin change" interrupt

**PCMSK0 – Pin Change Mask Register 0**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x6B) | PCINT7 | PCINT6 | PCINT5 | PCINT4 | PCINT3 | PCINT2 | PCINT1 | PCINT0 | PCMSK0 |
| 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 | |

**PCMSK1 – Pin Change Mask Register 1**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x6C) | – | PCINT14 | PCINT13 | PCINT12 | PCINT11 | PCINT10 | PCINT9 | PCINT8 | PCMSK1 |
| 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 | |

**PCMSK2 – Pin Change Mask Register 2**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x6D) | PCINT23 | PCINT22 | PCINT21 | PCINT20 | PCINT19 | PCINT18 | PCINT17 | PCINT16 | PCMSK2 |
| 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 | |

**Example : IT Pin Change for Port B and Port D**

```
// Pin Change Interrupt
void setup()
{
  cli();
  PCICR |= 0x05;    // Pin Change enabled port D and port B
  DDRB&=~0x03;      // PB1/PB0 inputs
  DDRD&=~0x0A;      // PD7/PD5 inputs
  PCMSK0=0x03;      // Pin Change enabled on PB0/PB1
  PCMSK2=0xA0;      // Pin Change enabled on PD7/PD5
  sei();
}

ISR(PCINT0_vect){   ...   }  // called for a pinchange on PB1 or PB0

ISR(PCINT2_vect){   ...   }  // called for a pinchange on PD7 or PD5
```

**Flags** for ISR "Pin Change"

**PCIFR – Pin Change Interrupt Flag Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x1B (0x3B) | – | – | – | – | – | PCIF2 | PCIF1 | PCIF0 | PCIFR |
| Read/Write | R | R | R | R | R | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Other example "PIN CHANGE"**

```
//Pin Change for PC4,PC5,PD2 and PD3

void setup()
{
  Serial.begin(9600);
  cli();
  PCICR |= 0x06;    // Pin Change enabled port D/port C
  PCMSK2=0x0C;      // Pin Change PD3/PD2
  PCMSK1=0x30;      // Pin Change PC5/PC4
  DDRD&=~0x0C;      // PD2/PD3 inputs
  PORTD|=0x0C;      // pull-up for PD2 PD3
  DDRC&=~0x30;      // PC4 PC5 inputs
  PORTC|=0x30;      // pull-up for PC4 PC5
  sei();
}

volatile int cpt1=0;
volatile int cpt2=0;

ISR(PCINT1_vect)       // Pin change port C
{
    cpt1++;
}

ISR(PCINT2_vect)       // Pin Change port D
{
    cpt2++;
}

void loop()
{
    delay(2000);
    Serial.print("cpt1=");
    Serial.println(cpt1,DEC);
    Serial.print("cpt2=");
    Serial.println(cpt2,DEC);
}
```

**Note**: On AVR, by default, an interrupt function cannot itself be interrupted. The CPU prevents ISR from being suspended.

### *6.3 Timers interrupts*

Built-in timers can trigger interruptions. A complete section on configuring the built-in timers and operating the associated interrupts is provided.

## 7. Timers/Counters on ATMega328

The ATMega328 microcontroller has several internal timer/counter modules (Timers), some with 8-bit count registers and others with 16-bit count registers. In all cases, each counting event leads to a change in the count register (+1). The count event can be a "tick" of the microcontroller clock, which is equivalent to measuring the passage of time. The count event can also be an edge on an input pin of the microcontroller (pins T0 and T1 can be used as count input).

**Timer function:** When counting "ticks" of the clock that clocks the microcontroller, the time elapsed is measured. The Timer/Counter modules provide this function. It is also possible to count the ticks of a lower frequency signal obtained by dividing the clock frequency by a prescaler.

**Note**: on the Arduino UNO board, the clock is at 16MHz, or 16,000,000 clock cycles per second, or 16 clock cycles per microsecond. These are the cycles that are counted as a timer function. 16000000 cycles = one second.

**Counter function**: when counting edges on a counter input (pins T0 or T1), the "counter" function of the module (not studied here) is used.

The choice between timer function (with prescaler or not) and counter function is made by configuring registers dedicated to the management of Timer/Counter modules. You will see, it's technical.

**Generating periodic signals**: the Timer/Counter modules are quite complex and each of these modules can generate two PWM (Pulse Width Modulation) signals whose duty cycle can be easily modified. In this case, use the Arduino analogWrite() function which generates a PWM signal. This PWM signal is only managed on the outputs linked to integrated Timers i.e. PD6,PD5,PD3, PB1,PB2 and PB3.

Arduino analogWrite( ) = generates a PWM signal managed by a Timer module.

**Note**: timers are complex embedded devices (about 70 pages of the ATMega datasheet). Only a simplified view is provided here.

### *7.1 Timer/Counter 0 (8 bits)*

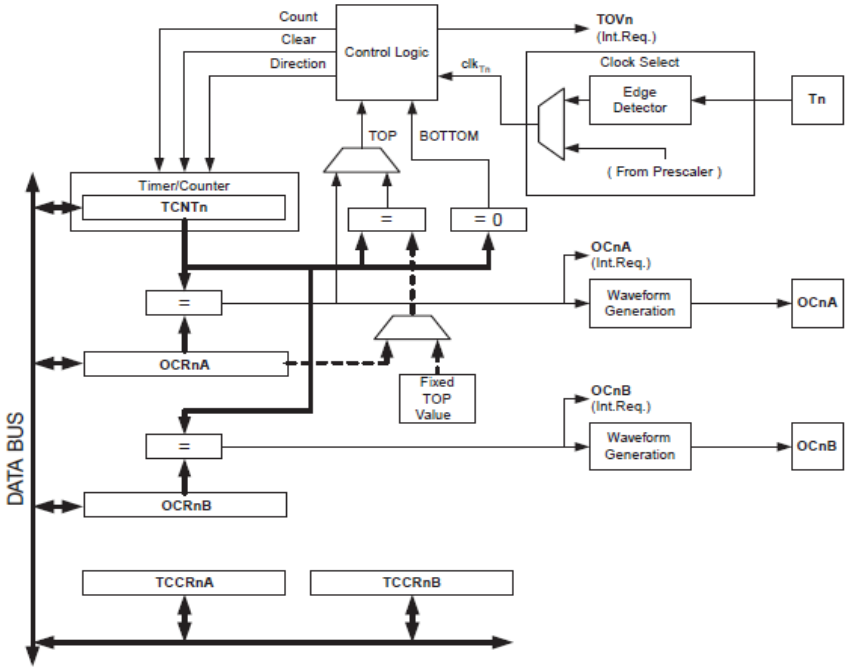
It is a Timer/Counter module with 8-bit count register. When using the Arduino IDE, timer 0 (and the associated interrupt) is implicitly used by the time management functions (delay(),millis() ...). This Timer/Counter module cannot therefore be used directly with the ARDUINO Uno board. Unless you accept to do without the Arduino time management functions.

*7.2 Timer/Counter 2 (comptage 8 bits)*

It is a Timer/Counter module with 8-bit count register. The general structure of the Timer/Counter 2 module is shown in the following diagram. The counter register is TCNT2 (8-bit register).

**Important points (Timer 2) :**

detection and IT overflow (TIMER2_OVF_vect)
count input = clock signal with prescaler or not
possibility to compare TCNT2 with two OCR2A/OCR2B comparison registers
the equality TCTN2=OCR2A can trigger an IT (TIMER2_COMPA_vect)
the equality TCTN2=OCR2B can trigger an IT (TIMER2_COMPB_vect)
Pins OC2A(PB3) and OC2B (PD3) can be activated by Timer/Counter 2 for generating periodic signals (PWM).




## Registers Timer/Counter 2

### TCNT2 – Timer/Counter Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0xB2) | TCNT2[7:0] | | | | | | | | TCNT2 |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

### OCR2A – Output Compare Register A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0xB3) | OCR2A[7:0] | | | | | | | | OCR2A |
| 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 | |

The Output Compare Register A contains an 8-bit value that is continuously compared with the counter value (TCNT2). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC2A pin.

### TCCR2A – Timer/Counter Control Register A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0xB0) | COM2A1 | COM2A0 | COM2B1 | COM2B0 | – | – | WGM21 | WGM20 | TCCR2A |
| Read/Write | R/W | R/W | R/W | R/W | R | R | R/W | R/W | |

**TCCR2B – Timer/Counter Control Register B**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0xB1) | FOC2A | FOC2B | – | – | WGM22 | CS22 | CS21 | CS20 | TCCR2B |
| Read/Write | W | W | R | R | R | R | R/W | R/W | |

**TIMSK2 – Timer/Counter2 Interrupt Mask Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x70) | – | – | – | – | – | OCIE2B | OCIE2A | TOIE2 | TIMSK2 |
| Read/Write | R | R | R | R | R | R/W | R/W | R/W | |

**TIFR2 – Timer/Counter2 Interrupt Flag Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x17 (0x37) | – | – | – | – | – | OCF2B | OCF2A | TOV2 | TIFR2 |
| Read/Write | R | R | R | R | R | R/W | R/W | R/W | |

## Operating Modes (Table 17-8) :

**Normal**: Register TCNT2 is incremented by 1 for each counting event. The register only returns to 0 after an overflow (0xFF to 0x00).

**CTC (Clear Timer on Compare)**: Register TCNT2 is incremented at each counting event AND is reset to 0 if TCNT2=OCR2A.

Other non-detailed modes, especially for PWM management.

The choice of the mode is made via the bits WGM22:20 (bits TCR2A and TCR2B).

**Table 17-8.** Waveform Generation Mode Bit Description

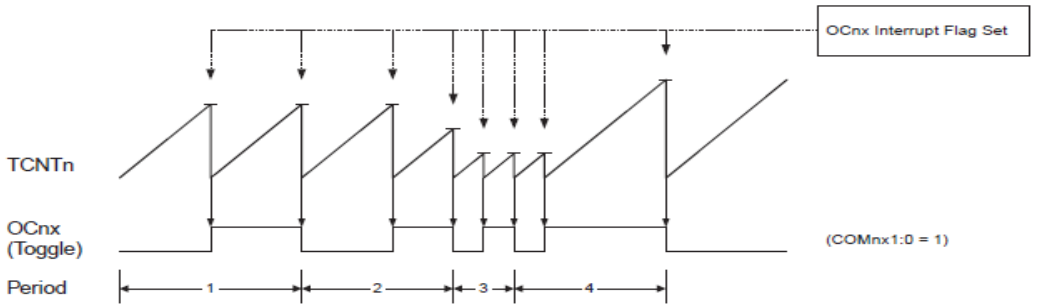
| Mode | WGM2 | WGM1 | WGM0 | Timer/Counter Mode of Operation | TOP | Update of OCRx at | TOV Flag Set on[1][2] |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | Normal | 0xFF | Immediate | MAX |
| 1 | 0 | 0 | 1 | PWM, Phase Correct | 0xFF | TOP | BOTTOM |
| 2 | 0 | 1 | 0 | CTC | OCRA | Immediate | MAX |
| 3 | 0 | 1 | 1 | Fast PWM | 0xFF | BOTTOM | MAX |
| 4 | 1 | 0 | 0 | Reserved | – | – | – |
| 5 | 1 | 0 | 1 | PWM, Phase Correct | OCRA | TOP | BOTTOM |
| 6 | 1 | 1 | 0 | Reserved | – | – | – |
| 7 | 1 | 1 | 1 | Fast PWM | OCRA | BOTTOM | TOP |

Notes: 1. MAX= 0xFF
2. BOTTOM= 0x00

### Clear Timer on Compare Match (CTC) Mode

In CTC mode (WGM22:0 = 2), register OCR2A sets the resolution. The counter TCTN2 is reset to zero after the match TCTN2=OCR2A. Register OCR2A defines the maximum value for the counter and thus its resolution.

**Figure 17-5.** CTC Mode, Timing Diagram




**Prescaler :** as a timer function, the counter register TCNT2 is incremented according to the clock cycles. The increment can be at each clock cycle (no prescaling) or at a lower frequency. Remember that the clock cycle is 1/16 microseconds. The prescaler indicates how many clock cycles are required for a TCNT2 increment.

## prescaler Timer 2 settings

**Table 17-9.** Clock Select Bit Description

| CS22 | CS21 | CS20 | Description |
|---|---|---|---|
| 0 | 0 | 0 | No clock source (Timer/Counter stopped). |
| 0 | 0 | 1 | $clk_{T2S}$/(No prescaling) |
| 0 | 1 | 0 | $clk_{T2S}$/8 (From prescaler) |
| 0 | 1 | 1 | $clk_{T2S}$/32 (From prescaler) |
| 1 | 0 | 0 | $clk_{T2S}$/64 (From prescaler) |
| 1 | 0 | 1 | $clk_{T2S}$/128 (From prescaler) |
| 1 | 1 | 0 | $clk_{T2S}$/256 (From prescaler) |
| 1 | 1 | 1 | $clk_{T2S}$/1024 (From prescaler) |

*7.3 Example Timer 2 with Interrupt*

**Enabling of interrupts on Timer 0 (3 sources n°14, 15 et 16)**

**TIMSK0 – Timer/Counter Interrupt Mask Register**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x6E) | – | – | – | – | – | OCIE0B | OCIE0A | TOIE0 | TIMSK0 |
| Read/Write | R | R | R | R | R | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

- **Bit 2 – OCIE0B: Timer/Counter Output Compare Match B Interrupt Enable**

When the OCIE0B bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter Compare Match B interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter occurs, i.e., when the OCF0B bit is set in the Timer/Counter Interrupt Flag Register – TIFR0.

- **Bit 1 – OCIE0A: Timer/Counter0 Output Compare Match A Interrupt Enable**

When the OCIE0A bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Compare Match A interrupt is enabled. The corresponding interrupt is executed if a Compare Match in Timer/Counter0 occurs, i.e., when the OCF0A bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

- **Bit 0 – TOIE0: Timer/Counter0 Overflow Interrupt Enable**

When the TOIE0 bit is written to one, and the I-bit in the Status Register is set, the Timer/Counter0 Overflow interrupt is enabled. The corresponding interrupt is executed if an overflow in Timer/Counter0 occurs, i.e., when the TOV0 bit is set in the Timer/Counter 0 Interrupt Flag Register – TIFR0.

**Note**: for Timers 1 and 2, the configurations are similar.

**Example**: It is desired that each overflow of Timer 0 should lead to an interruption.

SREG.7=1 (general bit for IT activation, without it no IT)
TIMSK0.0 (TOIE0)=1 (interrupt on timer 0 overflow)

For Timer 2, a TIMER2_OVF_vect interrupt can be triggered for each TCNT2 overflow (change from 0xFF to 0x00). This mechanism has to be activated

```
TIMSK2=0x01; // IT Timer2 Over Flow Active
```

**Provide IT routine:** ISR(TIMER2_OVF_vect){ ... } //ISR of the IT overflow Timer2 and set up Timer 2.

**Note** (volatile variables): When global variables are shared (read/write) between the main program and an ISR function, it is recommended to label them as **volatile**. Without this keyword, the compiler is likely to make code optimizations that would cause variable sharing to fail.

```
// ARDUINO UNO - IT Timer 2 Overflow

volatile unsigned char cpt=0; // counter of ISR executions

// ISR  n°9 = Timer 2 OverFlow
ISR(TIMER2_OVF_vect){
      cpt++;
      if(cpt==61){
            PORTB ^=0x20;
            cpt=0;
      }
}

void setup(){

      DDRB |= 0x20;    // PB5 as output
      PORTB &= ~0x20; // PORTB.5 <-0
      cli();
      // settings Timer 2
      TCCR2A=0;        // Normal
      TCCR2B=0x07;     // Prescaler 1024 (Clock/1024)
      TIMSK2=0x01;     // Timer2 Over Flow Enabled
      sei();
}

void loop() {    /* nothing */          }
```

Settings **Timer 2**

Mode 0 ( Normal) : WGM2=0 WGM1=0 WGM0=0  [TCCR2A=0]
Prescaler = 1024 : CS22=1 CS21=1 CS20=1  [TCCR2B=0x07=(111)b]

**Interrupts**

Interruption if Overflow = TIMSK2.0 =1

**Principle**: after the setup() function, the TCNT2 register (8bits) is incremented at each tick of the periodic clock/1024 signal. Each time register TCNT2 overflows, the overflow triggers interrupt n°10 called "Timer 2 Over Flow". Every 60 calls of this function, pin PB5 (LED) changes state. The LED therefore blinks.

How often does it blink?

Clock = 16MegaHz periodic signal (1600000 cycles per second)
Preset = 1024 -> TCNT2 increment frequency = (16/1024) MegaHz
Overflow frequency: TCNT2 only overflows every 256 increments
i.e. at the frequency of 16/(1024*256) MegaHZ ≈ 61 Hz
There is 1 Timer2 Over Flow interruption every 1/61 seconds.
It takes 61 Interruptions for the LED to change state.

The LED changes state (0->1 1->0) at intervals of about 1 second.

**Another example: timer2 in CTC mode and interrupts**

The CTC mode of Timer 2 is used here. Each time TCNT2=OCR2A, TCNT2 is reset to 0 and the equality triggers an interrupt. Autre exemple : timer2 en mode CTC et interruptions

```
// ARDUINO UNO - IT Timer 2
// Mode Clear Timer On Compare

volatile unsigned char cpt;

//ISR IT n°7 = Timer 2 COMPA
ISR(TIMER2_COMPA_vect){
      cpt++;
        if(cpt==40) PORTB|=0x20;
        if(cpt==50){
            PORTB &=~0x20;
            cpt=0;
         }
}

void setup(){

      DDRB |= 0x20;     // PB5 output
      PORTB &= ~0x20;   // PORTB.5 <-0

      // Settings Timer 2
      TCCR2A=0x02;     // Mode CTC (Clear Timer On Compare)
      OCR2A=156;       // comparison reg A = 156
      TCCR2B=0x07;     // Prescaler 1024 (Clock/1024)
      TIMSK2=0x02;     // IT when TCNT2=OCR2A
      sei();
}

void loop() {      /* */  }
```

**How often does it blink?**

Clock = 16MegaHz periodic signal (16 million ticks per second)
Preset = 1024 -> TCNT2 increment frequency = (16/1024) MegaHz
Overflow frequency :
TCNT2 only overflows every 156 increments (OCR2A value).
i.e. at the frequency of 16/(1024*156) MegaHZ ≈ 100 Hz
There is 1 Timer2 On Compare A interruption approximately every 1/100 seconds.

**The LED lights up 1/10 of a second and then goes out 4/10 of a second. It lights up briefly twice a second.**

**Timer2 configuration function in CTC mode**

The **SetTimer2CTC(CSB,period)** function provided below allows you to configure Timer 2 in CTC mode with a given period (on 8 bits) between two resets.

For example:

SetTimer2CTC(2,50); // prescaler = /8 period=50

In the example below :

presscaler /256 = TCNT2 incremented 62500 times per second
period 100 = the TIMER2_COMPA_vect interruption occurs 625 times per second

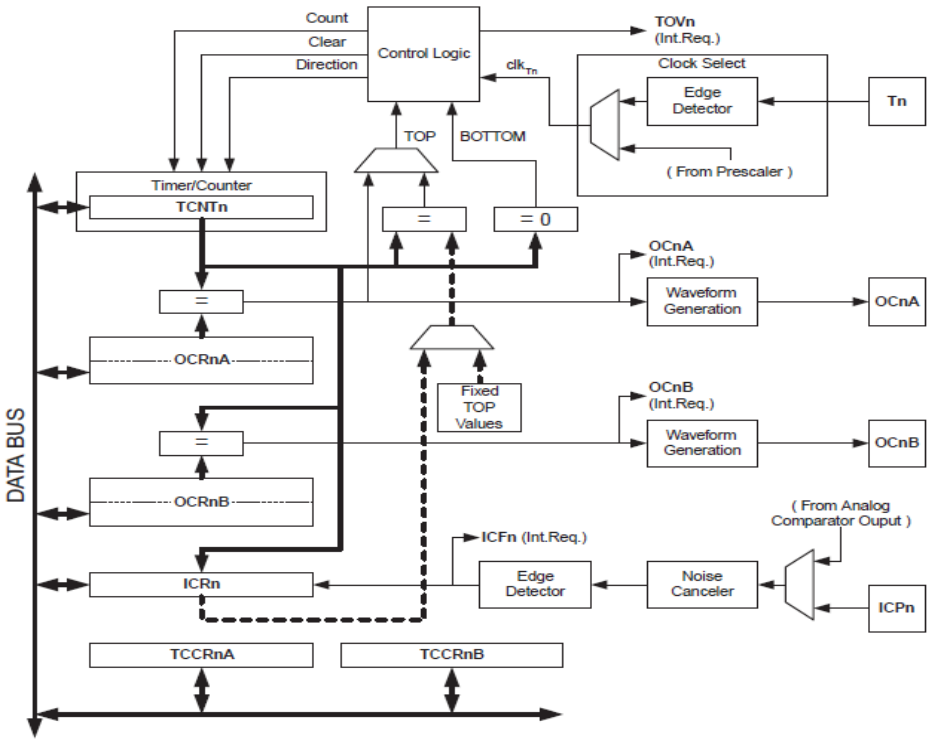
```
/* Config. Timer2 en Mode CTC
CSB(Clock Select)= 0(timer2 stop),1(=/1),2(=/8),3(=/32),
4(=/64),5(=/128),6 (=/256),7(=/1024)
period =
*/
void SetTimer2CTC(byte CSB,byte periode)
{
  TCCR2A=B010;    // Mode CTC (Clear Timer On Compare)
  OCR2A=periode; // comparison OCR2A (8bits)
  TCCR2B&=0xF0;
  TCCR2B|=(CSB&0x07); // prescaler choice
}

volatile unsigned int cpt=0;

ISR(TIMER2_COMPA_vect)  // IT when TCNT2==OCR2A
{
  cpt++;
}

void setup()
{
    Serial.begin(9600);
    cli();
    SetTimer2CTC(6,100);  // prescaler /256 periode 100
    TIMSK2|=0x02;       //IT Timer2 when TCNT2==OCR2A
    sei();
}

void loop()
{
   delay(1000);
   Serial.println(cpt,DEC);
}
```

*7.4 Timer/Counter 1 (16 bits)*

The TCNT1 count register, as well as the comparison registers OCR1A and OCR1B, are 16 bits this time.

**Note**: in assembly language, two 8-bit accesses are required to read/write these 16-bit registers. In C language, 16-bit data can be manipulated symbolically via TCNT1, OCR1A and OCR1B without worrying about how the code will be generated.

**Figure 15-1.** 16-bit Timer/Counter Block Diagram[1]




## Registers Timer/Counter 1

### TCNT1H and TCNT1L – Timer/Counter1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x85) | TCNT1[15:8] | | | | | | | | TCNT1H |
| (0x84) | TCNT1[7:0] | | | | | | | | TCNT1L |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

### OCR1AH and OCR1AL – Output Compare Register 1 A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x89) | OCR1A[15:8] | | | | | | | | OCR1AH |
| (0x88) | OCR1A[7:0] | | | | | | | | OCR1AL |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

### OCR1BH and OCR1BL – Output Compare Register 1 B

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x8B) | OCR1B[15:8] | | | | | | | | OCR1BH |
| (0x8A) | OCR1B[7:0] | | | | | | | | OCR1BL |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |

### TIFR1 – Timer/Counter1 Interrupt Flag Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| 0x16 (0x36) | – | – | ICF1 | – | – | OCF1B | OCF1A | TOV1 | TIFR1 |
| Read/Write | R | R | R/W | R | R | R/W | R/W | R/W | |

### TCCR1A – Timer/Counter1 Control Register A

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x80) | COM1A1 | COM1A0 | COM1B1 | COM1B0 | – | – | WGM11 | WGM10 | TCCR1A |
| Read/Write | R/W | R/W | R/W | R/W | R | R | R/W | R/W | |

### TCCR1B – Timer/Counter1 Control Register B

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| (0x81) | ICNC1 | ICES1 | – | WGM13 | WGM12 | CS12 | CS11 | CS10 | TCCR1B |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |

**Depending on the mode selected by bits WGM10:3, the following options are available (PWM mode Correct phase not described)**

**Table 15-4.** Waveform Generation Mode Bit Description(1)

| Mode | WGM13 | WGM12 (CTC1) | WGM11 (PWM11) | WGM10 (PWM10) | Timer/Counter Mode of Operation | TOP | Update of OCR1x at | TOV1 Flag Set on |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | Normal | 0xFFFF | Immediate | MAX |
| 1 | 0 | 0 | 0 | 1 | PWM, Phase Correct, 8-bit | 0x00FF | TOP | BOTTOM |
| 2 | 0 | 0 | 1 | 0 | PWM, Phase Correct, 9-bit | 0x01FF | TOP | BOTTOM |
| 3 | 0 | 0 | 1 | 1 | PWM, Phase Correct, 10-bit | 0x03FF | TOP | BOTTOM |
| 4 | 0 | 1 | 0 | 0 | CTC | OCR1A | Immediate | MAX |
| 5 | 0 | 1 | 0 | 1 | Fast PWM, 8-bit | 0x00FF | BOTTOM | TOP |
| 6 | 0 | 1 | 1 | 0 | Fast PWM, 9-bit | 0x01FF | BOTTOM | TOP |
| 7 | 0 | 1 | 1 | 1 | Fast PWM, 10-bit | 0x03FF | BOTTOM | TOP |
| 8 | 1 | 0 | 0 | 0 | PWM, Phase and Frequency Correct | ICR1 | BOTTOM | BOTTOM |
| 9 | 1 | 0 | 0 | 1 | PWM, Phase and Frequency Correct | OCR1A | BOTTOM | BOTTOM |
| 10 | 1 | 0 | 1 | 0 | PWM, Phase Correct | ICR1 | TOP | BOTTOM |
| 11 | 1 | 0 | 1 | 1 | PWM, Phase Correct | OCR1A | TOP | BOTTOM |
| 12 | 1 | 1 | 0 | 0 | CTC | ICR1 | Immediate | MAX |
| 13 | 1 | 1 | 0 | 1 | (Reserved) | – | – | – |
| 14 | 1 | 1 | 1 | 0 | Fast PWM | ICR1 | BOTTOM | TOP |
| 15 | 1 | 1 | 1 | 1 | Fast PWM | OCR1A | BOTTOM | TOP |

Note: 1. The CTC1 and PWM11:0 bit definition names are obsolete. Use the WGM12:0 definitions. However, the functionality and location of these bits are compatible with previous versions of the timer.

## Prescaler Timer 1

**Table 15-5.** Clock Select Bit Description

| CS12 | CS11 | CS10 | Description |
|---|---|---|---|
| 0 | 0 | 0 | No clock source (Timer/Counter stopped). |
| 0 | 0 | 1 | $clk_{I/O}$/1 (No prescaling) |
| 0 | 1 | 0 | $clk_{I/O}$/8 (From prescaler) |
| 0 | 1 | 1 | $clk_{I/O}$/64 (From prescaler) |
| 1 | 0 | 0 | $clk_{I/O}$/256 (From prescaler) |
| 1 | 0 | 1 | $clk_{I/O}$/1024 (From prescaler) |
| 1 | 1 | 0 | External clock source on T1 pin. Clock on falling edge. |
| 1 | 1 | 1 | External clock source on T1 pin. Clock on rising edge. |

In the example below

prescaler /256 = 62500 TCNT1 increments per second
period 10,000 = 6.25 interruptions TIMER1_COMPA_vect per second

```
/* Timer1 en Mode CTC
CSB = 0 (timer1 stop), 1(/1),2(/8),3(/64),4(/256),5(/1024)
period = */
void SetTimer1CTC(byte CSB,unsigned int period)
{
      TCCR1A=0;
      OCR1A=periode; // comparison OCR1A (16bits)
      TCCR1B=0x08;
      TCCR1B|=(CSB&0x07);
}

volatile unsigned int cpt=0;

ISR(TIMER1_COMPA_vect)  // IT when TCNT2==OCR2A
{
  cpt++;
}

void setup()
{
    Serial.begin(9600);
    cli();
    SetTimer1CTC(4,10000);  // prescaler /256 periode 10000
    TIMSK1|=0x02; //IT Timer1 when TCNT1==OCR1A
    sei();
}

void loop()
{
   delay(1000);
   Serial.println(cpt,DEC);
}
```