

Association for
Computing Machinery
Student Chapter, UPatras

Arduino Bare Metal Programming

Γρηγόρης Δεληπαλταδάκης

Content

- Introduction
- ATmega328P Microcontroller
- Embedded C Programming
- Blinking an LED
- Blinking an LED with Timer Interrupts

Arduino Programming vs Bare Metal Programming

- Στον προγραμματισμό Arduino μας παρέχεται high-level abstraction με συνάρτησης όπως `pinMode()`, `digitalWrite()`, `analogRead()` κλπ.
- Κρύβονται λεπτομερείς του υλικού για να δοθεί έμφαση στην ταχεία προτυποποίηση.
- Δεν απαιτείται γνώση της αρχιτεκτονικής του μικροελεγκτή.

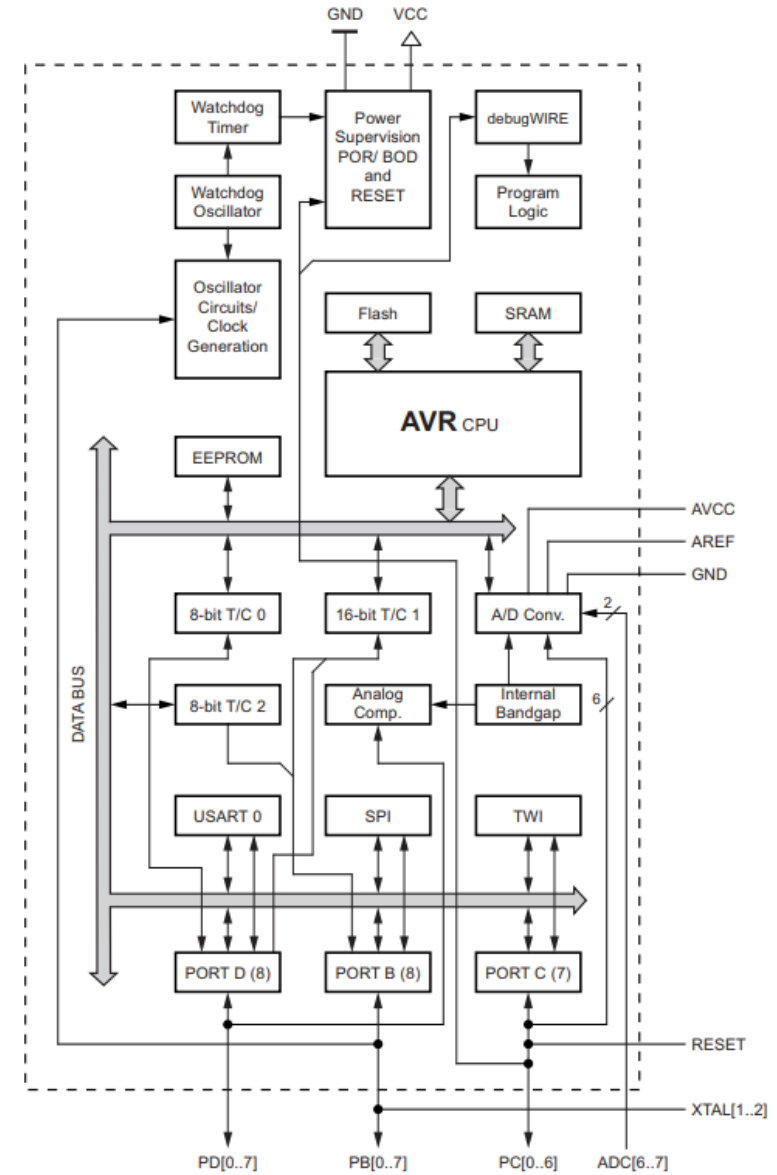
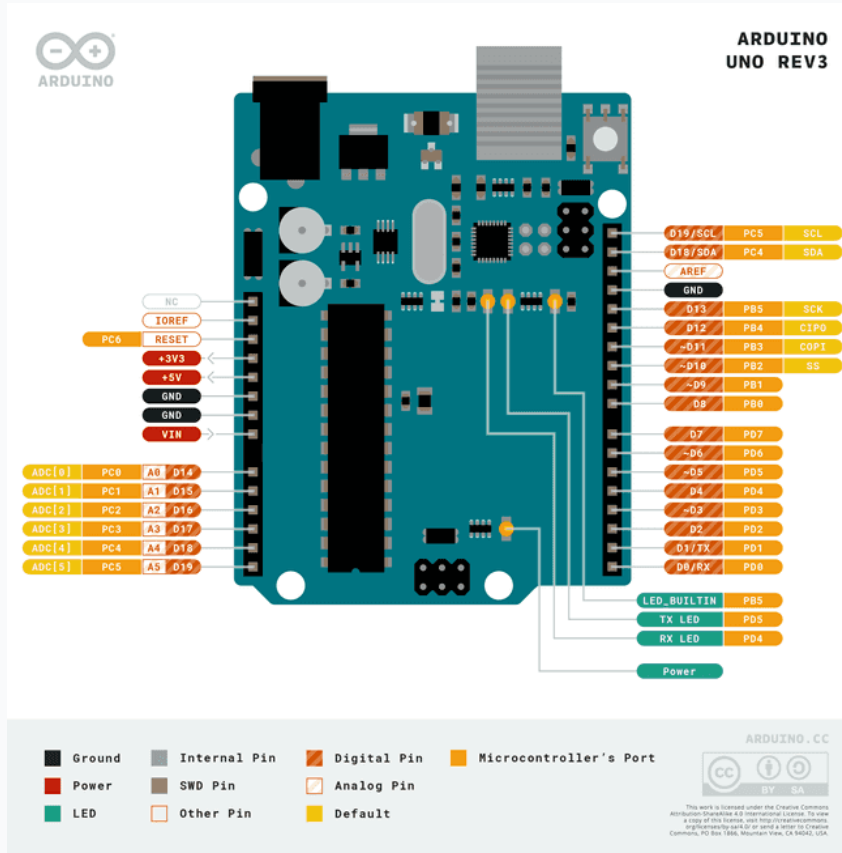
Bare Metal Programming Advantages

- Precise Control
- Optimal Performance
- Smaller Binary Size
- Better Understanding of Microcontroller Internals

ATmega328P

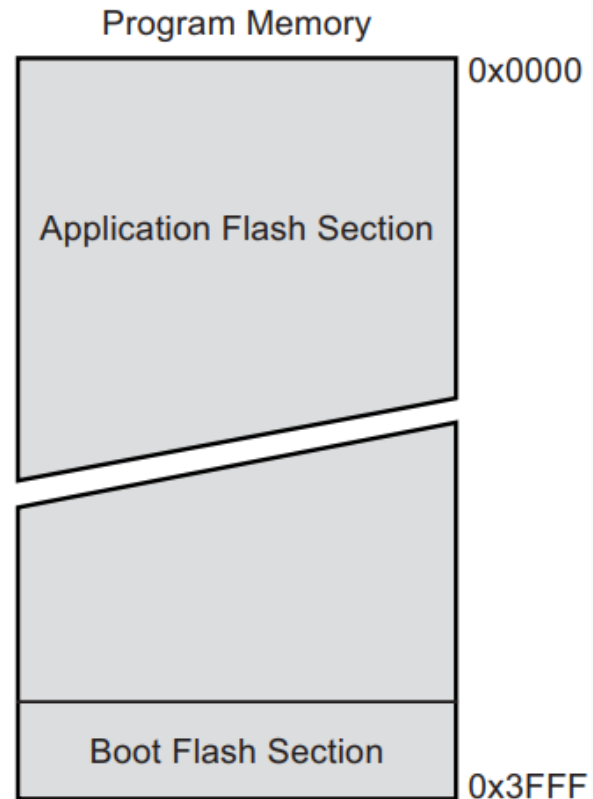
- Ο μικροελεγκτής του Arduino Uno.
- 8-bit AVR (RISC) μικροελεγκτής βασισμένος στην αρχιτεκτονική μνήμης Harvard.
- 32 General Purpose Registers.
- 23 GPIO Pins.
- 131 Instructions.

Diagrams



Memory

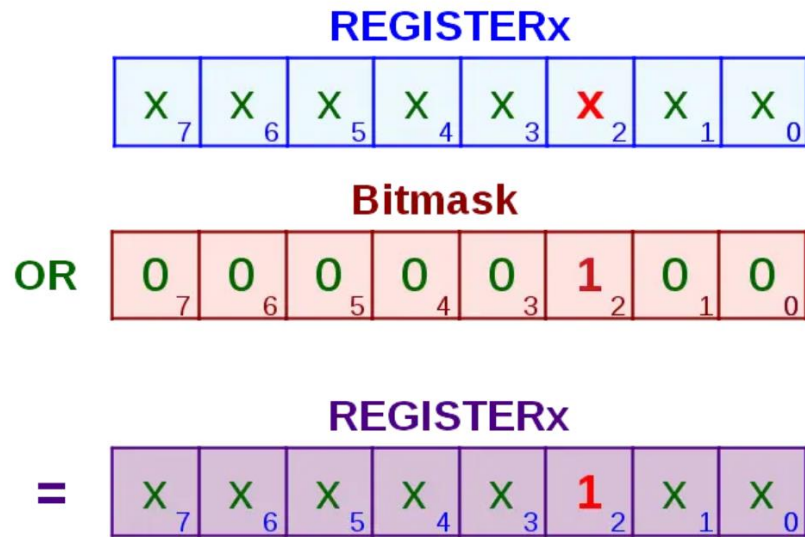
- 32KBytes Reprogrammable Flash Memory.
- Bootloader: Φορτώνει το νέο application code στην flash.
- 2408 Bytes SRAM Data Memory.
- Χρησιμοποιούμε τους I/O Registers για τον προγραμματισμό των pins.



Data Memory	
32 Registers	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F
160 Ext I/O Registers	0x0060 - 0x00FF
Internal SRAM (1048 x 8)	0x0100 0x08FF

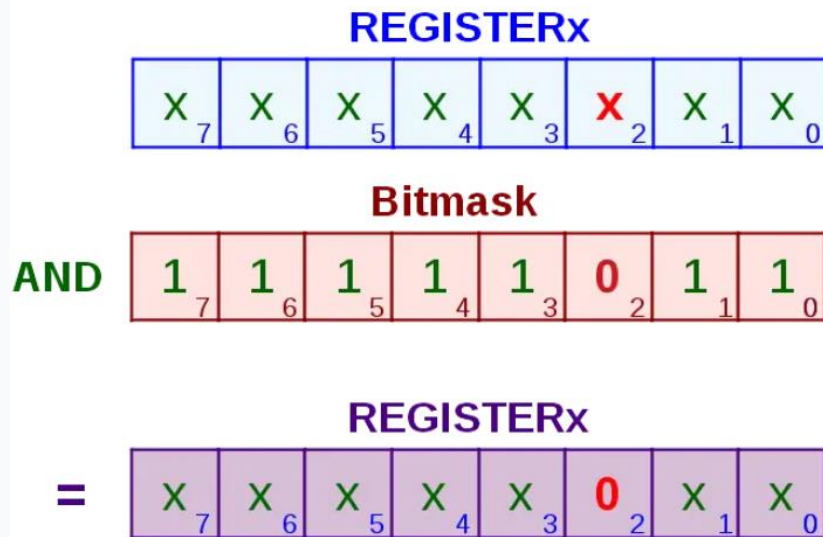
Programming Concepts

- Setting a bit of a register
- Mask Bitwise OR:
 - $\text{RegX} = \text{RegX} \mid 0b000000100;$
 - $\text{RegX} \mid= 0b000000100;$
 - $\text{RegX} \mid= (1 \ll 2);$

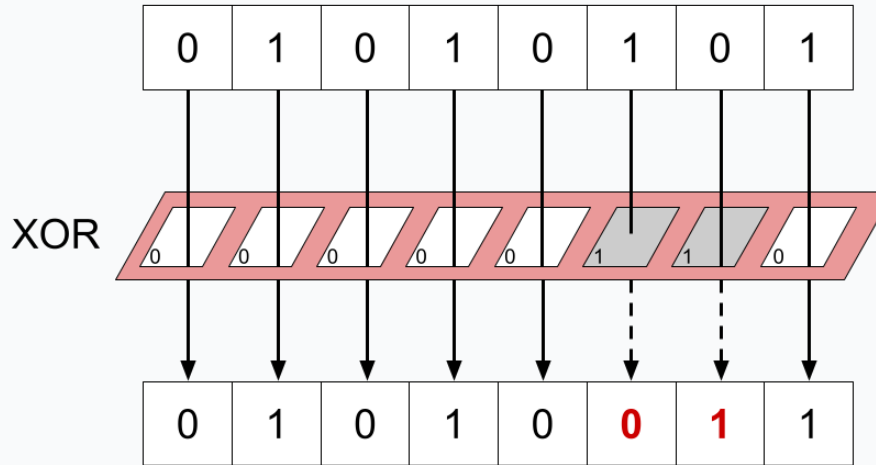


Programming Concepts

- Unsetting a bit of a register
- Mask Bitwise AND:
 - `RegX = RegX & 0b11111011;`
 - `RegX &= 0b11111011;`
 - `RegX &= ~0b00000100;`
 - `RegX &= ~(1<<2);`



Programming Concepts



- Toggling a bit of a register
- Mask Bitwise XOR:
- Ex. toggle bits 1,2.
- $\text{RegX} = \text{RegX} \wedge 0b00000110;$
- $\text{RegX} \wedge= 0b00000110;$
- $\text{RegX} \wedge= (3 \ll 1);$

Blinking an LED

- **Arduino Code:**

- Set up the LED pin as an output
- Write high.
- Delay for 1 second.
- Write low .
- Delay again.

```
#define led_pin LED_BUILTIN

void setup() {
  pinMode(led_pin, OUTPUT);
}

void loop() {
  digitalWrite(led_pin, HIGH);

  delay(1000);

  digitalWrite(led_pin, LOW);

  delay(1000);
}
```

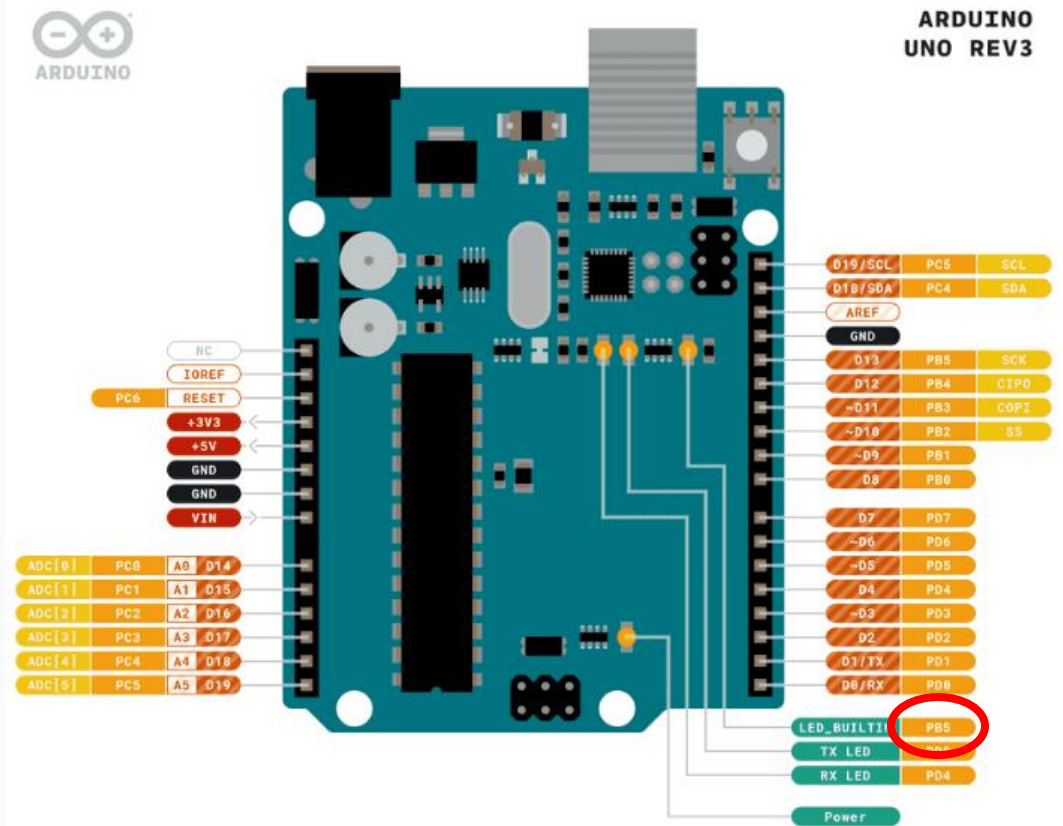
Compiling and Uploading the Arduino code

- Το πρόγραμμα έχει μέγεθος 924 bytes.
- Σχεδόν 1KB.
- Μπορούμε να κάνουμε κάτι καλύτερο;

```
Sketch uses 924 bytes (2%) of program storage space. Maximum is 32256 bytes.
```

Port and pin

- Το LED_BUILTIN pin αντιστοιχεί στο port B pin 5.
- Ο Atmega328P διαθέτει τρία I/O ports (Port B, Port C και Port D), που χρησιμοποιούνται για digital input/output και άλλες λειτουργίες (SPI, UART, Analog I/O, κλπ.)



Necessary Registers - DDRB

- Ορίζουμε αν ένα pin του port B θα είναι είσοδος ή έξοδος.
- Γράφοντας σε κάποια θέση:
- 0: Input
- 1: Output

13.4.3 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	

Necessary Registers - PORTB

- Ελέγχει την έξοδο δεδομένων των pins του port B.
- Γράφοντας σε κάποια θέση:
- 0: Θέτουμε το αντίστοιχο pin LOW.
- 1: Θέτουμε το αντίστοιχο pin HIGH.

13.4.2 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	0	0	0	

Command Line Tools

- **AVR-GCC:** C/C++ compiler για AVR μικροελεγκτές.
- **AVR-libC:** C βιβλιοθήκη που παρέχει essential utilities, macros και functions, που βοηθούν στην συγγραφή embedded κώδικα.
- **AVR-Binutils:** Συλλογή από binary utilities (assembler, linker και άλλα) με σκοπό την δημιουργία και διαχείριση κώδικα μηχανής για AVR μικροελεγκτές.
- **AVRdude (AVR Downloader Uploader):** Utility program, που επιτρέπει την μεταφόρτωση compiled κώδικα στην μνήμη flash ενός AVR μικροελεγκτή μέσω της θύρα USB του υπολογιστή.

```
$ sudo apt install avr-libc avrdude binutils-avr gcc-avr
```


Code – blink.c

- **DDB5**: Predefined macro.
- Αναφέρεται στο pin 5 του DDRB.
- Κάνει τον κώδικα πιο αναγνώσιμο.
- *iom328p.h*:

```
#define PORTB _SFR_I08(0x05)
#define PORTB0 0
#define PORTB1 1
#define PORTB2 2
#define PORTB3 3
#define PORTB4 4
#define PORTB5 5
#define PORTB6 6
#define PORTB7 7
```

```
#include <avr/io.h>
#include <util/delay.h>

int main(void){
    DDRB = DDRB | (1<<DDB5);

    while(1){
        PORTB |= (1<<PORTB5);

        _delay_ms(1000);

        PORTB &= ~(1<<PORTB5);

        _delay_ms(1000);
    }

    return 0;
}
```

Build and Upload

- Makefile:

```
TARGET = blink

default:
    avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o $(TARGET).o $(TARGET).c
    avr-gcc -mmcu=atmega328p $(TARGET).o -o $(TARGET)
    avr-objcopy -O ihex -R .eeprom $(TARGET) $(TARGET).hex

upload: default
    avrdude -F -V -c arduino -p ATMEGA328P -P /dev/ttyACM0 -b 115200 -U flash:w:$(TARGET).hex

clean:
    rm -f *.o *.bin *.hex
```

Build and Upload

- `avr-gcc -Os -DF_CPU=16000000UL -mmcu=atmega328p -c -o $(TARGET).o $(TARGET).c`
- Κάνει compile το source file σε object file.
- `-Os`: Optimization level. Optimizes the code for size (minimizing memory usage), which is critical for microcontrollers with limited flash memory.
- `-DF_CPU=16000000UL`: Defines a preprocessor macro `F_CPU` with the value 16MHz.
- `-mmcu=atmega328p`: Specifies the target microcontroller. Ensures the compiler uses the appropriate instruction set and memory mapping for the ATmega328P.
- `-c`: Compile-only flag. The compiler stops after generating the object file and does not perform linking.

Build and Upload

- `avr-gcc -mmcu=atmega328p blink.o -o blink`
- Links the object file and produces an executable. Combines the `blink.o` object file with any necessary system libraries or startup code.
- `avr-objcopy -O ihex -R .eeprom blink blink.hex`
- Converts the binary executable into an Intel HEX file, which can be uploaded to the microcontroller. AVRDUDE requires a HEX file for uploading.
- `-R`: Removes the `.eeprom` section from the output file. Για να μην γράψουμε στην eeprom.

Build and Upload

- `avrdude -c arduino -p ATMEGA328P -P /dev/ttyACM0 -b 115200 -U flash:w:blink.hex`
- Uploads `blink.hex` to the microcontroller via the specified port.
- `-c arduino`: Specifies the programmer type, “arduino” is for boards like the Arduino Uno, which use a bootloader to handle the flashing process.
- `-p ATMEGA328P`: Specifies the target microcontroller.
- `-P /dev/ttyACM0`: Specifies the port to use for communication. `/dev/ttyACM0` is the serial port typically assigned to Arduino devices on Linux systems.
- `-b 115200`: Sets the baud rate for communication. 115200 is the standard baud rate for Arduino boards using the bootloader.
- `-U flash:w:blink.hex`: Specifies the target memory (flash) and a write operation (w).

Compiling and Uploading the AVR-C code

```
avrdude: writing 176 bytes flash ...  
Writing | ##### | 100% 0.07 s  
avrdude: 176 bytes of flash written
```

- Ο κώδικας πιάνει 176 bytes στην μνήμη προγράμματος.
- **Μείωση μεγέθους 81%** συγκριτικά με τον Arduino κώδικα.

Why?

- **Arduino Framework Overhead:**
 - Arduino code uses functions from the Arduino core library, which introduces a layer of abstraction.
 - Αυτές οι συναρτήσεις είναι γενικευμένες για την υποστήριξη πολλών μικροελεγκτών, οπότε είναι μεγαλύτερες.
 - Επίσης συμπεριλαμβάνονται συναρτήσεις που δεν χρησιμοποιούνται.
- **Initialization Code:**
 - Χρησιμοποιούνται startup routines (πχ `init()`), για την διαμόρφωση περιφερειακών, που μπορεί να μην χρειάζονται σε ένα minimal πρόγραμμα.
- **Delay Implementation:**
 - Η υλοποίηση του `delay()` βασίζεται σε επιπρόσθετα layers (`millis()`, `micros()`), που είναι πιο σύνθετα από την απλή `_delay_ms()` συνάρτηση της AVR C.

Pointers

- Η πρόσβαση στους I/O καταχωρητές γίνεται με την χρήση των macros DDRB και PORTB.
- Τα οποία ουσιαστικά είναι pointers στις κατάλληλες θέσεις μνήμης των καταχωρητών.
- **volatile**: Αποτρέπει τον compiler από το να κάνει optimize out αυτή την έκφραση. Μιας και πρόκειται για μια θέση μνήμης, οπού μόνο γράφουμε.

```
int main(void){
    DDRB = DDRB | (1<<DDB5);

    while(1){
        PORTB |= (1<<PORTB5);

        _delay_ms(1000);

        PORTB &= ~(1<<PORTB5);

        _delay_ms(1000);
    }
}
```

```
#define DDRB _SFR_I08(0x04)
```

Expands to:

```
*(volatile uint8_t *)((0x04) + 0x20))
```

```
#define PORTB _SFR_I08(0x05)
```

Expands to:

```
*(volatile uint8_t *)((0x05) + 0x20))
```


Memory Address for DDRB and PORTB

- Σε ποιες θέσεις μνήμης ανήκουν τελικά οι καταχωρητές DDRB και PORTB;
- Στις 0x4 και 0x5 ή 0x24 και 0x25;

```
#define DDRB _SFR_I08(0x04)
```

Expands to:

```
(*(volatile uint8_t *)((0x04) + 0x20))
```

```
#define PORTB _SFR_I08(0x05)
```

Expands to:

```
(*(volatile uint8_t *)((0x05) + 0x20))
```

DDRB – The Port B Data Direction Register

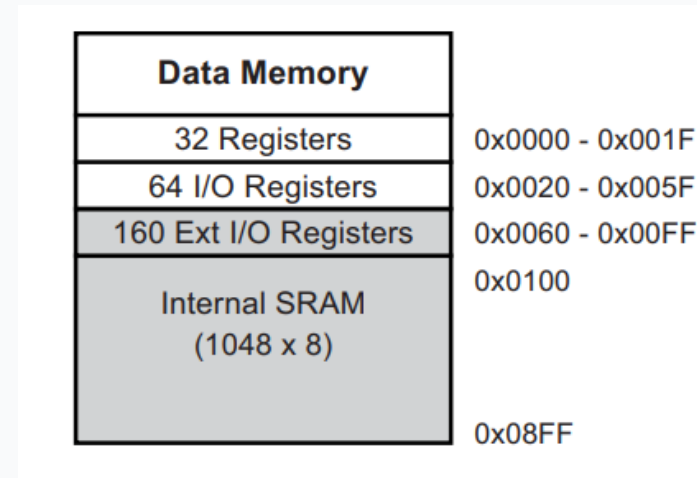
Bit	7	6	5	4	3	2	1	0	
0x04 (0x24)	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0	DDRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

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	

Memory Address for DDRB and PORTB

- Στον ATmega328p, οι I/O καταχωρητές, όπως οι DDRB και PORTB είναι mapped σε πολλαπλές περιοχές μνήμης.
- Στο **data memory space**: DDRB -> 0x24 και PORTB -> 0x25.
- Όμως για **συγκεκριμένες εντολές** (π.χ. SBI, CBI) αυτοί οι καταχωρητές είναι προσβάσιμοι στις θέσεις 0x4 και 0x5.



and ST/STS/STD instructions, transferring data between the 32 general purpose working registers and the I/O space. I/O registers within the address range 0x00 - 0x1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions. Refer to the instruction set section for more details. When using the I/O specific commands IN and OUT, the I/O addresses 0x00 - 0x3F must be used. When addressing I/O registers as data space using LD and ST instructions, 0x20 must be added to these addresses. The

Memory Address for DDRB and PORTB

- Ποιες εντολές χρησιμοποιούνται στην περίπτωση μας;
- Ο compiler χρησιμοποιεί εντολές **sbi** και **cbi**, οι οποίες χρησιμοποιούνται τις διευθύνσεις των I/O καταχωρητών στο εύρος 0x00 – 0xFF.

- blink.s:

```
DDRB = DDRB | (1<<DDB5);
```

```
PORTB |= (1<<PORTB5);
```

```
main:
/* prologue: function */
/* frame size = 0 */
/* stack size = 0 */
.L__stack_usage = 0
    sbi 0x4,5
.L2:
    sbi 0x5,5
    ldi r18,lo8(3199999)
    ldi r24,hi8(3199999)
    ldi r25,hlo8(3199999)
1:    subi r18,1
    sbci r24,0
    sbci r25,0
    brne 1b
```

Implementation of _delay_ms()

- `__tmp = ((F_CPU) / 1e3) * __ms;`
- Υπολογίζεται ο αριθμός των CPU clock cycles, που απαιτούνται για το delay.
- `__ticks_dc = (uint32_t)(ceil(fabs(__tmp)));`
- Πραγματοποιείται rounding up.
- `__builtin_avr_delay_cycles(__ticks_dc);`
- Καλείται η παραπάνω built-in συνάρτηση, η οποία παράγει inline assembly κώδικα για την δημιουργία του delay.

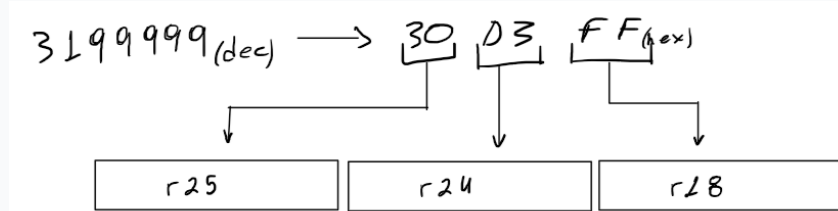
```
_delay_ms(double __ms)
{
    double __tmp;
    #if __HAS_DELAY_CYCLES && defined(__OPTIMIZE__) && \
        !defined(__DELAY_BACKWARD_COMPATIBLE__) && \
        __STDC_HOSTED__
        uint32_t __ticks_dc;
        extern void __builtin_avr_delay_cycles(unsigned long);
        __tmp = ((F_CPU) / 1e3) * __ms;

        #if defined(__DELAY_ROUND_DOWN__)
            __ticks_dc = (uint32_t)fabs(__tmp);
        #elif defined(__DELAY_ROUND_CLOSEST__)
            __ticks_dc = (uint32_t)(fabs(__tmp)+0.5);
        #else
            //round up by default
            __ticks_dc = (uint32_t)(ceil(fabs(__tmp)));
        #endif

        __builtin_avr_delay_cycles(__ticks_dc);
    }
```

Assembly

- **sbi 0x5, 5:** Set bit 5 of register 0x5 (PORTB).
- Δημιουργία ενός **24 – bit counter** με την τιμή



- **Μείωση του counter κατά 1.**
- **Branch** στο label 1, αν το αποτέλεσμα της αφαίρεσης δεν είναι μηδέν


```
while(1){  
    PORTB |= (1<<PORTB5);  
  
    _delay_ms(1000);  
}
```

```
.L2:  
    sbi 0x5,5  
    ldi r18,lo8(3199999)  
    ldi r24,hi8(3199999)  
    ldi r25,hlo8(3199999)  
1:  subi r18,1  
    sbci r24,0  
    sbci r25,0  
    brne 1b
```

Assembly

- `subi r18,1` (1 cycle)
- `subci r24,0` (1cycle)
- `subci r25,0` (1cycle)
- `brne 1b` (2 cycles when branching)
- **5 clock cycle σε κάθε loop.**
- **3199999 iterations του loop.**
- Αρά το **loop διαρκεί 15999995 clock cycles.**
- $15999995 \text{ cycles} * 62.5ns (T_{cpu}) = 0.99s \approx 1s$

```
while(1){  
    PORTB |= (1<<PORTB5);  
    _delay_ms(1000);  
}
```



```
.L2:  
    sbi 0x5,5  
    ldi r18,lo8(3199999)  
    ldi r24,hi8(3199999)  
    ldi r25,hlo8(3199999)  
1:  subi r18,1  
    sbci r24,0  
    sbci r25,0  
    brne 1b
```

} 5 cycles

Blocking delay

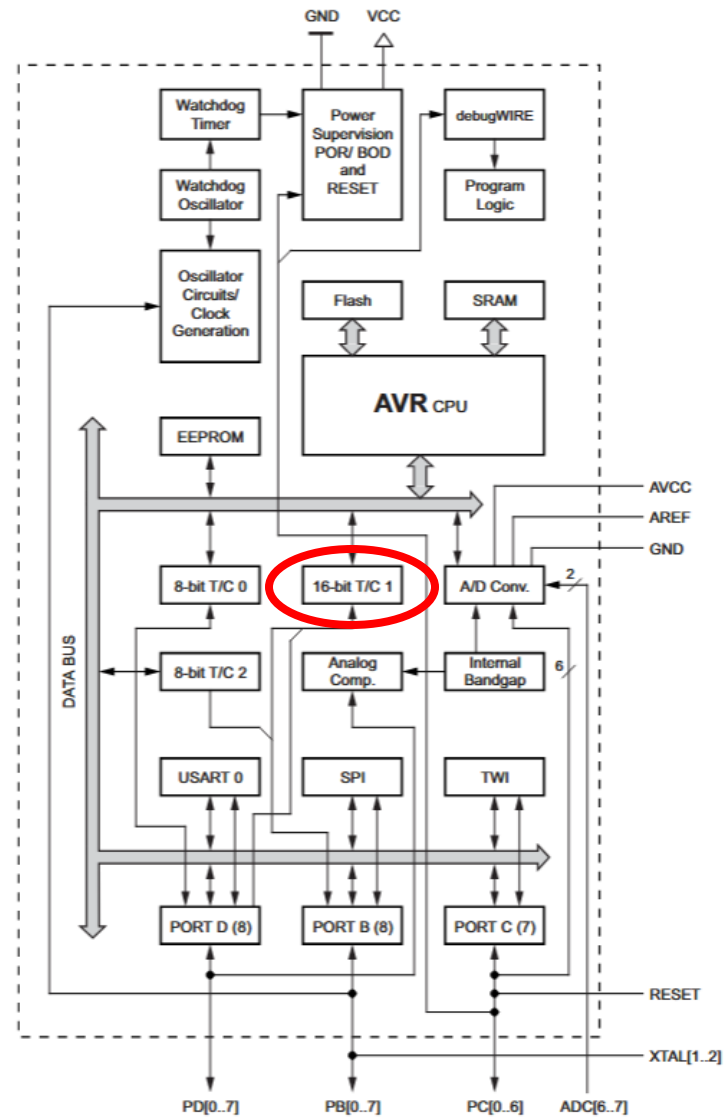
- Η συνάρτηση `_delay_ms()` είναι blocking, μιας και υλοποιείται με ένα busy-wait loop.
- Όσο εκτελείται το loop, η CPU δεν μπορεί να εκτελέσει άλλες διεργασίες.
- Αν η εφαρμογή μας απαιτεί concurrent operations πρέπει να χρησιμοποιήσουμε timers, interrupts ή κάποιο real-time operating system (RTOS).

Non-Blocking Blinking

- Μπορούμε να υλοποιήσουμε non-blocking LED blinking αξιοποιώντας timer interrupts.
- Ουσιαστικά προγραμματίζουμε έναν timer, να μετράει μέχρι μια συγκεκριμένη τιμή και όταν φτάσει σε αυτήν εκτελείται μια ISR (Interrupt Service Routine), που κάνει toggle το LED.

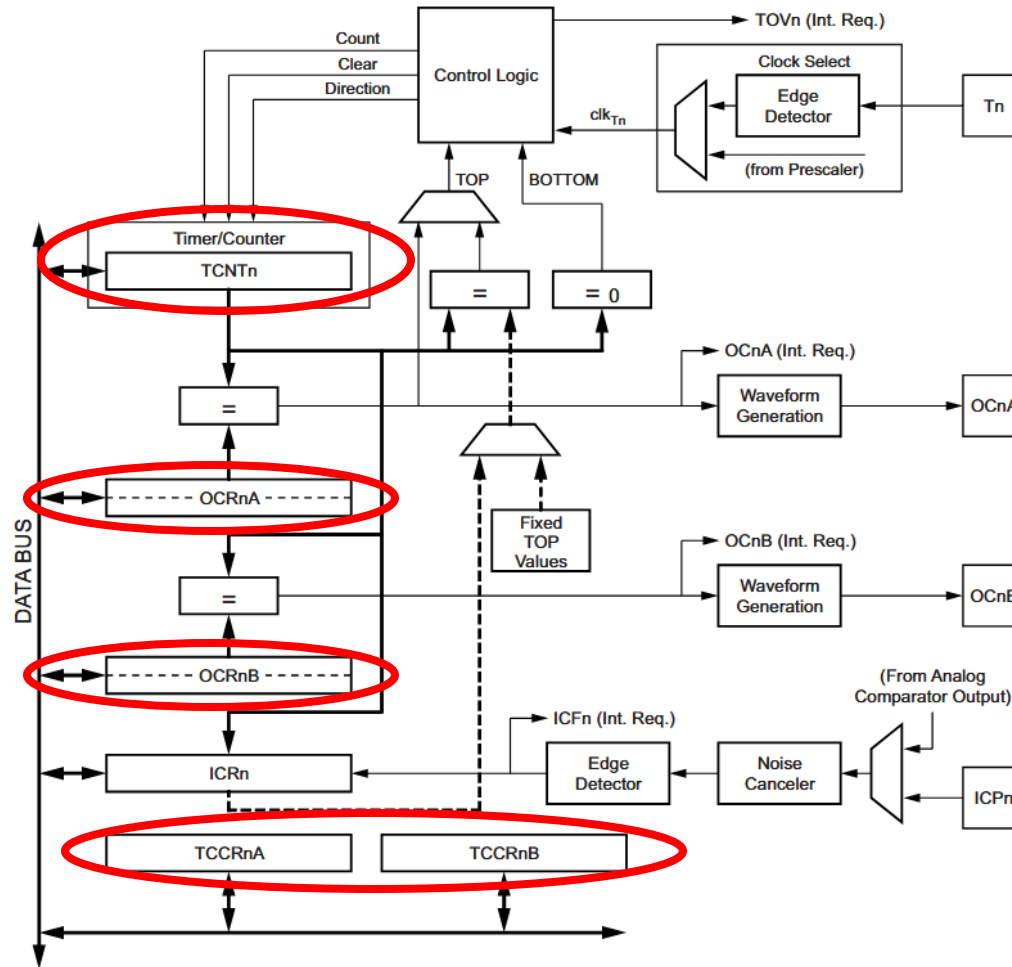
16-bit Timer/Counter1

- **Features:**
- True 16-bit design
- Two independent output compare units
- Clear timer on compare match (auto reload)
- Four independent interrupt sources (TOV1, OCF1A, OCF1B, and ICF1)



Timer/Counter1 Block Diagram

- **TCNT1:** The main 16-bit timer/counter unit. Split into two 8-bit registers (TCNT1H and TCNT1L)
- **TCCR1A, TCCR1B:** Timer/Counter1 Control Registers.
- **OCR1A, OCR1B:** Output Compare Registers.



Delay implementation using Timer/Counter 1

- **1. Επιλογή mode λειτουργίας.**
- Επιλέγουμε το CTC mode (Clear Timer on Compare), ώστε να κάνουμε clear τον timer όταν αυτός φτάσει στην τιμή που θέσουμε στον καταχωρητή OCR1A.

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



Delay implementation using Timer/Counter 1

- 2. Προγραμματισμός καταχωρητών ελέγχου για CTC mode.

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

15.11.2 TCCR1B – Timer/Counter1 Control Register B

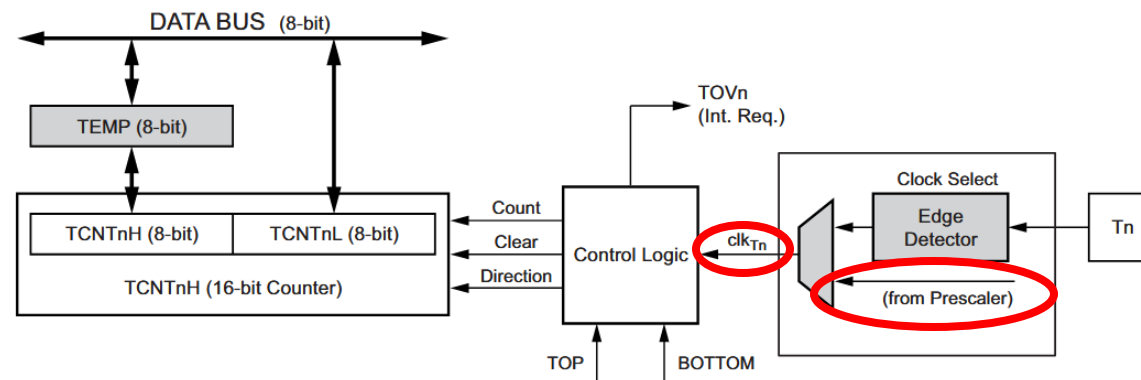
Bit	7	6	5	4	3	2	1	0	
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Set to '1'

Delay implementation using Timer/Counter 1

- 3. Επιλογή Prescaler.
- Μπορούμε να χρησιμοποιήσουμε ένα prescaled clock για τον timer, ώστε να μειώσουμε την συχνότητα μέτρησης. Συγκεκριμένα θέλουμε ένα timer clock, που να δίνει $f_{timer} = \frac{f_{system}}{1024}$.

Figure 15-2. Counter Unit Block Diagram



Delay implementation using Timer/Counter 1

- 3. Επιλογή Prescaler.
- Για να επιλέξουμε prescaler = 1024, προγραμματίζουμε και πάλι τον καταχωρητή TCCR1B.

CS12	CS11	CS10	Description
1	0	1	$\text{clk}_{\text{I/O}}/1024$ (from prescaler)

15.11.2 TCCR1B – Timer/Counter1 Control Register B

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

Handwritten annotations: Red arrows point to bits 2, 1, and 0 with values 1, 0, and 1 respectively. The bits CS12, CS11, and CS10 are circled in red.

Delay implementation using Timer/Counter 1

- 4. Ενεργοποίηση Output Compare A Match Interrupt.
- Όταν ο timer φτάσει στην τιμή που θα θέσουμε στον OCR1A, θέλουμε να πραγματοποιηθεί interrupt και η ροή εκτέλεσης να μεταφερθεί στο κατάλληλο ISR.
- Οπότε, προγραμματίζουμε κατάλληλα τον καταχωρητή TIMSK1 (Timer/Counter1 Interrupt Mask Register).

15.11.8 TIMSK1 – Timer/Counter1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	–	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Output Compare A Match Interrupt Enabled

Set to '1'

Delay implementation using Timer/Counter 1

- Λειτουργία του Output Compare A Match Interrupt.
- Με την ενεργοποίηση του Output Compare Match Interrupt, όταν η έξοδος του Timer/Counter1 φτάσει στην τιμή του OCR1A, μετά από έναν κύκλο ρολογιού, θα ενεργοποιηθεί το OCF1A flag και στην συνέχεια θα εκτελεστεί το ISR στην διεύθυνση μνήμης που ορίζει το Interrupt Vector TIMER1 COMPA.

Figure 15-6. CTC Mode, Timing Diagram

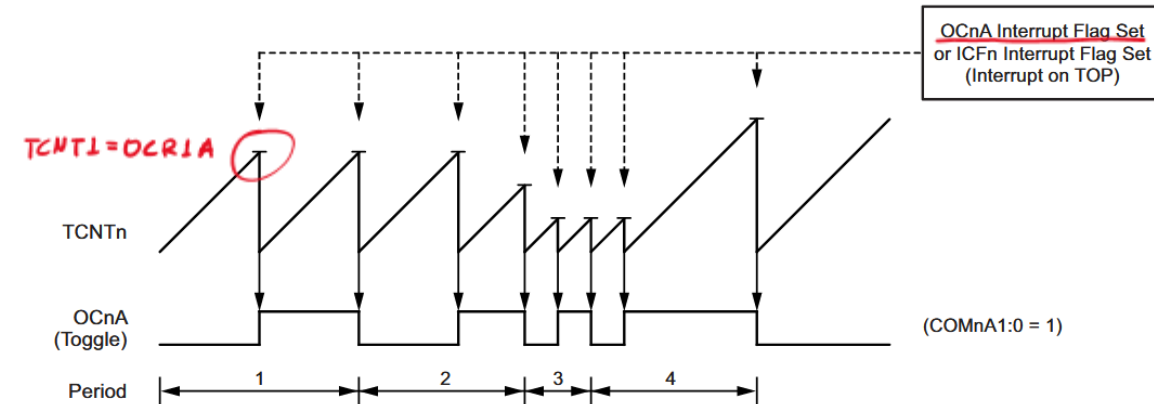


Table 11-1. Reset and Interrupt Vectors in ATmega328P

Vector No.	Program Address	Source	Interrupt Definition
1	0x0000	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

Delay implementation using Timer/Counter 1

- 5. Επιλογή τιμής του OCR1A.

- Έστω ότι θέλουμε interrupt μετά από $delay\ ms$ και $f_{timer} = \frac{f_{system}}{PRESCALER}$, $T_{timer} = \frac{PRESCALER}{f_{system}}$

- Πρέπει να βρούμε πόσο $ticks$ του timer απαιτούνται για να περάσουν $delay\ ms$.

- $ticks * T_{timer} = delay \Rightarrow ticks = \frac{delay}{T_{timer}} = delay(ms) * f_{timer(MHz)} \Rightarrow ticks = \frac{delay}{1000} * f_{timer}$

- Όμως ο timer μετράει από το 0 μέχρι την τιμή στον OCR1A.

- $OCR1A = ticks - 1 = \left(\frac{delay}{1000} * f_{timer} \right) - 1$

Delay implementation using Timer/Counter 1

- 5. Προγραμματισμός του OCR1A.

- Π.χ. για 1000ms delay, $ticks = 15625_{dec}$, $OCR1A = ticks - 1 = 15624 = 3D08_{hex} = 0011\ 1101\ 0000\ 1000_{bin}$

15.11.5 OCR1AH and OCR1AL – Output Compare Register 1 A

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A[15:8] = 0x3D								OCR1AH
(0x88)	OCR1A[7:0] = 0x08								OCR1AL
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	

Delay implementation using Timer/Counter 1

- **Μέγιστο delay.**
- Η μέγιστη τιμή που μπορεί να πάρει ο OCR1A είναι $FFFF_{hex} = 65535_{dec}$.
- Δηλαδή το μέγιστο delay με αυτή την μέθοδο είναι:
- $delay_{max} = (OCR1A_{max} + 1) * \frac{1000}{f_{timer}} = 65536 * \frac{1000}{16MHz} * 1024 = 4194.304 ms$

Blinking Led with Timer Interrupts

```
void initTimer1(uint16_t delay_ms) {
    // Configure Timer1 in CTC mode
    TCCR1B |= (1 << WGM12) | (1 << CS12) | (1 << CS10); // CTC mode, prescaler 1024

    uint32_t ticks = (F_CPU / 1024 * delay_ms / 1000) - 1; // Calculate number of ticks for the delay

    if (ticks > 0xFFFF) {
        ticks = 0xFFFF; // Ensure ticks is within the 16-bit range
    }

    OCR1AH = (ticks >> 8); // Load the high byte of the OCR1A register
    OCR1AL = ticks; // Load the low byte of the OCR1A register

    TIMSK1 = (1 << OCIE1A); // Enable Timer1 compare match A interrupt
}

ISR(TIMER1_COMPA_vect) {
    PORTB ^= (1 << PORTB5); // Toggle LED
}
```