

# Μικροελεγκτές AVR

## 2η ΕΝΟΤΗΤΑ: ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΣΕ ΓΛΩΣΣΑ C ΕΝΣΩΜΑΤΩΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ

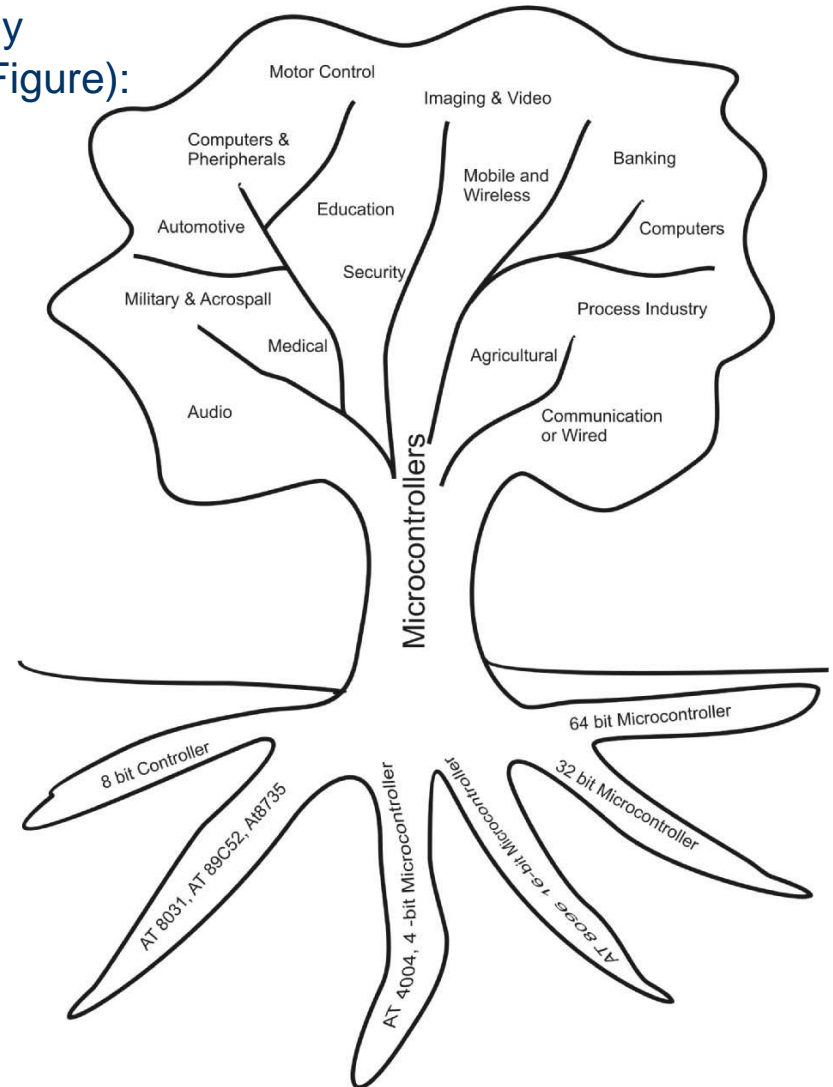
Ε.Μ.Π.

Εργ. Μικροϋπολογιστών & Ψηφιακών Συστημάτων  
Υπεύθυνος: Κ. ΠΕΚΜΕΣΤΖΗ Καθ.

# Microcontroller application tree

The microcontroller applications are mainly categorized into the following types (see Figure):

- ☐ Audio
- ☐ Automotive
- ☐ Communication/wired
- ☐ Computers and peripherals
- ☐ Consumer
- ☐ Industrial
- ☐ Imaging and video
- ☐ Medical
- ☐ Military/aerospace
- ☐ Mobile/wireless
- ☐ Motor control
- ☐ Security
- ☐ General Purpose
- ☐ Miscellaneous



# Γλώσσα C για Προγραμματισμό Ενσωματωμένων Συστημάτων

Η Γλώσσα Προγραμματισμού C είναι υψηλού επιπέδου αφαίρεσης. Επιτρέπει όμως αποδοτική περιγραφή και διαχείριση λειτουργιών χαμηλού επιπέδου, π.χ. bitwise-operators, pointer-based memory referencing κ.λπ.

Τα προγράμματα που αφορούν σε εφαρμογές πραγματικού χρόνου συχνά δεν περιλαμβάνουν κάποιο λειτουργικό σύστημα όπως αυτά των υπολογιστών γενικού σκοπού για τους παρακάτω λόγους:

- Συνήθως είναι συστήματα μικρού ή μεσαίου μεγέθους, δεδομένου ότι (όπως έγινε σαφές και από τα προηγούμενα) τα ενσωματωμένα συστήματα επιτελούν μια συγκεκριμένη εργασία.
- Η παρουσία ενός κλασικού λειτουργικού συστήματος αποτελεί σοβαρή χρονική επιβάρυνση. Έτσι για να αποκρίνεται σε κρίσιμες εφαρμογές πραγματικού χρόνου μπορούν να χρησιμοποιηθούν ειδικά λειτουργικά συστήματα πραγματικού χρόνου ή να υιοθετηθούν κατά περίπτωση λύσεις.
- Η απλούστερη δομή που υποκαθιστά ένα πλήρες λειτουργικό σύστημα είναι ένας ατέρμονας βρόχος που εκτελεί συνεχώς την εξειδικευμένη εργασία (λειτουργία) και οι χρονικά κρίσιμες λειτουργίες προσαρτώνται σε διακοπές.
- Η παρουσία ενός κλασικού λειτουργικού συστήματος “δυσκολεύει” την αποσφαλμάτωση της ενσωματωμένης εφαρμογής.

# One Software Mistake Is All It Takes

## Knight Capital Says Trading Glitch Cost It \$440 Million

By NATHANIEL POPPER AUGUST 2, 2012 9:07 AM 356 Comments

Runaway Trades Spread Turmoil Across Wall St.



Errant trades from the Knight Capital Group began hitting the New York Stock Exchange almost as soon as the opening bell rang on Wednesday. Brendan McDermid/Reuters

The [Knight Capital Group](#) announced on Thursday that it lost \$440 million when it sold all the stocks it accidentally bought Wednesday morning because a computer glitch.

<https://goo.gl/7dHOjO>

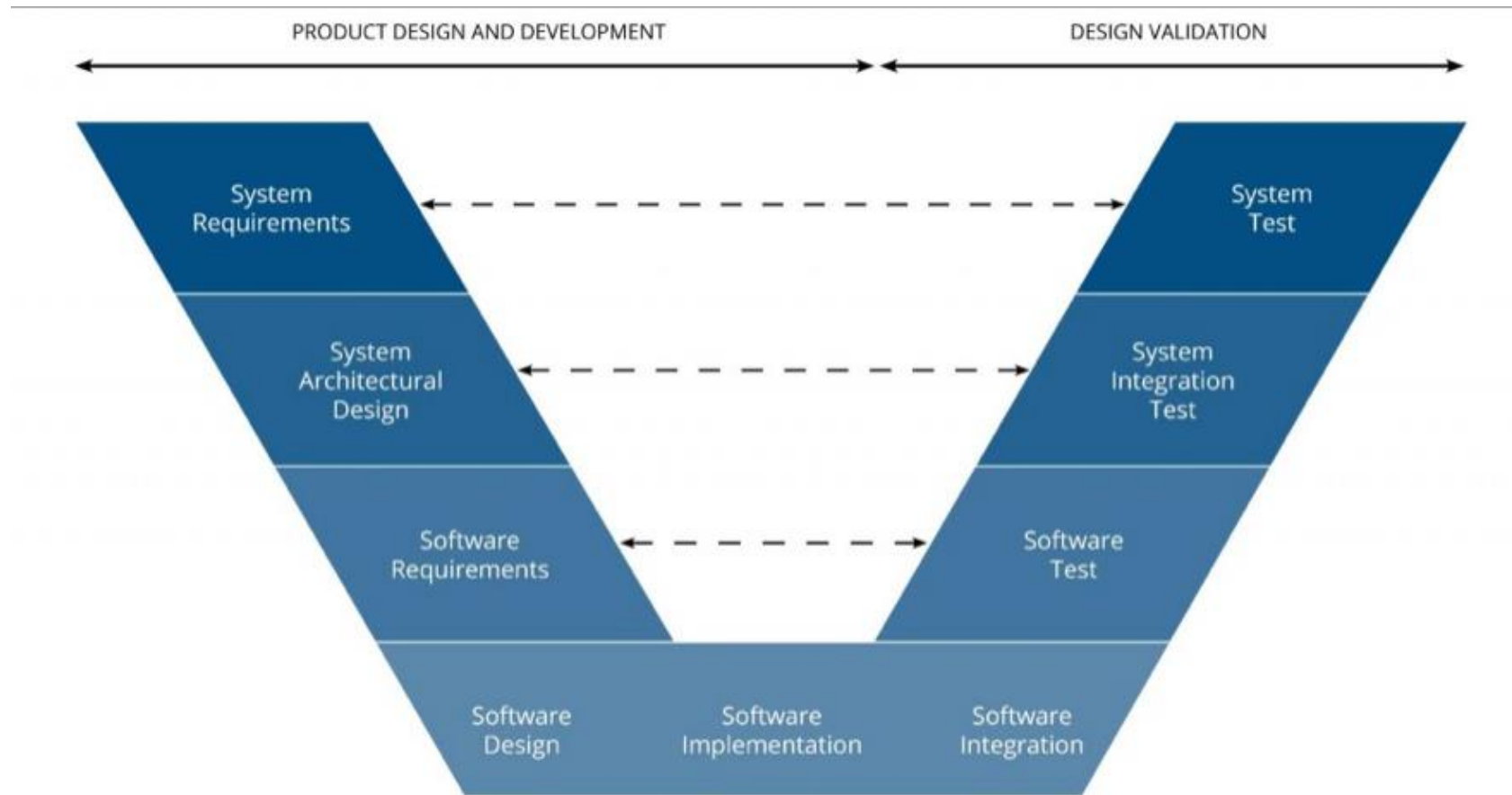


ist Martin Winterkorn his position as chief executive of VW Photo: AP/Richard

<https://goo.gl/T96ezC>

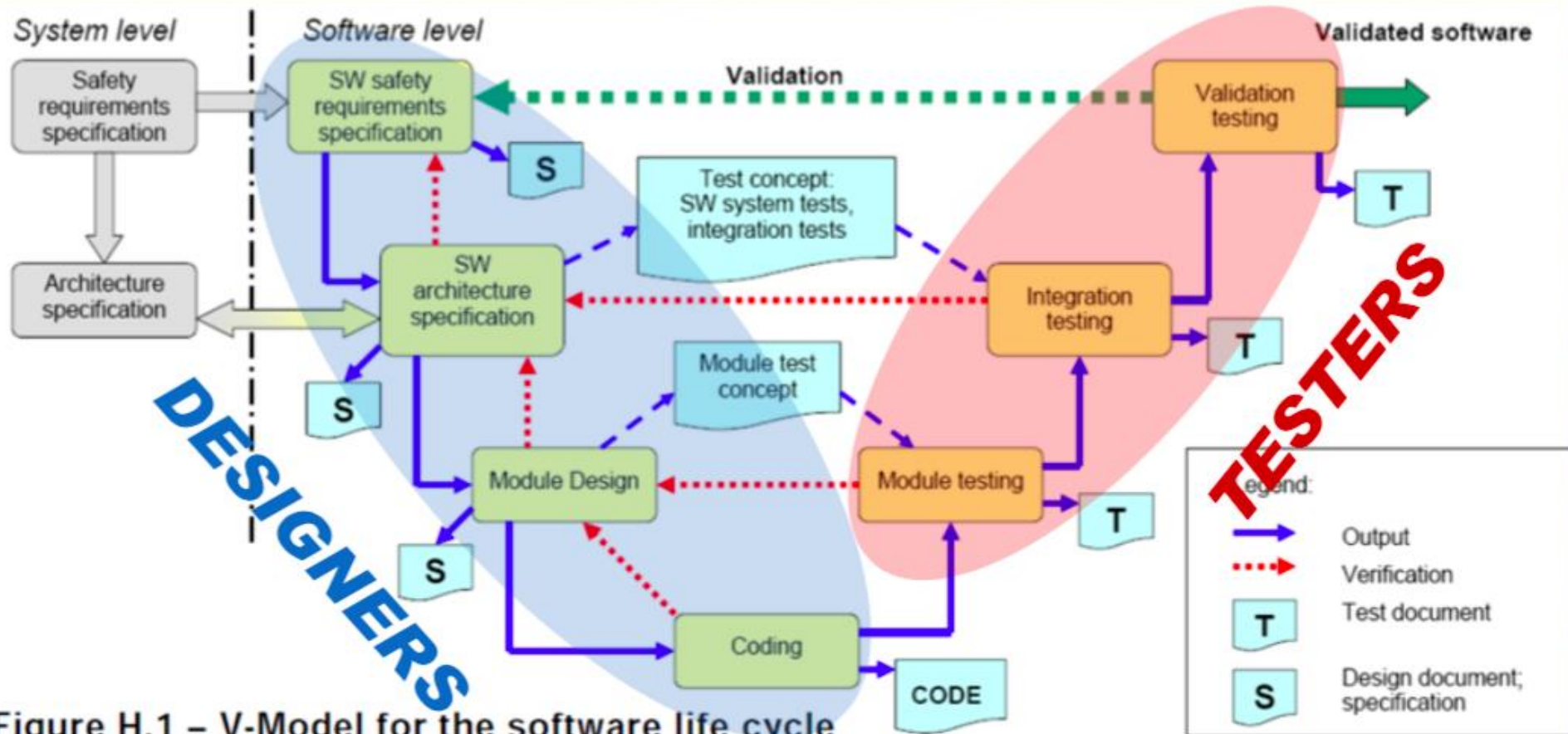
## Will "Diesel-Gate" Kill VW?

# Embedded System HW/SW Design Cycle





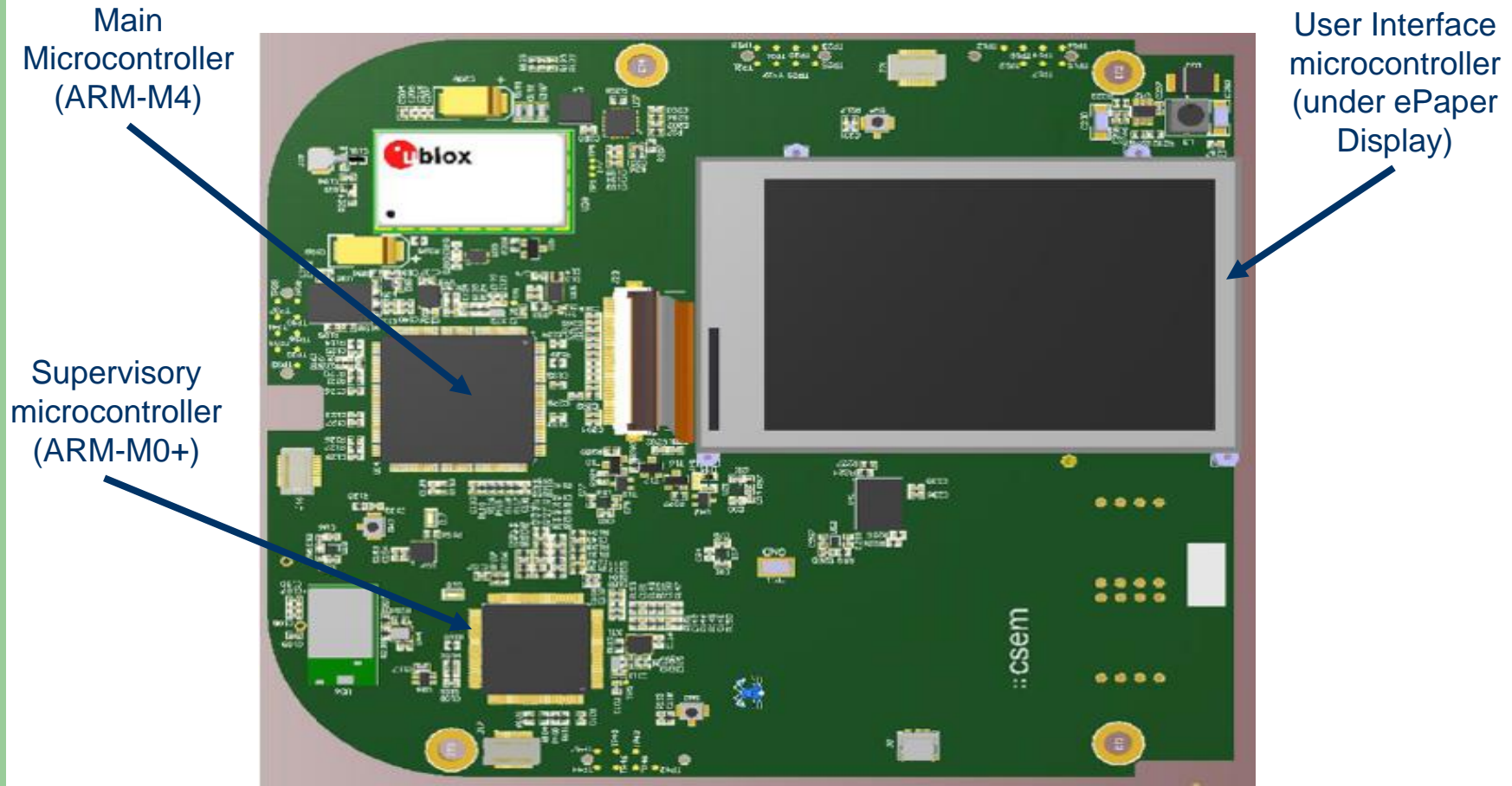
# The V-model for Embedded SW development



IEC 2510/13

## IEC 60730 Appliance Safety

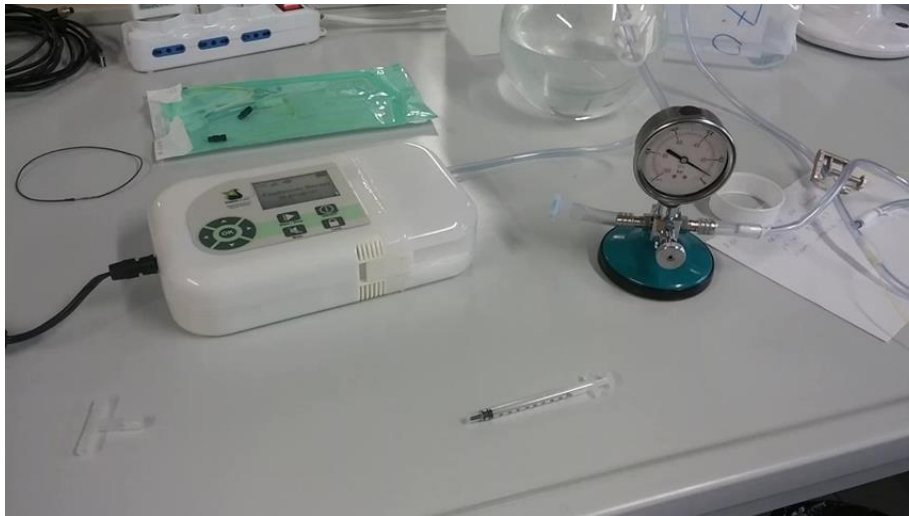
# Proof of concepts: Smart Negative Pressure Medical Device Programming in Microlab



# SNPD (Smart Negative Pressure Device)

- ❑ Software stack from drivers up to user application
- ❑ No RTOS (Real Time Operating System)
- ❑ C and assembly programming

## SNPD real case: Pump control

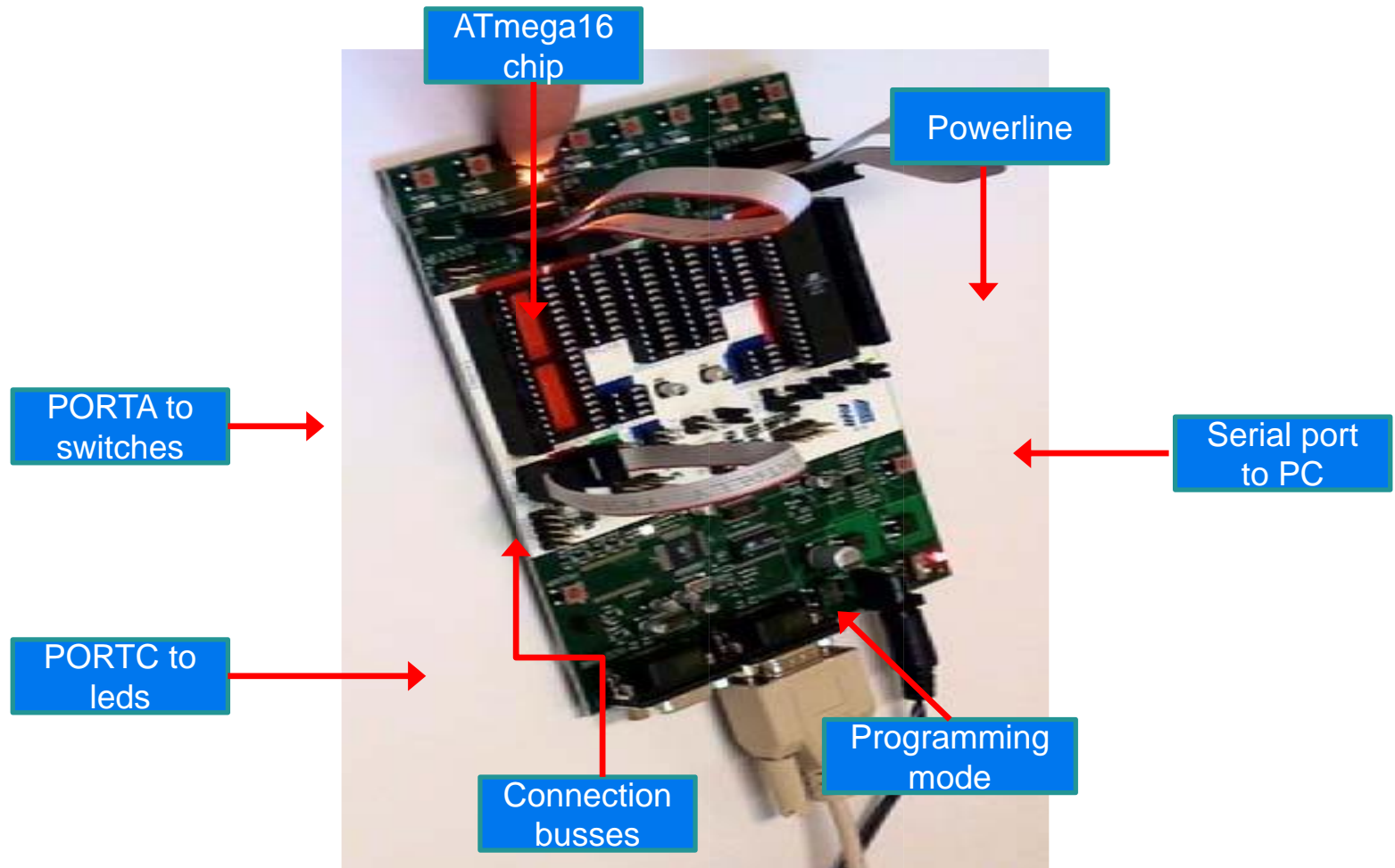


Verification in complex lab tubing system with pressure meter and on healthy volunteer

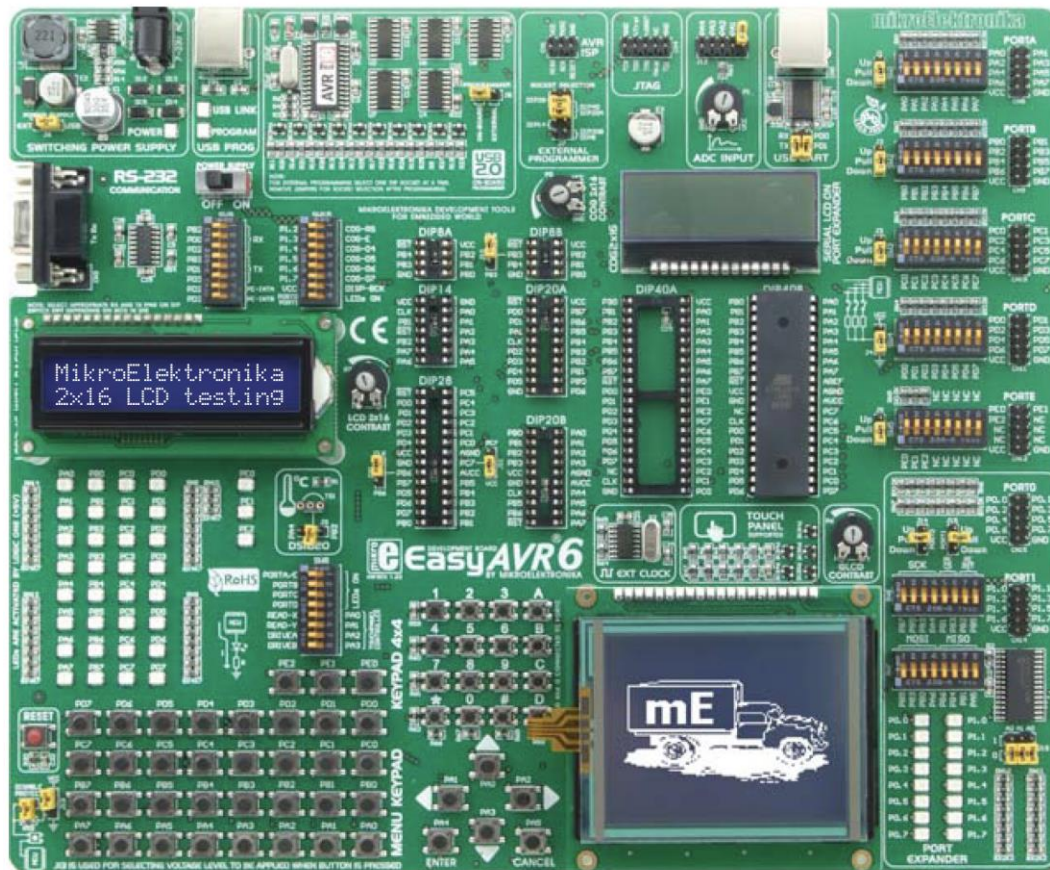




# AVR development kit



# EasyAVR6 Development System



Full-featured and user-friendly development board for AVR microcontrollers



High-Performance USB 2.0 On-Board Programmer



Port Expander provides easy I/O expansion (2 additional ports) using data format conversion



Alphanumeric On-Board 2x16 LCD with Serial Communication



Graphic LCD with backlight

# Χρήσιμα 'tips' για Embedded AVR-C

## Θέματα που θα μελετηθούν:

- AVR I/O
- Παραδείγματα Προγραμματισμού σε AVR-C
- Volatile variables
- AVR Memory space
- Stack and function call

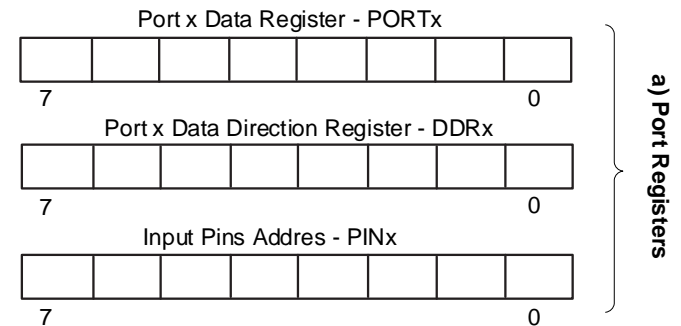
# I/O από AVR θύρες

## Output:

```
DDRB=0xf0;    // PORTB[7:4] as output,  
              // set PORTB[3:0] as input  
PORTB=0x00;   // disable PORTB pull-up resistors  
  
DDRC=0xff;    // set PORTC as output  
PORTC=0x00;   // initialize low
```

## Input:

```
Unsigned char new_PORTB; // PORTB new value  
:  
:  
:  
:  
new_PORTB = PINB;       // read PORTB
```



DDRxn	PORTxn	I/O	Comments	Pullup
0	0	input	Tri-state (Hi-Z)	No
0	1	input	Pullup	Yes
1	0	output	Output Low	No
1	1	output	Output High	No

x: port (A, B, C, D)  
n: pin (0 – 7)

## b) port pin configuration

# Ανάγνωση σε και εγγραφή από θύρες

```
#include <mega16.h> // Αρχείο επικεφαλίδας για τον AVR ATmega16
// το περιεχόμενο του αρχείο επικεφαλίδας περιλαμβάνει δηλώσεις του τύπου
// sfrb PORTB = 0x16, sfrb DDRB = 0x17, sfrb DDRD = 0x17,(special function register byte)
// δηλ. όλες τις διευθύνσεις των I/O καταχωρητών των περιφερειακών
// και τα διανύσματα διακοπών όλων των περιφερειακών

void main(void)
{

    DDRB=0xFF;           // Αρχικοποίηση της θύρας B σαν έξοδο
    DDRD=0x00;           // Αρχικοποίηση της θύρας D σαν είσοδο
    while (1)
    {
        PORTB=PIND + 5;  // Τα bits της θύρας PIND (switches) διαβάζονται και
                          // αφού προστεθούν με το 5 γράφονται στην PORTB (leds)
                          // Υποθέτουμε απεικόνιση με θετική λογική.
    };
}
```



# SFRB & SFRW compiler directives

- The SRAM area (Attention not the registers of the processor) of the AVR microcontroller includes a region called the Register File.
- The region contains IO ports, timers, and other peripherals, as well as some "working" or 'scratchpad' area.
- To instruct the compiler to allocate a variable to a register or registers, the storage class modifier register must be used.

## sfrb and sfrw

The I/O ports and peripherals are located in the register file. Therefore, special instructions are used to indicate to the compiler the difference between an SRAM location, an I/O port, or another peripheral in the register file.

The `sfrb` and `sfrw` keywords indicate to the compiler that the `IN` and `OUT` assembly instructions are to be used to access the AVR microcontroller's I/O registers:

```
/* Define the Special Function Registers (SFR)*/
```

```
sfrb PINA=0x19;           // 8-bit access to the SFR input port A
```

# Όλες τις διευθύνσεις των I/O καταχωρητών των περιφερειακών

Address	Name	Address	Name
\$3F (\$5F)	SREG	\$1F (\$3F)	EEARH-EEPROM Addr register high byte
\$3E (\$5E)	SPH	\$1E (\$3E)	EEARL-EEPROM Addr register low byte
\$3D (\$5D)	SPL	\$1D (\$3D)	EEDR - EEPROM Data Register
\$3C (\$5C)	OCR0 Timer/Counter0 -Output Compare Register	\$1C (\$3C)	EECR - EEPROM Control Register
\$3B (\$5B)	GICR Gener. Interrupt Control	\$1B (\$3B)	PORTA
\$3A (\$5A)	GIFR General Interrupt Flags	\$1A (\$3A)	DDRA
\$39 (\$59)	TIMSK Timers Interrupt Mask	\$19 (\$39)	PINA
\$38 (\$58)	TIFR Timers Interrupt Flags	\$18 (\$38)	PORTB
\$37 (\$57)	SPMCR Store Prog Con. Reg.	\$17 (\$37)	DDRB
\$36 (\$56)	TWCR - TWI Control Register	\$16 (\$36)	PINB
\$35 (\$55)	MCUCR Processor General Control Register	\$15 (\$35)	PORTC
\$34 (\$54)	MCUCSR (Pr Status Register)	\$14 (\$34)	DDRC
\$33 (\$53)	TCCR0 Timer0 Control Reg.	\$13 (\$33)	PINC
\$32 (\$52)	TCNT0Timer/Counter0	\$12 (\$32)	PORTD
\$31 (\$51)	OSCCAL Osc. Calibration Reg	\$11 (\$31)	DDRD
\$30 (\$50)	SFIOR	\$10 (\$30)	PIND
\$2F (\$4F)	TCCR1A Timer1 Control Register A	\$0F (\$2F)	SPDR - SPI Data Register
\$2E (\$4E)	TCCR1B Timer1 Control Register B	\$0E (\$2E)	SPSR - SPI Status Register
\$2D (\$4D)	TCNT1H Timer/Counter1-Counter Register High Byte	\$0D (\$2D)	SPCR - SPI Control Register
\$2C (\$4C)	TCNT1L Timer/Counter1-Counter Register Low Byte	\$0C (\$2C)	UDR USART I/O Data Register
\$2B (\$4B)	OCR1AH Timer/Counter1-Output Compare Register A	\$0B (\$2B)	UCSRA-USARTControl Status Register A
\$2A (\$4A)	OCR1AL Timer/Counter1-Output Compare Register A	\$0A (\$2A)	UCSRB-USARTControl Status Register B
\$29 (\$49)	OCR1BH Timer/Counter1-Output Compare Register B	\$09 (\$29)	UBRR - USART Baud Rate Register Low Byte
\$28 (\$48)	OCR1BL Timer/Counter1-Output Compare Register B	\$08 (\$28)	ACSR - Analog Control Status Register
\$27 (\$47)	ICR1H Timer/Counter1 Input Capture Register High Byte1	\$07 (\$27)	ADMUX - ADC Multiplexer Selection Register
\$26 (\$46)	ICR1L Timer/Counter1 Input Capture Register Low Byte1	\$06 (\$26)	ADCSRA - ADC Control Status Register
\$25 (\$45)	TCCR2 Timer2 Control Reg.	\$05 (\$25)	ADCH - ADC Data Register High Byte
\$24 (\$44)	TCNT2 Timer/Counter2 (8 bit)	\$04 (\$24)	ADCL - ADC Data Register Low Byte
\$23 (\$43)	OCR2Timer/Counter2 Output Compare Register	\$03 (\$23)	TWDR Two-wire Serial - Interface Data Register
\$22 (\$42)	ASSR Asynchr. Status Reg.	\$02 (\$22)	TWAR Two-wire Address Register
\$21 (\$41)	WDTCSR- WatchDog Con. Reg	\$01 (\$21)	TWSR Two-wire Status Register
\$20 (\$40)	UBRRH/URSEL (USART Baud Rate Register High Byte)	\$00 (\$20)	TWBR Two-wire bit Rate Register

# ΠΑΡΑΔΕΙΓΜΑ 1ο: I/O και Χειρισμός bit

## Τα 3 LSB της θύρας PORTD στα 3 MSB της θύρας PORTB

```
#include <mega16.h> // Αρχείο επικεφαλίδας για τον AVR ATmega16
unsigned char z;
void main(void)
{
    DDRB = 0xFF;           // Αρχικοποίηση της θύρας B ως έξοδο
    DDRD = 0x00;           // Αρχικοποίηση της θύρας D ως είσοδο
    while (1)
    {
        z = PIND & 0x7;    // Τα bits της θύρας PIND(switches) διαβάζονται
                           // και απομονώνονται τα 3 τελευταία bits (masking)
        z << 5;            // ολίσθηση 5 θέσεις αριστερά και
                           // τα 3 LSB πάνε στα 3 MSB
        z = z & 0xE0;      // Τα bits της θύρας PORTB(leds) διαβάζονται,
                           // μηδενίζονται τα 3 MSB και στη συνέχεια μπαίνουν τα
        PORTB |= z;        // αντίστοιχα του PORTD αφού γίνουν OR με το z,
                           // γράφονται ξανά πίσω στην PORTB(leds).
    };
}
```

# ΠΑΡΑΔΕΙΓΜΑ 2ο: Blink LED on bit PortB0 at a rate of 1Hz.

```
#include <mega16.h>
#include <delay.h>

void main()
{
    DDRB = 0xFF;          //PB as output
    PORTB= 0x00;          //keep all LEDs off

    while(1)
    {
        PORTB &= 0b11111110;    //turn LED off
        delay_ms(500);    //wait for half second
        PORTB |= 0b00000001;    //turn LED on
        delay_ms(500);    //wait for half second
    };
}
```

## ΠΑΡΑΔΕΙΓΜΑ 2ο: Διαφορά πλήθους μονάδων από το πλήθος των μηδενικών της θύρας PORTD

```
#include <mega16.h> // Αρχείο επικεφαλίδας για τον AVR ATmega16
unsigned char x_p, x_m, y, z; //
void main(void)
{
    int i;
    DDRB=0xFF; // Αρχικοποίηση της θύρας B ως έξοδο
    DDRD=0x00; // Αρχικοποίηση της θύρας D ως είσοδο
    while (1) // Ατέρμων βρόγχος. Η συνθήκη είναι πάντα αληθής.
    {
        z = PIND; // Διαβάζονται τα bits της θύρας PIND (switches)
        x = 0; // Αρχικοποίηση του μετρητή των '1' – '0'.
        for(i = 0; i < 8; i++) // Βρόγχος με 8 επαναλήψεις
        {
            // Απομονώνεται το LSB του z.
            y = z & 0x1; // Η εντολή αυτή μπορεί να παραληφθεί και να πάει στην if.
            z >> 1; // ολίσθηση του καταχωρητή z, 1 θέση δεξιά
            if (y = 1) // Δομή ελέγχου για διακλάδωση της ροής
                x_p = x_p + 1; // προγράμματος με βάση την τιμή του y
            else
                x_m = x_m + 1; // Μετρητής ψηφίων 0
        }
        if (x_p > x_m)
            PORTB = (x_p - x_m) ^ 0xFF; // Τα bits της θύρας PORTB (leds)
        else
            PORTB = (x_m - x_p) ^ 0xFF; // γράφονται με το συμπλήρωμα ως προς 1 του z
            // Υποθέτουμε απεικόνιση με αρνητική λογική.
    };
}
```

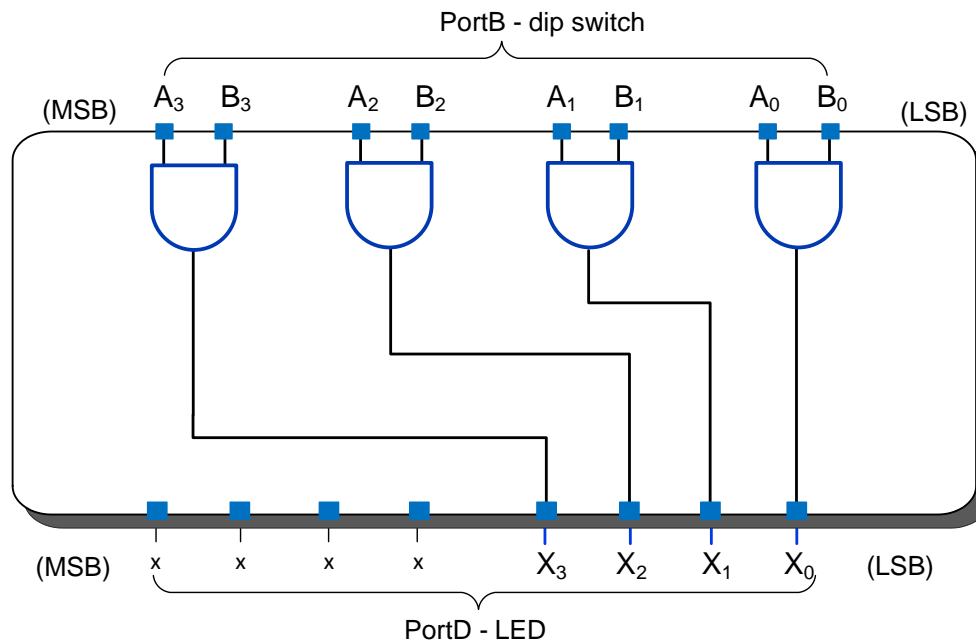


## ΠΑΡΑΔΕΙΓΜΑ 3ο: Με βάση τον αριθμό (0-7) από τα 3 LSB του PORTD να ανάβει το αντίστοιχης τάξης bit του PORTB

```
#include <mega16.h> // Αρχείο επικεφαλίδας για τον AVR ATmega16
unsigned char x, y, z;
void main(void)
{
    int i;
    DDRB=0xFF; // Αρχικοποίηση της θύρας B ως έξοδο
    DDRD=0x00; // Αρχικοποίηση της θύρας D ως είσοδο
    while (1) // Ατέρμων βρόγχος. Η συνθήκη είναι πάντα αληθής.
    {
        z = PIND & 0x7; // Διαβάζονται τα bits της θύρας PIND (switches)
        x = 1; // Αρχικοποίηση του καταχωρητή x.
        if(z > 0)
        {
            for(i = 0; i < z; i++) // Βρόγχος με z επαναλήψεις
            {
                x << 1; // ολίσθηση του καταχωρητή x, 1 θέση αριστερά
            }
        }
        PORTB = x ^ 0xFF; // Τα bits της θύρας PORTB (leds)
                          // γράφονται με το συμπλήρωμα ως προς 1 του z
                          // Υποθέτουμε απεικόνιση με αρνητική λογική.
    }
}
```

# ΠΑΡΑΔΕΙΓΜΑ 4ο: Εξομοίωση πυλών

Να εξομοιωθεί σε C η λειτουργία ενός υποθετικού I.C. που περιλαμβάνει 4 πύλες AND όπως φαίνεται στο σχήμα. Τα bits εισόδου πρέπει να αντιστοιχούν ακριβώς όπως φαίνονται στο σχήμα με τα dip switches της πόρτας εισόδου PortB, και οι έξοδοι με τα LEDs που πρέπει να είναι τα τέσσερα LSB της πόρτας εξόδου PortD ενός Μικροελεγκτή AVR. Υποθέτουμε θετική λογική.



# ΠΑΡΑΔΕΙΓΜΑ 4ο: Εξομοίωση πυλών

```
#include <mega16.h> // Αρχείο επικεφαλίδας για τον AVR ATmega16
unsigned char x, y, z;
void main(void)
{
    int i;
    DDRB=0x00; // Αρχικοποίηση της θύρας B ως είσοδο
    DDRD=0xFF; // Αρχικοποίηση της θύρας D ως έξοδο
    while (1) // Ατέρμων βρόγχος. Η συνθήκη είναι πάντα αληθής.
    {
        x = PINB; // Διαβάζονται τα 8 bits της θύρας PINB (switches)
        z = 0; // Αρχικοποίηση του καταχωρητή z
        for(i = 0; i < 4; i++) // Βρόγχος με 4 επαναλήψεις
        {
            y=x & 0x3; // Απομονώνω τα 2 LSB
            if(y = 3)
            {
                z = z | 0x10; // Θέτει '1' αν η κάθε πύλη δίνει αυτή τη τιμή
            }
            z >> 1; // ολίσθηση του καταχωρητή z, 1 θέση δεξιά
            x >> 2; // ολίσθηση του καταχωρητή x, 2 θέσεις δεξιά
        }
        PORTD = z & 0xF; // Το αποτέλεσμα στα 4 LSbits της θύρας PORTD (leds)
    }; // Υποθέτουμε απεικόνιση με θετική λογική.
}
```

# Volatile variables

Η δεσμευμένη λέξη `volatile` είναι ένας ANSI-C προσδιοριστής μεταβλητών για κώδικες οδήγησης περιφερειακών και διαχείρισης διακοπών. Κάθε δεδομένο που η τιμή του δύναται να ενημερώνεται με ασύγχρονο τρόπο και όχι μέσα από κώδικα του μικροεπεξεργαστή πρέπει να δηλώνεται ως `volatile`.

```
#define PER_PORT 0x1775 // διεύθυνση θύρα εισόδου  
// ενός περιφερειακού με 16-bit είσοδο
```

...

```
volatile int *port = PER_PORT; // Δείκτης χειρισμού της θύρας  
int data_a, data_b;
```

```
data_a = *port ;  
data_b = *port ;
```

// 1η Ανάγνωση PER\_PORT  
// 2η Ανάγνωση PER\_PORT

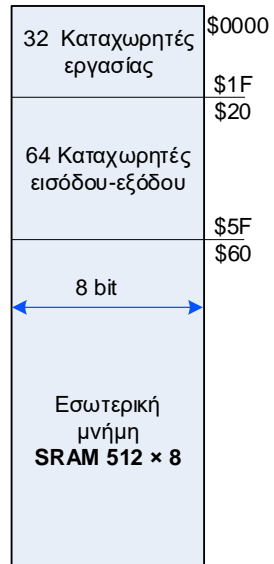
Η PORT θα 'διαβαστεί'  
δύο φορές σε  
περίπτωση όπου η  
τιμή της έχει αλλάξει  
στο ενδιαμέσο.

Χωρίς τη δήλωση του δείκτη `port` ως `volatile`, ο μεταγλωττιστής θα συμπεραίνει την εξής συμπεριφορά: `data_a = data_b = *port` (εξοικονομώντας έτσι μια θέση μνήμης), κάτι το οποίο δεν ισχύει στην προκειμένη περίπτωση καθώς στη δεύτερη ανάγνωση της θύρας το δεδομένο μπορεί να έχει αλλάξει.

Στην περίπτωση που  
**data\_a** και **data\_b** δεν  
δηλωθούν `volatile` τότε η  
PORT θα 'διαβαστεί' μόνο  
μια φορά θεωρώντας ότι:  
**data\_a = data\_b**

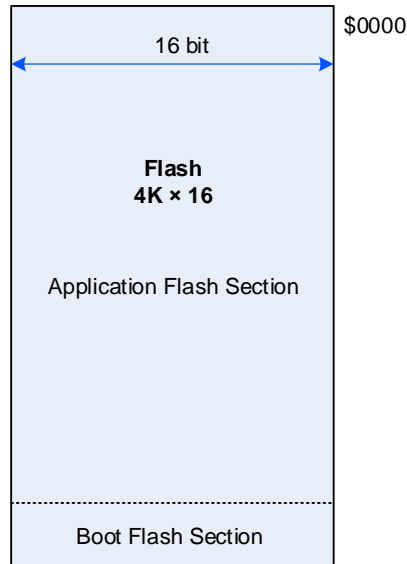
# AVR memory space

Μνήμη δεδομένων 8-bit



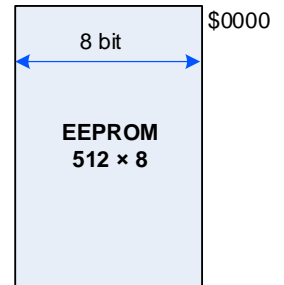
Τελική διεύθυνση: \$025F

Μνήμη προγράμματος 16-bit



Τελική διεύθυνση: \$0FFF

Μνήμη EEPROM 8-bit



Τελική διεύθυνση: \$01FF

## AVR Memory Segments

- Data Memory Segment (.DSEG)
- Program Memory Segment (.CSEG)
- EEPROM Memory (.ESEG)

## Data/Variable Allocation Directives:

- **Variables in .DSEG**
- **Constants in .CSEG or .ESEG**  
(προκειμένου η τιμή να παραμένει και μετά το power off )



# ΠΙΝΑΚΑΣ ΜΕ ΤΑ AVR ASSEMBLER DIRECTIVES

DIRECTIVES για  
μετάφραση και  
υλοποίηση τύπων  
δεδομένων σε AVR  
ASSEMBLY

ΠΕΡΙΣΣΟΤΕΡΕΣ ΠΛΗΡΟΦΟΡΙΕΣ:

[[http://www.atmel.com/webdoc/avrasmblr/avrasmblr.wb\\_directives.html](http://www.atmel.com/webdoc/avrasmblr/avrasmblr.wb_directives.html)]

Directive	Description
BYTE	Reserve byte to a variable
CSEG	Code Segment
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DEVICE	Define ...nch device to assemble for
DSEG	Data Segment
DW	Define constant word(s)
ENDMACRO	End macro
EQU	Set a symbol equal to an expression
ESEG	EEPROM Segment
EXIT	Exit from file
INCLUDE	Read source from another file
LIST	Turn listfile generation on
LISTMAC	Turn macro expansion on
MACRO	Begin macro
NOLIST	Turn listfile generation off
ORG	Set program origin
SET	Set a symbol to an expression

# C to AVR Assembly: allocation and placements of constants and variables

## C Source Code

```
int a;  
const char b[ ]="COMP9032";  
const int c=9032;
```



## AVR Assembly

```
.dseg  
.org 0x100  
a: .byte 4  
  
.cseg  
b: .DB 'C', 'O', 'M', 'P', '9', '0', '3', '2', 0  
c: .DW 9032
```

- Μεταβλητές → .dseg
- σταθερές → .cseg ( ΓΙΑΤΙ ? )

# More on data/variable allocations

The default or automatic allocation of variables, where no memory descriptor keyword is used, is in SRAM. Constants can be placed in FLASH memory (program space) with the `flash` or `const` keywords. For variables to be placed in EEPROM, the `eeeprom` keyword is used.

## Example of SRAM data allocation in C

```
char mystring[30] = "This string is placed in SRAM";
```

## Example of FLASH data allocation in C

```
flash char string_constant1[] = "This is a string constant";  
const char string_constant2[] = "This is also a string constant";
```

## Example of EEPROM data allocation in C

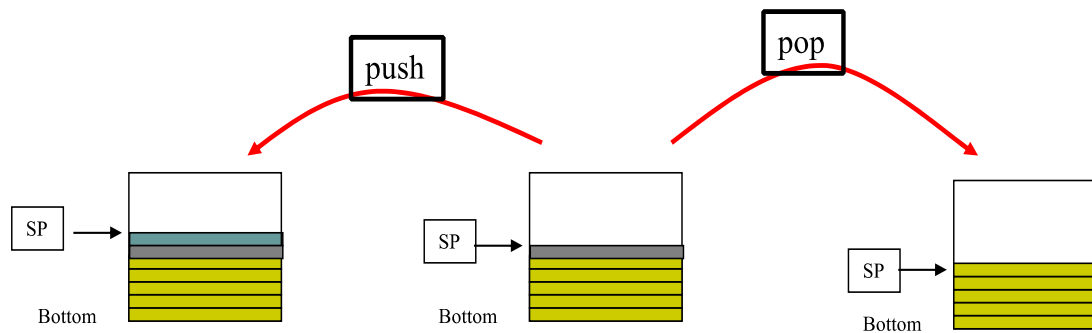
```
eeeprom int cycle_count;      // allocates an integer space in EEPROM  
eeeprom char ee_string[20];   // allocates a 20-byte area in EEPROM  
  
eeeprom struct {  
    char a;  
    int  b;  
    char c[15];  
} se;                          // allocates 18-byte structure "se" in EEPROM
```

# GCC Assembler vs Atmel AVR Assembler

GCC ASSEMBLER	ATMEL AVR ASSEMBLER
#INCLUDE	.INCLUDE
ENABLE DATA INITIALIZATION IN .DATA SEGMENT	DATA INITIALIZATION NOT ALLOWED IN .DSEG
HI8()	HIGH()
LO8()	LOW()
.ASCIZ	.DB
.SECTION .DATA	.DSEG
.SECTION .TEXT	.CSEG
<AVR/IO.H>	"M16DEF.INC"

# Avr Stack

- ❑ Στους επεξεργαστές AVR, η στοίβα (stack) υλοποιείται από ένα σύνολο συνεχόμενων BYTES στην μνήμη SRAM.
- ❑ Stack pointer (SP): I/O register SPH:SPL
- ❑ Η στοίβα χρησιμοποιείται στην υλοποίηση κλήσεων συναρτήσεων.



## POP instruction

- Syntax: **pop Rd**
- Operands:  $Rd \in \{r0, r1, \dots, r31\}$
- Operation:  $SP \leftarrow SP + 1$   
 $Rd \leftarrow (SP)$
- Words: 1
- Cycles: 2

## PUSH instruction

- Syntax: **push Rr**
- Operands:  $Rr \in \{r0, r1, \dots, r31\}$
- Operation:  $(SP) \leftarrow Rr$   
 $SP \leftarrow SP - 1$
- Words: 1
- Cycles: 2



# Conventions on register usage

- **Call-used registers (r18-r27, r30-r31):** May be allocated by gcc for local data. You *may* use them freely in assembler subroutines. Calling C subroutines can clobber any of them - **the caller is responsible for saving and restoring.**
- **Call-saved registers (r2-r17, r28-r29):** May be allocated by gcc for local data. Calling C subroutines leaves them unchanged. **Assembler subroutines are responsible for saving and restoring these registers, if changed.**
  - r29:r28 (Y pointer) is used as a frame pointer (points to local data on stack) if necessary. The requirement for the callee to save/preserve the contents of these registers even applies in situations where the compiler assigns them for argument passing.
- **Fixed registers (r0, r1):**  
Never allocated by gcc for local data, but often used for fixed purposes:

# Function call conventions

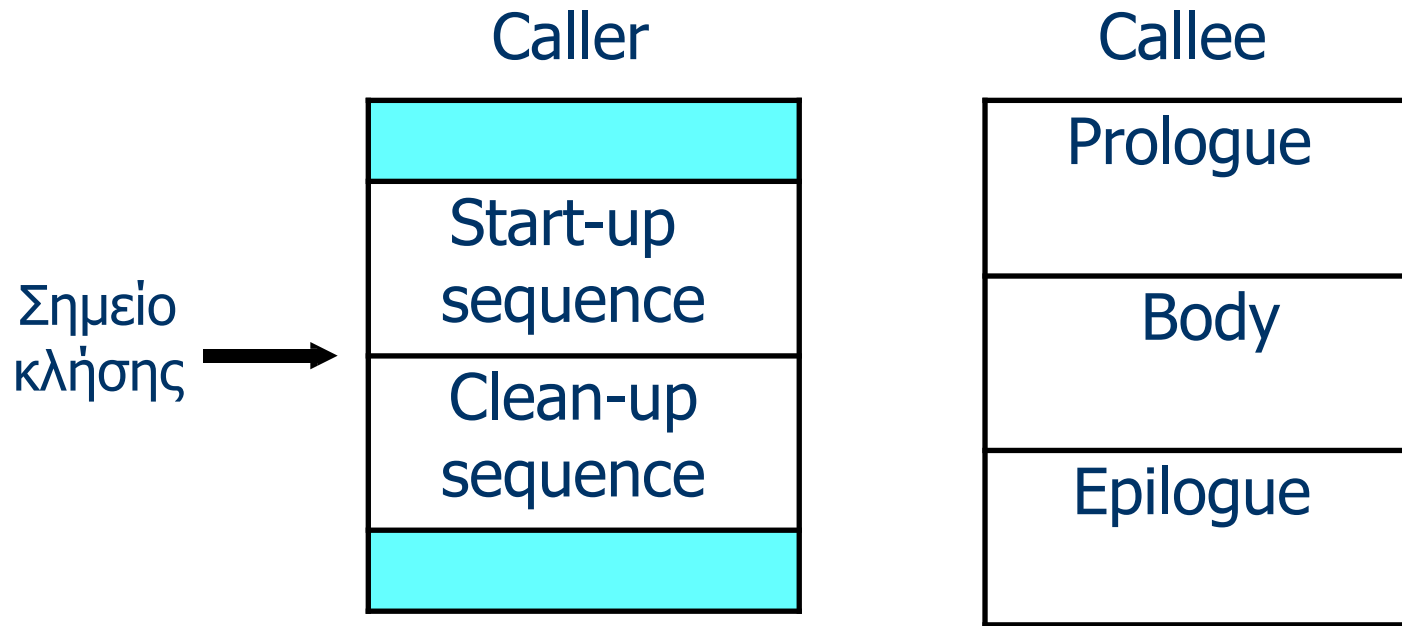
- **Arguments:** allocated left to right, **r25 to r8**.
  - **All arguments are aligned to start in even-numbered registers** (odd-sized arguments, including char, have one free register above them). This allows making better use of the movw instruction on the enhanced core.
  - If too many, those that don't fit are passed on the stack.
- **Return values:**
  - 8-bit in r24 (not r25!),
  - 16-bit in r25:r24,
  - up to 32 bits in r22-r25,
  - up to 64 bits in r18-r25.
  - 8-bit return values are zero/sign-extended to 16 bits by the called function
  - Arguments to functions with variable argument lists (printf etc.) are all passed on stack, and char is extended to int.

# Deprecated calling conventions

- ❑ Οι registers **r31:r30:r27:r26** → Επιστροφή αποτελέσματος συνάρτησης ανάλογα με τον τύπο δεδομένων (char, short, int).
- ❑ Register **y=r29:r28** → stack frame pointer (δείκτης στο τοπικό stack frame κάθε συνάρτησης).
- ❑ Οι conflict registers του CALLER αποθηκεύονται στη στοίβα.
  - Register conflicts → τουλάχιστον ένας από τους καταχωρητές του AVR χρησιμοποιείται και στη συνάρτηση CALLER και στη συνάρτηση CALLEE.

**ΠΡΟΣΟΧΗ**  
Το βιβλίο σας θεωρεί αυτά τα  
calling conventions

# Κλήση Διαδικασίας



# A Template for Caller

- Caller:
  - Before calling the callee, store actual parameters in designated registers.
  - Call the callee.
    - Using instructions for subroutine call rcall, ical, call.

# A Template for Callee (1/2)

- Prologue:
  - Store conflict registers, including the stack frame register Y, on the stack by using *push* instruction
  - Reserve space for local variables and passing parameters
  - Update the stack pointer and stack frame pointer Y to point to the top of its stack frame
  - Pass the actual parameters to the formal parameters on the stack
- Function body:
  - Do the normal task of the function on the stack frame and general purpose registers.



# A Template for Callee (2/2)

- Epilogue:
  - Store the return value in designated registers r25:r24.
  - De-allocate the stack frame
  - Deallocate the space for local variables and parameters by updating the stack pointer SP.  
SP=SP + the size of all parameters and local variables.  
Using *OUT* instruction
  - Restore conflict registers from the stack by using *pop* instruction
    - The conflict registers must be popped in the reverse order that they are pushed on the stack. The stack frame register of the caller is also restored.
  - Return to the caller by using *ret* instruction

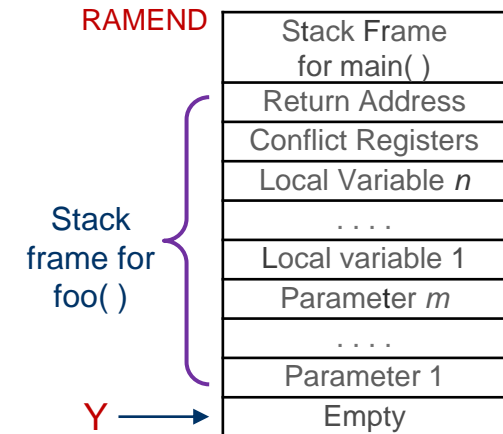
# Stack frame

## Stack frame:

- Return address
- Conflict registers
- Local variables
- Parameters (arguments)

```
int main(void)
{ ...
  foo(arg1, arg2, ..., argm);
}

void foo(arg1, arg2, ..., argm)
{
  int var1, var2, ..., varn;
  ...
}
```



# Παράδειγμα κλήσης συνάρτησης στη γλώσσα C

```
// int parameters b & e,  
// returns an integer  
  
unsigned int pow(unsigned int b, unsigned int e)  
{  
    unsigned int i, p;      // local variables  
    p = 1;  
    for (i = 0; i < e; i++) // p = be  
        p = p * b;  
    return p;               // return value of the function  
}  
  
int main(void)  
{  
    unsigned int m, n;  
    m = 2;  
    n = 3;  
    m = pow(m, n);  
    return 0;  
}
```

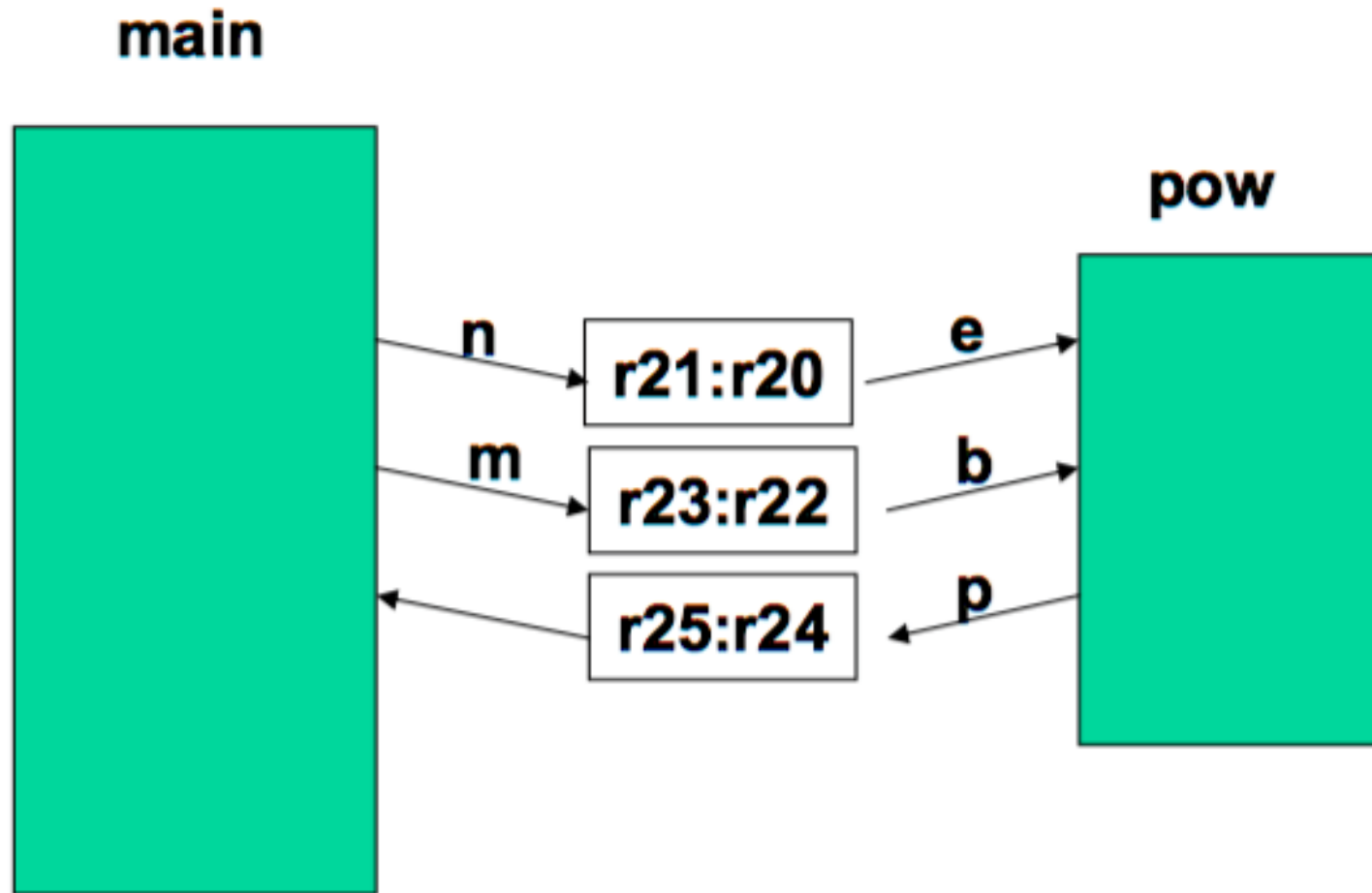
# Παράδειγμα κλήσης συνάρτησης στη γλώσσα C

```
// int parameters b & e,  
// returns an integer  
  
unsigned int pow(unsigned int b, unsigned int e)  
{  
    unsigned int i, p;    // local variables  
    p = 1;  
    for (i = 0; i < e; i++) // p = be  
        p = p * b;  
    return p;    // return value of the function  
}  
  
CALLER  
int main(void)  
{  
    unsigned int m, n;  
    m = 2;  
    n = 3;    CALLEE  
    m = pow(m, n);  
    return 0;  
}
```

Parameter Type → Passing By Value

Return Value

# Parameter passing



# Translating C to AVR assembly

```
.include "m64def.inc"
```

```
.def zero = r15 ; to store constant value 0
```

```
; Macro mul2: multiplication of two 2-byte unsigned numbers with the  
; results of 2-bytes.
```

```
; All parameters are registers, @5:@4 should be in the form: rd+1:rd,  
; where d is the even number, and they are not r1 and r0
```

```
; Operation: (@5:@4) = (@1:@0)*(@3:@2)
```

```
.macro mul2 ; a * b
```

```
    mul  @0, @2 ; al * bl
```

```
    movw @5:@4, r1:r0
```

```
    mul  @1, @2 ; ah * bl
```

```
    add  @5, r0
```

```
    mul  @0, @3 ; bh * al
```

```
    add  @5, r0
```

```
.endmacro
```

; continued



# Translating C to AVR assembly – cont.

**; continued**

**main:**

**ldi r28, low(RAMEND-4)**  
**ldi r29, high(RAMEND-4)**  
**out SPH, r29**  
**out SPL, r28**

**; 4 bytes to store local variables,**  
**; assume an integer is 2 bytes;**  
**; Adjust stack pointer to point to**  
**; the new stack top.**

**; function body of the main**

**ldi r24, low(2)**  
**ldi r25, high(2)**  
**std Y+1, r24**  
**std Y+2, r25**

**; m = 2;**

**ldi r24, low(3)**  
**ldi r25, high(3)**  
**std Y+3, r24**  
**std Y+4, r25**

**; n=3;**

**; continued**

# Translating C to AVR assembly – cont.

**; continued**

**ldd r20, Y+3**

**ldd r21, Y+4**

**ldd r22, Y+1**

**ldd r23, Y+2**

**rcall pow**

**std Y+1, r24**

**std Y+2, r25**

**end:**

**rjmp end**

**; end of function main()**

**; prepare parameters for function call**

**; r21:r20 keep the actual parameter n**

**; r23:r22 keep the actual parameter m**

**; call subroutine pow**

**; get the return result**

**; continued**

# Translating C to AVR assembly – cont.

**; continued**

**pow:**

**; prologue:**

**push r28  
push r29  
push r16  
push r17  
push r18  
push r19  
push zero  
in r28, SPL  
in r29, SPH  
sbiw r29:r28, 8**

**; r29:r28 will be used as the frame pointer  
; save r29:r28 in the stack**

**; save registers used in the function body**

**; initialize the stack frame pointer value**

**; reserve space for local variables  
; and parameters.**

**; continued**

# Translating C to AVR assembly – cont.

**; continued**

**out SPH, r29  
out SPL, r28**

**; update the stack pointer to the new stack top**

**std Y+1, r22  
std Y+2, r23  
std Y+3, r20  
std Y+4, r21**

**; pass the actual parameters**

**; pass m to b**

**; pass n to e**

**; end of prologue**

**; continued**

# Translating C to AVR assembly – cont.

**; continued**

**; function body**

**clr zero**

**clr r23;**

**clr r22;**

**clr r21;**

**ldi r20, 1**

**...**

**ldd r25, Y+4**

**ldd r24, Y+3**

**ldd r17, Y+2**

**ldd r16, Y+1**

**; use r23:r22 for i and r21:r20 for p,  
; r25:r24 temporarily for e. and r17:r16 for b**

**;initialize i to 0**

**;initialize p to 1**

**;store the local values to the stack  
;if necessary**

**;load e to registers**

**;load b to registers**

**; continued**

# Translating C to AVR assembly – cont.

**; continued**

```
loop:    cp r22, r24                ; compare i with e
         cpc r23, r25
         brsh done                 ; if i >= e
         mul2 r20, r21, r16, r17, r18, r19 ; p *= b
         movw r21:r20, r19:r18
         ;std Y+8, r21              ; store p
         ;std Y+7, r20
         inc r22                   ; i++, (can we use adiw?)
         adc r23, zero
         ;std Y+6, r23              ; store i
         ;std Y+5, r22
         rjmp loop

done:
         ; ...                      ; save the local to the stack
         movw r25:r24, r21:r20      ; save the result
         ; end of function body

                                           ; continued
```



# Translating C to AVR assembly – cont.

**; continued**

**; Epilogue**

**;ldd r25, Y+8**

**; the return value of p is stored in r25,r24**

**;ldd r24, Y+7**

**adiw r29:r28, 8**

**; de-allocate the reserved space**

**out SPH, r29**

**out SPL, r28**

**pop zero**

**pop r19**

**pop r18**

**; restore registers**

**pop r17**

**pop r16**

**pop r29**

**pop r28**

**ret**

**; return to main()**

**; end of epilogue**

**; End**

# Πρόγραμμα διαχείρισης διακοπών σε C

Στη συνέχεια δίνεται ένα παράδειγμα προγράμματος διαχείρισης διακοπών σε C που επιτρέπει εξωτερικές διακοπές στην είσοδο INT0 του Μικροελεγκτή AVR. Αρχικά τα led της θύρας PORTB είναι ON και στη συνέχεια κάθε φορά που προκαλείται διακοπή αλλάζει η κατάσταση τους.

```
#include <mega16.h>
unsigned char chLed;

// External Interrupt 0 service routine. Η EXT_INT0 αντιστοιχεί στη θέση 02-03
interrupt [EXT_INT0] void ext_int0_isr(void)
{
    chLed = chLed ^ 0xFF;
    PORTB=chLed;
}

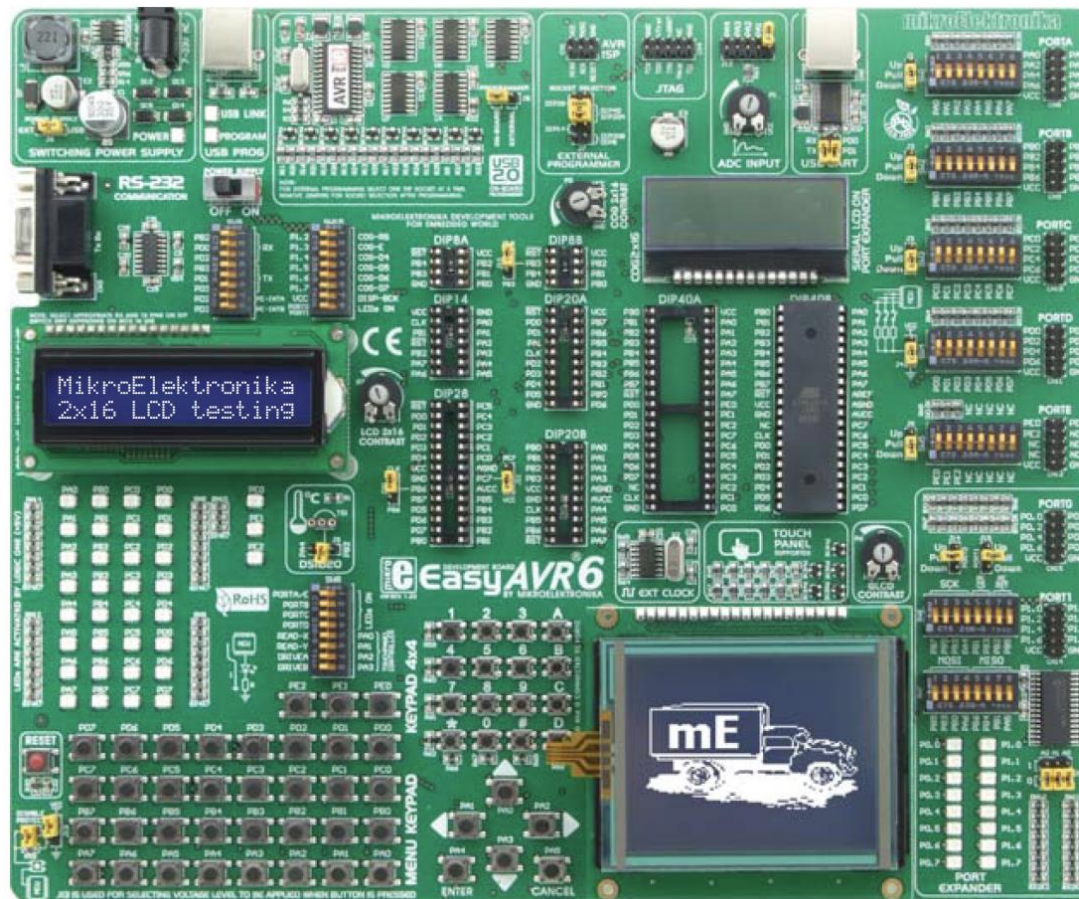
void main(void)
{
    PORTB=0x00;           // Η PORTB τίθεται ως έξοδος
    DDRB=0xFF;
    GIMSK=0x40;           // Ενεργοποίηση εξωτερικής διακοπής 0, INT0: On
    MCUCR=0x02;           // INT0 Mode: στην ακμή πτώσης
    #asm("sei")           // ενεργοποίηση συνολικά των διακοπών
    chLed=0x00;
    PORTB=chLed;
    while (1)
    {
        // Μπορούμε τοποθετώντας πρόσθετο κώδικα στο σημείο αυτό,
        // το κύριο πρόγραμμα να κάνει επιπλέον λειτουργίες εκτός του ON – OFF των led.
    };
}
```

# Αναφορές

- Κ. Πεκμεστζή, “Συστήματα Μικροϋπολογιστών – Μικροελεγκτής AVR και PIC”
- William Barnekow, “Mixing C and assembly language programs”, Lecture Notes, Cornell University
- “Embedded C Programming and the Atmel AVR”, R. Barnett, Thomson and Delmar Learning.
- Sri Parameswaran, Annie Guo, Hui Wu, “Assembly Programming (iii)”, Lecture Notes on Microprocessors and Interfacing, University of New South Wales
- <http://www.atmel.com/webdoc/avr assembler/>

- **Η ΑΝΑΠΤΥΞΙΑΚΗ  
ΠΛΑΚΕΤΑ EasyAVR6**

# EasyAVR6 Development System



Full-featured and user-friendly development board for AVR microcontrollers



High-Performance USB 2.0 On-Board Programmer



Port Expander provides easy I/O expansion (2 additional ports) using data format conversion



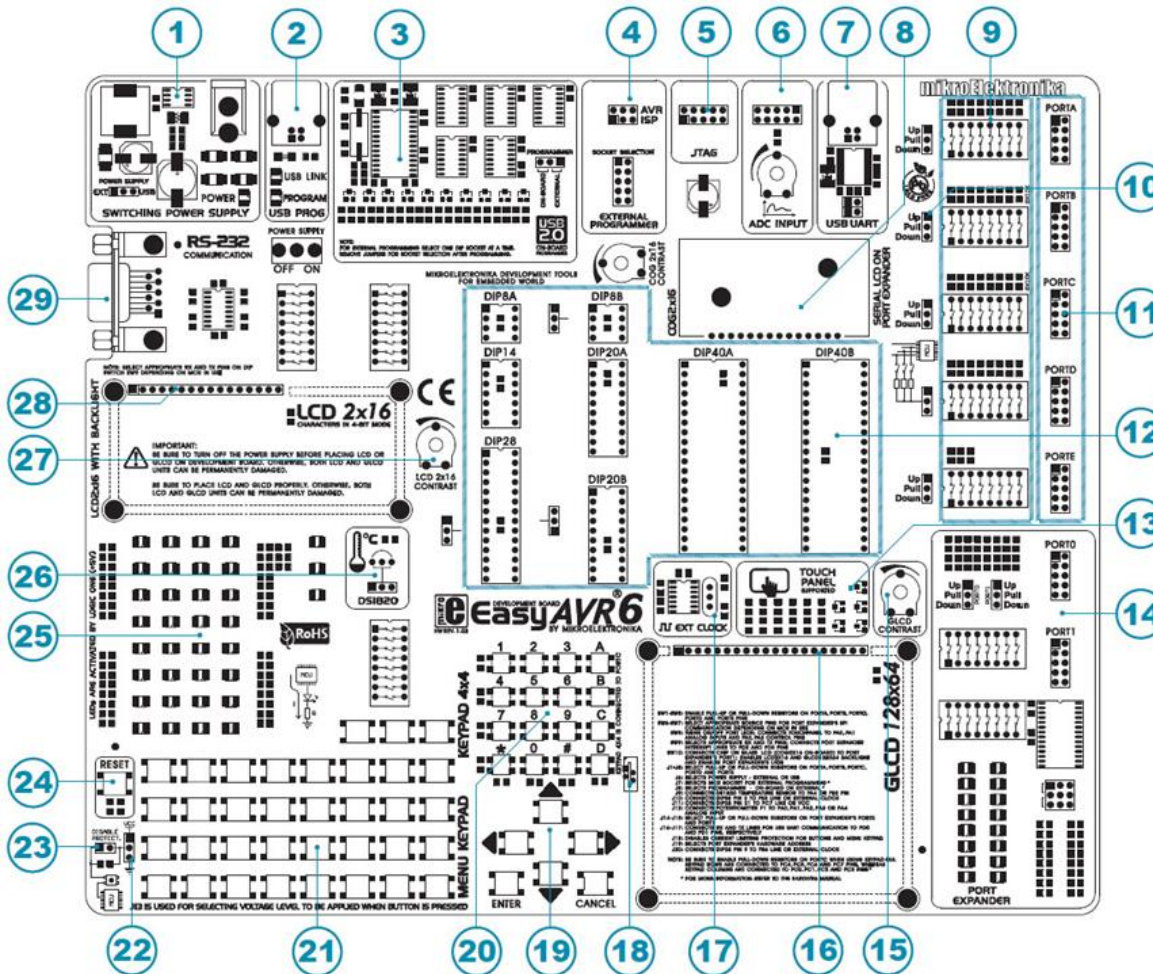
Alphanumeric On-Board 2x16 LCD with Serial Communication



Graphic LCD with backlight



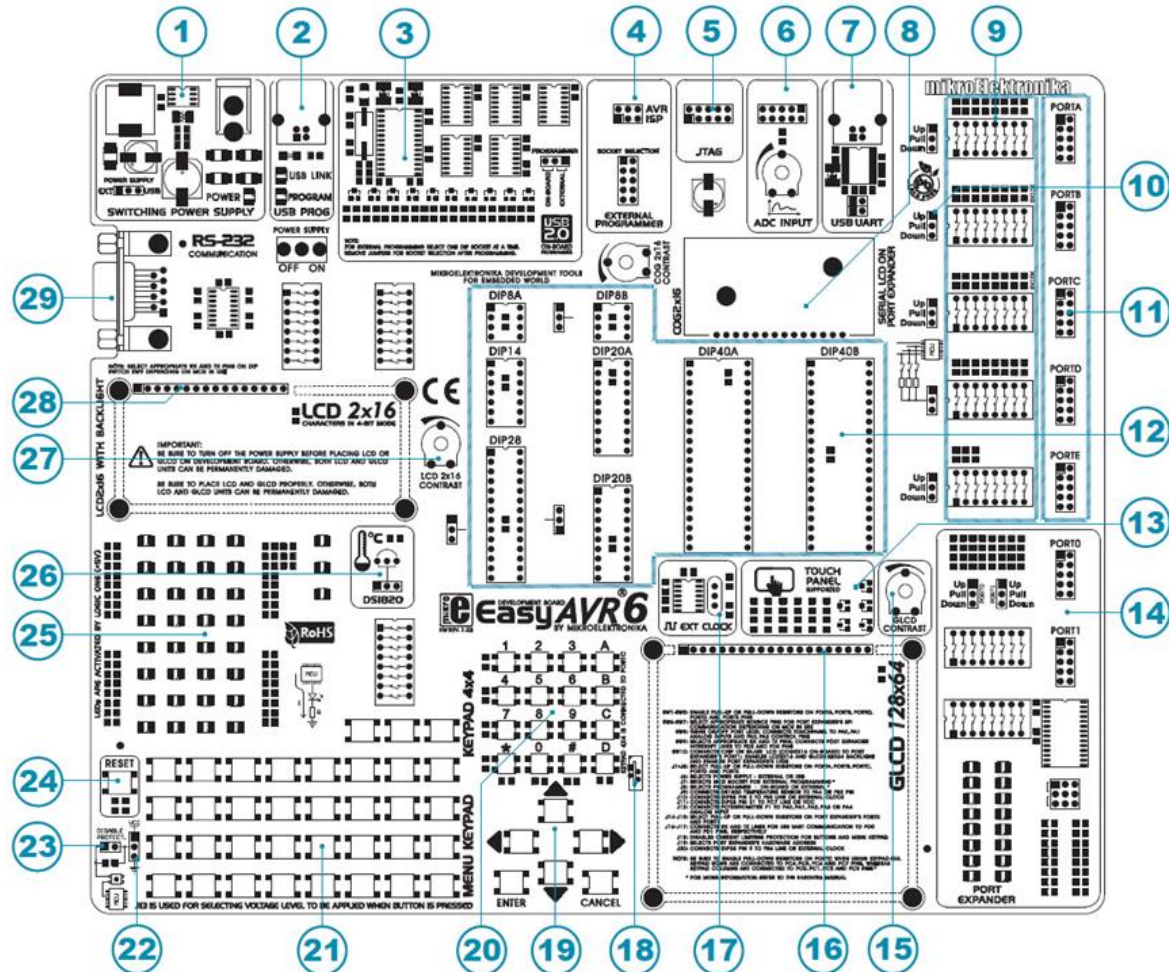
# Η ΑΝΑΠΤΥΞΙΑΚΗ ΠΛΑΚΕΤΑ EasyAVR6



1. Κυκλώμα τροφοδοσίας
2. Συνδετήρας USB προγραμματιστή εντός κυκλώματος
3. Κύκλωμα προγραμματιστή εντός κυκλώματος
4. Συνδετήρας εξωτερικού προγραμματιστή AVRSIP
5. Συνδετήρας JTAG για εξωτερικό προγραμματισμό και αποσφαλμάτωση
6. Κύκλωμα μετατροπέα Α/Ψ και αναλογικές εισόδους
7. Θύρα σειριακής επικοινωνίας UART με συνδετήρα USB
8. Ενσωματωμένη οθόνη χαρακτήρων 2x16
- 9-10. Συρόμενοι διακόπτες επίτρεψης σύνδεσης με αντιστάσεις πρόσδεσης
11. Συνδετήρες θυρών E/E
12. Βάσεις μικροελεγκτών AVR
13. Ελεγκτής μεμβράνης αφής
14. Κύκλωμα επέκτασης θυρών E/E

# Η ΑΝΑΠΤΥΞΙΑΚΗ ΠΛΑΚΕΤΑ EasyAVR6

15. Ρυθμιστής αντίθεσης οθόνης γραφικών 128x64
16. Συνδετήρας οθόνης γραφικών 128x64
17. Ταλαντωτής
18. Συνδετήρας μεμβράνης αφής
19. Πληκτρολόγιο μενού
20. Πληκτρολόγιο 4x4
21. Πληκτρολόγιο ψηφιακών εισόδων
22. Επιλογέας λογικής στάθμης
23. Βραχυκυκλωτήρας προστατευτικής αντίστασης
24. Πλήκτρο επανεκκίνησης
25. Φωτοδίοδοι
26. Βάση αισθητήρα θερμοκρασίας
27. Ρυθμιστής αντίθεσης οθόνης χαρακτήρων 2x16
28. Συνδετήρας οθόνης χαρακτήρων 2x16
29. Θύρα σειριακής επικοινωνίας UART με συνδετήρα RS-232



# Χαρακτηριστικά της πλακέτας EasyAVR6

- Δεν απαιτεί εξωτερική πηγή τροφοδοσίας καθώς μπορεί να τροφοδοτηθεί και από μια θύρα USB.
- Δεν απαιτεί ειδικό υλικό προγραμματισμού καθώς διαθέτει κύκλωμα προγραμματισμού εντός κυκλώματος που συνδέεται με τη θύρα τροφοδοσίας USB.
- Δεν απαιτεί εξωτερική πηγή χρονισμού.
- Μπορεί να λειτουργήσει πλήρως με μόνο μια σύνδεση USB με προσωπικό υπολογιστή.
- Μπορεί να χρησιμοποιηθεί με διαφορετικούς μικροελεγκτές AVR σε συσκευασίες 8, 14, 20, 28 και 40 ακροδεκτών.
- Διαθέτει μια πλούσια ποικιλία περιφερειακών μονάδων όπως οθόνη χαρακτήρων 2×16, οθόνη γραφικών 128×64 με πρόσθετη μεμβράνη αφής, μετατροπέα Α/Ψ, αισθητήρα θερμοκρασίας, θύρα σειριακής επικοινωνίας UART RS-232 και USB, πιεστικούς διακόπτες με διαφορετικές διατάξεις και φωτοδιόδους.
- Διαθέτει συρόμενους διακόπτες για τη διαμόρφωση των συνδέσεων των περιφερειακών μονάδων.
- Μπορεί να συνδυαστεί με σύνθετες εξωτερικές περιφερειακές μονάδες που συνδέονται στις θύρες E/E του μικροελεγκτή όπως ελεγκτές Ethernet, WiFi, GSM, ZigBee, Bluetooth, CAN, IrDA, κάρτες αποθήκευσης microSD και Compact Flash, αποκωδικοποιητές MP3, αισθητήρες κίνησης και αναγνώρισης RFID κ.α.



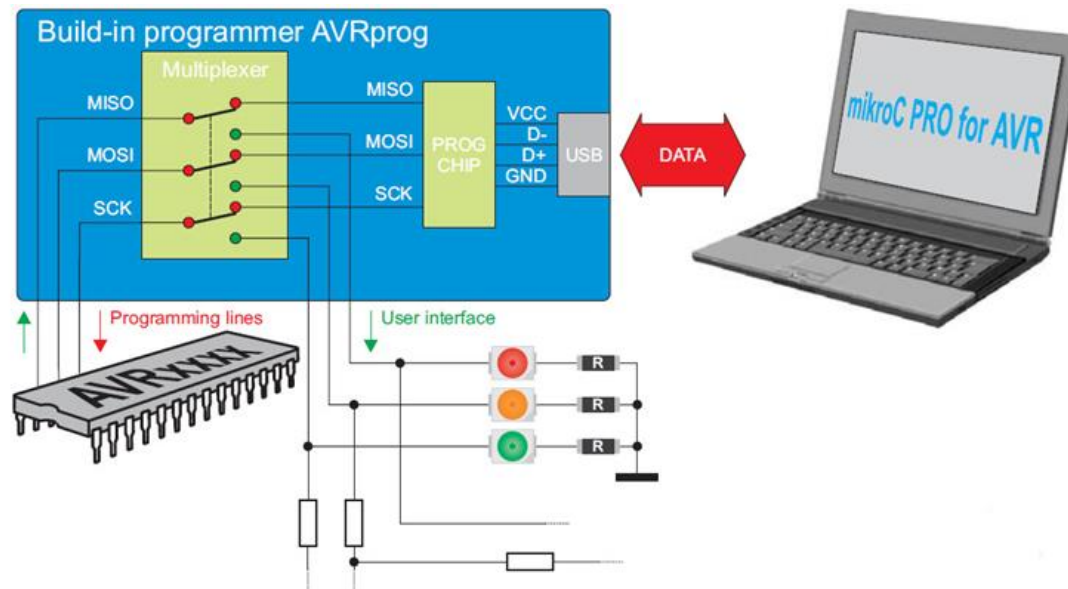
# Βάσεις μικροελεγκτών



# Ενσωματωμένος Προγραμματιστής εντός Κυκλώματος

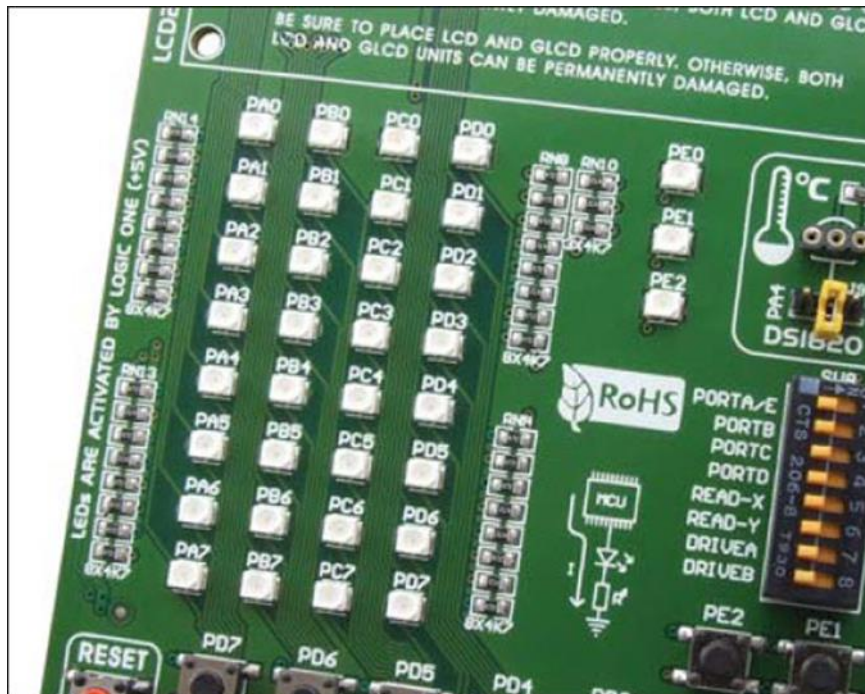
Ο προγραμματισμός με τον ενσωματωμένο προγραμματιστή AVRprog γίνεται με 3 απλά βήματα.

- Συγγραφή κώδικα και αποσφαλμάτωση με τη βοήθεια σχετικών εργαλείων όπως το AVR Studio.
- Παραγωγή εκτελέσιμου αρχείου .hex.
- Μεταφορά του εκτελέσιμου αρχείου στη μνήμη προγράμματος του μικροελεγκτή με την κατάλληλη εφαρμογή (AVRFlash). Η μεταφορά γίνεται με σειριακό πρωτόκολλο SPI, όμως εικονίζεται στο σχήμα.



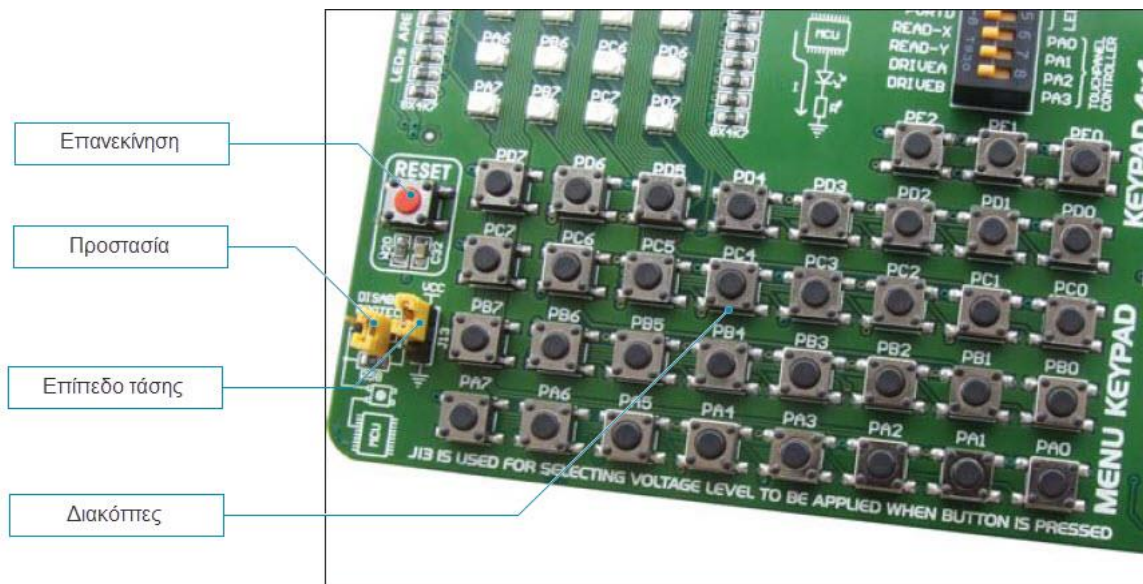
# Φωτοδιόδοι

Η αναπτυξιακή πλακέτα EasyAVR6 χρησιμοποιεί 35 φωτοδιόδους που μπορούν να απεικονίζουν ανά πάσα στιγμή τη λογική στάθμη σε οποιονδήποτε ακροδέκτη E/E του μικροελεγκτή. Μια ενεργή (αναμμένη) φωτοδιόδος σημαίνει ότι ο αντίστοιχος ακροδέκτης βρίσκεται σε λογικό 1. Για να είναι επιτρεπτή αυτή η συνεχής απεικόνιση πρέπει να έχουν τοποθετηθεί στη θέση ON οι συρόμενοι διακόπτες PORTA/E, PORTB, PORTC και PORTD.



# Πιεστικοί Διακόπτες

Αντίστοιχα με τις φωτοδιόδους, το EasyAVR6 περιλαμβάνει 35 πιεστικούς διακόπτες, που χρησιμοποιούνται για να αλλάξουν τη στάθμη τάσης σε οποιονδήποτε ακροδέκτη E/E του μικροελεγκτή. Η στάθμη αυτή ρυθμίζεται από το βραχυκυκλωτήρα J13. Όταν ο J13 βρίσκεται στη θέση VCC, το πάτημα οποιουδήποτε πιεστικού διακόπτη έχει ως αποτέλεσμα η στάθμη στον αντίστοιχο ακροδέκτη να γίνει λογικό 1 (5V). Όταν ο J13 βρίσκεται στη θέση GND (σύμβολο της γείωσης), το ίδιο πάτημα έχει ως αποτέλεσμα η στάθμη στον ακροδέκτη να γίνει λογικό 0.



# **ΤΟ ΠΕΡΙΒΑΛΛΟΝ ΑΝΑΠΤΥΞΗΣ ΕΦΑΡΜΟΓΩΝ AVR Studio 5.0**

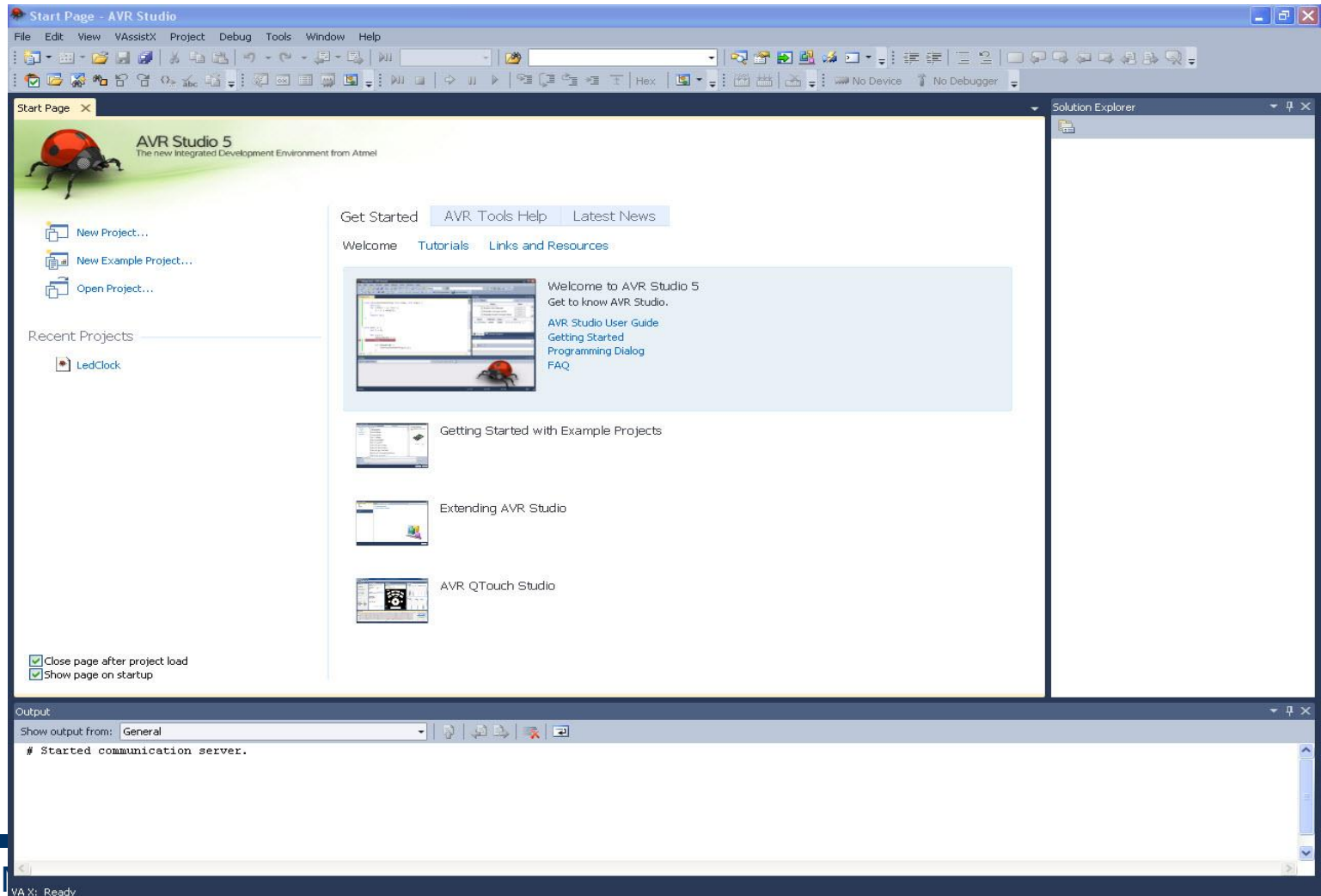


# AVR Studio 5.0

- Το περιβάλλον ανάπτυξης εφαρμογών AVR Studio 5.0 (είναι μια ολοκληρωμένη πλατφόρμα ανάπτυξης λογισμικού της εταιρείας Atmel. Καλύπτει όλο το φάσμα των μικροελεγκτών AVR, από 8 έως 32 bit.
- Περιλαμβάνει:
  - εκδότη κειμένου με υποβοήθηση συγγραφής κώδικα, βοηθός δημιουργίας και οργάνωσης έργου, βιβλιοθήκη με πάνω 400 έτοιμα παραδείγματα, συμβολομεταφραστή και μεταγλωττιστή GNU C/C++, ισχυρή μηχανή προσομοίωσης με λειτουργίες αποσφαλμάτωσης καθώς και σύνδεση με αποσφαλματωτές εντός κυκλώματος.
- Παράλληλα προσφέρει στο χρήστη εύκολη πρόσβαση σε πληροφορίες αναρτημένες στο διαδίκτυο όπως εγχειρίδια χρήσης, φύλλα προδιαγραφών, παραδείγματα έργων καθώς και σύνδεση με το ηλεκτρονικό κατάστημα της Atmel αλλά και ιστοσελίδες άλλων εταιρειών και ομάδων χρηστών, που προσφέρουν άρθρώματα για την κάλυψη επιπλέον λειτουργιών.
- Το περιβάλλον AVR Studio 5.0 είναι διαθέσιμο στον υπερσύνδεσμο:  
[http://www.atmel.com/dyn/products/tools\\_card.asp?tool\\_id=17212](http://www.atmel.com/dyn/products/tools_card.asp?tool_id=17212)

# AVR Studio 5.0 αρχική σελίδα επιλογής/δημιουργίας έργου λογισμικού

Η αρχική σελίδα μπορεί να εμφανιστεί και από το μενού View→Start Page  
ή πατώντας **Alt+V+G**.



# Τεχνικές προδιαγραφές και εγχειρίδια χρήσης

Get Started

AVR Tools Help

Latest News

Starter kits

Debuggers

Programmers

Touch kits

Wireless kits

Evaluation kits

Software



## AVR Studio 5

AVR Studio 5 is the new Integrated Development Environment (IDE) from Atmel. Packed with usefull and user friendly features help you perform at your peak in writing programs, testing and debugging

[Product Updates](#)

[AVR Studio 5 User Guide](#)



AVR GNU Toolchain



AVR Simulator



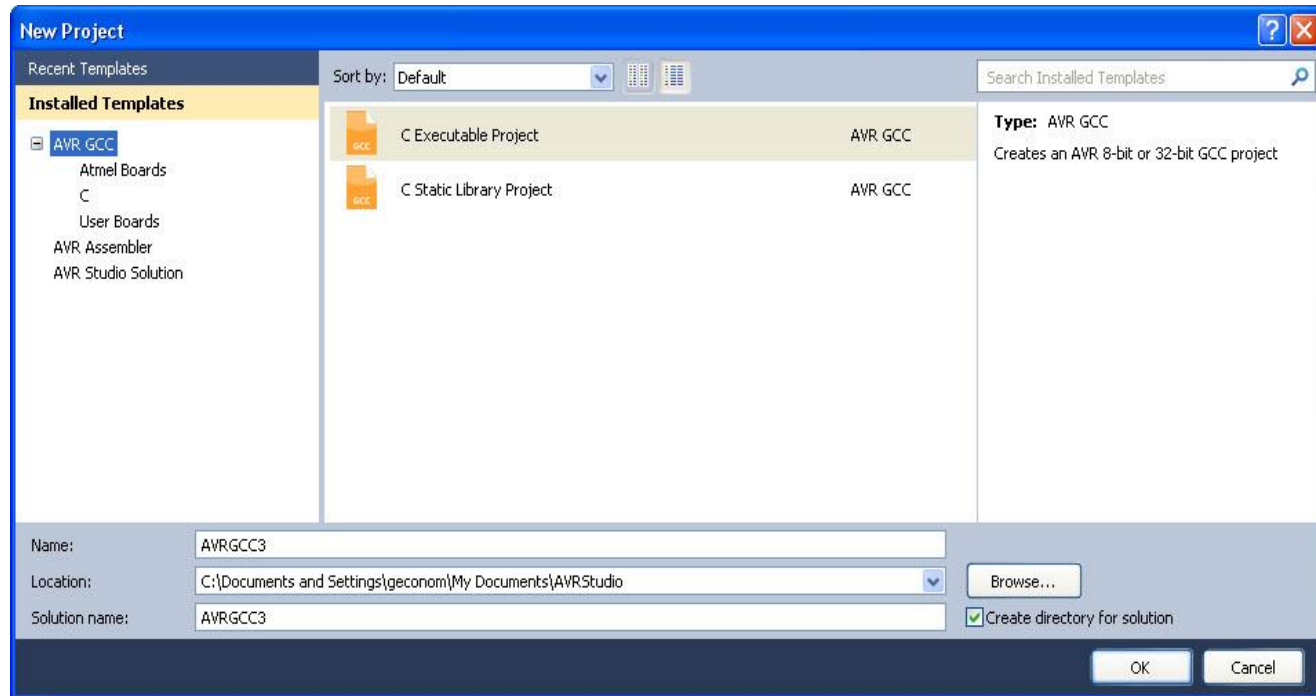
AVR Assembler

Η καρτέλα AVR Tools Help περιέχει συνδέσμους για τεχνικά κείμενα και εγχειρίδια χρήσης, που αφορούν λογισμικό και υλικό της Atmel. Υποκατηγορίες:

- Starter kits=εισαγωγικές αναπτυξιακές πλακέτες,
- Debuggers=αποσφαλματωτές,
- Programmers=προγραμματιστές μικροελεγκτών,
- Touch kits=αναπτυξιακές πλακέτες με αισθητήρες αφής,
- Wireless kits=αναπτυξιακές πλακέτες με πομποδέκτες RF,
- Evaluation kits=αναπτυξιακές πλακέτες αξιολόγησης ειδικών αρχιτεκτονικών μικροελεγκτών και
- Software=λογισμικό.



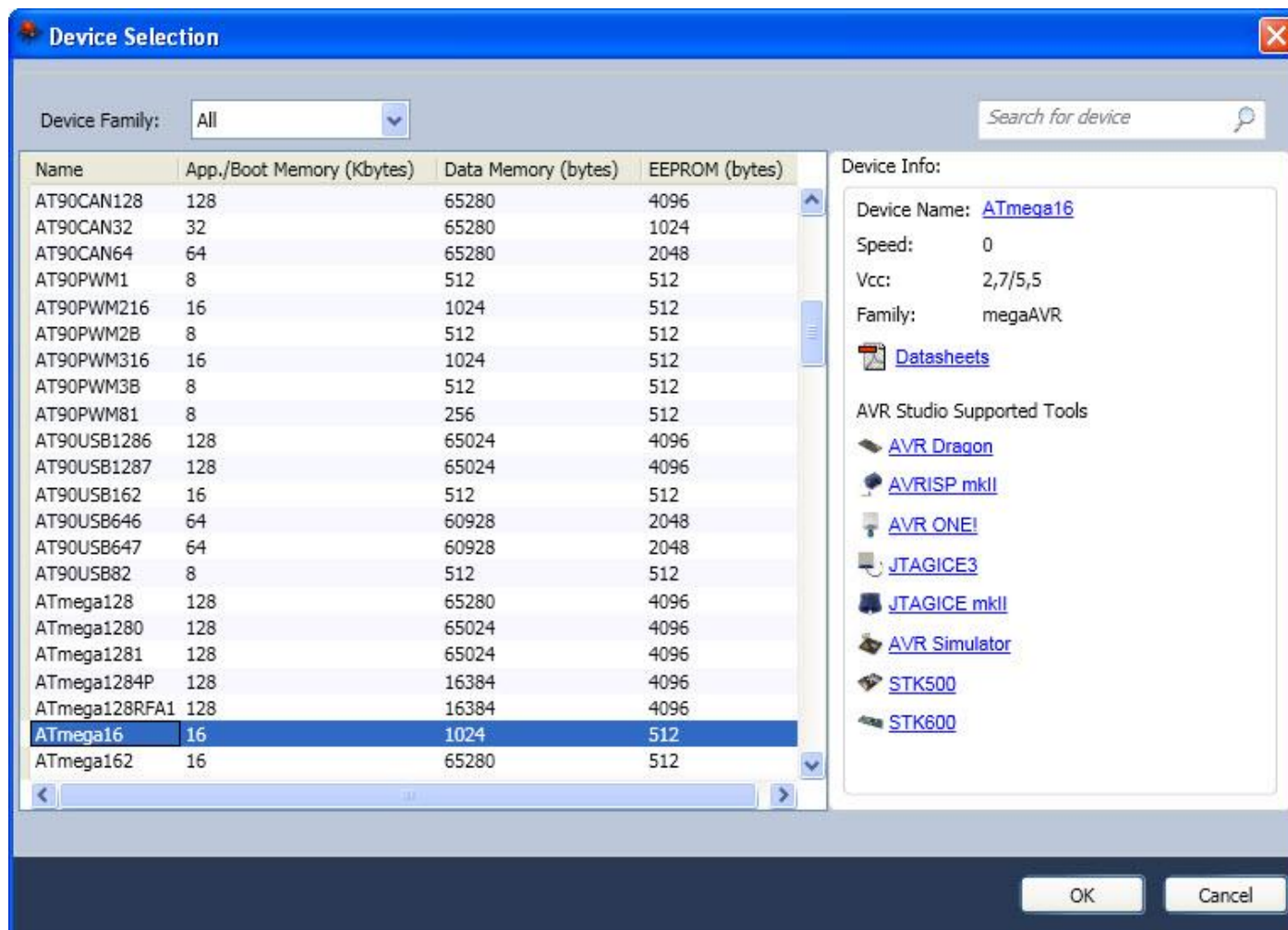
# Δημιουργία νέου έργου λογισμικού



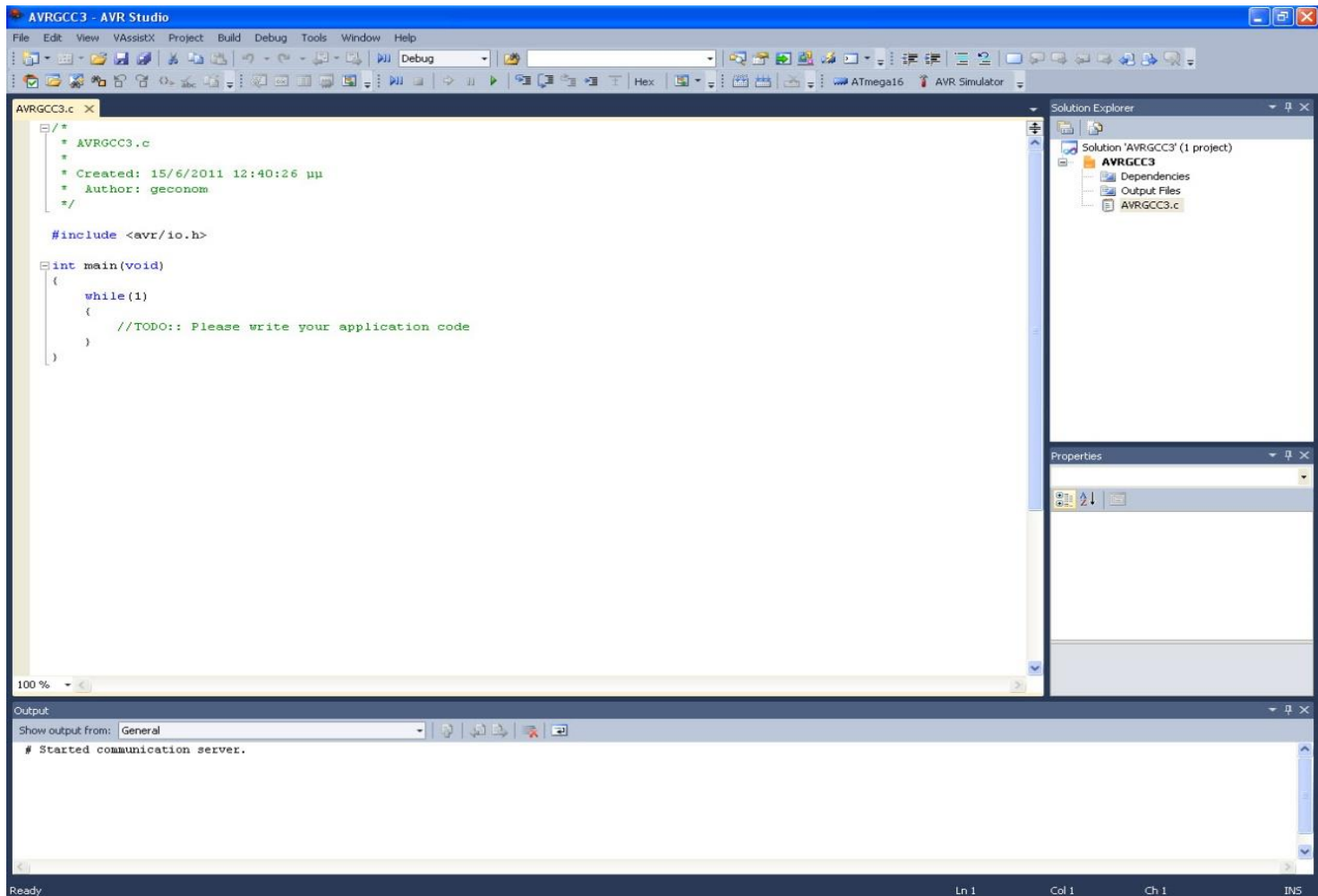
Επιλέγοντας New Project από την αρχική σελίδα του AVR Studio εμφανίζεται ο βοηθός νέου έργου, που παρουσιάζεται:

- στην αριστερή πλευρά του βοηθού εμφανίζονται τα προεγκατεστημένα πρότυπα,
- στη μέση οι βασικές επιλογές κάθε πρότυπου,
- δεξιά σχόλια και στο κάτω μέρος το όνομα του έργου, της επίλυσης προβλήματος και των υποκαταλόγων στους οποίους αυτά θα δημιουργηθούν.

# Επιλογή συσκευής



# Code editing



# Code building

- **Build και Debug** γίνεται με μία κίνηση έλεγχος του κώδικα, παραγωγή εκτελέσιμου και έναρξη αποσφαλμάτωσης .
- Συντομεύσεις πληκτρολογίου:
  - F7 (παραγωγή εκτελέσιμου κώδικα),
  - F5 (έναρξη αποσφαλμάτωσης),
  - Alt+F5 (έναρξη αποσφαλμάτωσης με σημείο διακοπής την αρχή του προγράμματος)
  - F10 (βηματική εκτέλεση κώδικα),.
- Ο εκτελέσιμος κώδικας μπορεί να ακολουθήσει δύο διαφορετικές διαμορφώσεις, μία με επιπλέον πληροφορία συμβολικής αποσφαλμάτωσης (debug) και μία χωρίς (release).

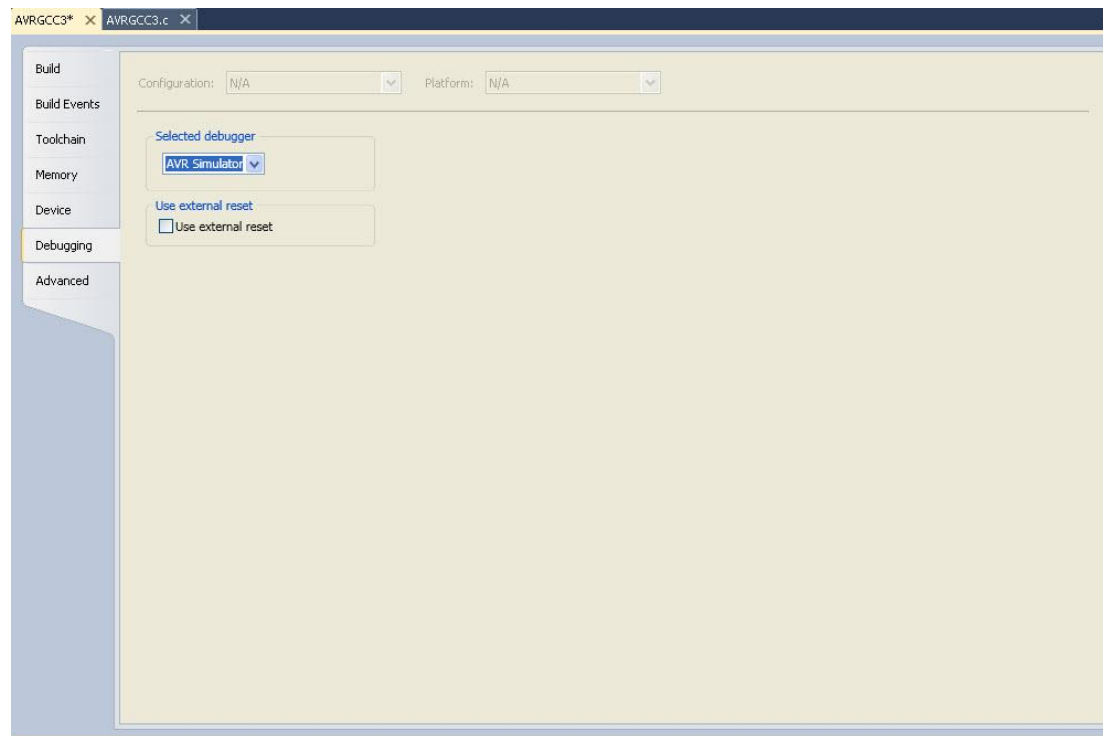
# Αποσφαλμάτωση κώδικα

- Η αποσφαλμάτωση μπορεί να πραγματοποιηθεί μόνο εφόσον έχει καθοριστεί πλατφόρμα εκτέλεσης. Σε περίπτωση που δεν έχει επιλεγεί, αυτό φαίνεται από την ένδειξη No Debugger στο τελευταίο κουμπί της γραμμής εργαλείων του AVR Studio



# Επιλογή Debugger

- Η επιλογή αυτή μπορεί να αλλάξει, πατώντας το δεξί κουμπί στο όνομα του έργου (δεξιά στο κεντρικό παράθυρο του AVR Studio) ώστε να εμφανιστούν οι ιδιότητες του έργου. Εκεί και από την καρτέλα Debugging μπορεί να επιλεγεί μια από τις διαθέσιμες πλατφόρμες εκτέλεσης. Η πιο συχνά χρησιμοποιούμενη, ειδικά στις περιπτώσεις που δεν υπάρχει ειδικό υλικό αποσφαλμάτωσης, είναι ο προσομοιωτής AVR Simulator.



# Αποσφαλμάτωση Κώδικα με Προσομοιωτή Avr Simulator

- Η εντολή **Debug→Continue** ή η συντόμευση F5 εκτελεί την αποσφαλμάτωση κάνοντας προσομοίωση του κώδικα μέχρι να βρεθεί σημείο διακοπής ή να μέχρι να δοθεί η εντολή **Debug→Stop Debugging**.
- Η εντολή **Debug→Start Debugging and Break** ή η συντόμευση Alt+F5 ξεκινάει τη διαδικασία αποσφαλμάτωσης και σταματάει την προσομοίωση του κώδικα στην πρώτη γραμμή του. Ο χρήστης μπορεί τότε να ορίσει σημεία διακοπής με την εντολή **Debug→New Breakpoint** ή πατώντας το δεξί κουμπί του ποντικιού σε μια γραμμή κώδικα, και να συνεχίσει την προσομοίωση πατώντας F5.
- Η εντολή **Debug→Start Without Debugging** είναι παρόμοια με την προηγούμενη, δηλαδή αρχικοποιεί την αποσφαλμάτωση, αλλά δεν ξεκινάει καθόλου την προσομοίωση, αφήνοντας πάλι το χρήστη να ορίσει σημεία διακοπής.
- Ο τερματισμός της αποσφαλμάτωσης μπορεί να γίνει με τις εντολές **Debug→Break All** (συντόμευση Ctrl+F5) προσωρινά ή **Debug→Stop Debugging** (συντόμευση Ctrl+Shift+F5) οριστικά.
- Τέλος αξιοσημείωτες είναι και οι εντολές **Debug→Step Over** (συντόμευση F10) και **Debug→Step Into** (συντόμευση F11), που χρησιμοποιούνται για βηματική εκτέλεση κώδικα υπερπηδώντας ή διεισδύοντας στον κώδικα υπορουτινών και μακροεντολών.

# Debugging views

Name	Value
Program Counter	0x00000002
Stack Pointer	0x00000000
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	I <input type="checkbox"/> H <input type="checkbox"/> S <input type="checkbox"/> N <input type="checkbox"/> Z <input type="checkbox"/> C
Cycle Counter	2
Frequency	1,000 MHz
Stop Watch	2,00 µs

Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00
R17	0x00
R18	0x00
R19	0x00
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0xFF

Οι όψεις αυτές ενημερώνονται σε κάθε στάση της αποσφαλμάτωσης (σημεία διακοπής) και μπορούν να τροποποιηθούν από το χρήστη. Εάν δεν είναι ορατές μπορούν να γίνουν με τις εντολές Debug→Windows→Processor View, Debug→Windows→I/O View και Debug→Windows→Memory→... αντίστοιχα.

Memory 1

Memory: 0x0,prog Address: 0x0000,prog

Address	Hex Value	ASCII
prog 0x0000	8f ef 87 bb a9 b3 a8 bb fd cf ff ff ff ff ff ff ff	.o.»@."»»00....
prog 0x000F	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff	.....
prog 0x001E	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff	.....
prog 0x002D	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff	.....
prog 0x003C	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff	.....
prog 0x004B	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff	.....
prog 0x005A	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff	.....
prog 0x0069	ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff	.....

Breakpoints Memory 1 Call Stack Command Window Immediate Window Output

## Memory view

The screenshot shows the 'IO View' window in Proteus. The left pane lists various I/O components, including AD\_CONVERTER, ANALOG\_COMPARATOR, BOOT\_LOAD, CPU, EEPROM, EXTERNAL\_INTERRUPT, JTAG, PORTA, PORTB, PORTC, PORTD, SPI, TIMER\_COUNTER\_0, TIMER\_COUNTER\_1, TIMER\_COUNTER\_2, and TWI. PORTA is selected and highlighted in blue. The right pane displays the details for the selected component, PORTA, showing its Name, Address (0x36), Value (0xE0), and a bit field representation (11100000).

Name	Address	Value	Bits
PORTA	0x36	0xE0	11100000

## Processor view

## I/O view



# AnrFlash: Μεταφορά Και Εκτέλεση Κώδικα Σε Αναπτυξιακή Πλακέτα

