ΕΡΓΑΣΤΗΡΙΟ 3

# ♦ Μέθοδοι απασφαλμάτωσης

## Προϋποθέσεις

Το εργαστήριο αυτό προϋποθέτει το διάβασμα και χρήση των εξής:

- **Αρχείο mcbstr9.chm HTML,** Κεφάλαιο "Writing Programs->Debugging Programs"

- **Αρχείο debug_test.c,** μέσα από τον φάκελο **Lab3_Debug.rar**.

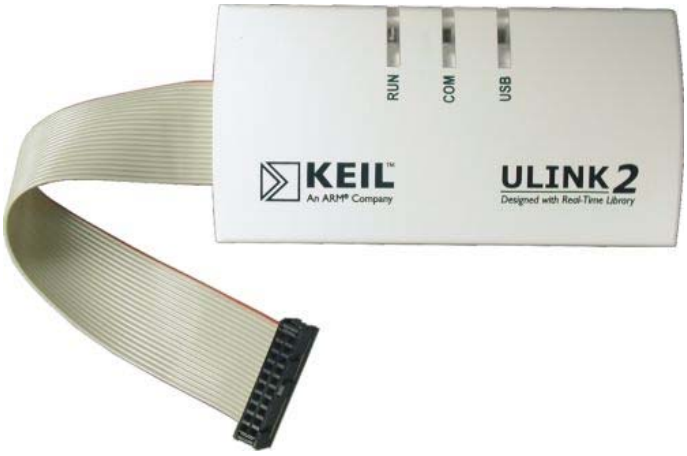
- **Αρχείο STR91xFA_Reference_Manual.pdf**. Θα το χρησιμοποιούμε ως αναφορά για καταχωρητές.

- **Βιβλίο Θεωρίας Wayne Wolf, "Οι Υπολογιστές ως Συστατικά Στοιχεία".** Κεφάλαιο 4, παράγραφος 7. Ανάπτυξη και Αποσφαλμάτωση.

## Εισαγωγή

Στο εργαστήριο αυτό θα ακολουθήσουμε διάφορες μεθόδους απασφαλμάτωσης, χρησιμοποιώντας τον απασφαλματωτή (debugger) του μVision4 σε συνδυασμό με την συσκευή ULINK2. Η συσκευή αυτή είναι ουσιαστικά ένας USB-JTAG μετατροπέας και επιτρέπει στο λογισμικό μVision4 να επικοινωνεί με την JTAG διεπαφή του μικροελεγκτή. Χρησιμοποιεί **JTAG (Joint Test Action Group) εντολές** διεπαφής για να πραγματοποιήσει τα παρακάτω:

- Φόρτωση προγραμμάτων λογισμικού στη πλακέτα
- Απασφαλμάτωσης του τρέχοντος προγράμματος με διάφορες μεθόδους.
- Προγραμματισμό της εσωτερικής μνήμης FLASH του μικροελεγκτή.
- Προγραμματισμό εξωτερικής μνήμης FLASH στην πλακέτα.




**Σχήμα 1:** ULINK2 USB-JTAG μετατροπέας

Παράλληλα το λογισμικό μVision4 μπορεί να χρησιμοποιηθεί και χωρίς κάποια πλακέτα και μικροελεγκτή, επιλέγοντας το “Use Simulator” στο παράθυρο “Options for Target “STR910 Flash”, όπως φαίνεται στο επόμενο σχήμα. Με την επιλογή αυτή, ο χρήστης μπορεί να δοκιμάσει τον κώδικα του καθώς το μVision4 έχει την δυνατότητα να εξομοιώσει το μεγαλύτερο κομμάτι του μικροελεγκτή.


**Σχήμα 2:** Επιλογή εξομοίωσης στο uVision4

Παρακάτω, θα χρησιμοποιήσουμε τρεις διαφορετικές μεθόδους απασφαλμάτωσης.

- Άμεση παρακολούθηση των καταχωρητών του μικροελεγκτή.
- Breakpoints.
- Παράθυρα Dissasembly και Register.

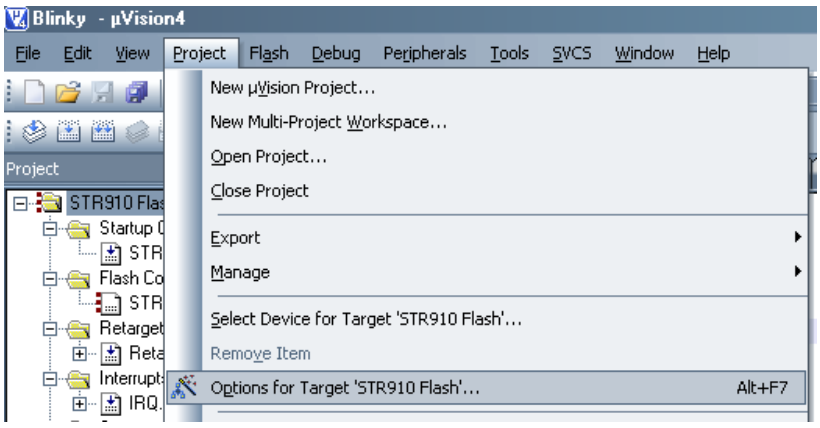
## 1.1 Ρυθμίσεις uVision για χρήση του αποσφαλματωτή

Ανοίγουμε το **Project debug_test.uvproj** αποσυμπιέζοντας το **Lab3_Debug.rar** αρχείο, που δίδεται με τα υπόλοιπα αρχεία του εργαστηρίου. Ανοίγουμε το αρχείο **debug_test.c**

```
int main (void)
{
  int i, dummy_loop;

  for (i=0; i<100; i++)
    dummy_loop = i;

  /* System Control Unit Set-up */
  SCU->PRR1     =  0x00EE2803;
  SCU->PCGR1    =  0x00EE2803;

  /* Buttons Setup */
  SCU->GPIOIN[3]     = 0x00;   /* P3.5 input - mode 1    */
  SCU->GPIOOUT[3]   &= 0xF3FF; /* P3.5 input - mode 0    */
  GPIO3->DDR         = 0xFF;   /* P3.5 direction - input */

  SCU->GPIOOUT[7]  = 0x5555;              /* P7.0..7 output - mode 1          */
  GPIO7->DDR       = 0xFF;                /* P7.0..7 Outputs (LED Data)       */

  i = GPIO3->DR[0x180];

  while (1) {                             /* Loop forever                     */
    for (i = 0x01; i <= 0xFF; i <<= 1) {
      GPIO7->DR[0x3FC] = i;               /* Turn on LED                      */

      wait();
    }
  }
}
```

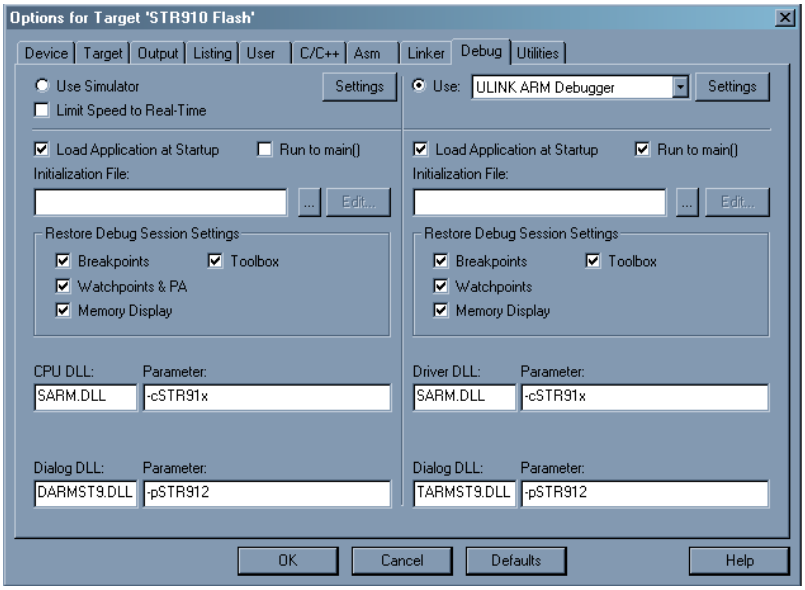
**Χτίζουμε και κατεβάζουμε τον κώδικα στο αναπτυξιακό ακολουθώντας τα βήματα των Κεφαλαίων 2.1 και 2.2.**

Επιλέγουμε το Options for Target 'STR910 Flash' όπως φαίνεται στο επόμενο σχήμα.



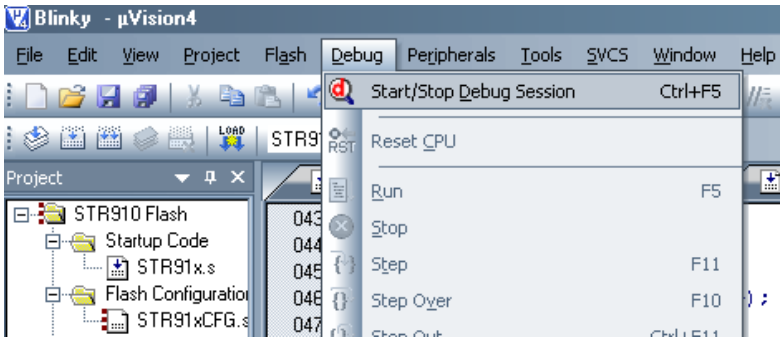

**Σχήμα 3:** Επιλογές debugger

Από το παράθυρο που αναδύεται επιλέγουμε το tab **Debug** και το ρυθμίζουμε όπως φαίνεται στο επόμενο σχήμα.



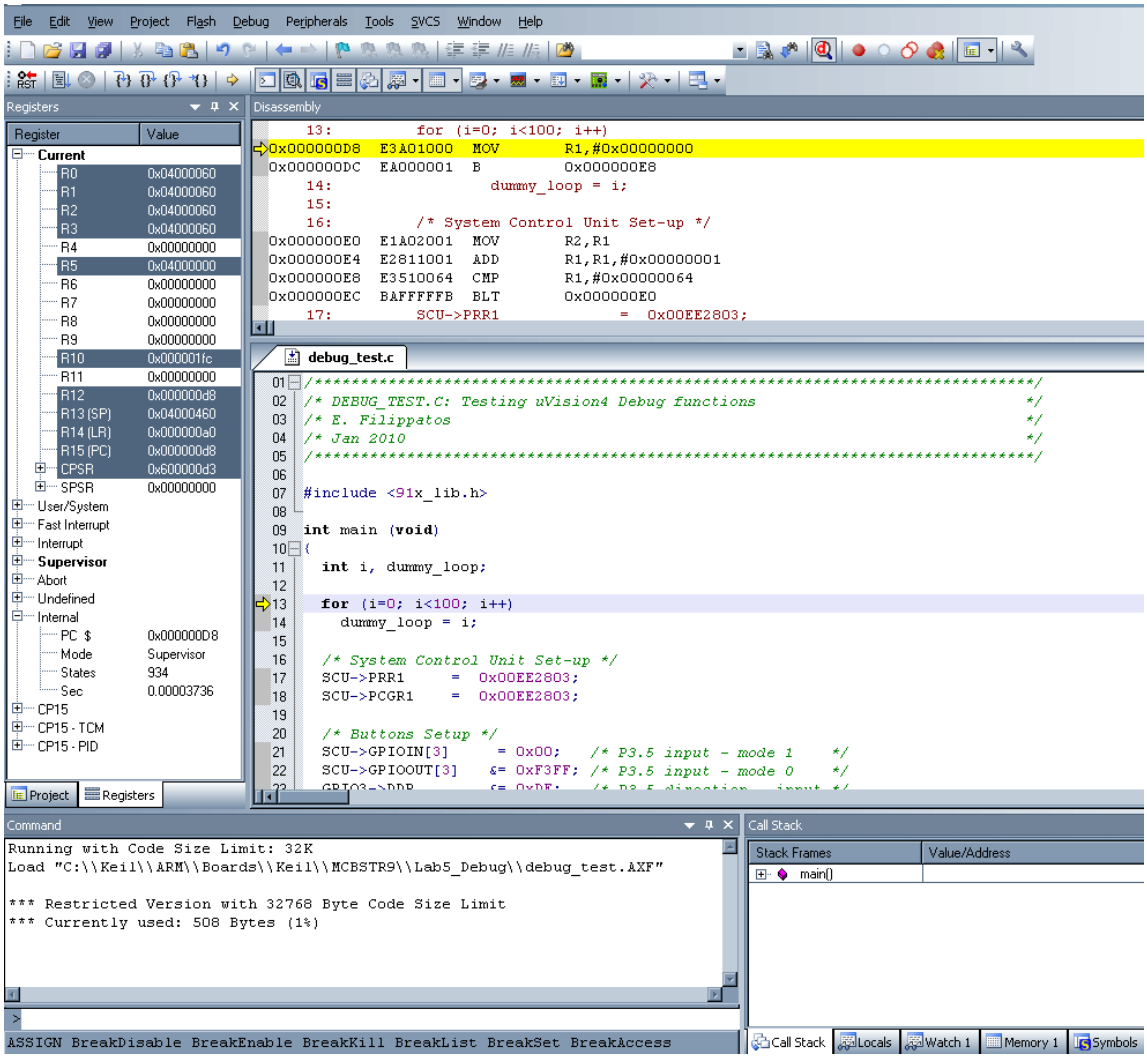

**Σχήμα 4:** Επιλογές debugger

Κλείνουμε το παράθυρο και επιλέγουμε το **Start/Stop Debug Session** όπως φαίνεται στην παρακάτω εικόνα.




**Σχήμα 5:** Επιλογή Εκκίνησης Debugger

Πατάμε OK στο μήνυμα: *EVALUATION MODE, Running with Code Size Limit : 32K*. Ο Debugger ξεκινάει και αφού κάνει ένα reset στον μικροελεγκτή, περιμένει στην πρώτη εντολή της main του προγράμματος μας. Αυτό φαίνεται παρακάτω.




**Σχήμα 6:** Παράθυρο Debugger μVision4

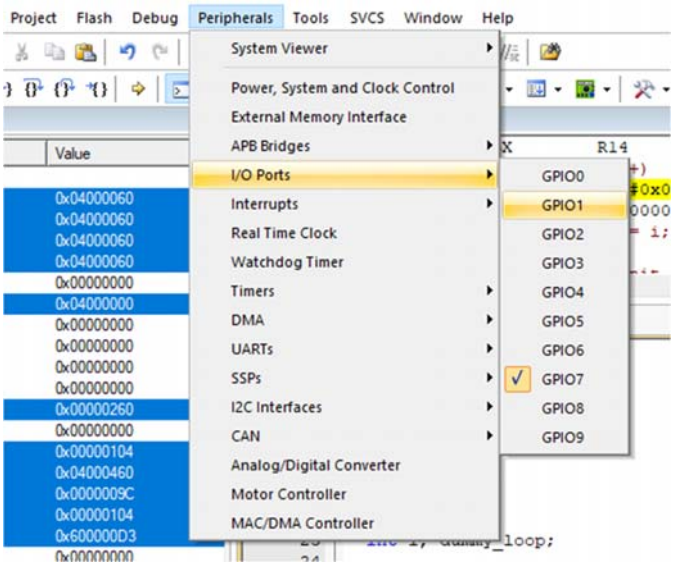
## 1.2 Παρακολούθηση κατάστασης καταχωρητών

Μέσω της χρήσης του debugger θα προσπαθήσουμε να παρακολουθήσουμε την κατάσταση του GPIO 7 στο οποίο είναι συνδεδεμένα τα led. Για να μπορέσουμε να παρακολουθούμε την κατάσταση του GPIO 7 πρέπει να παρακολουθούμε τον αντίστοιχο καταχωρητή που κρατάει τις καταστάσεις ή αλλιώς τα δεδομένα του GPIO Port 7.

| **Ερώτημα 1** |
| --- |
| **Σε ποιόν καταχωρητή αποθηκεύεται η κατάσταση του GPIO 7;** |
| |

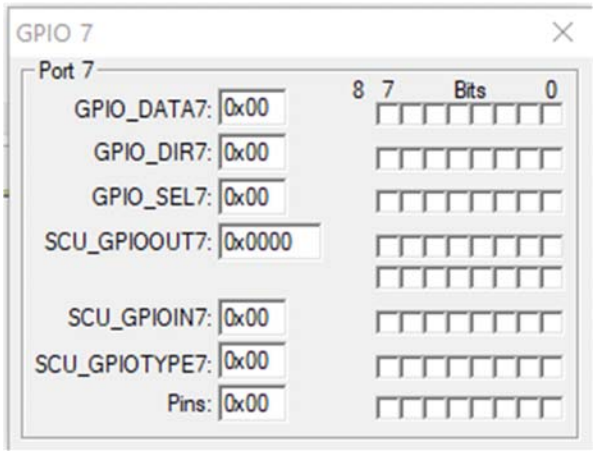
Ο έλεγχος του καταχωρητή μπορεί να γίνει με δύο διαφορετικούς τρόπους.

**a.** Επιλέγουμε τα παράθυρα των περιφερειακών όπως φαίνεται παρακάτω.



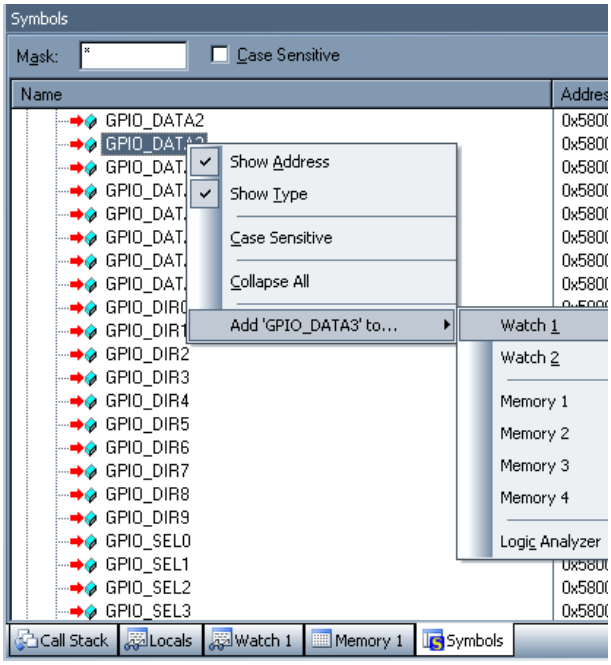

**Σχήμα 7:** Επιλογή παρακολούθηση κατάστασης GPIO

Το επόμενο παράθυρο θα πρέπει να εμφανιστεί.




**Σχήμα 8:** Παράθυρο καταχωρητών για το GPIO7

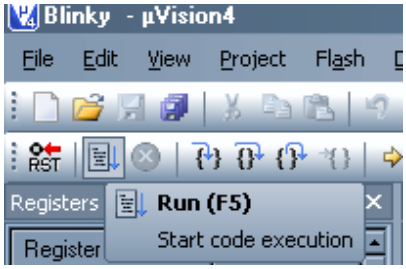
**b.** Επιλέγουμε να παρακολουθούμε τον καταχωρητή GPIO_DATA7 μέσα από την επιλογή Symbols -> Peripheral Registers όπως φαίνεται στο παρακάτω σχήμα.




**Σχήμα 9:** Επιλογή παρακολούθησης καταχωρητή

Στο παράθυρο Watch 1 θα έχουμε τώρα τον καταχωρητή GPIO_DATA7. Έχοντας και τα δυο παράθυρα παρακολούθησης του καταχωρητή GPIO_DATA7 μπορούμε να παρατηρούμε την κατάσταση των led, αλλά και όλων των GPIO, από 0 έως 7, στο Port 7.

- Ο μικροελεγκτής, με το που ξεκινάμε την διαδικασία του Debug γίνεται reset. Πατάμε το **Run ή το F5** για να ξεκινήσει να εκτελείται ο κώδικας. Η ανακύλιση των LED ξεκινάει και θα πρέπει να αυξομειώνεται καθώς μεταβάλουμε το ποτενσιόμετρο. Δοκιμάστε το.




**Σχήμα 10:** Εκτέλεση κώδικα μέσω του Debugger

- Πατήστε το **Stop**, που βρίσκεται δίπλα στο Run.
- Παρακολουθήστε την τιμή του GPIO_DATA7 ενώ τρέχει η for loop και κατανοήστε την αλλαγή των τιμών.
- Πατήστε πάλι το Run ή το F5.

| **Ερώτημα 2** |
| --- |
| **Ποιά η μέγιστη τιμή του GPIO7 DATA register;** |
| |

| **Ερώτημα 24** |
| --- |
| **Τροποποιήστε τον κώδικα ώστε ο GPIO7 DATA register να δουλεύει σαν δυαδικός μετρητής.** |
| |

Μέσω της χρήσης του debugger θα προσπαθήσουμε να ελέγξουμε την κατάσταση του GPIO 3.5 στο οποίο είναι συνδεδεμένο το πλήκτρο INT5. Για να μπορέσουμε να παρακολουθούμε την κατάσταση του GPIO 3.5, πρέπει να παρακολουθούμε τον αντίστοιχο καταχωρητή που κρατάει τις καταστάσεις ή αλλιώς τα δεδομένα του GPIO Port 3 και συγκεκριμένα το bit 5 (η αριθμηση

ξεκινάει από το 0) του DR. Για το λόγο γίνεται ανάθεση της τιμής του του DR σε μια μεταβλητή.

| **Ερώτημα 25** |
| --- |
| **Τι τιμή παίρνει η μεταβλητή i όταν ο χρήστης έχει τσεκάρει το 5ο bit του GPIO3 DR και τι όταν δεν το έχει τσεκάρει; Είναι αναμενόμενο;** |
| |

## 1.3 Χρησιμοποίηση των breakpoints

Τα breakpoints μπορούν να οριστούν σε οποιαδήποτε μέρος του κώδικα μας. Η λειτουργία των breakpoint είναι αρκετά απλή. Όταν η εκτέλεση του κώδικα μας φθάσει στο σημείο στο οποίο έχουμε ορίσει το breakpoint και πριν εκτελεστεί ο κώδικας στο σημείο αυτό, τότε η εκτέλεση σταματάει ή αλλιώς κάνει παύση. Μπορούν να οριστούν και breakpoints υπό-συνθήκη (conditional breakpoints ή αλλιώς watchpoints), τα οποία μπορούν να ελέγχουν εναλλαγή συγκεκριμένων τιμών σε καταχωρητές, σε μεταβλητές του κώδικα μας και άλλα πολλά.

Παρακάτω θα ορίσουμε ένα απλό breakpoint στο σημείο του κώδικα που προγραμματίζουμε τον Peripheral Reset Register 1, δηλαδή στην εντολή: **SCU->PRR1 = 0x00EE2803;**

| **Ερώτημα 26** |
| --- |
| **Ποιος ο σκοπός της SCU->PRR1 = 0x00EE2803 εντολής** |
| |
| |
| |
| |

Για να ορίσουμε και να ελέγξουμε το breakpoint ακολουθούμε τα παρακάτω βήματα:

- Σταματάμε την εκτέλεση του κώδικα μας πατώντας το **Stop.**

- Κάνουμε δεξί κλικ στην γραμμή του κώδικα που θέλουμε να ορίσουμε το Breakpoint, στην γραμμή SCU->PRR1 = 0x00EE2803 και επιλέγουμε το **Insert/Remove Breakpoint** ή πατάμε το F9.

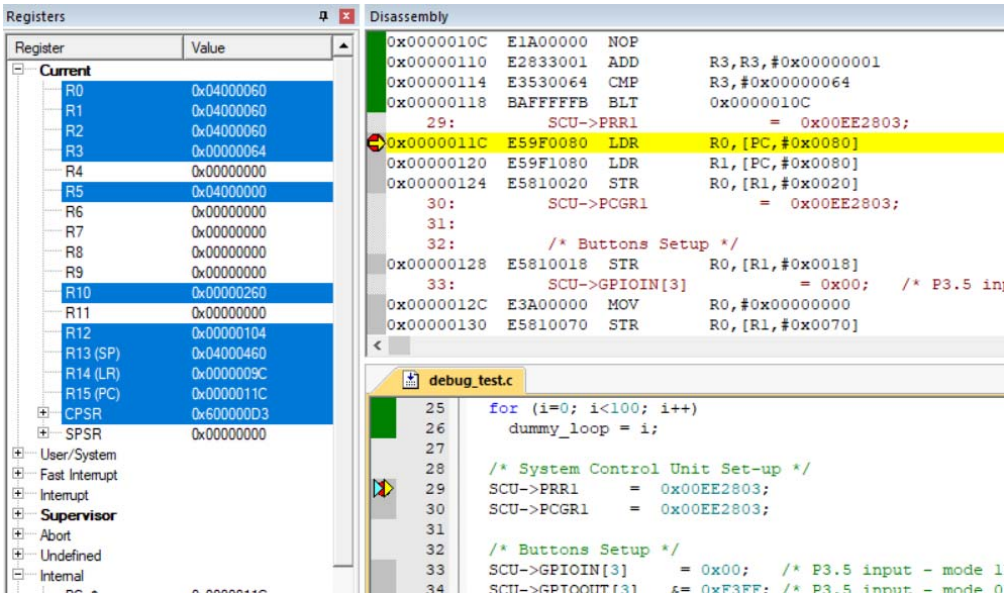
- Πατάμε το **Run** για να ξεκινήσει πάλι η εκτέλεση.
- Παρατηρούμε εάν και σε ποιο σημείο έχει σταματήσει η εκτέλεση του κώδικα μας.

- Πατάμε το πλήκτρο **Rst** και μετά το **Run** για επανεκκίνηση του κώδικα και επανέλεγχο της λειτουργία του breakpoint.

## 1.4 Χρησιμοποίηση των παραθύρων Disassembly και Registers

Το παράθυρο Dissasembly μας δείχνει τον C κώδικα μας και την αντίστοιχη μετάφραση του σε assembly ή μόνο την assembly εάν εκτελούμε απλή assembly και όχι C. Το παράθυρο Registers μας δείχνει τους καταχωρητές του ARM CPU που χρησιμοποιούνται για την επεξεργασία των δεδομένων και για την εκτέλεση του κώδικα μας. Για αυτούς θα μιλήσουμε αργότερα. Τώρα θα προσπαθήσουμε να παρατηρήσουμε το παράθυρο Dissasembly και παράλληλα την λειτουργία του παραθύρου Registers.

Εκτελούμε τον κώδικα μας με τις ρυθμίσεις της προηγούμενης παραγράφου. Η εκτέλεση, λόγο του breakpoint, σταματάει πριν εκτελεστεί η εντολή: SCU->PRR1 = 0x00EE2803.




**Σχήμα 11:** Dissasembly και Registers

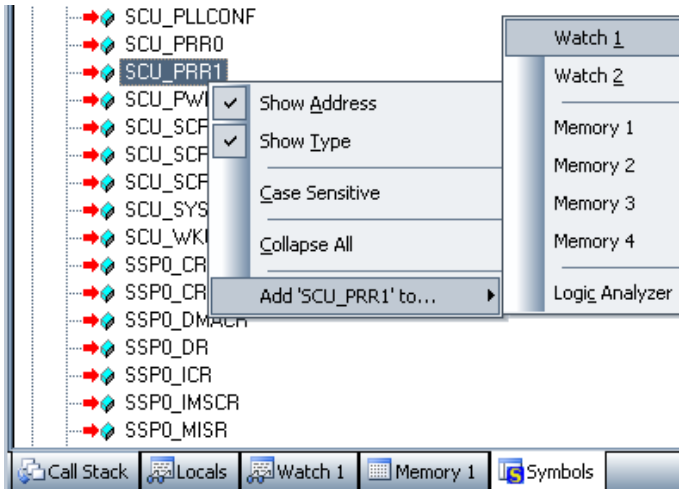
Στο προηγούμενο σχήμα παρατηρούμε δύο κίτρινα βελάκια. Τα βελάκια αυτά υποδεικνύουν ποιά είναι η επόμενη εντολή που ο CPU θα εκτελέσει. Επίσης παρατηρούμε ότι βρίσκονται πάνω στα κόκκινα κουτιά του breakpoint γιατί η εκτέλεση έχει σταματήσει στο breakpoint που έχουμε ορίσει, δηλαδή στην εντολή SCU->PRR1 = 0x00EE2803.

| **Ερώτημα 27** |
| --- |
| **Σε ποια θέση μνήμης του MCU βρίσκεται ο καταχωρητής SCU_PRR1;** |
| |

Η παραπάνω εντολή της C, στην οποία ο CPU θα καταχωρήσει στην θέση μνήμης του SCU->PRR1 την τιμή 0x00EE2803, έχει μεταφραστεί σε μια σειρά εντολών assembly. Για να δούμε ποιες εντολές είναι αυτές πρέπει να παρακολουθήσουμε τα παράθυρα Disassembly και Registers. Επίσης πρέπει να ορίσουμε τον SCU_PRR1 καταχωρητή στο Watch 1 παράθυρο.

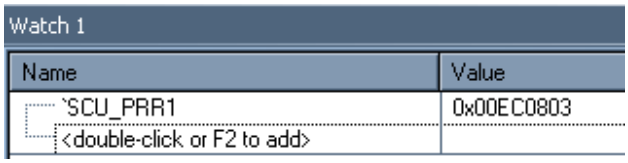
Για να θέσουμε τον καταχωρητή SCU_PRR1 στο Watch 1 παράθυρο ακολουθούμε την διαδικασία της παραγράφου 5.2. Πηγαίνουμε στο

παράθυρο Symbols και από την επιλογή Peripheral Registers θέτουμε τον καταχωρητή SCU_PRR1 στο παράθυρο Watch 1 όπως φαίνεται παρακάτω:




**Σχήμα 12:** Καταχωρητής στο παράθυρο Watch 1

Η αρχική τιμή του SCU_PRR1, είναι αυτή που φαίνεται στο Watch 1.




**Σχήμα 13:** Παράθυρο Watch 1 και αρχική τιμή του SCU_PRR1

Λόγω του breakpoint, η εκτέλεση του κώδικα μας έχει σταματήσει ακριβώς πριν εκτελεστεί η εντολή: SCU->PRR1 = 0x00EE2803. Η μετάφραση της εντολής αυτής από C σε assembly αποτελείται από τις παρακάτω τρεις εντολές, που διακρίνονται στο Disassembly παράθυρο.

```
0x0000011C  E59F0080  LDR      R0,[PC,#0x0080]
0x00000120  E59F1080  LDR      R1,[PC,#0x0080]
0x00000124  E5810020  STR      R0,[R1,#0x0020]
```

Η πρώτη στήλη περιέχει την θέση του Μετρητή Προγράμματος – Program Counter (PC). Ο Program Counter αυξάνεται κατά 4bytes (32bit) για κάθε μια εντολή που εκτελείται εκτός και εάν εκτελεστεί κάποιο παρακλάδι (Branch).

Την τρέχουσα θέση του Program Counter (PC) μπορούμε να την δούμε από τον καταχωρητή R15 (PC) στο παράθυρο των καταχωρητών (Registers).

| **Ερώτημα 28** |
| --- |
| **Σε ποια θέση βρίσκεται ο PC όταν η εκτέλεση σταματήσει λόγω του breakpoint που έχουμε θέσει;** |
| |

Μπορούμε επίσης να βρούμε την τρέχουσα τιμή του PC χρησιμοποιώντας το Disassembly παράθυρο και κοιτώντας την πρώτη στήλη στην οποία το κίτρινο βέλος δείχνει. Παρατηρώντας το Σχήμα 24 μπορούμε να δούμε ότι ο PC, το κίτρινο βέλος, έχει σταματήσει στο breakpoint, στο κόκκινο κουτί του Disassembly παραθύρου. Η διεύθυνση του PC την στιγμή αυτή είναι η 0xF0. Η ίδια διεύθυνση υπάρχει και στον καταχωρητή R15 (PC) στο παράθυρο Registers.

Η assembly εντολή που θα εκτελεστεί στην διεύθυνση 0x11C είναι η:

LDR R0,[PC,#0x0080]

Η οποία σημαίνει:
Φόρτωσε τον καταχωρητή R0 με ένα word (4bytes ή 32bits) από την διεύθυνση PC + 0x0080 + 8.

Πηγαίνοντας στην διεύθυνση PC + 0x0080 + 8= 0x11C + 0x0080 + 8 = 0x1A4 θα βρούμε την εξής assembly εντολή:

0x000001A4 00EE2803 …….

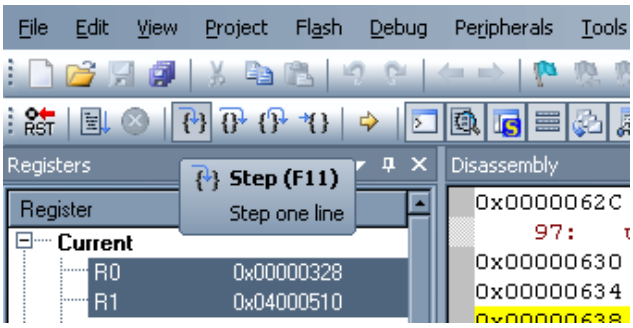
Η συγκεκριμένη λοιπόν είναι μια περιοχή στην μνήμη που χρησιμοποιείται για να αποθηκευτούν σταθερές τιμές, όπως είναι η τιμή 0x00EE2803 που θα καταχωρηθεί στον SCU_PRR1.

Οπότε η εντολή:

LDR R0,[PC,#0x00080]

πηγαίνει στην περιοχή μνήμης εκείνη, που προκύπτει από την πρόσθεση PC + 0x0080 + 8, στην οποία έχει αποθηκευτεί η τιμή 0x00EE2803. Την τιμή αυτή την φορτώνει στον καταχωρητή R0.

Παρακολουθώντας τα δεδομένα του καταχωρητή R0 πατήστε το πλήκτρο **Step (F11)**, όπως φαίνεται στο επόμενο σχήμα. Ο CPU θα εκτελέσει ένα βήμα, δηλαδή θα εκτελέσει μόνο την τρέχουσα εντολή του PC και θα κάνει παύση.




**Σχήμα 14:** Εκτέλεση Step

| **Ερώτημα 29** |
| --- |
| **Ποια εντολή θα εκτελεστεί όταν πατηθεί το Step (F11); Σε ποια θέση βρισκόταν ο PC και σε ποια θέση βρίσκεται τώρα;** |
| |
| |

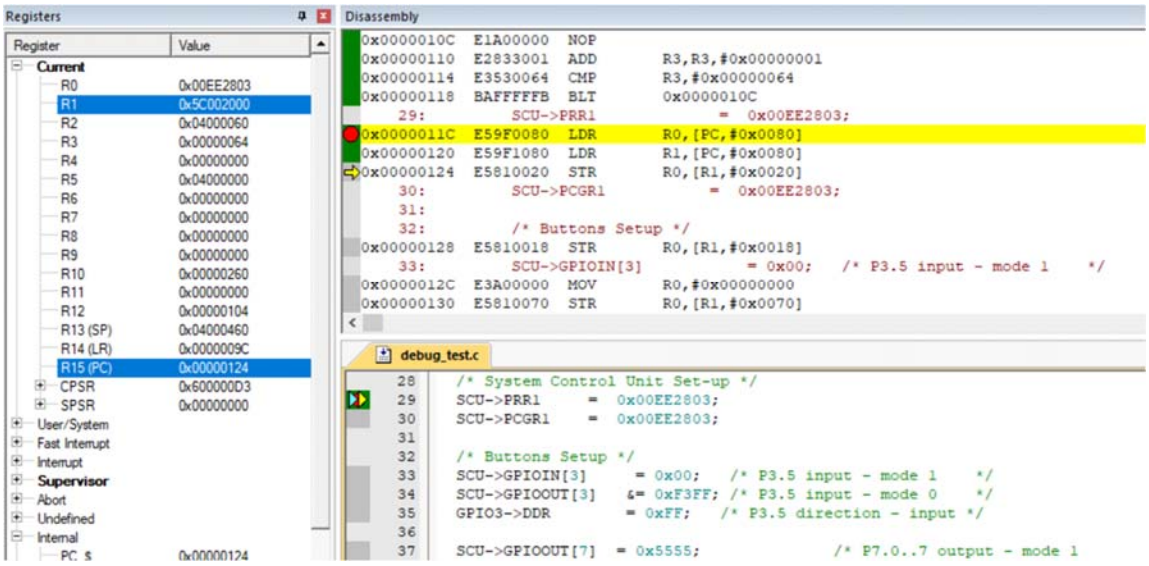
| **Ερώτημα 30** |
| --- |

| **Τι τιμή έχει τώρα ο R0; Ήταν αυτό αναμενόμενο; Δικαιολογήστε το.** |
| --- |
| |
| |
| |
| |
| |

| **Ερώτημα 31** |
| --- |
| **Ποια η εντολή που υπάρχει στην νέα θέση του PC;**<br>**Ποια η λειτουργία της;** |
| |
| |
| |
| |
| |

Ξαναπατάμε το πλήκτρο Step ή το F11. Η τρέχουσα εντολή στην διεύθυνση του PC εκτελείται.

| **Ερώτημα 32** |
| --- |
| **Ποια είναι τα αποτελέσματα της νέας εντολής που εκτελέστηκε;**<br>**Που βλέπουμε τα αποτελέσματα αυτά;** |
| |
| |




**Σχήμα 15:** Παράθυρα στην επιλογή Debug

Τώρα ο Program Counter πρέπει να βρίσκεται στην διεύθυνση 0x124. Τα debug παράθυρα πρέπει να είναι όπως φαίνεται παρακάτω.

Παρατηρούμε ότι στο παράθυρο με τον C κώδικα το κίτρινο βέλος έχει παραμείνει στην εντολή SCU->PRR1 = 0x00EE2803, παρόλο που εκτελέσαμε 2 εντολές assembly. Αυτό συμβαίνει γιατί η εντολή αυτή χρειάζεται 3 assembly εντολές να πραγματοποιηθεί και εμείς έχουμε εκτελέσει τις 2 από αυτές. Όταν λογικά εκτελέσουμε και την επόμενη και ο PC φτάσει στην διεύθυνση 0x128 τότε η εντολή θα έχει πλήρως εκτελεστεί.

Στο παράθυρο Registers παρατηρούμε ότι ο R0 έχει την τιμή 0x00EE2803. Την απέκτησε όταν εκτελέστηκε ή εντολή LDR R0,[PC,#0x0080] στην διεύθυνση 0x11C. Αντίστοιχα ο R1 έχει την τιμή 0x5C002000 και την απέκτησε όταν εκτελέστηκε η εντολή LDR R1,[PC,#0x0080].

Η assembly εντολή στην διεύθυνση 0x124, στην διεύθυνση στην οποία πρέπει να βρίσκεται τώρα ο PC, είναι η εξής:

STR R0,[R1,#0x0020]

Η λειτουργία της είναι να αποθηκεύσει (STR = store) τα δεδομένα του καταχωρητή R0 στην διεύθυνση που υπάρχει στα περιεχόμενα του R1 συν 0x20.

| **Ερώτημα 33** |
| --- |
| **Ποια διεύθυνση τελικά προκύπτει από το [R1, #0x0020].**<br>**Ποια διεύθυνση καταχωρητή του MCU είναι αυτή;** |
| |
| |
| |

| **Ερώτημα 34** |
| --- |
| **Τι αποτέλεσμα θα περιμέναμε με την εκτέλεση της STR R0,[R1,#0x0020] εντολής;** |
| |
| |
| |

Πατάμε για τελευταία φορά το F11 ώστε να εκτελεστεί η παραπάνω εντολή;

| **Ερώτημα 35** |
| --- |
| **Πώς μπορούμε να δούμε εάν ο καταχωρητής SCU_PRR1 πήρε τελικά την τιμή που περιμέναμε;** |
| |