Εργαστήριο 7-8

Ονομα: Μαυρογιώργης Δημήτρης
ΑΜ: 2016030016
ΗΡΥ 411 - Ενσωματωμένα Συστήματα Μικροεπεξεργατών
Πολυτεχνειο Κρητης

April 19, 2021

Σκοπός

Σκοπός του έβδομου και όγδοου εργαστηρίου είναι να δημιουργήσουμε ένα απλό δρομολογητή με μικρή πολυπλοκότητα. Γι' αυτό το λόγο δημιουργήθηκαν τρεις απλές ρουτίνες οι οποίες κάνουν κάποιες αλλαγές στο PORTA. Επιπλέον, στο κυρίως πρόγραμμα, αντί να έχουμε ένα κενό while loop, κάνουμε κάποιους ελέγχουν για να δουμε πια ρουτίνα εκτελείται και ποια είναι η επόμενη, ώστε να κάνουμε το context switching μεταξύ αυτών των διεργασιών.

Περιγραφή της υλοποίησης

Αρχικά, έγιναν κάποια define των χαρακτήρων της κάθε εντολής, προκειμένου να γίνουν οι έλεγχου στον interrupt handler της USART ότι ήρθε σωστά η εντολή. Επίσης, δηλώθηκαν κάποια flag τα οποία ενεργοποιούν ή απενεργοποιούν οι TIMER0 και TIMER1, ενώ δεσμεύτηκε και χώρος στη μνήμη που αποθηκεύουμε κατα το context switch τα δεδομένα του PORTA για κάθε διεργασία ξεχωριστά. Επίσης, δηλώθηκαν και ένα array για τα enables των διεργασιών και ένα βοηθητικό byte στο οποίο αποθηκεύονται τα enable που λαμβάνονται από τη USART.

Πιο συγκεκριμένα, όσον αφορά την υλοποίηση, από τα προηγούμενα εργαστήρια έχουμε κρατήσει αυτούσια την αρχικοποίηση του timer0 με σκοπό να αφήνουμε την κάθε διεργασία να αλλάζει το PORTA κάθε 1ms και να μην γίνονται πολλές αλλαγές σε πολύ μικρό αριθμό κύκλων. Επιπλέον, από το προηγούμενο εργαστήριο χρησιμοποιήθηκαν αυτούσιες οι ρουτίνες για receive και transmit από τη σειριακή θύρα, καθώς και η αρχικοποίηση του USART.

Παράλληλα, για την υλοποίηση του timeslice των 100ms, χρησιμοποιήθηκε ο timer1. Για την αρχικοποίησή του ορίζουμε στον καταχωρητή TCCR1B prescaler 64, ενεργοποιούμε στον TIMSK το OVF interrupt του timer1 (bit TOIE1), ενώ αρχικοποιούμε τον 16-bit καταχωρητή TCNT1 με την τιμή 0xC2F6, η οποία προκύπτει ως εξής:

$$TCNT1 = MAX\_VAL - \frac{DELAY * FREQ}{PRESCALE} = 65535 - \frac{100ms * 10MHz}{64} = 49910$$

Στη συνέχεια, δημιουργήθηκε και μια βοηθητική συνάρτηση, η οποία ελέγχει αν αλλαξε κάποιο από τα enable bits στη global μεταβλητή func_enable ο USART handler και ανάλογα θέτει TRUE ή FALSE στην κατάλληλη θέση του array func_num. Με αυτό τον τρόπο, εξασφαλίζουμε ότι μια διεργασία δε θα τελειώσει πρόωρα, δίχως να κάνει context switch με την επόμενη στη σειρά.

```c
#include<avr/io.h>
#include<avr/interrupt.h>
#include <avr/wdt.h>

#define F_CPU 10000000
//#define FOSC 10000000
#define S_HEX 0x53
#define Q_HEX 0x51
#define X_HEX 0x58      // Used to indicate that we have stored a number - we don't care what number
#define O_HEX 0x4F
#define K_HEX 0x4B
#define CR_HEX 0x0D
#define LF_HEX 0x0A

#define TRUE 0x01
#define FALSE 0x00

#define BAUD 9600                       //Definition of the desired baud rate
#define UBRR_BAUD F_CPU/16/BAUD-1       //Calculation of UBRR's initial value
```

(a) C code for defines

```c
register unsigned char input_char asm ("15");

uint8_t prev_instr_char = 0;        //Keeps the char we receive

uint8_t OK_MSG[4] = {0x4F, 0x4B, 0xD, 0xA};     //Here we store OK<CR><LF> in memory

uint8_t process_data_1 = 0x00;
uint8_t process_data_2 = 0x00;
uint8_t process_data_3 = 0x00;

uint8_t timer_flag = FALSE;
uint8_t context_switch_flag = FALSE;

uint8_t current_proc = 0;
uint8_t func_enable = 0x00;
uint8_t func_num[3] = {FALSE, FALSE, FALSE};
```

(b) C code for declaring global variables

```c
void TCNT0_INIT(){
    TCNT0 = 0x63;               // Set TCNT0 the value of 99(0x63) = 255 - (1 ms)*(1 MHz)/64(prescale)
    TCCR0 = (1<<CS01)|(1<<CS00);    //Set prescale 64 - these bits of TCCR0 are set - CS02 = 0  CS01 = 1  CS00 = 1
    TIMSK |= (1<<TOIE0);            // TOIE1 is set to 1 in order to enable the timer overflow interrupt

    DDRA = 0xFF;    //Set as output PORTA AND PORT C
    PORTA = 0x00;   //Initialize PORTC with 0x00 WHICH mean that none of the digits has enable

    return;
}

void TCNT1_INIT(){
    TCNT1 = 0xC2F6;     //0xF2F6;
    TCCR1A = 0x00;
    TCCR1B = (1<<CS11)|(1<<CS10);   // Timer mode with 64 prescaler - these bits of TCCR1B are set - CS12 = 0  CS11 = 1  CS10 = 1

    TIMSK |= (1 << TOIE1);   // Enable timer1 overflow interrupt(TOIE1)

    return;
}
```

(c) C code for TIME0/TIMER1 initiliazation

```c
void CHECK_ENABLES(){       //Checking if there are changes on function's enables
    if((func_enable & (1<<0))){
        func_num[0] = TRUE;
    }else{
        func_num[0] = FALSE;
        process_data_1 = 0x00;
    }

    if((func_enable & (1<<1))){
        func_num[1] = TRUE;
    }else{
        func_num[1] = FALSE;
        process_data_2 = 0x00;
    }

    if((func_enable & (1<<2))){
        func_num[2] = TRUE;
    }else{
        func_num[2] = FALSE;
        process_data_3 = 0x00;
    }
    return;
}
```

(d) C code for checking new function enables

Όσον αφορά τις τρεις διεργασίες, η πρώτη διεργασία κάνει την εναλλαγή του PORTA από "10101010" σε " 01010101", η δεύτερη διεργασία είναι μια απλή υλοποίηση ενός ring counter, ενώ η τρίτη και τελευταία διεργασία είναι η υλοποίηση ενός μετρητή που μετρά από το "0x00" ως το "0xFF".

```c
void Process_1(){
    if(timer_flag == TRUE){
        return;
    }else{
        if(PORTA == 0x00){
            PORTA = 0b10101010; //Initialize first time with tis value
        }else{
            PORTA =~ PORTA;     //Inverse all bits every other time
        }
        timer_flag = TRUE;      //Don't change the output until timer0 makes it false
    }
    return;
}
```

(a) C code for Process 1

```c
void Process_2(){
    if(timer_flag == TRUE){
        return;
    }else{
        if(PORTA == 0x00 || PORTA == 0x80){ //Initialize the first time and when "1" reach the 8th bit
            PORTA = 0x01;
        }else{
            PORTA = (PORTA<<1); //Shift "1" to left
        }
        timer_flag = TRUE;  //Don't change the output until timer0 makes it false
    }
    return;
}
```

(b) C code for Process 2

```c
void Process_3(){
    if(timer_flag == TRUE){
        return;
    }else{
        if(PORTA == 0xFF){  //Implementation of a simple counter that counts from 0x00 to 0xFF
            PORTA = 0x00;
        }else{
            PORTA += 0x01;
            timer_flag = TRUE;  //Don't change the output until timer0 makes it false
        }
    }
    return;
}
```

(c) C code for Process 3

Στο κυρίως πρόγραμμα πέρα από τα initialize των timers και usart, έχουμε ένα loop στο οποίο αρχικά ελέγχουμε αν έχει έρθει κάποια διεργασία από τις τρεις για πρώτη φορά. Σε αυτή την περίπτωση, όποια διεργασία έρθει πρώτη την θεωρούμε ως current process.

Στη συνέχεια, αν είναι ενεργοποιημένη κάποια ρουτίνα, την εκτελούμε και ελέγχουμε αν έχει τελειώσει το timeslice των 100ms. Στην περίπτωση που έχει τελειώσει, ελέγχουμε για κάποια αλλαγή στα enables και αναλόγως πηγαίνουμε στην επόμενη. Αν πχ είμαστε στο Process 1 και είναι απενεργοποιημένη η 2 και ενεργοποιημένη η 3, τότε η επόμενη στη σειρά είναι η 3 οπότε κάνουμε context switch μεταξύ της 1 και της 3 παραλείποντας την 2. Συνεπώς, μέσα στο while loop ελέγχουμε όλους τους πιθανούς συνδυασμούς για το ποια είναι ενεργοποιημένη και ποια απενεργοποιημένη και αναλόγως κάνουμε το contex switch.

```c
int main(void){
    cli();                      //Turn off interrupts before initialization
    TCNT0_INIT();               //Init the timer0
    TCNT1_INIT();
    USART_INIT(UBRR_BAUD);      //Init the usart
    sei();                      //Enable interrupts after initilization

    while (1){
        if(current_proc == 0){  // The first time we check which function is enabled
            CHECK_ENABLES();
            if(func_num[0] == TRUE){
                current_proc = 1;
            }else if(func_num[1] == TRUE){
                current_proc = 2;
            }else if(func_num[2] == TRUE){
                current_proc = 3;
            }
        }
```

(a) C code for main initialization

```c
if(func_num[0] == TRUE && current_proc == 1){
    Process_1();
    if(context_switch_flag == TRUE){
        CHECK_ENABLES();                //Changing enables
        if(func_num[1] == TRUE){        // Next proccess is No2
            process_data_1 = PORTA;     // Store PORTA to process_data_1
            PORTA = process_data_2;     // Display process_data_2 to PORTA
            current_proc = 2;

        }else if(func_num[2] == TRUE && func_num[1] == FALSE){  //Only 1 and 3 are enabled
            process_data_1 = PORTA;                             // Store PORTA to process_data_1
            PORTA = process_data_3;                             // Display process_data_2 to PORTA
            current_proc = 3;
        }else{
            current_proc = 1;   //Only No1 is enabled
        }
        context_switch_flag = FALSE;
    }
```

(b) C code for Process 1 context switching

```c
}else if(func_num[1] == TRUE  && current_proc == 2){
    Process_2();
    if(context_switch_flag == TRUE){
        CHECK_ENABLES();                                //Changing enables
        if(func_num[0] == TRUE && func_num[2] == FALSE){    //Only process 1 and 2 are enabled
            process_data_2 = PORTA;                         // Store PORTA to process_data_2
            PORTA = process_data_1;                         // Display process_data_1 to PORTA
            current_proc = 1;

        }else if(func_num[2] == TRUE){  //Next process is No3
            process_data_2 = PORTA;     // Store PORTA to process_data_1
            PORTA = process_data_3;     // Display process_data_3 to PORTA
            current_proc = 3;
        }else{
            current_proc = 2;   //Only No2 is enabled
        }
        context_switch_flag = FALSE;
    }
```

(c) C code for Process 2 context switching

```c
}else if(func_num[2] == TRUE  && current_proc == 3){
    Process_3();
    if(context_switch_flag == TRUE){
        CHECK_ENABLES();                                //Changing enables
        if(func_num[1] == TRUE && func_num[0] == FALSE){    //Only process 2 and 3 are enabled
            process_data_3 = PORTA;                         // Store PORTA to process_data_2
            PORTA = process_data_2;                         // Display process_data_1 to PORTA
            current_proc = 2;

        }else if(func_num[0] == TRUE){  //Next process is No1
            process_data_3 = PORTA;     // Store PORTA to process_data_1
            PORTA = process_data_1;     // Display process_data_3 to PORTA
            current_proc = 1;
        }else{
            current_proc = 3;   //Only No1 is enabled
        }
        context_switch_flag = FALSE;
    }
}
```

(d) C code for Process 3 context switching

Οι interrupt handler των δύο TIMER0 και TIMER1, είναι πολύ απλοί και αυτό που κάνουν είναι να θέτουν TRUE κάποια flags για το πότε να αλλάξει μια διεργασία το PORTA και πότε γίνεται το context switch μεταξύ των διεργασιών. Για τον handler του USART, αυτό που κάνουμε είναι κα ελέγχουμε αν ήρθε κάποιο enable ή disable κάποιας διεργασίας και αναλόγως.

Όσον αφορά το USART, αυτό που κάνουμε είναι να διαβάζουμε χαρακτήρα χαρακτήρα την κάθε εντολή. Στην περίπτωση που λαμβάνουμε κάποια εντολή Sx<CR><LF>, αποθηκεύουμε στο (x-1)-bit του byte func_enable την τιμή "1". Αν πχ έρθει η τιμή 1, κάνουμε set το bit 0 στο "1", αν έρθει 2 κάνουμε το bit 1 set "1" και, τέλος, αν έρθει 3 κάνουμε το bit 2 set στον "1".

Αντίθετα, στην περίπτωση που έρθει κάποιο Qx<CR><LF>, αποθηκεύουμε στο (x-1)-bit του byte func_enable την τιμή "0". Έτσι, αν έρθει η τιμή 1, κάνουμε clear το bit 0, αν έρθει 2 κάνουμε clear το bit 1 και αν έρθει 3 κάνουμε clear το bit 2.

```c
ISR(TIMER1_OVF_vect){
    TCNT1 = 0xC2F6;             // 0xF2F6
    context_switch_flag = TRUE;     //This flag indicates that we have to make context switch
}

ISR(TIMER0_OVF_vect){
    TCNT0 = 0x63;
    timer_flag = FALSE; //This flag indicates that current process has to change PORTA
}
```

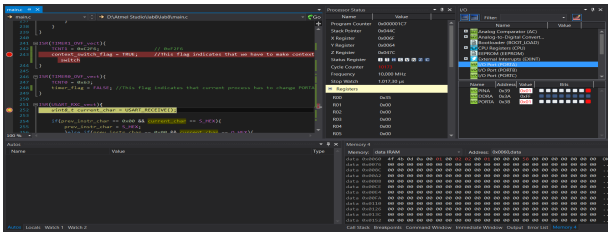
(a) C code for TIMER0/TIMER1 OVF interrupt handlers

```c
ISR(USART_RXC_vect){
    uint8_t current_char = USART_RECEIVE();

    if(prev_instr_char == 0x00 && current_char == S_HEX){
        prev_instr_char = S_HEX;
        }else if(prev_instr_char == 0x00 && current_char == Q_HEX){
        prev_instr_char = Q_HEX;
    }else if(prev_instr_char == S_HEX && (current_char > 0x30 && current_char < 0x34)){
        if(current_char == 0x31){
            func_enable |= (1<<0);  //Set "1" bit 0 of this "register"
        }else if(current_char == 0x32){
            func_enable |= (1<<1);  //Set "1" bit 1 of this "register"
        }else{
            func_enable |= (1<<2);  //Set "1" bit 2 of this "register"
        }
        prev_instr_char = X_HEX;

    }else if(prev_instr_char == Q_HEX && (current_char > 0x30 && current_char < 0x34)){
        if(current_char == 0x31){
            func_enable &= ~(1<<0); //Clear bit 0 of this "register"
        }else if(current_char == 0x32){
            func_enable &= ~(1<<1); //Set "1" bit 0 of this "register"
        }else{
            func_enable &= ~(1<<2); //Set "1" bit 0 of this "register"
        }
        prev_instr_char = X_HEX;

    }else if(prev_instr_char == X_HEX && current_char == CR_HEX){
        prev_instr_char = CR_HEX;

    }else if(prev_instr_char == CR_HEX && current_char == LF_HEX){
        prev_instr_char = LF_HEX;
        USART_TRANSMIT();   //Send OK<CR><LF>
    }else{
        prev_instr_char = 0x00;
    }
}
```

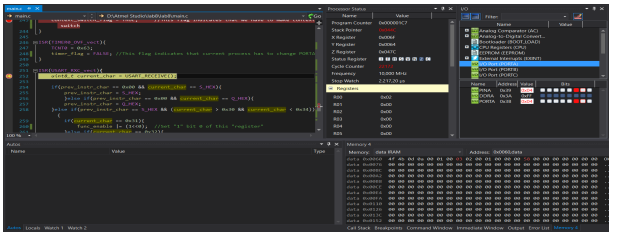
(b) C code for USART RXC interrupt handler

Προσομοίωση Αποτελεσμάτων

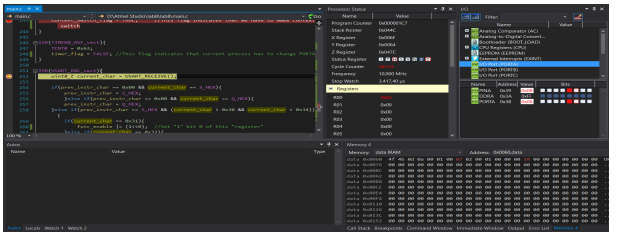
Αρχικά, στην εικόνα (a) βλέπουμε ότι λαμβάνουμε σωστά από τη USART την εντολή S2<CR><LF> και αποθηκεύονται σωστά οι τιμές των enable τόσο στο array func_num όσο και στη μεταβλητή func_enable. Στο array αποθηκεύεται η τιμή "0x01" στη θέση 1, ενώ στη μεταβλητή η τιμή είναι "0x02" που σημαίνει οτι το bit 1 είναι "1". Ομοίως, στην εικόνα (b) βλέπουμε ότι βλέπουμε ότι λαμβάνουμε το S1<CR><LF> και στο array αποθηκεύεται η τιμή "0x01" στη θέση 2, ενώ στη μεταβλητή η τιμή είναι "0x03" που σημαίνει οτι το bit 0 και 1 είναι "1". Τέλος, βλέπουμε και η εντολή S3<CR><LF> λαμβάνεται σωστά και το array έχει την τιμή "0x01" σε όλες τις θέσεις του και η μεταβλητή έχει την τιμή "0x07" που σημαίνει ότι τα bit 0,1 και 2 είναι "1".



(a) Results from Atmel Studio 7 - Only Process 2 is enabled

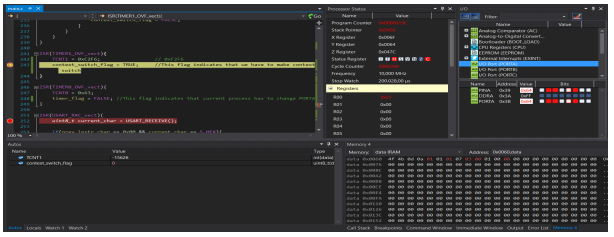


(b) Results from Atmel Studio 7 - Process 1 is enabled

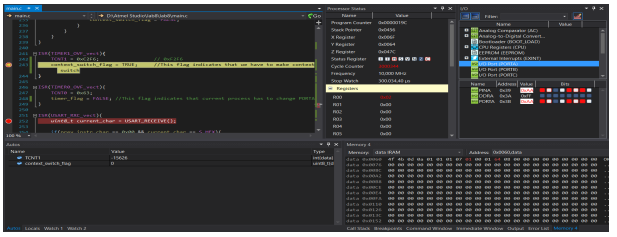


(c) Results from Atmel Studio 7 - Process 3 is enabled

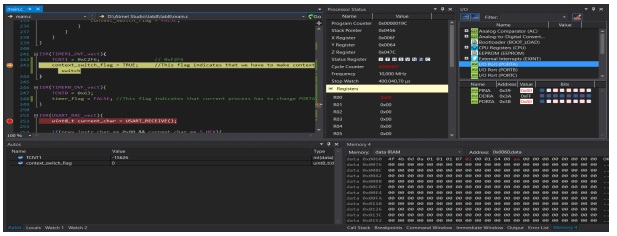
Στις παρακάτω εικόνες βλέπουμε το context switch μεταξύ των διεργασιών. Στην εικόνα (a) φαίνεται ότι τα δεδομένα που είχε επεξεργαστεί το Process 2 στο PORTA, έχουν αποθηκευτεί στη μνήμη. Παράλληλα, βλέπουμε ότι έχει έρθει και ο TIMER1 και το timeslice της διεργασίας 3 τελείωσε και πρέπει να γίνει context switch μεταξύ της 3 και της 1. Στην επόμενη εικόνα (b), φαίνεται ότι τα δεδομένα του Process 3 αποηθηκεύτηκαν και αυτά στη μνήμη, ενώ ήρθε και πάλι ο TIMER1, αυτή τη φορά για το context switch μεταξύ της 1 και της 2. Τέλος, όπως φαίνεται στην εικόνα (c), και τα δεδομένα του Process 1 αποθηκεύτηκαν στη μνήμη και γίνεται context switch μεταξύ της 2 και της 3 για δεύτερη φορά.



(a) Results from Atmel Studio 7 - Context Switch between Process 2 and 3

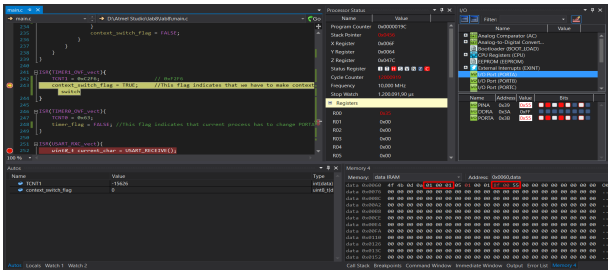


(b) Results from Atmel Studio 7 - Context Switch between Process 3 and 1

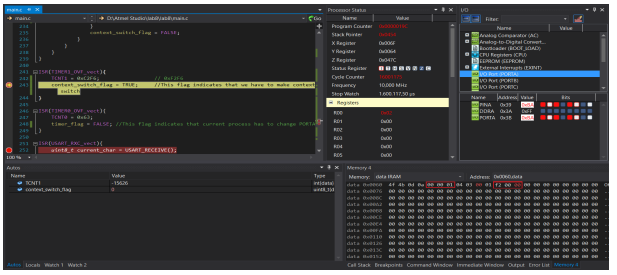


(c) Results from Atmel Studio 7 - Context Switch between Process 1 and 2

Στην εικόνα (a) φαίνεται ότι μετά την εντολή Q2<CR><LF> η διεργασία 2 απενεργοποιείται και τα δεδομένα της καθαρίζονται από τη μνήμη. Επίσης, βλέπουμε ότι μετά το context switch μεταξύ της 2 και της 3 έγινε σωστά και η 2 έγινε quit αμέσως μετά το context switch. Επίσης, στην ίδια εικόνα φαίνεται ότι είμαστε στην περίοδο που το timeslice της 1 τελείωσε και τα δεδομένα της θα αποθηκευτούν στη μνήμη. Στην εικόνα (b) βλέπουμε ότι γίνεται σωστά και η εντολή Q1<CR><LF> και η μοναδική διεργασία που έμεινε είναι η 3. Σε αυτή την περίπτωση αφού είναι η μοναδική διεργασία που αλλάζει το PORTA, δε χρειάζεται να αποθηκεύσουμε κάτι στη μνήμη εφόσον δεν γίνεται κάποιο context switch.

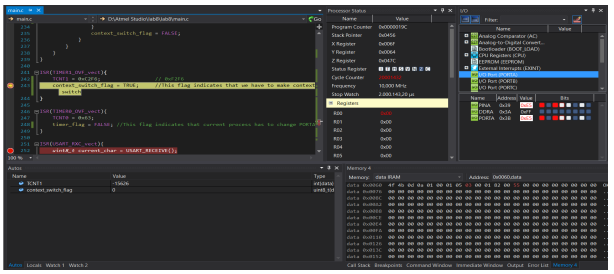


(a) Results from Atmel Studio 7 - Quit process 2

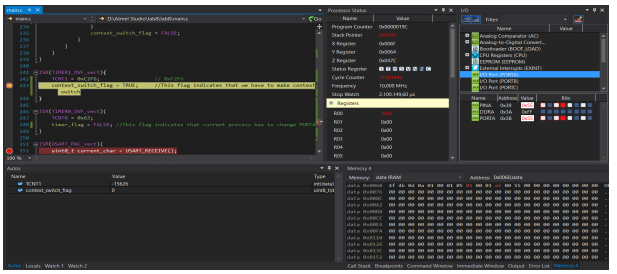


(b) Results from Atmel Studio 7 - Quit process 1

Στις παρακάτω εικόνες, κάνουμε ξανά start την διεργασία 1 την οποία είχαμε σταματήσει. όπως φαίνεται στην εικόνα (a) η εντολή start εκτελέστηκε σωστά και οι ενεργές διεργασίες είναι η 1 και η 3. Επίσης, φαίνεται ότι στο προηγούμενο timeslice τα δεδομένα της διεργασίας 1 αποθηκεύτηκαν στη μνήμη, ενώ ήρθε ο TIMER1, καθώς το timeslice της 3 τελείωσε. Στην τελευταία εικόνα βλέπουμε το αντίθετο, δηλαδή στο προηγούμενο timeslice τα δεδομένα της διεργασίας 3 αποθηκεύτηκαν στη μνήμη, ενώ ήρθε ο TIMER1, καθώς το timeslice της 1 τελείωσε και συνεπώς πρέπει να γίνει context switch μεταξύ των διεργασιών.



(a) Results from Atmel Studio 7 - Start process 1



(b) Results from Atmel Studio 7 - Context Switch between Process 3 and 1