

Θεόδωρος Μπάρκας 2016030050 LAB 7 και 8

Σχόλια

Παρακατώ βρίσκετε ολόκληρη η αναφορά και του εργαστηρίου 7 και του εργαστηρίου 8.

Σκοπός άσκησης

Στη Συγκεκριμένη άσκηση σκοπός ήταν η υλοποίηση ενός απλού δρομολογητή. Πιο συγκεκριμένα ενός δρομολογητή από 0 έως 3 διεργασιών .Οι ενεργές διεργασίες δίνονται από την USART μέσω της εντολής Sx<CR><LF> , Qx<CR><LF> για Start και Quit αντίστοιχα .

Νέα δέσμευση μνήμης και MEMORY MAP

0x60 to 0x67	displayNumbers	
0x68 to 0x72	SSD decoder	
0x73	curNumberIndex	
0x74	ringCounter	
0x75	curState	FSM of USART RXC interrupt
0x76	input	Input of USART (only for testing)
0x77	sliceCounter(NEW)	Helps to count from counter0 4ms -> 100ms (values:0 to 24)
0x78	sliceFlag(NEW)	Set equal to one when we want to run a process (values:0 or 1)
0x79	curProcess(NEW)	When sliceFlag==1 in main we run curProcess (values:0,1,2)
0x7A	processesFlags(NEW)	Flags of Open(1) or Close Processes(0) bitwise (ex.if process 1(LS) and 3(MS) running processesFlags=0b00000101)
0x7B	ringCounterRight	For Process1
0x7C	ringCounterLeft	For Process2
0x7D	valueMem	For Process3

Για τις θέσεις μνήμης αξίζει να αναφερθεί ότι θα μπορούσαν να περιοριστούν (όλες οι NEW για παράδειγμα χωράνε εντός 2 Bytes).Ακόμα και έτσι θα χάναμε σε κύκλους και σε παραπάνω Flash για κωδικά.

Υλοποίηση - Μετατροπή FSM της USART

Εισάγαμε τις δυο νέες καταστάσεις Sstate και Qstate οι οποίες αν και εφόσον βρεθούμε σε αυτές Αφού “μασκάρουμε” :

maskedInput=input * 0x0F;

κάνουμε ένα local variable:

compProcessFlag=1<<(maskedInput-1);

δηλαδή ενεργοποιούμε το κατάλληλο bit με κλειστά όλα τα άλλα

Αυτά τα κάνει η συνάρτηση `unsigned char maskAndSetBit(unsigned char input)`

→ Κατόπιν στην περίπτωση του start αποθηκεύουμε στη μνήμη :

`*processesFlagsMem=processesFlags | compProcessFlag;`

→ Στην περίπτωση του quit αποθηκεύουμε στη μνήμη :

`*processesFlagsMem=processesFlags & ~compProcessFlag;`

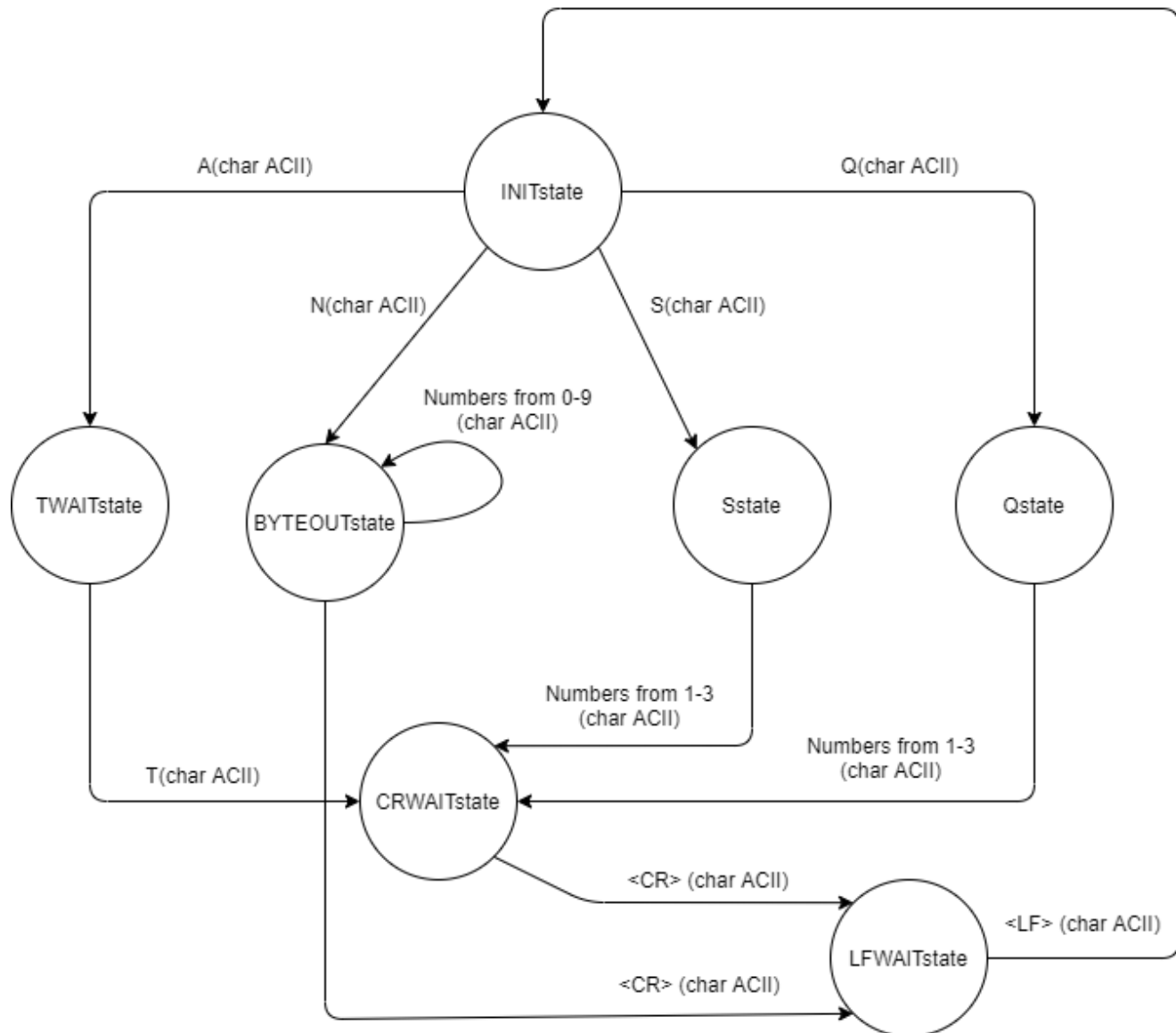
```
case Sstate:          //S state wait for number 1-3
compProcessFlag=maskAndSetBit(input);
*processesFlagsMem=processesFlags|compProcessFlag;
*stateMem=CRWAITstate;
break;
case Qstate:          //LF Just wait for number 1-3
compProcessFlag=maskAndSetBit(input);
compProcessFlag=~compProcessFlag;
*processesFlagsMem=processesFlags & compProcessFlag;
*stateMem=CRWAITstate;
break;
```

κώδικας εντός της FSM

```
unsigned char maskAndSetBit(unsigned char input){
    unsigned char maskedInput=input & 0x0F;
    unsigned char compProcessFlag;
    compProcessFlag = 1<<(maskedInput-1);
    return compProcessFlag;
}
```

`unsigned char maskAndSetBit(unsigned char input)`

FSM



Υλοποίηση του δρομολογητή

Ο δρομολογητής περιλαμβάνει δύο Functions που καλούνται από το εσωτερικό του TIMER0_OVF:

```
//SLICE COUNTER
unsigned char *processesFlagsMem=(unsigned char *) 0x80;//3FlagBits
unsigned char *sliceFlagMem=(unsigned char *) 0x78;
unsigned char processesFlags=processesFlagsMem;

unsigned char sliceFlag = sliceCounterFlag();
//even if we got a sliceFlag we don't want to change Process if we don't have one
//Because we gonna stuck in while loop of findNextProcess
if(sliceFlag==0x01 && processesFlags!=0){
    findNextProcess();
}
else if(sliceFlag==0x01){
    *sliceFlagMem=0x00;
}

void findNextProcess();
unsigned char sliceCounterFlag();
```

Πιο συγκεκριμένα:

- Ο `sliceCounterFlag()` καλείτε κάθε φορά που δίνετε interrupt ο TIMER0_OVF .Θυμίζουμε ότι το συγκεκριμένο Interrupt άρχετε κάθε 4ms έτσι ώστε να επιτυγχάνουμε λίγο πάνω από το ελάχιστο Refresh rate των 30Hz .Επομένως ο `sliceCounterFlag` μετράει 25 κύκλους και επιστρέφει τιμή είτε 1 είτε 0 ανάλογα με το αν ολοκλήρωσε ή όχι ο `sliceCounter` δηλαδή αν έφτασε στην τιμή 24 .(1 TIMER0_OVF → 4ms, 2 TIMER0_OVF → 8ms,...,24 TIMER0_OVF → 100ms κ.ο.κ.) εξ ου και η σύγκριση γίνεται ως `sliceCounter==sliceCounterSize-1`

```
unsigned char sliceCounterFlag(){
    //MEMORY LOADING
    unsigned char *sliceCounterMem=(unsigned char *) 0x77;
    unsigned char *sliceFlagMem=(unsigned char *) 0x78;
    unsigned char sliceCounter=*sliceCounterMem;
    //LOGIC
    if(sliceCounter==sliceCounterSize-1){
        *sliceFlagMem=1;//MEMORY STORING
        sliceCounter=0;
    }else{
        sliceCounter++;
    }
    *sliceCounterMem=sliceCounter;//MEMORY STORING
    return *sliceFlagMem;
}
```

- Ο `findNextProcess()`; Καλείτε όταν δίνεται Flag και ταυτόχρονα δεν είναι άδεια όλα τα `processesFlags` της μνήμης.

Το πότε καλείτε φαίνεται ευκολότερα από τον κωδικά μέσα στον `TIMER0_OVF` interrupt.

Λειτουργιά

Αφού ο `nextProcessFlag` αρχικοποιηθεί στο `0x00` `unsigned char nextProcessFlag=0x00;` .

Τρέχουμε ένα `whileLoop` : `while(nextProcessFlag==0)`

Περνάμε με τη σειρά από τον επόμενο του τωρινού `curProcess` και εκτελούμε τις επόμενες τιμές και εκτελούμε :

→ `nextProcessFlagComp=1<<curProcess;`

και κατόπιν

→ `nextProcessFlag=processesFlags & nextProcessFlagComp;`

Αυτό δίνει 0 αν το `curProcess` δεν υπάρχει ή κάτι άλλο (1 bit ανοικτό στην αντίστοιχη θέση (bitwise) του `Process`) αν βρέθηκε το επόμενο `Process`.

Σε περίπτωση που βρέθηκε επομένως γίνεται `break` και έχουμε στον `curProcess` τι τιμή της επόμενης διεργασίας που πρέπει να εκτελέσουμε.

Επομένως το μόνο που μένει να κάνουμε είναι να τον αποθηκεύσουμε στη μνήμη.

```
void findNextProcess(){
    //MEMORY LOADING
    unsigned char *curProcessMem=(unsigned char *) 0x79;
    unsigned char *processesFlagsMem=(unsigned char *) 0x80;//3FlagBits
    unsigned char processesFlags=*processesFlagsMem;
    unsigned char curProcess=*curProcessMem;
    //LOCAL VARIABLES
    unsigned char nextProcessFlag=0x00;
    unsigned char nextProcessFlagComp;
    //LOGIC
    while(nextProcessFlag==0){
        curProcess++;
        if(curProcess==totalProcesses){
            curProcess=0x00;
        }
        nextProcessFlagComp=1<<curProcess;
        nextProcessFlag=processesFlags & nextProcessFlagComp;
    }
    //MEMORY STORING
    *curProcessMem=curProcess;
}
```

Τέλος αξίζει να σημειωθεί ότι σε περίπτωση που έχουμε `sliceFlag==1` και είναι άδειος ο `processesFlags` δηλαδή δεν τρέχουμε `Process` δεν πρέπει να τρέξουμε την `findNextProcess()`; καθώς θα μπει σε `InfiniteLoop` .

Επίσης σε αυτή τη περίπτωση θέτουμε το `sliceFlag==0` καθώς δεν χρειάζεται να τρέξουμε καμία διεργασία.

Λειτουργία main() : Idle Loop και δρομολογήση διεργασιών εντός του main()

Σε περίπτωση που μας δρομολογείτε κάτι θέλουμε η main() να το τρέξει και αφού το τρέξει να μείνει σε IDLE LOOP έως ότου έρθει κάτι καινούργιο (sliceFlag==0x01)σε κάθε άλλη περίπτωση παραμένει σε IDLE LOOP .Σε περίπτωση που δρομολογηθεί μια διεργασία τσεκάρουμε πια είναι μεσώ μιας switch(curProcess) την εκτελούμε.

Επίσης κάθε φορά που μας δρομολογείτε μια διεργασία επαναφέρουμε το sliceFlag της μνήμης στο 0.

```
int main(void)
{
    INIT();
    unsigned char *sliceFlagMem=(unsigned char *) 0x78;
    unsigned char *curProcessMem=(unsigned char *) 0x79;
    unsigned char volatile sliceFlag;
    unsigned char volatile curProcess=*curProcessMem;
    while (1)
    {
        if(sliceFlag==0x01){
            //run code
            curProcess=*curProcessMem;
            switch(curProcess){
                case 0x00:
                    asm("call ringCounterRightProc");
                    break;
                case 0x01:
                    asm("call ringCounterLeftProc");
                    break;
                case 0x02:
                    asm("call zerosToOnesAndBack");
                    break;
            }
            *sliceFlagMem=0x00;
        }
        sliceFlag = *sliceFlagMem;
    }
}
```

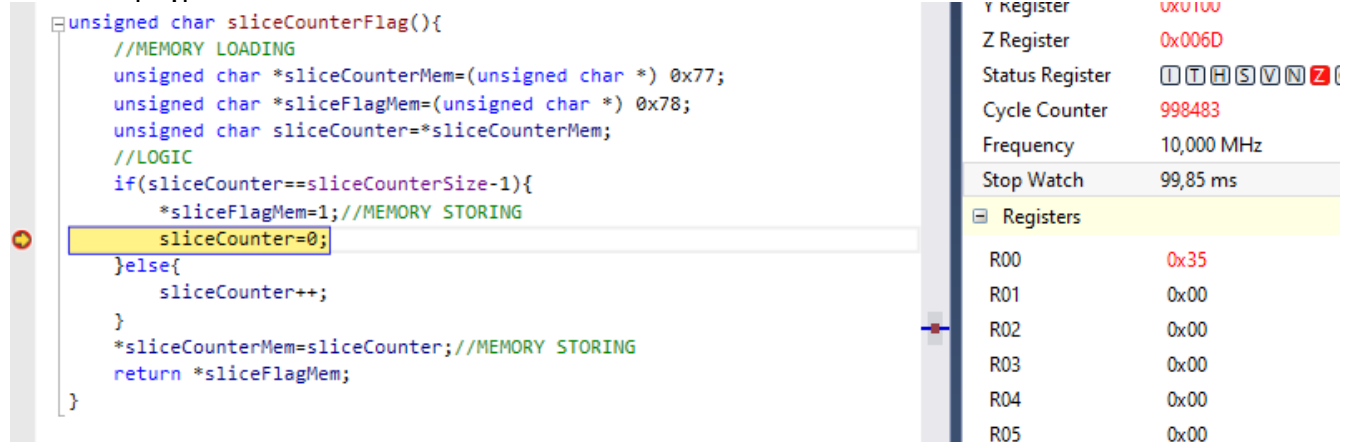
Οι 3 διεργασίες που καλούνται απο την main είναι πολύ άπλες και τα δεδομένα τους εμπεριεχουν για αποθήκευση στη RAM από ένα Byte το καθένα όπως φαίνεται και στο Memory Map στην αρχή.

```
RingCounterRightProc();    //0x01 → 0x02 → ... → 0x80→ 0x01 → ...
ringCounterLeftProc();    //0x80 → 0x40 → ... → 0x01→ 0x80 → ...
zerosToOnesAndBack();    //makes 0x00 → 0xFF → 0x00 → ...
```

Επίσης κάθε διεργασία τρέχει μόνο από μια φορά δηλαδή θα πάμε από 0x01 σε 0x02 και στο επόμενο sliceFlag που εμπλέκετε η διεργασία θα πάμε από 0x02 σε 0x04 κ.ο.κ.

TESTING

Σε πρώτη φάση έθεσα το sliceCounterSize=0x19=25(Δεκαδικό) για να δω αν το Flag ανοίγει κάθε 100ms .Πράγματι:



```
unsigned char sliceCounterFlag(){
    //MEMORY LOADING
    unsigned char *sliceCounterMem=(unsigned char *) 0x77;
    unsigned char *sliceFlagMem=(unsigned char *) 0x78;
    unsigned char sliceCounter=*sliceCounterMem;
    //LOGIC
    if(sliceCounter==sliceCounterSize-1){
        *sliceFlagMem=1;//MEMORY STORING
        sliceCounter=0;
    }else{
        sliceCounter++;
    }
    *sliceCounterMem=sliceCounter;//MEMORY STORING
    return *sliceFlagMem;
}
```

Y Register	0x0100
Z Register	0x006D
Status Register	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
Cycle Counter	998483
Frequency	10,000 MHz
Stop Watch	99,85 ms
Registers	
R00	0x35
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00

βλέπουμε ότι ο χρόνος είναι αρκετά κοντά στα 100ms.

Υστερα έθεσα το sliceCounterSize=0x02 (TESTING) έτσι ώστε να μην καθυστερεί η προσομοίωση (η προσομοίωση των 100ms πήρε στον υπολογιστή μου τουλάχιστον 20sec).

```
#define sliceCounterSize 0x02 //just for testing Set to 0x19=25 for 100ms
```

Για το σωστό Testing έχουμε φτιάξει μερικά stim file για να εισάγουμε δεδομένα από τη Usart και να δούμε αν λειτουργεί σωστά.

Βρίσκονται στο φάκελο debug :

και έχουμε τα :

- stimEnableProcess1.stim → όπου εκτελεί S1<CR><LF>
- stimEnableProcess3.stim → όπου εκτελεί S3<CR><LF>
- stimQuitProcess1.stim → όπου εκτελεί Q1<CR><LF>

Επίσης θέτουμε τα 3 break Points στη main από την οποία θα παρακολουθούμε τη σωστή λειτουργία ελέγχοντας τις θέσεις μνήμης αλλά και το αποτέλεσμα στο PORTB από το οποίο επίσης φαίνεται η σωστή λειτουργία του προγράμματος.

Επίσης σε κάθε μήνυμα βάζουμε τα ανάλογα BreakPoint για να δούμε ότι μπαίνει σωστά στα States και κάνει όλες τις απαραίτητες διαδικασίες όπως πχ αποστέλνει OK<CR><LF> κ.ο.κ.

→ Προτού τρέξω οποιοδήποτε stimFile βλέπω ότι το πρόγραμμα μου μένει σε IDLE LOOP περιμένοντας να ενεργοποιηθεί το flag. Δηλαδή παρότι ο Slice Counter ολοκληρώνει η τιμή του sliceFlag ξανά γίνεται 0x00 αφού δεν υπάρχει κάποιο Process να τρέξουμε.

(Σχόλιο: Στα παρακάτω test οι δυο πρώτες τιμές της δεύτερης γραμμής της μνήμης εμπεριέχουν το curProcess , το processesFlags) και οι υπόλοιπες της δεύτερης γραμμής τα Process Variables ακριβώς όπως ορίζετε και από το Memory Map

→ Εν συνεχεία τρέχοντας το stimEnableProcess1.stim:

Βλέπω ότι εκτελείτε το Process1 όπου κάνει shiftRight τον ring Counter ότι αποθηκεύεται στη μνήμη και ότι εμφανίζεται ως έξοδος στο PORTB.

Αφήνοντας το ως έχει βλέπω ότι όταν ξαναέρχεται το Flag ξανατρέχει από το σημείο που έμεινε οπότε αυτή είναι η αναμενόμενη λειτουργία.

I/O	DDRB	0x37	0x00	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I/O	PORTB	0x38	0x02	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

```
Memory 4
Memory: data IRAM
ata 0x0060 05 05 05 05 05 05 05 05 01 4f 1
ata 0x0079 00 01 02 00 00 00 00 00 00 00 0
```


→ Ύστερα έτρεξα το stimEnableProcess3.stim:

και παρατήρησα εναλλαγή κάθε φορά που έρχονταν sliceFlag μεταξύ των δύο διεργασιών .Επίσης η κάθε διεργασία συμφωνά με τις προδιαγραφές κρατάει το State της αφού οι πληροφορίες τους είναι αποθηκευμένες στη μνήμη.

1ο sliceFlag μετά την ενημέρωση από USART(PROCESS 3):

I/O	PORTB	0x30	0x00	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I/O	DDRB	0x37	0x00	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I/O	PORTB	0x38	0xFF	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Memory 4											
Memory:	data IRAM										Addr
0xa 0x0060	05	05	05	05	05	05	05	05	01	4f	
0xa 0x0079	02	05	02	00	ff	00	00	00	00	00	

2ο sliceFlag μετά την ενημέρωση από USART(PROCESS 1):

I/O	PORTB	0x30	0x00	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I/O	DDRB	0x37	0x00	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
I/O	PORTB	0x38	0x04	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Memory 4											
Memory:	data IRAM										Addr
0xa 0x0060	05	05	05	05	05	05	05	05	01	4f	
0xa 0x0079	00	05	04	00	ff	00	00	00	00	00	

→ Τέλος έτρεξα το stimQuitProcess1.stim :

και παρατήρησα ότι σταμάτησε να εκτελείτε η διεργασία 1 και εκτελούνταν μόνο η 3 η οποία κάθε φορά που έρχονταν το sliceFlag πήγαινε στο επόμενο βήμα.

1ο sliceFlag μετά την ενημέρωση από USART(PROCESS 3):

I/O DDRB 0x37 0x00 ☐☐☐☐☐☐☐☐
I/O PORTB 0x38 0x00 ☐☐☐☐☐☐☐☐

Memory 4											
Memory:			data IRAM								
					Address						
0x0060	05	05	05	05	05	05	05	05	01	4f	12
0x0079	02	04	04	00	00	00	00	00	00	00	00

2ο sliceFlag μετά την ενημέρωση από USART(PROCESS 3):

I/O DDRB 0x37 0x00 ☐☐☐☐☐☐☐☐
I/O PORTB 0x38 0xFF ☒☒☒☒☒☒☒☒

Memory 4											
Memory:			data IRAM								
					Address						
0x0060	05	05	05	05	05	05	05	05	01	4f	12
0x0079	02	04	04	00	ff	00	00	00	00	00	00

Με παρόμοιο τρόπο επιβεβαιώνουμε τη σωστή συμπεριφορά και του Process2 καθώς και άλλα πιθανά cases .