Μελέτη Λειτουργικού Συστήματος FreeR

1. Ζητούμενο

Ο σκοπός αυτής της εργασίας είναι η δημιουργία μιας εφαρμογής στο λειτουργικό σύστημα πραγματικού χρόνου ανοιχτού κώδικα FreeRTOS με γλώσσα C κάτω από το περιβάλλον AVR32 Studio της Atmel και ο προγραμματισμός με το μεταγλωττισμένο πρόγραμμα του μικροελεγκτή AT32UC3A0512ES που βρίσκεται πάνω στην πλακέτα δοκιμών EVK1100.

Η εφαρμογή θα αποτελείται από διάφορες εργασίες (tasks) του FreeRTOS που θα επικοινωνούν μεταξύ τους και θα συνεργάζονται. Ο χρονοδρομολογητής του FreeRTOS να ρυθμιστεί σε **περίοδο εναλλαγής εργασιώ ν (tick time) ίση με 1ms** με χρονισμό του μικροελεγκτή από τον κρυσταλλικό ταλαντωτή (FOSC0), δηλαδή
συχνότητα ρολογιού 12MHz
.

Η πρώτη από τις εργασίες θα επεξεργάζεται δείγματα αναλογικού σήματος που θα φέρνει ο ADC από το ποτενσιόμετρο που υπάρχει πάνω στην πλακέτα (**εργασία ADC**). Ο ADC θα έχει **συχνότη τα δειγματοληψίας 10Hz**
. Δηλαδή το σήμα διέγερσης (trigger) θα δίνεται κάθε 100ms και η διέγερση θα γίνεται από το υλικό και συγκεκριμένα από ένα κανάλι του περιφερειακού Timer/Counter που θα έχει προγραμματιστεί κατάλληλα για αυτό το σκοπό. Η συχνότητα ρολογιού του ADC (δηλαδή η ταχύτητα μετατροπής του κάθε δείγματος) να είναι η μέγιστη δυνατή. Με το τέλος της μετατροπής κάθε αναλογικού δείγματος σε ψηφιακό από τον ADC θα πρέπει να προκαλείται μία διακοπή που θα εξυπηρετείται από μία ρουτίνα εξυπηρέτησης διακοπής (Interrupt Service Routine). Η ρουτίνα εξυπηρέτησης διακοπής θα διαβάζει την τιμή από τον ADC και με τη χρήση σημαφόρου θα ενεργοποιεί την εργασία επεξεργασίας των δειγμάτων και φυσικά θα επιστρέφει τον έλεγχο στο χρονοδρομολογητή. Η εργασία επεξεργασίας

δειγμάτων το μόνο που θα κάνει θα είναι μία τεχνητή καθυστέρηση που θα εξαρτάται από την τιμή που θα διαβάζεται από ποτενσιόμετρο. Η τιμή αυτή να προγραμματιστεί έτσι ώστε να καλύπτει την περιοχή από 0-120ms σε γραμμική αντιστοιχία με τις τιμές 0-1023 που θα δίνει ο ADC (ανάλυση 10 bits). Αφού εκτελεστεί η τεχνητή καθυστέρηση η εργασία θα μπλοκάρεται μέχρι να ενεργοποιηθεί ξανά από το σημαφόρο της ρουτίνας εξυπηρέτησης διακοπής στην επόμενη παραλαβή του δείγματος από τον ADC.

Ταυτόχρονα με την προηγούμενη εργασία θα εκτελείται μία άλλη **εργασία σποραδικώ ν συμβάντων** που γενικά θα είναι μπλοκαρισμένη, αλλά θα ξεμπλοκάρεται όποτε παραλαμβάνει έναν άλλο σημαφόρο από μία διαφορετική ρουτίνα εξυπηρέτησης διακοπής. Η ρουτίνα αυτή θα ενεργοποιείται όποτε υπάρχει διακοπή εξ΄ αιτίας πίεσης του διακόπτη PB0 που υπάρχει πάνω στην πλακέτα. Το ξεμπλοκάρισμα της εργασίας θα έχει σαν αποτέλεσμα την εκτέλεση μίας άλλης εικονικής καθυστέρησης ακριβώς 20ms μέχρι να ξανασυμβεί μπλοκάρισμα.

Και οι δύο παραπάνω εργασίες θα επικοινωνούν μέσω μίας κοινής ουράς FIFO με μία τρίτη **εργασία επικοινωνίας**
με τον έξω κόσμο που θα κάνει χρήση της σειριακής σύνδεσης (USART0) προκειμένου να αποστέλλονται δεδομένα σε έναν υπολογιστή. Τα δεδομένα που θα αποστέλλονται από τις δύο πρώτες εργασίες προς την εργασία επικοινωνίας θα είναι τα ακόλουθα. Για την εργασία του ADC ο πραγματικός χρόνος απόκρισης (response time) της εργασίας του ADC από την στιγμή που έγινε η διακοπή για την παραλαβή του δείγματος μέχρι τη στιγμή που μπλοκάρεται η εργασία καθώς και η τιμή που διάβασε ο ADC. Για τη σποραδική εργασία επίσης ο χρόνος από τη στιγμή που έγινε αντιληπτή η διακοπή μέχρι τη στιγμή που ξαναμπλοκαρίστηκε η εργασία μαζί με ένα ακέραιο σταθερής τιμής ?1 που θα δηλώνει ότι ο χρόνος αυτός αφορά τη δεύτερη εργασία. Η μέτρηση αυτών των χρόνων μπορεί να γίνει με πολύ μεγάλη ακρίβεια με την ανάγνωση του καταχωρητή συστήματος Cycle Count Register (COUNT) και υπολογισμό της διαφοράς διαδοχικών μετρήσεων. Η εργασία επικοινωνιών θα στέλνει στη σειριακή θύρα τα δεδομένα (σε μορφή κειμένου ASCII) με ταχύτητα 57600bps και ρυθμίσεις του πρωτοκόλλου UART: no Parity, 8 data bits, 1 stop bit.

Οι προτεραιότητες των εργασιών θα είναι με φθίνουσα σειρά: μέγιστη προτεραιότητα θα έχει η εργασία του ADC, ακολουθεί η εργασία των σποραδικών γεγονότων, και τέλος η εργασία επικοινωνιών. Το διάγραμμα που υπάρχει παρακάτω εμφανίζει μία περίπτωση εναλλαγής διεργασιών, χωρίς να περιλαμβάνεται η εργασία επικοινωνιών.

Το ζητούμενο είναι να γίνουν οι εξής στατιστικές μετρήσεις:

(α) Να βρεθεί προσεγγιστικά η κατανομή πιθανότητας του χρόνου απόκρισης (response time) για την **εργασία σποραδικώ ν γεγονότων** για την περίπτωση που η εργασία ADC έχει χρόνο επεξεργασίας **20ms**, **40ms**, **60 ms**
,
70ms
,
80ms, 90ms
.

(β) Να βρεθεί προσεγγιστικά η κατανομή πιθανότητας του χρόνου απόκρισης (response time) για την **εργασία ADC** για την περίπτωση που η εργασία ADC έχει χρόνο επεξεργασίας **20 ms**
και
90ms
.

(γ) Να βρεθεί προσεγγιστικά η κατανομή πιθανότητας του χρόνου απόκρισης (response time) για την **εργασία σποραδικώ ν γεγονότων** για την περίπτωση που η εργασία ADC έχει χρόνο επεξεργασίας **20ms**, **90ms**, όταν όμως η σποραδική εργασία τεθεί σε
υψηλότερη προτεραιότητα
σε σχέση με την εργασία ADC.

Τέλος να σχολιαστούν οι μετρήσεις.

ΔΟΜΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

1. Φόρτωση απαραίτητων βιβλιοθηκών (Οι κυριότερες είναι:)
1. usart.h = Βιβλιοθήκη για την χρήση του περιφερειακού σειριακής επικοινωνίας USART.
2. Tc.h = Βιβλιοθήκη για την χρήση του περιφερειακού χρονιστή TC.
3. Intc.h = Βιβλιοθήκη για την χρήση διακοπών του AVR.
4. Gpio.h = Βιβλιοθήκη για την χρήση του περιφερειακού I/O.
5. Adc.h = Βιβλιοθήκη για την χρήση του περιφερειακού ADC.
6. Pm.h = Βιβλιοθήκη για την χρήση του περιφερειακού Power Manager (Clocks).
7. Cycle_Counter.h = Βιβλιοθήκη για την χρήση του περιφερειακού Cycle_Counter (Μετρητή Κύκλων Ρολογιού).
8. FreeRTOS.h, task.h, queue.h, semphr.h = Βιβλιοθήκες για την χρήση του FreeRTOS.

9. Δήλωση σταθερών
10. Δήλωση μεταβλητών
11. Ορισμός της δομής που μεταφέρει της μετρήσεις από τις εργασίες του ADC και του PB0 στην εργασία USART
12. Ορισμός Σημαφόρων που χρησιμοποιούνται από τις εργασίες επεξεργασίας του ADC και PB0
13. Ορισμός Ουράς για την μεταφορά μετρήσεων μεταξύ των διεργασιών
14. Ορισμός Συνάρτησης αρχικοποίησης του TC για συχνότητα 10Hz
15. Ορισμός Συνάρτησης αρχικοποίησης του ADC
16. Ορισμός Συνάρτησης αρχικοποίησης του PB0
17. Ορισμός Συνάρτησης αρχικοποίησης του USART0
18. Ορισμός Συνάρτησης Εργασίας χειρισμού διακοπής ADC που θα τρέχει από το λειτουργικό (ADC Task)
19. Ορισμός Συνάρτησης Εργασίας χειρισμού διακοπής PB0 που θα τρέχει από το λειτουργικό (PB0 Task)
20. Ρουτίνα Χειρισμού διακοπής ISR_ADC
21. Ρουτίνα Χειρισμού διακοπής ISR_PB0
22. Ορισμός Συνάρτησης Εργασίας που θα εκτελείτε από την κενή διεργασία (Idle Task)

23. Ορισμός της διεργασίας USART που θα τρέχει από το λειτουργικό (USART Task)
24. Ορισμός της διεργασίας Init που θα τρέχει από το λειτουργικό (Init Task), (Μόλις τρέξει για την ενεργοποίηση του ADC σβήνει των εαυτό της)

25. Κύρια Συνάρτηση
1. Απενεργοποίηση όλων των διακοπών
2. Αρχικοποίηση διακοπών
3. Κύριο Ρολόι από τον ταλαντωτή 0
4. Αρχικοποίηση του περιφερειακού TC για συχνότητα 10Hz
5. Αρχικοποίηση του περιφερειακού ADC
6. Αρχικοποίηση του περιφερειακού USART
7. Αρχικοποίηση του περιφερειακού PB0
8. Ενεργοποίηση όλων των διακοπών

9. Συσχέτιση ρουτίνας χειρισμού με διακοπή από PB0 Συσχέτιση ρουτίνας χειρισμού με διακοπή από ADC
10. Δημιουργία της κοινής ουράς μεγέθους 5 και τύπου "δομής xData"
11. Δημιουργία σημαφόρου που χρησιμοποιήτε από την εργασία ADC
12. Δημιουργία σημαφόρου που χρησιμοποιήτε από την εργασία PB0
13. Δημιουργία εργασίας PB0 με προτεραιότητα 2 (Η παράμετρος είναι η δομή που στέλνει η εργασία)
14. Δημιουργία εργασίας ADC με προτεραιότητα 3 (Η παράμετρος είναι η δομή που στέλνει η εργασία)
15. Δημιουργία εργασίας USART με προτεραιότητα 1 (Χαμηλή προτεραιότητα, μεγαλώνουμε το μέγεθος της στοίβας για να έχουμε χώρο που απαιτεί η συνάρτηση sprintf)

16. Δημιουργία εργασίας Init με προτεραιότητα 4 (Υψηλή προτεραιότητα)

q. Έναρξη του Χρονοδρομολογητή

1. Ατέρμονος βρόχος

ΣΧΟΛΙΑ-ΠΑΡΑΤΗΡΗΣΕΙΣ ΣΧΕΤΙΚΑ ΜΕ ΤΗΝ ΑΝΑΠΤΥΞΗ ΚΑΙ ΛΕΙΤΟΥΡΓΙΑ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ

Με την επιλογή μηδενικής βελτιστοποίησης του μεταγλωττιστή (-O0) ο scheduler λειτουργούσε για λίγο και μετά κόλλαγε, το πρόβλημα αυτό λύθηκε επιλέγοντας βελτιστοποίηση (-O1) και χρησιμοποιώντας μεταβλητές τύπου volatile και για να μην ακυρωθούν οι βρόχοι καθυστέρησης ενσωματώσαμε σε αυτούς την λειτουργία tonggle του led6.

Το LED2 ανάβει όποτε έχουμε διακοπή από τον ADC, κάθε 100ms, Το LED1 ανάβει όποτε πατάμε το PB0 προκαλούμε διακοπή, το LED4 ανάβει όταν τρέχει η εργασία του USART. Το LED3 ανάβει όποτε τρέχει η εργασία Idle, δηλαδή δεν πρέπει να ανάβει ποτέ.

Για το ζήτημα α) προσοχή στην περίπτωση 80ms και 90ms ((ADC=80) + (PB0=20) =100ms οπότε και έχουμε νέα διακοπή από τον ADC.

Για το ερώτημα γ) και για να αλλάξουμε τις προτεραιότητες, κάνουμε τις ακόλουθες αλλαγές στον κώδικα:

xTaskCreate(vHandlerTask_PB0, (**signed char** *) "Handler_PB0", 200, &(xStructsToSend2),
3
, NULL);
// Δημιουργία εργασίας PB0 με προτεραιότητα 3 (Η παράμετρος είναι η δομή που στέλνει η εργασία)

xTaskCreate(vHandlerTask_ADC, (**signed char** *) "Handler_ADC", 200,
&(xStructsToSend1),
2
, NULL);
// Δημιουργία εργασίας ADC με προτεραιότητα 2 (Η παράμετρος είναι η δομή που στέλνει η εργασία)

xTaskCreate(vHandlerTask_USART, (**signed char** *) "Handler_USART", 3000, NULL,
1, NULL); //
Δημιουργία εργασίας USART με προτεραιότητα 1 (Χαμηλή προτεραιότητα, μεγαλώνουμε το μέγεθος της στοίβας για να έχουμε χώρο που απαιτεί η συνάρτηση sprintf)

xTaskCreate(vTask_Init, (**signed char** *) "Handler_Init", 100, NULL, 4, NULL);//
Δημιουργία εργασίας Init με προτεραιότητα 4 (Υψηλή προτεραιότητα)

Ο ΚΩΔΙΚΑΣ

/***

```
 Name          : main.c

 Author        : JM

 Copyright     : Γιάννης Μαλτέζος

 Description   : Μελέτη Λειτουργικού Συστήματος FreeRtos σε Μικροελεγκτή AVR32

 ********************************************************************************/

// Include Files

#include <string.h>

#include "usart.h"

#include "preprocessor.h"

#include "tc.h"

#include "compiler.h"

#include "board.h"
```

```
#include "intc.h"

#include "gpio.h"

#include "adc.h"

#include "pm.h"

#include "cycle_counter.h"

#include <avr32/io.h>

//#include <stdio.h> "Αυτήν την βιβλιοθήκη την αγνοούμε λόγο του μεγάλου χώρου που

                //στοίβας που καταλαμβάνει η συνάρτηση sprintf"

#include "FreeRTOS.h"

#include "queue.h"

#include "task.h"
```

```
#include "semphr.h"

// Σταθερές

#define ADC_CHANNEL  1 //Ορισμός χρήσης του καναλιού 1 του ADC

#define TC_CHANNEL   0 //Ορισμός χρήσης του καναλιού 0 του TC

//Ορισμός απαραίτητων σταθερών για την χρήση του USART

#define EXAMPLE_USART              (&AVR32_USART0)

#define EXAMPLE_USART_RX_PIN       AVR32_USART0_RXD_0_PIN

#define EXAMPLE_USART_RX_FUNCTION  AVR32_USART0_RXD_0_FUNCTION

#define EXAMPLE_USART_TX_PIN       AVR32_USART0_TXD_0_PIN

#define EXAMPLE_USART_TX_FUNCTION  AVR32_USART0_TXD_0_FUNCTION

// Μεταβλητές, ορίζονται σαν τύπος Volatile, γιατί χρησιμοποιούμε βελτιστοποίηση

// επιπέδου -O1 του μεταγλωττιστή επιλογή απαραίτητη για να τρέξει σωστά ο scheduler
```

```
// Η τιμή του ποτενσιόμετρου που διαβάζει ο ADC

 volatile unsigned int  pot;




 // *********************************

 //timeADC_START= Ο χρόνος έναρξης ρουτίνας εξυπηρέτησης διακοπής του ADC

 //timeADC_BLOCK= Ο χρόνος που μπλοκάρεται η εργασία επεξεργασίας του ADC

 //timePB0_START= Ο χρόνος έναρξης ρουτίνας εξυπηρέτησης διακοπής του PB0

 //timePB0_BLOCK= Ο χρόνος που μπλοκάρεται η εργασία επεξεργασίας του PB0

 //timePB0_RESPONSE= Ο χρόνος απόκρισης της εργασίας σποραδικών γεγονότων

 //timeADC_START_DELAY= Ο χρόνος έναρξης της καθυστέρησης της εργασίας επεξεργασίας του ADC

 //timeADC_STOP_DELAY= Ο χρόνος λήξης της καθυστέρησης της εργασίας επεξεργασίας του ADC
```

//timeADC_DELAY= Ο χρόνος της καθυστέρησης της εργασίας επεξεργασίας του ADC

//timePB0_START_DELAY= Ο χρόνος έναρξης της καθυστέρησης της εργασίας επεξεργασίας του PB0

//timePB0_STOP_DELAY= Ο χρόνος λήξης της καθυστέρησης της εργασίας επεξεργασίας του PB0

//timePB0_DELAY= Ο χρόνος της καθυστέρησης της εργασίας επεξεργασίας του PB0

//timeADC_RESPONSE= Ο χρόνος απόκρισης της εργασίας επεξεργασίας του ADC

//timeADC_DELAY_MS= Ο χρόνος της καθυστέρησης της εργασίας επεξεργασίας του ADC σε ms

//timePB0_DELAY_MS= Ο χρόνος της καθυστέρησης της εργασίας επεξεργασίας του PB0 σε ms

//*********************************

volatile float timeADC_START,timeADC_BLOCK,timePB0_START,timePB0_BLOCK,timePB0_RESPONSE;

volatile float timeADC_START_DELAY,timeADC_STOP_DELAY,timeADC_DELAY,timePB0_START_DELAY,timePB0_STOP_DELAY,timePB0_DELAY;

```
volatile float timeADC_RESPONSE,timeADC_DELAY_MS,timePB0_DELAY_MS;

//*********************************



//Ορισμός της δομής που μεταφέρει της μετρήσεις από τις εργασίες του ADC και του PB0 στην εργασία USART

volatile typedef struct

{

        unsigned int ADC_VALUE; //Η τιμή του ADC

        float ADC_DELAY; //Η καθυστέρηση της εργασίας επεξεργασίας του ADC

        float ADC_RESPONSE; // Η απόκριση της εργασίας επεξεργασίας του ADC

        unsigned int PB0_INT; // Δήλωση εργασίας προέλευσης 0=ADC , 1=PB0

        float PB0_DELAY; //Η καθυστέρηση της εργασίας επεξεργασίας του PB0
```

```
    float PB0_RESPONSE;//Η απόκριση της εργασίας επεξεργασίας του PB0

} xData;

xData xStructsToSend1; //Ορισμός της δομής που μεταφέρει της μετρήσεις από την εργασία του ADC

xData xStructsToSend2; //Ορισμός της δομής που μεταφέρει της μετρήσεις από την εργασία του PB0

//Ορισμός Σημαφόρων

xSemaphoreHandle xBinarySemaphoreADC; //Ορισμός σημαφόρου που χρησιμοποιήτε από την εργασία επεξεργασίας του ADC

xSemaphoreHandle xBinarySemaphorePB;  //Ορισμός σημαφόρου που χρησιμοποιήτε από την εργασία επεξεργασίας του PB0

//Ορισμός Ουράς

xQueueHandle xQueue;

//TC Initialization function

void init_tc(int rate)
```

```
{

  volatile avr32_tc_t *tc = &AVR32_TC;



  // Options for waveform generation.

  static const tc_waveform_opt_t WAVEFORM_OPT =

  {

    .channel  = TC_CHANNEL,                   // Channel selection.



    .bswtrg   = TC_EVT_EFFECT_NOOP,           // Software trigger effect on TIOB.

    .beevt    = TC_EVT_EFFECT_NOOP,           // External event effect on TIOB.

    .bcpc     = TC_EVT_EFFECT_NOOP,           // RC compare effect on TIOB.
```

```
    .bcpb     = TC_EVT_EFFECT_NOOP,             // RB compare effect on TIOB.

    .aswtrg   = TC_EVT_EFFECT_NOOP,             // Software trigger effect on TIOA.

    .aeevt    = TC_EVT_EFFECT_NOOP,             // External event effect on TIOA.

    .acpc     = TC_EVT_EFFECT_TOGGLE,           // RC compare effect on TIOA: toggle.

    .acpa     = TC_EVT_EFFECT_NOOP,             // RA compare effect on TIOA: none.

    .wavsel   = TC_WAVEFORM_SEL_UPDOWN_MODE_RC_TRIGGER,// Waveform
selection: Up mode without automatic trigger on RC compare.

    .enetrg   = FALSE,                          // External event trigger enable.

    .eevt     = 0,                              // External event selection.

    .eevtedg  = TC_SEL_NO_EDGE,                 // External event edge selection.

    .cpcdis   = FALSE,                          // Counter disable when RC compare.
```

```
    .cpcstop  = FALSE,                  // Counter clock stopped with RC compare.

    .burst    = FALSE,                  // Burst signal selection.

    .clki     = FALSE,                  // Clock inversion.

    .tcclks   = TC_CLOCK_SOURCE_TC5     // Internal source clock 5 - connected to
PBA/128

  };

  // Initialize the timer/counter.

  tc_init_waveform(tc, &WAVEFORM_OPT);  // Initialize the timer/counter waveform.

  // Set the compare triggers.
```

```
  // Remember TC counter is 16-bits, so counting second is not possible.

  // We configure it to count ms.

  // Set the compare triggers.

 tc_write_ra(tc, TC_CHANNEL, 0);

 tc_write_rc(tc, TC_CHANNEL, 46875/(rate/2));     // Set RC value. (RATE=hz)



  // Start the timer/counter.

 //tc_start(tc, TC_CHANNEL);

}

//ADC Initialization function

void init_ADC()

{
```

```
// Ρύθμιση ADC


    //AVR32_ADC.MR.trgen = 1;

    //AVR32_ADC.MR.trgsel = TC_CHANNEL;


// Σύνδεση καναλιού 1 του ADC με το GPIO pin 22 (PA22-ποτενσιόμετρο)

 gpio_enable_module_pin(AVR32_ADC_AD_1_PIN, AVR32_ADC_AD_1_FUNCTION);


 volatile avr32_adc_t *adc = &AVR32_ADC;

// set Sample/Hold time to max so that the ADC capacitor should be loaded entirely

adc->mr |= 0xF << AVR32_ADC_SHTIM_OFFSET;
```

```
// set Startup to max so that the ADC capacitor should be loaded entirely

adc->mr |= 0x1F << AVR32_ADC_STARTUP_OFFSET;

// Χρονισμός ADC = MCK/64

adc->mr |= 0x3F << AVR32_ADC_PRESCAL_OFFSET;

// Δημιουργία διακοπής με το πρώτο κανάλι

adc->ier |= 1 << AVR32_ADC_IER_EOC1_OFFSET;

// Ενεργοποίηση καναλιών 0,1,2

adc->cher = 2; //(1+2+4)

}



//PB0 Initialization function

void init_PB0()
```

```
{


        gpio_enable_pin_glitch_filter(GPIO_PUSH_BUTTON_0);                    // E
νεργοποίηση
φίλτρου
παρασίτων
για
το
PB0


        gpio_enable_pin_interrupt (GPIO_PUSH_BUTTON_0,GPIO_FALLING_EDGE );   // Ev
εργοποίηση
παραγωγής
διακοπών
για
το
PB0









}




//USART Initialization function


void init_USART()
```

```
{

        static const gpio_map_t USART_GPIO_MAP =

                {

                  {EXAMPLE_USART_RX_PIN, EXAMPLE_USART_RX_FUNCTION},

                  {EXAMPLE_USART_TX_PIN, EXAMPLE_USART_TX_FUNCTION}

                };

               // USART options.

                static const usart_options_t USART_OPTIONS =

                {

                  .baudrate     = 57600,

                  .charlength   = 8,

                  .paritytype   = USART_NO_PARITY,
```

```
    .stopbits     = USART_1_STOPBIT,

    .channelmode  = USART_NORMAL_CHMODE

  };

  // Assign GPIO to USART.

   gpio_enable_module(USART_GPIO_MAP,

                      sizeof(USART_GPIO_MAP) / sizeof(USART_GPIO_MAP[0]));

   // Initialize USART in RS232 mode.

     usart_init_rs232(EXAMPLE_USART, &USART_OPTIONS, FOSC0);

}
```

// Ορισμός συνάρτησης εργασίας χειρισμού διακοπής ADC

```
 void vHandlerTask_ADC( void *pvParameters ) {



        //Ορισμός της δομής που μεταφέρει της μετρήσεις από την εργασία του ADC

         extern xData xStructsToSend1;



         float delay,k;// Ορισμός των μεταβλητών καθυστέρησης της ADC



       delay=6.09;

for( ;; )

       {

       // Περίμενε επ' αόριστον μέχρι να παραλάβεις των σημαφόρο

       xSemaphoreTake( xBinarySemaphoreADC, portMAX_DELAY  );
```

```
timeADC_START_DELAY=Get_sys_count() ;//Διάβασμα χρόνου έναρξης
καθυστέρησης

gpio_tgl_gpio_pin(LED1_GPIO);        // Αναβόσημα LED2 ένδειξης διακοπής από ADC

for (k = 0; k < pot*delay; k += 1)

{

                gpio_tgl_gpio_pin(LED5_GPIO);        // Αναβόσημα LED6 (σαν ε
ντολή
καθυστέρησης
)

        }

timeADC_STOP_DELAY=Get_sys_count() ;  //Διάβασμα χρόνου λήξης καθυστέρησης

timeADC_DELAY=timeADC_STOP_DELAY-timeADC_START_DELAY;//Υπολογισμός
χρόνου
καθυστέρησης
```

```
timeADC_BLOCK=Get_sys_count() ;//Διάβασμα χρόνου μπλοκαρίσματος της εργασίας

//Μετατροπή χρόνων σε msec

timeADC_RESPONSE=((timeADC_BLOCK-timeADC_START)/(12000));

timeADC_DELAY_MS=( (timeADC_DELAY)/(12000));



//Απόδοση στην δομή του ADC των τιμών του ποτενσιόμετρου,

//της καθυστέρησης και της απόκρισης της εργασίας ADC.

//Στο PB0_INT δίνουμε τιμή 0 για να δηλώσουμε στην εργασία USART,

//ότι τα δεδομένα προέρχονται από την εργασία ADC.

xStructsToSend1.ADC_VALUE=pot;
```

```
xStructsToSend1.ADC_DELAY=timeADC_DELAY_MS;

xStructsToSend1.ADC_RESPONSE=timeADC_RESPONSE;

xStructsToSend1.PB0_INT=0;

xStructsToSend1.PB0_DELAY=0;

xStructsToSend1.PB0_RESPONSE=0;




//Αποστολή της παραπάνω δομής στην κοινή ουρά.

//Η παράμετρος pvParameters της εργασίας ADC έχει δηλωθεί σαν &(xStructsToSend1),

//κατά την δημιουργία της εργασίας (task) του ADC.

//Τα δεδομένα προοθούνται στην ουρά συνεχώς χωρίς έλεγχο για το αν αυτή είναι γεμάτη (0).
```

```
        xQueueSend(xQueue,pvParameters,0 );




        }

}

// Ορισμός συνάρτησης εργασίας χειρισμού διακοπής PB0

 void vHandlerTask_PB0( void *pvParameters  ) {



          //Ορισμός της δομής που μεταφέρει της μετρήσεις από την εργασία του PB0

                extern xData xStructsToSend2;



          float delay2,t;// Ορισμός των μεταβλητών καθυστέρησης της ADC
```

```
for( ;; )

        {

        // Περίμενε επ' αόριστον μέχρι να παραλάβεις των σημαφόρο

        xSemaphoreTake( xBinarySemaphorePB, portMAX_DELAY  );




        delay2=1.26128;

        gpio_clr_gpio_pin(LED0_GPIO);//Αναμα του LED1 ένδειξης διακοπής από PB0




        timePB0_START_DELAY=Get_sys_count() ;//Διάβασμα χρόνου έναρξης καθυστέρησης

        for (t=0; t < delay2*1755; t +=1)//Ορισμός καθυστέρησης 20ms

                {
```

```
gpio_clr_gpio_pin(LED5_GPIO);// Αναβόσημα LED6 (σαν εντολή καθυστέρησης)

}

timePB0_STOP_DELAY=Get_sys_count() ;//Διάβασμα χρόνου λήξης καθυστέρησης

timePB0_DELAY=timePB0_STOP_DELAY-timePB0_START_DELAY;//Υπολογισμός χρόνου καθυστέρησης

timePB0_BLOCK=Get_sys_count() ;//Διάβασμα χρόνου μπλοκαρίσματος της εργασίας

//Μετατροπή χρόνων σε msec

timePB0_RESPONSE=((timePB0_BLOCK-timePB0_START)/(12000));

timePB0_DELAY_MS=( (timePB0_DELAY)/(12000));

//Απόδοση στην δομή του PB0 των τιμών :
```

```
		//της καθυστέρησης και της απόκρισης της εργασίας PB0.

		//Στο PB0_INT δίνουμε τιμή 1 για να δηλώσουμε στην εργασία USART,

		//ότι τα δεδομένα προέρχονται από την εργασία PB0.

xStructsToSend2.ADC_VALUE=0;

		xStructsToSend2.ADC_DELAY=0;

		xStructsToSend2.ADC_RESPONSE=0;

		xStructsToSend2.PB0_INT=1;

		xStructsToSend2.PB0_DELAY=timePB0_DELAY_MS;

		xStructsToSend2.PB0_RESPONSE=timePB0_RESPONSE;



		//Αποστολή της παραπάνω δομής στην κοινή ουρά.
```

```
                //Η παράμετρος pvParameters της εργασίας PB0 έχει δηλωθεί σαν
&(xStructsToSend2),

                //κατά την δημιουργία της εργασίας (task) του PB0.

                //Τα δεδομένα προοθούνται στην ουρά συνεχώς χωρίς έλεγχο για το αν αυτή
είναι γεμάτη (0).

                xQueueSend(xQueue,pvParameters,0 );          // send data to queue

                gpio_set_gpio_pin(LED0_GPIO);//Σβήσιμο του LED1 ένδειξης διακοπής από
PB0

        }

 }

 // Ρουτίνα χειρισμού διακοπής (ISR_ADC)

 // Λειτουργία εκτός FreeRTOS
```

```
// Η ρουτίνα είναι "γυμνή" δηλάδή δεν γίνονται οι συνηθισμένες

// αποθηκεύσεις στη στοίβα από τον μεταγλωττιστή.

__attribute__((__naked__))

 void  vInterruptHandlerADC( void )

{

        timeADC_START=Get_sys_count();//Ανάγνωση χρόνου έναρξης ρουτίνας εξυπηρέτησης διακοπής του ADC

        portSAVE_CONTEXT_OS_INT();

        static signed portBASE_TYPE xHigherPriorityTaskWoken;

        xHigherPriorityTaskWoken = pdFALSE;
```

```
volatile avr32_adc_t *adc = &AVR32_ADC;

        // Ανάγνωση ADC

         pot=adc_get_value(adc,1);

//Εισαγωγή σε προστατευμένη κατάσταση

portENTER_CRITICAL();

//Ενεργοποίηση σημαφόρου εργασίας εξυπηρέτησης ADC

xSemaphoreGiveFromISR( xBinarySemaphoreADC, xHigherPriorityTaskWoken );

portEXIT_CRITICAL();//Εξοδος από προστατευμένη κατάσταση
```

```
        portRESTORE_CONTEXT_OS_INT();

}



// Ρουτίνα χειρισμού διακοπής (ISR_PB0)

// Λειτουργία εκτός FreeRTOS

// Η ρουτίνα είναι "γυμνή" δηλάδή δεν γίνονται οι συνηθισμένες

// αποθηκεύσεις στη στοίβα από τον μεταγλωττιστή.

__attribute__((__naked__))

 void  vInterruptHandler_PB0( void )

{
```

timePB0_START=Get_sys_count() ;//Ανάγνωση χρόνου έναρξης ρουτίνας εξυπηρέτησης διακοπής του PB0

portSAVE_CONTEXT_OS_INT();

static signed portBASE_TYPE xHigherPriorityTaskWoken;

xHigherPriorityTaskWoken = pdFALSE;

// 'Give' the semaphore to unblock the task. //

portENTER_CRITICAL();//Εισαγωγή σε προστατευμένη κατάσταση

xSemaphoreGiveFromISR(xBinarySemaphorePB, xHigherPriorityTaskWoken); //Ενεργοποίηση σημαφόρου εργασίας εξυπηρέτησης PB0

portEXIT_CRITICAL();//Εξοδος από προστατευμένη κατάσταση

```
        gpio_clear_pin_interrupt_flag(GPIO_PUSH_BUTTON_0);//Απενεργοποίηση σημαίας διακοπής

        portRESTORE_CONTEXT_OS_INT();

}

// Ορισμός συνάρτησης που θα εκτελείται από την "κενή" εργασία (idle)

void vApplicationIdleHook(void) {

for (;;)                    // Συνεχώς

        {

                // Αναβόσβηνε το LED3
```

```
                gpio_tgl_gpio_pin(LED2_GPIO);

            }

}
```

```
//Ορισμός της εργασίας USART

void vHandlerTask_USART( void *pvParameters ) {


    //Ορισμός χαρακτήρων απεικόνισης

    char s_pot[40];

    char s_timePB0[40];
```

```
//Ορισμός δομής λήψης δεδομένων από την κοινή ουρά

xData xReceivedStructure;

for (;;)

{

gpio_tgl_gpio_pin(LED3_GPIO);//Ενδειξη από το LED4 εκτέλεσης της εργασίας USART

xQueueReceive(xQueue,&xReceivedStructure,0); //Λήψη δεδομένων από την κοινή ουρά

if (xReceivedStructure.PB0_INT==0)//Ελεγχος από που προέρχονται τα δεδομένα της ουράς (από ADC ή από PB0)
```

```
            {


          //Αποστολή των δεδομένων της ADC στην USART


          sprintf(s_pot,"The value of ADC is: %d the Delay is: %4.2f and the Response is: %4.2f
.n"                          ,xReceivedStructure.ADC_VALUE,xReceivedStructure.ADC_DELAY,
xReceivedStructure.
ADC_RESPONSE
);


          usart_write_line(EXAMPLE_USART, s_pot);


                              }


           else


           {


          //Αποστολή των δεδομένων της PB0 στην USART


          sprintf(s_timePB0,"The Delay of Sporadic is: %4.2f and the Response is: %4.2f  -1 .n"
,xReceivedStructure.
PB0_DELAY
,xReceivedStructure.
PB0_RESPONSE
);


          usart_write_line(EXAMPLE_USART, s_timePB0);
```

```
                }

                        }

}
```

```
//Ορισμός της εργασίας Init (Μόλις τρέξει για την ενεργοποίηση του ADC σβήνει των εαυτό της)

void vTask_Init( void *pvParameters )

{

        volatile avr32_tc_t *tc = &AVR32_TC;

        tc_start(tc, TC_CHANNEL);//Εναρξη του TimerCounter0
```

```
        //Ορισμός έναρξης μετατροπής ADC με σκανδαλισμό από TC0

        AVR32_ADC.MR.trgen = 1;

        AVR32_ADC.MR.trgsel = TC_CHANNEL;



         //Αναβόσβημα  του LED7 για οπτικό έλεγχο λειτουργίας της εργασίας Init

                gpio_clr_gpio_pin(LED6_GPIO);

                gpio_set_gpio_pin(LED6_GPIO);



                vTaskDelete(NULL);//Σβήσιμο της εργασίας Init

}
```

```
int main(void)

{

        Disable_global_interrupt();                 // Απενεργοποίηση όλων των διακοπών

        INTC_init_interrupts();             // Αρχικοποίηση διακοπών

         pm_switch_to_osc0(&AVR32_PM, FOSC0, OSC0_STARTUP);// Κύριο Ρολόι από τον ταλαντωτή  0

                                                init_tc(10); //Αρχικοποίηση του περιφερειακού TC για συχνότητα 10Hz

                                                init_ADC();  //Αρχικοποίηση του περιφερειακού ADC

                                                init_USART(); //Αρχικοποίηση του περιφερειακού USART
```

init_PB0(); ////Αρχικοποίηση του περιφερειακού PB0

Enable_global_interrupt(); //Ενεργοποίηση όλων των διακοπών

INTC_register_interrupt((__int_handler)&vInterruptHandler_PB0,AVR32_GPIO_IRQ_11, AVR32_INTC_INT0); // Συσχέτιση ρουτίνας χειρισμού με διακοπή από PB0

INTC_register_interrupt((__int_handler)&vInterruptHandlerADC,AVR32_ADC_IRQ, AVR32_INTC_INT0); // Συσχέτιση ρουτίνας χειρισμού με διακοπή από ADC

xQueue=xQueueCreate(5,**sizeof**(xData));//Δημιουργία της κοινής ουράς μεγέθους 5 και τύπου "δομής xData"

vSemaphoreCreateBinary(xBinarySemaphoreADC);// Δημιουργία σημαφόρου που χρησιμοποιήτε από την εργασία ADC

vSemaphoreCreateBinary(xBinarySemaphorePB); // Δημιουργία σημαφόρου που χρησιμοποιήτε από την εργασία PB0

xTaskCreate(vHandlerTask_PB0, (**signed char** *) "Handler_PB0", 200, &(xStructsToSend2), 2, NULL);
// Δημιουργία εργασίας PB0 με προτεραιότητα 2 (Η παράμετρος είναι η δομή που στέλνει η εργασία)

xTaskCreate(vHandlerTask_ADC, (**signed char** *) "Handler_ADC", 200, &(xStructsToSend1), 3, NULL);
// Δημιουργία εργασίας ADC με προτεραιότητα 3 (Η παράμετρος είναι η δομή που στέλνει η εργασία)

xTaskCreate(vHandlerTask_USART, (**signed char** *) "Handler_USART", 3000, NULL, 1, NULL); // Δημιουργία εργασίας USART με προτεραιότητα 1 (Χαμηλή προτεραιότητα, μεγαλώνουμε το μέγεθος της στοίβας για να έχουμε χώρο που απαιτεί η συνάρτηση sprintf)

xTaskCreate(vTask_Init, (**signed char** *) "Handler_Init", 100, NULL, 4, NULL);// Δημιουργία εργασίας Init με προτεραιότητα 4 (Υψηλή προτεραιότητα)

vTaskStartScheduler();//Έναρξη του Χρονοδρομολογητή

for(;;) ; // Ατέρμονος βρόχος

}

ΜΕΤΡΗΣΕΙΣ :

Για το ερώτημα α) και για αντίστοιχα χρόνους επεξεργασίας 20, 40, 60,70, 80 ms έχουμε την ακόλουθη κατανομή των μετρήσεων:

Για 90ms η σποραδική εργασία δεν προλαβαίνει να εξυπηρετηθεί γιατί έχουμε 90ms από ADC και 20ms από PB0 =110ms >100ms οπότε και συμβαίνει νέα διακοπή από τον ADC,

20 ms

40 ms

60 ms

70 ms

80ms

ms

Rates

ms

Rates

ms

Rates

ms

Rates

ms

Rates

20-22

69,00%

20-22

43,00%

20-22

19,50%

20-22

12,00%

100-102

24,50%

22-24

1,00%

22-24

2,00%

22-24

1,00%

22-24

3,00%

102-104

2,00%

24-26

5,00%

24-26

3,50%

24-26

0,00%

24-26

2,50%

104-106

1,50%

26-28

1,50%

26-28

2,50%

26-28

1,50%

26-28

2,00%

106-108

2,00%

28-30

1,50%

28-30

2,50%

28-30

1,00%

28-30

1,50%

108-110

1,00%

30-32

2,00%

30-32

2,50%

30-32

1,00%

30-32

2,50%

110-112

1,00%

32-34

1,50%

32-34

0,00%

32-34

2,00%

32-34

1,50%

112-114

4,00%

34-36

2,00%

34-36

2,00%

34-36

4,00%

34-36

1,50%

114-116

1,50%

36-38

2,50%

36-38

2,50%

36-38

0,50%

36-38

2,00%

116-118

2,00%

38-40

1,50%

38-40

3,00%

38-40

1,00%

38-40

4,50%

118-120

2,50%

40-42

17,00%

40-42

2,00%

40-42

1,50%

40-42

2,00%

120-122

1,00%

42-44

2,00%

42-44

2,00%

42-44

2,50%

122-124

1,00%

44-46

1,50%

44-46

1,50%

44-46

1,50%

124-126

2,00%

46-48

2,50%

46-48

3,50%

46-48

1,50%

126-128

1,50%

48-50

1,00%

48-50

3,00%

48-50

2,00%

128-130

1,00%

50-52

1,50%

50-52

4,50%

50-52

2,00%

130-132

2,50%

52-54

1,50%

52-54

3,00%

52-54

1,00%

132-134

2,50%

54-56

2,00%

54-56

0,50%

54-56

3,00%

134-136

1,00%

56-58

1,50%

56-58

2,00%

56-58

1,00%

136-138

2,50%

58-60

1,50%

58-60

2,50%

58-60

3,00%

138-140

2,50%

60-62

18,50%

60-62

0,50%

60-62

0,50%

140-142

2,50%

62-64

1,50%

62-64

1,50%

142-144

3,00%

64-66

2,50%

64-66

0,50%

144-146

1,50%

66-68

2,50%

66-68

1,00%

146-148

2,00%

68-70

1,00%

68-70

0,50%

148-150

2,00%

70-72

2,50%

70-72

2,00%

150-152

3,00%

72-74

3,00%

72-74

4,00%

152-154

0,50%

74-76

2,50%

74-76

2,50%

154-156

4,00%

76-78

1,50%

76-78

2,50%

156-158

0,50%

78-80

2,50%

78-80

2,00%

158-160

1,00%

80-82

24,00%

80-82

1,50%

160-162

0,50%

82-84

1,50%

162-164

1,50%

84-86

1,00%

164-166

1,50%

86-88

1,50%

166-168

4,00%

88-90

1,00%

168-170

2,50%

90-92

24,00%

170-172

3,00%

172-174

4,00%

174-176

1,50%

176-178

1,50%

178-180

0,00%

180-182

0,50%

Για το β) ερώτημα οι χρόνοι παραμένουν σταθεροί και είναι αντίστοιχα 20ms και 90 ms περίπου, γι

Για το γ) ερώτημα ο χρόνος απόκρισης παραμένει σταθερός 20ms περίπου, για την ακρίβεια είναι 0.

Όλα τα παραπάνω φαίνονται στα σχετικά αρχεία μετρήσεων.

Σχόλια:

Από τα διαγράμματα παρατηρούμε ότι τα μεγαλύτερα ποσοστά είναι ή στα 20ms χρόνος καθυστέρ

Δηλαδή ή η εργασία PB0 σποραδική εξυπηρετείτε άμεσα (1ή περίπτωση) ή περιμένει να τελειώσει α

Λογικό είναι όπως φαίνεται και στα διαγράμματα για τιμή καθυστέρησης 20 ms της εργασίας ADC ε

Συμπερασματικά όσο μεγαλώ νει ο χρόνος επεξεργασίας της εργασίας ADC, μικραίνει ο αν

*(Σε αυτή την εργασία με τον όρο interrupt latency εννοούμε τον χρόνο από την στιγμή που συμβαίνε

1ή περίπτωση

2ή περίπτωση

3ή περίπτωση

20ms

70%

18%

12%

40ms

43%

17%

40%

60ms

20%

24%

56%

70ms

12%

24%

64%

80ms

0%

24%

76%