ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ

2η ΑΣΚΗΣΗ

31/03/2022

ΠΑΠΑΔΟΠΟΥΛΟΥ ΑΙΚΑΤΕΡΙΝΗ 1067535

ΠΡΑΠΠΑΣ ΤΡΑΝΤΑΦΥΛΛΟΣ 1067504

Πείραμα

Ο κώδικας της άσκησης είναι ο ακόλουθος:

Ερώτημα Α

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdlib.h>

#define ped 10

int interrupt = 1;
int count = 0;

int main()
{

	// PIN right is output
	PORTD.DIR |= PIN0_bm;

	// PIN forward is output
	PORTD.DIR |= PIN1_bm;

	// PIN left is output
	PORTD.DIR |= PIN2_bm;

	// Initialize the ADC for Free-Running mode

	// 10-bit resolution
	ADC0.CTRLA |= ADC_RESSEL_10BIT_gc;

	// Free-Running mode enabled
	ADC0.CTRLA |= ADC_FREERUN_bm;

	// Enable ADC
	ADC0.CTRLA |= ADC_ENABLE_bm;

	// The bit
	ADC0.MUXPOS |= ADC_MUXPOS_AIN7_gc;

	// Enable Debug Mode
	ADC0.DBGCTRL |= ADC_DBGRUN_bm;

	// Window Comparator Mode

	// Set threshold
	ADC0.WINLT |= 10;

	// Enable Interrupts for WCM
	ADC0.INTCTRL |= ADC_WCMP_bm;

	// Interrupt when RESULT < WINLT
	ADC0.CTRLE |= ADC_WINCM0_bm;

	sei();

	// Start Conversion
	ADC0_COMMAND |= ADC_STCONV_bm;
```

```
	while(1)
	{
		// If count is equal to 4..
		if(count == 4)
		{
			// Exits the program
			exit(0);
		}

		_delay_ms(5);
		// LED0 right is off
		PORTD.OUT |= PIN0_bm;
		_delay_ms(5);
		// LED2 left is off
		PORTD.OUT |= PIN2_bm;
		_delay_ms(5);
		// LED1 forward is on
		PORTD.OUTCLR = PIN1_bm;
		_delay_ms(5);

		// If interrupt is equal to 0...
		if(interrupt == 0)
		{
			// Clear counter
			TCA0.SINGLE.CNT = 0;
			// Normal Mode
			TCA0.SINGLE.CTRLB = 0;
			// When reaches this value
			-> interrupt CLOCK FRENQUENCY/1024
			TCA0.SINGLE.CMP0 = ped;
			TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1024_gc;
			// Enable
			TCA0.SINGLE.CTRLA |=1;
			// Interrupt Enable (=0x10)
			TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm;
			// Begin accepting interrupt signals
			sei();

			// While the interrupt flag is 0...
			while (interrupt == 0)
			{
				// Wait...
			}
		}
	}
}

// ADC..
ISR(ADC0_WCOMP_vect)
{

	int intflags = ADC0.INTFLAGS;
	ADC0.INTFLAGS = intflags;


	// If RES < 10...
	if(ADC0.RES < 10)
	{
	_delay_ms(5);
	// LED0 right is off
	PORTD.OUT |= PIN0_bm;
	// LED forward is off
```

```
        PORTD.OUT |= PIN1_bm;
        _delay_ms(5);
        // LED left is on
        PORTD.OUTCLR = PIN2_bm;
        _delay_ms(5);

        // Set the interrupt flag to 0
        interrupt = 0;
        // Increments count
        count += 1;
        }
}

// Timer..
ISR(TCA0_CMP0_vect)
{
        // Disable
        TCA0.SINGLE.CTRLA = 0;
        // Clear flags
        int intflags = TCA0.SINGLE.INTFLAGS;
        TCA0.SINGLE.INTFLAGS = intflags;

        // Set the interrupt flag to 1
        interrupt = 1;
}
```

Σχετικά με το ερώτημα Α ο περισσότερος κώδικας έχει αναλυθεί μέσω σχολίων. Παρόλα αυτά θα θέλαμε να τονίσουμε τι συμβαίνει σε μερικά σημεία. Αρχικά έχουμε θέσει το threshold μας ως 10, δηλαδή για οποιεσδήποτε τιμές μεγαλύτερες του 10 θα βρισκόμαστε μέσα στην while(1). Αν στο RES βάλουμε τιμή μικρότερη του 10 τότε θα κάνει fire το interrupt και θα τρέξει την ISR που αφορά τον ADC. Στην ISR θα σβήσει το LED που αφορά την μπροστά πορεία και θα ανοίξει το LED που αφορά την αριστερή στροφή, παράλληλα το count θα αυξηθεί κατά μια μονάδα. Στην συνέχεια αφού έχουμε θέση τιμή μεγαλύτερη του 10 στον RES θα επιστρέψουμε στην while(1) και αφού το interrupt έχει πάρει την τιμή 0 από την ISR θα μπούμε μέσα στην if(interrupt == 0) και θα ξεκινήσει ο timer για τον χρόνο που του έχουμε ορίσει. Στην συνέχεια θα μπει στην ISR που αφορά τον timer και αφού την τρέξει το interrupt θα γίνει 1 οπότε θα βγούμε από τον timer και την if(interrupt == 0) και η συσκευή θα μπορεί να συνεχίσει ευθεία (ανάβει το αντίστοιχο LED), μέχρι να ξανασυναντήσει τοίχο και να ξαναστρίψει. Τέλος αφού τρέξει η ISR του ADC0 4 φορές το count λόγω του increment θα πάρει την τιμή 4 οπότε όταν μπει ξανά στην while(1) θα τρέξει το if(count == 4) και θα τερματίσει το πρόγραμμα αφού θα έχουμε φτάσει στην αρχική θέση της συσκευής.

Ερώτημα Α με Switch

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdlib.h>

#define ped 10

int interrupt = 1;
int isBackward = 0;
int isBackwardInterrupt = 1;
int count = 0;

int main()
{

	// PIN right is output
	PORTD.DIR |= PIN0_bm;

	// PIN forward is output
	PORTD.DIR |= PIN1_bm;

	// PIN left is output
	PORTD.DIR |= PIN2_bm;

	// Initialize the ADC for Free-Running mode

	// 10-bit resolution
	ADC0.CTRLA |= ADC_RESSEL_10BIT_gc;

	// Free-Running mode enabled
	ADC0.CTRLA |= ADC_FREERUN_bm;

	// Enable ADC
	ADC0.CTRLA |= ADC_ENABLE_bm;

	// The bit
	ADC0.MUXPOS |= ADC_MUXPOS_AIN7_gc;

	// Enable Debug Mode
	ADC0.DBGCTRL |= ADC_DBGRUN_bm;

	// Window Comparator Mode

	// Set threshold
	ADC0.WINLT |= 10;

	// Enable Interrupts for WCM
	ADC0.INTCTRL |= ADC_WCMP_bm;

	// Interrupt when RESULT < WINLT
	ADC0.CTRLE |= ADC_WINCM0_bm;

	// Pull-up enable and interrupt enabled with sense on both edges
	PORTF.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc;

	sei();

	// Start Conversion
```

```
	ADC0_COMMAND |= ADC_STCONV_bm;

	while(1)
	{
		// If count = 4 and isBackward = 0...
		// or count = 0 and isBackward = 1...
		if((count == 4 && isBackward == 0)
		|| (count == 0 && isBackward == 1))
		{
			// Exits the program
			exit(0);
		}

		_delay_ms(5);
		// LED0 right is off
		PORTD.OUT |= PIN0_bm;
		_delay_ms(5);
		// LED2 left is off
		PORTD.OUT |= PIN2_bm;
		_delay_ms(5);
		// LED1 forward is on
		PORTD.OUTCLR = PIN1_bm;
		_delay_ms(5);

		// If interrupt = 0 or isBackward = 1
		and isBackwardInterrupt = 0...
		if(interrupt == 0 || (isBackward == 1 && isBackwardInterrupt == 0))
		{
			// Clear counter
			TCA0.SINGLE.CNT = 0;
			// Normal Mode
			TCA0.SINGLE.CTRLB = 0;
			// When reaches this value
			-> interrupt CLOCK FRENQUENCY/1024
			TCA0.SINGLE.CMP0 = ped;
			TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1024_gc;
			// Enable
			TCA0.SINGLE.CTRLA |=1;
			// Interrupt Enable (=0x10)
			TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm;
			// Begin accepting interrupt signals
			sei();

			// While the interrupt flag is 0 or isBackward = 1
			and isBackwardInterrupt = 0...
			while (interrupt == 0 || (isBackward == 1 &&
			isBackwardInterrupt == 0))
			{
				// Wait...
			}
		}
	}
}

// ADC..
ISR(ADC0_WCOMP_vect)
{
	int intflags = ADC0.INTFLAGS;
	ADC0.INTFLAGS = intflags;
```

```
        // If RES < 10...
        if(ADC0.RES < 10)
        {

        // LED0 forward is off
        _delay_ms(5);
        // LED0 forward is off
        PORTD.OUT |= PIN1_bm;
        _delay_ms(5);

        // If isBackward = 0...
        // Else if isBackward = 1...
        if(isBackward == 0)
        {
                // LED0 right is off
                PORTD.OUT |= PIN0_bm;
                // LED2 left is on
                PORTD.OUTCLR = PIN2_bm;
                _delay_ms(5);

                // count = count + 1
                count += 1;
        }
        else if(isBackward == 1)
        {
                // LED2 left is off
                PORTD.OUT |= PIN2_bm;
                // LED0 right is on
                PORTD.OUTCLR = PIN0_bm;
                _delay_ms(5);

                // count = count - 1
                count -= 1;
        }

        // Set the interrupt flag to 0
        interrupt = 0;
        // Set the isBackwardInterrupt to 0
        isBackwardInterrupt = 0;
}

// Timer..
ISR(TCA0_CMP0_vect)
{
        // Disable
        TCA0.SINGLE.CTRLA = 0;
        // Clear flags
        int intflags = TCA0.SINGLE.INTFLAGS;
        TCA0.SINGLE.INTFLAGS = intflags;

        // Set the interrupt flag to 1
        interrupt = 1;

        // Set the isBackwardInterrupt flag to 1
        isBackwardInterrupt = 1;


}
```

```
// Switch..
ISR(PORTF_PORT_vect)
{
	// Clear the interrupt flag
	int intflags = PORTF.INTFLAGS;
	PORTF.INTFLAGS = intflags;

	// LED0 right is on
	PORTD.OUTCLR = PIN0_bm;
	_delay_ms(5);
	// LED1 forward is on
	PORTD.OUTCLR = PIN1_bm;
	_delay_ms(5);
	// LED2 left is on
	PORTD.OUTCLR = PIN2_bm;
	_delay_ms(5);

	// Set the isBackward flag to 1
	isBackward = 1;
	// Set the isBackwardInterrupt to 0
	isBackwardInterrupt = 0;
}
```

Σχετικά με το ερώτημα Α μαζί με την ανάποδη πορεία ο περισσότερος κώδικας έχει αναλυθεί μέσω σχολίων καθώς και οι λειτουργίες των ISR στο παραπάνω ερώτημα. Το μόνο που έχει αλλάξει είναι ότι στην ISR του ADC έχουμε προσθέσει μια if(isBackward == 0) else if(isBackward==1) η οποία ανάλογα την μεταβλητή isBackward επιλέγει αν θα τρέξει την κανονική πορεία δηλαδή στροφή αριστερά και count = count + 1 ή αν θα τρέξει την ανάποδη πορεία δηλαδή στροφή δεξιά και count = count – 1. Σε περίπτωση που έχουμε ανάποδη πορεία θα πρέπει να αφαιρούμε από το count για να δούμε πότε θα φτάσει στην αρχική του θέση δηλαδή στο count = 0. Τέλος να αναφέρουμε πως λειτουργεί το switch το οποίο ενεργοποιεί την ανάποδη πορεία. Για να ενεργοποιήσουμε την ανάποδη πορεία θα πρέπει να θέσουμε την τιμή ‘1’ στο bit 5 του intflags του PORTF. Αφού το κάνουμε αυτό πηγαίνουμε στην ISR του Switch ενεργοποιούνται και τα τρία LED που έχουμε το οποίο συμβολίζει ότι η συσκευή κάνει στροφή 180 μοίρες. Στην συνέχεια επιστρέφουμε στην while(1) και θα τρέξει η if(interrupt == 0 || (isBackward == 1 && isBackwardInterrupt == 0)) αφού λόγω της ISR έχουμε isBackward = 1 και isBackwardInterrupt = 0 με αποτέλεσμα να ενεργοποιηθεί ο timer για ένα ορισμένο χρονικό διάστημα και στην συνέχεια αφού τελειώσει να επιστρέψουμε στην while(1) και να συνεχίσει η συσκευή την ανάποδη πορεία από εκεί που ενεργοποιήσαμε το switch.

Ερώτημα Β

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdlib.h>

#define ped 10

int interrupt = 1;
int count = 0;

int main()
{
	int randomInt = 0;
	// PIN right is output
	PORTD.DIR |= PIN0_bm;

	// PIN forward is output
	PORTD.DIR |= PIN1_bm;

	// PIN left is output
	PORTD.DIR |= PIN2_bm;

	while(1)
	{
		randomInt++;
		// Initialize the ADC for Free-Running mode

		// 10-bit resolution
		ADC0.CTRLA |= ADC_RESSEL_10BIT_gc;

		// Free-Running mode enabled
		ADC0.CTRLA |= ADC_FREERUN_bm;

		// Enable ADC
		ADC0.CTRLA |= ADC_ENABLE_bm;

		// The bit
		ADC0.MUXPOS |= ADC_MUXPOS_AIN7_gc;

		// Enable Debug Mode
		ADC0.DBGCTRL |= ADC_DBGRUN_bm;

		// Window Comparator Mode

		// Set thresholds
		ADC0.WINLT = 10;

		// Enable Interrupts for WCM
		ADC0.INTCTRL |= ADC_WCMP_bm;

		// Interrupt when RESULT < WINLT
		ADC0.CTRLE = ADC_WINCM0_bm;

		// Pull-up enable and interrupt enabled with sense on both edges
		PORTF.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc;

		sei();

		// Start Conversion
		ADC0_COMMAND |= ADC_STCONV_bm;
```

```
_delay_ms(5);

if(randomInt % 2 == 0)
{
	// Single-Running mode enabled
	ADC0.CTRLA = 0;
	ADC0.CTRLA |= ADC_ENABLE_bm;
	ADC0.CTRLA |= ADC_RESSEL_10BIT_gc;

	// Enable Interrupts for WCM
	ADC0.INTCTRL |= ADC_WCMP_bm;
	ADC0.WINHT = 20;

	// Interrupt when RESULT > WINHT
	ADC0.CTRLE = 0x02;

	// Start Conversion
	ADC0_COMMAND |= ADC_STCONV_bm;
	_delay_ms(5);

}

// If count is equal to 8
if((count == 8)
{
	// Exit the app...
	exit(0);
}

_delay_ms(5);
// LED0 right is off
PORTD.OUT |= PIN0_bm;
_delay_ms(5);
// LED2 left is off
PORTD.OUT |= PIN2_bm;
_delay_ms(5);
// LED1 forward is on
PORTD.OUTCLR = PIN1_bm;
_delay_ms(5);

// If interrupt = 0
if(interrupt == 0)
{
	// Clear counter
	TCA0.SINGLE.CNT = 0;
	// Normal Mode
	TCA0.SINGLE.CTRLB = 0;
	// When reaches this value
	-> interrupt CLOCK FRENQUENCY/1024
	TCA0.SINGLE.CMP0 = ped;
	TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1024_gc;
	// Enable
	TCA0.SINGLE.CTRLA |=1;
	// Interrupt Enable (=0x10)
	TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm;
	// Begin accepting interrupt signals
	sei();

	// While the interrupt flag is 0...
	while (interrupt == 0)
	{
```

```
                        // Wait...
                }
        }
}

// ADC..
ISR(ADC0_WCOMP_vect)
{
        // If the result is less than 10 ...
        // Or if result is higher than 20...
        if(ADC0.RES < 10 || ADC0.RES > 20)
        {
                int intflags = ADC0.INTFLAGS;
                ADC0.INTFLAGS = intflags;

                _delay_ms(5);
                // LED forward is off
                PORTD.OUT |= PIN1_bm;

                        // If the result is less than 10...
                        if(ADC0.RES < 10)
                        {
                                // LED0 right is off
                                PORTD.OUT |= PIN0_bm;
                                _delay_ms(5);
                                // LED left is on
                                PORTD.OUTCLR = PIN2_bm;
                        }
                        // If the result is higher than 20...
                        else if(ADC0.RES > 20)
                        {
                                // LED0 left is off
                                PORTD.OUT |= PIN2_bm;
                                _delay_ms(5);
                                // LED right is on
                                PORTD.OUTCLR = PIN0_bm;
                        }
                        _delay_ms(5);
                        count += 1;
                        interrupt = 0;
        }
}

// Timer..
ISR(TCA0_CMP0_vect)
{
        // Disable
        TCA0.SINGLE.CTRLA = 0;
        // Clear flags
        int intflags = TCA0.SINGLE.INTFLAGS;
        TCA0.SINGLE.INTFLAGS = intflags;

        // Set the interrupt flag to 1
        interrupt = 1;
}
```

Σχετικά με το ερώτημα Β ο περισσότερος κώδικας έχει αναλυθεί με σχόλια και οι ISR που αφορούν τον ADC και τον Timer έχουν αναλυθεί και στα προηγούμενα ερωτήματα. Το κομμάτι που θα θέλαμε να αναλύσουμε είναι η δεύτερη λειτουργία του ADC και το

if(randomInt % 2 == 0). Αρχικά στον κώδικα έχουμε προσθέσει int randomInt = 0; και randomInt++; Το οποίο κάθε φορά που τρέχει ο κώδικας θα αυξάνει την μεταβλητή κατά 1. Ο λόγος που το προσθέσαμε αυτό είναι επειδή δεν θέλουμε κάθε φορά να κοιτάει για ο αισθητήρας για τοίχο δεξιά. Όταν το randomInt γίνεται άρτιος τότε θα μπαίνουμε στην If, μέσα στην if απενεργοποιούμε το **free-running mode** και μπαίνουμε σε **single-running mode** (δηλαδή αν ισχύουν οι συνθήκες και μπούμε στην ISR θα τρέξει μόνο μια φορά). Στην συνέχεια αφού στρίψει θα επιστρέψει στην while(1) και θα συνεχίσει μέσα στον Timer, και θα ακολουθήσει τα βήματα που έχουν αναλυθεί και στα προηγούμενα ερωτήματα. Τέλος μετά την στροφή δεξιά θα αλλάξουμε την τιμή στο RES διότι πρέπει να συνεχίσει ευθεία με αποτέλεσμα να ξανατρέξει ο κώδικας της while(1) και να ενεργοποιηθεί το **free-running mode** και να δέχεται συνεχώς τιμές.

Ερώτημα Β με Switch

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdlib.h>

#define ped 10

int interrupt = 1;
int isBackward = 0;
int isBackwardInterrupt = 1;
int count = 0;

int main()
{
	int randomInt = 0;
	// PIN right is output
	PORTD.DIR |= PIN0_bm;

	// PIN forward is output
	PORTD.DIR |= PIN1_bm;

	// PIN left is output
	PORTD.DIR |= PIN2_bm;

	while(1)
	{
		randomInt++;
		// Initialize the ADC for Free-Running mode

		// 10-bit resolution
		ADC0.CTRLA |= ADC_RESSEL_10BIT_gc;

		// Free-Running mode enabled
		ADC0.CTRLA |= ADC_FREERUN_bm;

		// Enable ADC
		ADC0.CTRLA |= ADC_ENABLE_bm;

		// The bit
		ADC0.MUXPOS |= ADC_MUXPOS_AIN7_gc;

		// Enable Debug Mode
		ADC0.DBGCTRL |= ADC_DBGRUN_bm;
```

```
// Window Comparator Mode

// Set thresholds
ADC0.WINLT = 10;

// Enable Interrupts for WCM
ADC0.INTCTRL |= ADC_WCMP_bm;

// Interrupt when RESULT < WINLT
ADC0.CTRLE = ADC_WINCM0_bm;

// Pull-up enable and interrupt enabled with sense on both edges
PORTF.PIN5CTRL |= PORT_PULLUPEN_bm | PORT_ISC_BOTHEDGES_gc;

sei();

// Start Conversion
ADC0_COMMAND |= ADC_STCONV_bm;

_delay_ms(5);

if(randomInt % 2 == 0)
{
	// Single-Running mode enabled
	ADC0.CTRLA = 0;
	ADC0.CTRLA |= ADC_ENABLE_bm;
	ADC0.CTRLA |= ADC_RESSEL_10BIT_gc;

	// Enable Interrupts for WCM
	ADC0.INTCTRL |= ADC_WCMP_bm;
	ADC0.WINHT = 20;

	// Interrupt when RESULT > WINHT
	ADC0.CTRLE = 0x02;

	// Start Conversion
	ADC0_COMMAND |= ADC_STCONV_bm;
	_delay_ms(5);

}

// If count is equal to 8 and isBackward = 0 or...
// If count is equal to 0 and isBackward = 1...
if((count == 8 && isBackward == 0)
|| (count == 0 && isBackward == 1))
{
	// Exit the app...
	exit(0);
}

_delay_ms(5);
// LED0 right is off
PORTD.OUT |= PIN0_bm;
_delay_ms(5);
// LED2 left is off
PORTD.OUT |= PIN2_bm;
_delay_ms(5);
// LED1 forward is on
PORTD.OUTCLR = PIN1_bm;
_delay_ms(5);
```

```
                // If interrupt = 0 or isBackward = 1 …
                // And isBackwardInterrupt = 0…
                if(interrupt == 0
                || (isBackward == 1 && isBackwardInterrupt == 0))
                {
                        // Clear counter
                        TCA0.SINGLE.CNT = 0;
                        // Normal Mode
                        TCA0.SINGLE.CTRLB = 0;
                        // When reaches this value
                        // -> interrupt CLOCK FRENQUENCY/1024
                        TCA0.SINGLE.CMP0 = ped;
                        TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV1024_gc;
                        // Enable
                        TCA0.SINGLE.CTRLA |=1;
                        // Interrupt Enable (=0x10)
                        TCA0.SINGLE.INTCTRL = TCA_SINGLE_CMP0_bm;
                        // Begin accepting interrupt signals
                        sei();

                        // While the interrupt flag is 0...
                        while (interrupt == 0
                        || (isBackward == 1 && isBackwardInterrupt == 0))
                        {
                                // Wait...
                        }
                }
        }
}

// ADC..
ISR(ADC0_WCOMP_vect)
{
        // If the result is less than 10 ...
        // Or if result is higher than 20...
        if(ADC0.RES < 10 || ADC0.RES > 20)
        {
                int intflags = ADC0.INTFLAGS;
                ADC0.INTFLAGS = intflags;

                _delay_ms(5);
                // LED forward is off
                PORTD.OUT |= PIN1_bm;

                if(isBackward == 0)
                {

                        // If the result is less than 10...
                        if(ADC0.RES < 10)
                        {
                                // LED0 right is off
                                PORTD.OUT |= PIN0_bm;
                                _delay_ms(5);
                                // LED left is on
                                PORTD.OUTCLR = PIN2_bm;
                        }
                        // If the result is higher than 20...
                        else if(ADC0.RES > 20)
                        {
                                // LED0 left is off
```

```
                        PORTD.OUT |= PIN2_bm;
                        _delay_ms(5);
                        // LED right is on
                        PORTD.OUTCLR = PIN0_bm;
                }
                _delay_ms(5);
                count += 1;
        }
        else if(isBackward == 1)
        {

                // If the result is less than 10...
                if(ADC0.RES < 10)
                {
                        // LED2 left is off
                        PORTD.OUT |= PIN2_bm;
                        _delay_ms(5);
                        // LED0 right is on
                        PORTD.OUTCLR = PIN0_bm;
                }
                // If the result is higher than 20...
                else if(ADC0.RES > 20)
                {
                        // LED0 right is off
                        PORTD.OUT |= PIN0_bm;
                        _delay_ms(5);
                        // LED2 left is on
                        PORTD.OUTCLR = PIN2_bm;
                }
                _delay_ms(5);
                count -= 1;
        }

        interrupt = 0;
        isBackwardInterrupt = 0;
    }
}

// Timer..
ISR(TCA0_CMP0_vect)
{
    // Disable
    TCA0.SINGLE.CTRLA = 0;
    // Clear flags
    int intflags = TCA0.SINGLE.INTFLAGS;
    TCA0.SINGLE.INTFLAGS = intflags;

    // Set the interrupt flag to 1
    interrupt = 1;
    isBackwardInterrupt = 1;
}

// Switch..
ISR(PORTF_PORT_vect)
{
    // Clear the interrupt flag
    int intflags = PORTF.INTFLAGS;
    PORTF.INTFLAGS = intflags;

    // LED0 right is on
    PORTD.OUTCLR = PIN0_bm;
    _delay_ms(5);
```

```
        // LED1 forward is on
        PORTD.OUTCLR = PIN1_bm;
        _delay_ms(5);
        // LED2 left is on
        PORTD.OUTCLR = PIN2_bm;
        _delay_ms(5);

        // Set the isBackward flag to 1
        isBackward = 1;
        // Set the isBackwardInterrupt to 0
        isBackwardInterrupt = 0;
}
```

Σχετικά με το ερώτημα Β μαζί με το switch για την ανάποδη πορεία τα περισσότερα είναι αναλυμένα με τα σχόλια που έχουμε προσθέσει καθώς και τις περιγραφές που έχουμε δώσει στα προηγούμενα ερωτήματα.