ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΕΠΕΞΕΡΓΑΣΤΩΝ

3η ΑΣΚΗΣΗ

14/05/2022

ΠΑΠΑΔΟΠΟΥΛΟΥ ΑΙΚΑΤΕΡΙΝΗ 1067535

ΠΡΑΠΠΑΣ ΤΡΑΝΤΑΦΥΛΛΟΣ 1067504

Πείραμα

Ο κώδικας της άσκησης είναι ο ακόλουθος:

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>

#define TCB_CMP_EXAMPLE_VALUE (0x80ff)
int risingEdgeCounter = 0;

void CLOCK_init (void);
void PORT_init (void);
void TCB0_init (void);
void TCB1_init (void);
void TCB0_SW5 (void);
void TCB1_SW6 (void);

void CLOCK_init (void)
{
	// Enable writing to protected register
	CPU_CCP = CCP_IOREG_gc;
	// Disable CLK_PER Prescaler
	CLKCTRL.MCLKCTRLB = 0 << CLKCTRL_PEN_bp;

	// Enable writing to protected register
	CPU_CCP = CCP_IOREG_gc;
	// Select 32KHz Internal Ultra Low Power Oscillator (OSCULP32K)
	CLKCTRL.MCLKCTRLA = CLKCTRL_CLKSEL_OSCULP32K_gc;

	// Wait for system oscillator changing to finish
	while (CLKCTRL.MCLKSTATUS & CLKCTRL_SOSC_bm)
	{
		;
	}
}

// Initializes the ports
void PORT_init (void)
{
	// Initializes the LED for the right wheel
	PORTD.DIR |= PIN0_bm;
	// Initializes the LED for the left wheel
	PORTD.DIR |= PIN1_bm;
	// The LED for the ADC
	PORTD.DIR |= PIN2_bm;

}

void TCB0_init (void)
{
	// Load the Compare or Capture register with the timeout value
	TCB0.CCMP = TCB_CMP_EXAMPLE_VALUE;

	// Enable TCB and set CLK_PER divider to 1 (No Pre scaling)
	TCB0.CTRLA = TCB_CLKSEL_CLKDIV1_gc | TCB_ENABLE_bm;

	// Enable Capture or Timeout interrupt
	TCB0.INTCTRL = TCB_CAPT_bm;

	// Enable Pin Output and configure TCB in 8-bit PWM mode
	TCB0.CTRLB |= TCB_CCMPEN_bm;
```

```
	TCB0.CTRLB |= TCB_CNTMODE_PWM8_gc;
}

void TCB1_init (void)
{
	// Load the Compare or Capture register with the timeout value
	TCB1.CCMP = TCB_CMP_EXAMPLE_VALUE;

	// Enable TCB and set CLK_PER divider to 1 (No Pre scaling)
	TCB1.CTRLA = TCB_CLKSEL_CLKDIV1_gc | TCB_ENABLE_bm;

	// Enable Capture or Timeout interrupt
	TCB1.INTCTRL = TCB_CAPT_bm;

	// Enable Pin Output and configure TCB in 8-bit PWM mode
	TCB1.CTRLB |= TCB_CCMPEN_bm;
	TCB1.CTRLB |= TCB_CNTMODE_PWM8_gc;
}

void TCB0_SW5 (void){
	// Load the Compare or Capture register with the timeout value
	TCB0.CCMP = TCB_CMP_EXAMPLE_VALUE;

	// Enable TCB and set CLK_PER divider to 2 (No Pre scaling)
	TCB0.CTRLA = TCB_CLKSEL_CLKDIV2_gc | TCB_ENABLE_bm;

	// Enable Capture or Timeout interrupt
	TCB0.INTCTRL = TCB_CAPT_bm;

	// Enable Pin Output and configure TCB in 8-bit PWM mode
	TCB0.CTRLB |= TCB_CCMPEN_bm;
	TCB0.CTRLB |= TCB_CNTMODE_PWM8_gc;
}

void TCB1_SW6 (void)
{
	// Load the Compare or Capture register with the timeout value
	TCB1.CCMP = TCB_CMP_EXAMPLE_VALUE;

	// Enable TCB and set CLK_PER divider to 2 (No Pre scaling)
	TCB1.CTRLA = TCB_CLKSEL_CLKDIV2_gc | TCB_ENABLE_bm;

	// Enable Capture or Timeout interrupt
	TCB1.INTCTRL = TCB_CAPT_bm;

	// Enable Pin Output and configure TCB in 8-bit PWM mode
	TCB1.CTRLB |= TCB_CCMPEN_bm;
	TCB1.CTRLB |= TCB_CNTMODE_PWM8_gc;
}

int main(void)
{
	CLOCK_init();
	PORT_init();
	TCB0_init();
	TCB1_init();

	// 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 enable debug mode
	ADC0.MUXPOS |= ADC_MUXPOS_AIN7_gc;
	// Window comparator mode
	ADC0.DBGCTRL |= ADC_DBGRUN_bm;
	// Set threshold
	ADC0.WINLT |= 10;
	// Enable Interrupts for WCM
	ADC0.INTCTRL |= ADC_WCMP_bm;
	// Interrupt when RESULT < WINLT
	ADC0.CTRLE |= ADC_WINCM0_bm;

	// Enable Global Interrupts
	sei();

	ADC0.COMMAND |= ADC_STCONV_bm; // Start Conversion

	while (1)
	{
		;
	}
}


ISR(TCB0_INT_vect)
{
	// Clear the interrupt flag
	TCB0.INTFLAGS = TCB_CAPT_bm;
	// Open the LED 0
	PORTD.OUTCLR |= PIN0_bm;
	// If the TCB1 is also on a rising edge...
	if(TCB1.INTFLAGS == 1)
	{
		// Clear the interrupt flag
		TCB1.INTFLAGS=TCB_CAPT_bm;
		// If the temp is an even number...
		if(risingEdgeCounter % 2 == 0)
		{
			// Open the LED 1
			PORTD.OUTCLR |= PIN1_bm;
		}
		// Else if the variable is an odd number...
		else if(risingEdgeCounter % 2 == 1)
		{
			// Close LED 0
			PORTD.OUT |= PIN0_bm;
			// Close LED 1
			PORTD.OUT |= PIN1_bm;
		}
	}
	// Increments the value by one
	risingEdgeCounter += 1;
}


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

	// Closes the LED 0
```

```
        PORTD.OUT |= PIN0_bm;
        // Closes the LED 1
        PORTD.OUT |= PIN1_bm;
        // Opens LED 2
        PORTD.OUTCLR |= PIN2_bm;

        // Initializes the PIN5
        PORTF.DIR |= PIN5_bm;
        // Initializes the PIN6
        PORTF.DIR |= PIN6_bm;

        // If the int flags of the PORTF AND the mask is equal to 5
        if(PORTF.INTFLAGS && 00100000 == 00100000)
        {
                // sw5 is on
                TCB0_SW5();
                TCB1_init();
        }
        // Else if the int flags of the PORTF AND the mask is equal to 6
        else if(PORTF.INTFLAGS && 01000000 == 01000000 )
        {
                //SW6 is on
                TCB1_SW6();
                TCB0_init();
        }
}
```

Για την υλοποίηση του πρώτου ερωτήματος, επιλέξαμε να χρησιμοποιήσουμε 2 TCB timers/counters, ο καθένας από τους οποίους περιέχει ένα 8-bit PWM Mode. Συγκεκριμένα τους TCB0 και TCB1. Παρόλα αυτά επειδή δεν μπορούν και οι δύο ISR των TCB να τρέξουν παράλληλα, χρησιμοποιήσαμε την ISR του TCB0 μόνο και μέσα σ' αυτή με τη βοήθεια των INTFLAGS του TCB1 ελέγχουμε αν έχει φτάσει σε υψηλή στάθμη και ο TCB1. Σε αυτή την περίπτωση, γίνεται ένας έλεγχος με τη βοήθεια ενός μετρητή(στατικής μεταβλητής `risingEdgeCounter`) και αν είναι άρτιος ο μετρητής ανάβει τα LED0 και LED1. Αν είναι περιττός σβήνει τα δύο αυτά LEDs.

Για την υλοποίηση του δεύτερου ερωτήματος, προσθέσαμε τον ADC0, του οποίου το threshold είναι 10. Δηλαδή όταν στο result του θέσουμε μια τιμή μικρότερη του 10 ενεργοποιείται η IRS του. Σε αυτήν, ανάβουμε ένα τρίτο LED το LED2, καθώς και απενεργοποιούμε τα LED0 και LED1. Όταν προφανώς τεθεί μια τιμή μεγαλύτερη του 10 στο result του ADC0 εξερχόμαστε από την ISR του και κάνει fire ξανά η ISR του TCB0.

Για το τρίτο ερώτημα, μέσα στον ADC0 προσθέσαμε και τη λειτουργία δύο διακοπτών. Συγκεκριμένα γίνεται ένας έλεγχος για τα INTFLAGS του PORTF πραγματοποιώντας το λογικό AND μαζί με μία μάσκα κάθε φορά αν το επιθυμητό bit είναι ίσο με 1. Συγκεκριμένα για το PIN5 η μάσκα είναι το δεκαεξαδικό 0x20, δηλαδή το δυαδικό 00100000 και για το PIN6 η μάσκα είναι το δεκαεξαδικό 0x40, δηλαδή το δυαδικό 01000000. Αν το PIN5 έχει πατηθεί ενεργοποιείται η δεξιά περιστροφή, ενώ αντίστοιχα για το PIN6 η αριστερή.

Για το τέταρτο ερώτημα, αν ο έλεγχος του τρίτου ερωτήματος ισχύει, καλείται η αντίστοιχη μέθοδος για το LED και σε αυτήν υποδιπλασιάζεται η περίοδος του ρολογιού για το αντίστοιχο TCB. Δηλαδή, αν το SWTCH6 πατηθεί καλείται η `TCB1_SW6()` στην οποία υποδιπλασιάζεται η περίοδος του ρολογιού του TCB1, οπότε το LED1 θα αναβοσβήνει 2 φορές πιο αργά από το LED0, αφού ο παλμός του 0 είναι πλέον διπλάσιος του 1.