

/*

Puls-Trigger für Servomotoren
mit Mikroprozessor ATtiny13

heinz@gfai.de

Dev.-Environment: AVR-STUDIO 4.19 Built 730

Ziel
Wenn ein Servo zwei Endlagen mit maximaler Kraft bedienen soll, so ist die Ruhestromaufnahme des Servos hinderlich. Ein Servo kann durchbrennen, wenn eine Endlage angeschlagen wird.

Idee
(abschalt) Servo wird voll gegen Anschlag laufen lassen. Nach ganz kurzer Zeit wird der steuernde Puls abgeschaltet.

Aufbau
Der Modul auf Basis AT13 besitzt je einen dreipoligen Servostecker und einen dreipoligen Stiftsockel. Er wird zwischen Empfänger und Servo gesteckt.

Funktion
Pulslänge an Input-Pin PB3 wird geprüft. Übersteigt diese eine gewisse Größe (pulsschwelle), wird Pulslänge am Ausgang PB4 von pulsmin auf pulsmx umgeschaltet:

Pulslänge		
	Input PB3	Output PB4
	[mal 0,1 ms]	[mal 0,1 ms]
	<hr/>	
	<pulsschwelle	pulsmin
	>pulsschwelle	pulsmx
Pulsabschaltung nach "abschalt" [mal 0,1 ms]		

Board AT13 (Basis ATtiny13)

Output (Stecker, ST1)		
6	PB4 out	
5	VCC +5V	
4	GND	
Input (Kabel mit Buchse, ST1)		
3	PB3 inp	
2	VCC +5V	
1	GND	
ISP-MKII (BU2)		
6	/RESET	
5	SCK	~ PB2
4	MISO	~ PB1
3	MOSI	~ PB0
2	VCC	
1	GND	

*/

```
// Pulsschwelle Eingangssignal:
// Schwellwert für Eingangs-Pulsdauer hier einstellen (mal 100µs)
#define pulsschwelle 17
// Neutralstellung wäre 1,5 ms ~ 15.
```

```

                                pulstrg.c
// Aber Senderschalter arbeiten oft zwischen neutral und max bzw. zwischen
// neutral und min. Deshalb sind 12 oder 17 die besseren Werte.

// Pulslängen des Ausgabesignals hier einstellen:
#define pulsmax 21                // maximale Pulslänge (mal 100µs)
#define pulsmín 9                 // minimale Pulslänge (mal 100µs)

// Begrenzung der Anzahl auszugebender Impulse hier wählen:
#define abschalt 18               // Anzahl der auszugebenden Pulse (abh. vom
Servo)

/*
Algorithmus:
    Timer wird mit Takt 0,1ms gestartet
    Messung Input-Pulsdauer an PB3 über Timer-Zählung
    Prüfen, ob Schwelldauer überschritten
    Festlegung, ob min oder max-PulsAusgabe
    Pulsausgabe mit Compare match
    Prüfung, ob sich Ausgabe verändert hat
    wenn nicht, Pulsabschaltung.

Zeitbasis des Timers auf 0,1ms einstellen: zwei Varianten:
*/

// Variante 1: clock = 9,6 MHz, ohne Fuses, kostet 6,5 mA
// Fuses sind: SUT_CKSEL = 9,6 MHz, CKDIV8 = off (default)
// #define prescal 0x05 // Vorteiler /1024 ~ 0.1ms

// Variante 2: stromreduziert auf 0,5...1 mA
// !!! dazu Fuses setzen: !!!
// SUT_CKSEL = 4,8 MHz
// CKDIV8 = on (div/8)
#define prescal 0x03 // Vorteiler /64 ~ 0.1ms

// vorteiler für simulation: ohne (1)
// #define prescal 1

/*****/

// Setzen/Löschen von Port-Bits:
#define bset(PORT,BITNUM) ((PORT) |= (1<<(BITNUM))) // Set I/O bit;
OR
#define bclr(PORT,BITNUM) ((PORT) &= ~(1<<(BITNUM))) // Clear I/O bit;
AND
#define btog(PORT,BITNUM) ((PORT) ^= (1<<(BITNUM))) // Toggle I/O bit;
XOR
// Merke:      neg ~ oder !; OR |; AND &; XOR ^

/*****/

#define __AVR_ATtiny13_
#define F_CPU 4800000 // interner RC-Oszillator

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

////////////////////////////////////

void init_timer(void) { // timer auf 0,1 ms einstellen
    // CPU-clock von 9,6 auf 10,24 bzw. von 4.8 auf 5,12 MHz erhöhen

```

```

                                pulstrg.c
// dazu Kalibrierung des internen RC-Oszillators ändern:
OSCCAL = OSCCAL + 0x0e; // empirisch optimiert, typabhängig
// 10,24 / 9,6 = 1,0666 = 16/15
// OSCCAL ~ dez.105 ~ 7*15 -> +7
// (Änderung gilt nur im Programm, Flash bleibt unberührt)

// Vorteiler einstellen, s.73, Tab.11-9 (CS0x), Timer-Start
TCCR0B = prescal; // Vorteiler /1024 ~ 0.1ms: 5
}

void init_ports(void) { // Ports initialisieren
    DDRB = 0b00010000; // PB4 als Output, alle anderen Input
    PORTB = 0x00; // Ausgabe PB4 = low
}

ISR(TIM0_COMPA_vect) { // Timer0 compare match interrupt
    // Puls an Ausgang PB4 beendet
    bclr(PORTB,4);

    // Interrupts sperren, damit nichts unerwünschtes passiert
    TIMSK0 = 0;
}

////////////////////////////////////

int main(void) {

    uint8_t pulsdauer = 0; // gemessene Pulsdauer
    uint8_t pulsanzahl = 0; // mitzählen, wieviel gleiche kamen
    uint8_t pulsvorher = 0; // Zwischenspeicherung, Änderung erkennen
    uint8_t pulsvorvor = 0; // Zwischenspeicherung, Änderung erkennen
    uint8_t pulsvorvvr = 0; // Zwischenspeicherung, Änderung erkennen

    init_ports();
    init_timer(); // Timer-Vorteiler einstellen -> Timer läuft

    while (1) { // Endlose Schleife

        // Pulsdauer an PB3 messen
        // LH-Flanke an PB3 abwarten (Start):
        loop_until_bit_is_clear (PINB,3); // Pulsende an PB3 abwarten
        loop_until_bit_is_set (PINB,3); // Pulsbeginn an PB3 abwarten

        // Timer starten, dazu Timer auf Null setzen:
        TCNT0 = 0; // PB3-Puls wird ab jetzt ausgezählt

        // Interrupts sperren, damit nichts unerwünschtes passiert
        TIMSK0 = 0;

        // HL-Flanke von PB3 erkennen (Pulsende):
        loop_until_bit_is_clear (PINB,3); // Pulsende an PB3 abwarten

        // Timer auslesen (retten)
        pulsdauer = TCNT0;

        // mit Schwellwert vergleichen
        if (pulsdauer < (pulsschwelle + 1)) // für kleiner gleich <=

            // kurzen Puls ausgeben
            pulsdauer = pulsmin;

        else
            // langen Puls ausgeben

```

```

                                pulstrg.c
    pulsdauer = pulsmx;

    // Puls nur bei zeitlicher Veränderung ausgeben
    if (pulsvorvvr != pulsvorvor)
        pulsanzahl = abschalt; // Veränderung erkannt

    // Wackler eliminieren
    if ((pulsdauer != pulsvorher) | (pulsvorher != pulsvorvor))
        pulsanzahl = 0; // Wackler zurücksetzen

    if (pulsanzahl > 0)
    {
        // Puls an PB4 starten: (PB4 ist low)
        bset(PORTB,4); // Puls starten an PB4
        pulsanzahl--; // einer weniger ist auszugeben
    }

    else { // Puls-STOP
        bclr(PORTB,4); // keinen Puls starten an PB4
    }

    // Timer starten
    TCNT0 = 0; // Timer rücksetzen
    OCR0A = pulsdauer; // zu liefernde Pulsdauer ins output-compare
register

    // Wertekette der letzten Ausgaben speichern
    pulsvorvvr = pulsvorvor;
    pulsvorvor = pulsvorher;
    pulsvorher = pulsdauer;

    // Interrupt-Bug des ATtiny13 behandeln (Spike von 2 µs auf PB4):
    _delay_us(20); // sonst wird BOTTOM-Interrupt ausgelöst
    TIFR0 = 0xff; // latentes Interruptflag löschen

    // Interrupt freigeben:
    sei(); // Interrupts freigeben
    TIMSK0 = (1<<OCIE0A); // OCM-Interrupt freigeben

    // auf compare-match Interrupt warten an nächster loop
}
return 0;
};

```









