ECE 375 LAB 7

Introduction to AVR Development Tools

Lab Time: Tuesday 7-9

Aaron Rito

Introduction

Lab 7 uses the timer/counter and button interrupts to change the speed of the tekbot using the fast PWM feature on the at128.

Program Overview

The program makes the bot drive forward at full speed. When one of the speed buttons is hit, the PWM duty cycle is changed, altering the speed of the motors. When the speed is changed the LED's are updated.

Initialization Routine

The program initializes the at128 chip , and then runs a routine to initialize the stack pointer and direct access registers. Then it defines the locations for the interrupt routines. Then after the typical tekbot initialization, the timer/counters are set up. Since the motors are on the output compare register OCR1A and OCR1B output pins, timer counter 1 is set up for fast PWM using timer/counter 1 control registers TCCR1A and TCCR1B. The clock prescaler (set to x1024), and output modes are also set using these two registers. An initial value of $FF is loaded into the compare registers, to set the motors starting at full speed.

Main Routine

The main routine is only 3 lines long. It does two nops then loops over and over, waiting for an interrupt.

Full_fast, Full_slow Functions

When buttons 0 or 3 are hit the motors will either shut off or go to full speed. R20 keeps track of the current speed. When the button is hit, r20 is updated with either 0 or $FF, and then it's value is pushed out to the LED's on PORTB. This value is then pushed out to the output compare registers.

Tick_fast, Tick_slow

When buttons 1 or 2 are hit the motors will speed up or slow down by 1 value. R20 keeps track of the current speed. When the button is hit, r20 is updated with either 0 or $FF, and then it's value is pushed out to the LED's on PORTB. Next r20 is multiplied with $0F to scale the value 0-255. This value is then pushed out to the output compare registers.

Study Questions

1. In this lab, you used the Fast PWM mode of both 8-bit Timer/Counters, which is only one of many possible ways to implement variable speed on a TekBot. Suppose instead that you used just one of the 8-bit Timer/Counters in Normal mode, and had it generate an interrupt for every overflow. In the overflow ISR, you manually toggled both Motor Enable pins of the TekBot, and wrote a new value into the Timer/Counter's register. (If you used the correct sequence of values, you would be manually performing PWM.) Give a detailed assessment (in 1-2 paragraphs) of the advantages and disadvantages of this new approach, in comparison to the PWM approach used in this lab.

This approach requires manually updating the T/C in code. It also requires setting the motor pins manually. I can't see an advantage to this. All of the same code from the lab method must be included, as well as the code

to manually update the timers and the motor pins. The interrupt and T/C compare method automatically generates the waveform, and the duty cycle can be easily changed by updating the output compare register. The proposed method would be far less efficient.

2. The previous question outlined a way of using a single 8-bit Timer/Counter in Normal mode to implement variable speed. How would you accomplish the same task (variable TekBot speed) using one or both of the 8- bit Timer/Counters in CTC mode? Provide a rough-draft sketch of the Timer/Counter-related parts of your design, using either a flow chart or some pseudocode (but not actual assembly code).

The CTC mode uses the OCR to tell the timer when to toggle the pin and start the timer over. It is frequency based as opposed to the duty cycle base of the fast PWM. This allows for higher frequency waveforms.

Initialize the timer counter for CTC mode.

Set the OCR for the desired frequency.

Set the interrupts for the buttons.

When the button is hit update the OCR.

Difficulties

Setting up the T/C is time consuming.

Conclusion

The at128 has 3 timer counters that can be used to generate PWM signals, or keep track of time in a program. These counters can be used to trigger interrupts internally, or keep track of duration between external events. Fast PWM and the OCR allows for easy switching of speed on the motor pins.

Source Code

```
;
; lab7.asm
;
; Created: 2/26/2017 5:44:30 PM
; Author : aaron
;



.def    mpr = r16
.def    waitcnt = r17                ; Wait Loop Counter
```

```
.def    ilcnt = r18                         ; Inner Loop Counter
.def    olcnt = r19                         ; Outer Loop Counter
.def    speed = r20                         ; Multi-Purpose Register
.def    speed_out = r21                     ; Holds the scaler value
.equ    EngEnR = 4                          ; Right Engine Enable Bit
.equ    EngEnL = 7                          ; Left Engine Enable Bit
.equ    EngDirR = 5                         ; Right Engine Direction Bit
.equ    EngDirL = 6                         ; Left Engine Direction Bit
.equ    MovFwd = (1<<EngDirR|1<<EngDirL)    ; Move Forward Command
.equ    WTime = 25                          ; Time to wait in wait loop
.equ    port_forward = $F             ; scaler


.cseg


;-----------------------------------------------------------------
; Interrupt Vectors
;-----------------------------------------------------------------
.org    $0000                               ; Beginning of IVs
                rjmp    INIT                ; Reset interrupt


.org INT0addr                               ; setting up int0
                rcall full_slow
                reti
.org INT1addr                               ; setting up int1
                rcall tick_slow
                reti
.org INT2addr                               ; setting up int2
                rcall tick_fast
                reti
.org INT3addr                               ; setting up int3
                rcall full_fast
                reti
```

```
.org    $0046                           ; End of Interrupt Vectors

;-----------------------------------------------------------
; Program Initialization
;-----------------------------------------------------------
INIT:
            cli
    ; Initialize the Stack Pointer (VERY IMPORTANT!!!!)
            ldi         mpr, low(RAMEND)
            out         SPL, mpr            ; Load SPL with low byte of RAMEND
            ldi         mpr, high(RAMEND)
            out         SPH, mpr            ; Load SPH with high byte of RAMEND

    ; Initialize Port B for output
            ldi         mpr, $FF            ; Set Port B Data Direction Register
            out         DDRB, mpr           ; for output
            ldi         mpr, $00            ; Initialize Port B Data Register
            out         PORTB, mpr          ; so all Port B outputs are low
       ; timer counters
            ldi         mpr, 0b10100001     ; Activate Fast PWM mode with toggle
            out         TCCR1A, mpr
            ldi         mpr, 0b00001101     ; Activate Fast PWM mode with toggle
            out         TCCR1B, mpr
            ldi         mpr,250             ; Set initial compare value
            out         OCR1AL, mpr
            ldi         mpr, 0x00
            out         OCR1AH, mpr
            ldi         mpr,250             ; Set initial compare value
            out         OCR1BL, mpr
            ldi         mpr, 0x00
            out         OCR1BH, mpr
            ldi         r21, $0F            ;scaler
```

```
            mov         r20, r21
            out         PORTB, r21          ;loading in the intial value

        ; Initialize Port D for input
            ldi         mpr, $00            ; Set Port D Data Direction Register
            out         DDRD, mpr           ; for input
            out         PIND, mpr
            ldi         mpr, $FF            ; Initialize Port D Data Register
            out         PORTD, mpr          ; so all Port D inputs are Tri-Stat

            ; set the Interrupt control state in EIRCA to falling
            ldi mpr, (1 << ISC31) |(1 << ISC21) | (1 << ISC11) | (1 << ISC01)
        sts EICRA, mpr

            ; Set the External Interrupt Mask for int0,1,2,3
            ldi mpr, 0b00001111
            out EIMSK, mpr
            ; Turn on global interrupts
            sei

MAIN:
            rjmp    MAIN
Wait:

Loop:   ldi         olcnt, 224          ; load olcnt register
OLoop:  ldi         ilcnt, 237          ; load ilcnt register
ILoop:  dec         ilcnt               ; decrement ilcnt
            brne    ILoop               ; Continue Inner Loop
            dec         olcnt               ; decrement olcnt
            brne    OLoop               ; Continue Outer Loop
            dec         waitcnt             ; Decrement wait
            brne    Loop                ; Continue Wait loop
```

```
                ret


full_slow:
                cli
                ldi mpr, 0x00
                out EIMSK, mpr
                out EIFR, mpr
                rcall wait                                      ;debounce
                ldi mpr, 0x00
                mov r20, mpr                            ; set the speed to 0
                out PORTB, r20                          ; update the LED's
                out     OCR1AL, mpr                             ; update the control registers
                out     OCR1BL, mpr
                ldi mpr, 0b00001111
                out EIMSK, mpr
                out EIFR, mpr
                sei
                ret
tick_slow:
                cli
                ldi mpr, 0x00
                out EIMSK, mpr
                out EIFR, mpr
                rcall wait              ;debounce
                cpi r20, 0
                BREQ skip               ;  if  the  speed  is  already  at  0,  skip  the  rest  of  the
function.
                dec r20                 ; set the speed to 1 less than it is
                out PORTB, r20 ; update teh LED's
                mul r21, r20    ; multiply by the scaler
                mov mpr, r0             ; update the control registers
                out     OCR1AL, mpr     ; update the control registers
```

```
                out     OCR1BL, mpr     ; update the control registers
                skip:
                ldi mpr, 0b00001111
                out EIMSK, mpr
                out EIFR, mpr
                nop
                sei
                ret
tick_fast:
                cli
                ldi mpr, 0x00
                out EIMSK, mpr
                out EIFR, mpr
                rcall wait              ;debounce
                cpi r20, 15
                BREQ skip1              ; if the speed is already at 15, skip the rest of the
function.
                inc r20                 ; set the speed to 1 more than it is
                out PORTB, r20 ; update the LED's
                mul r21, r20    ; multiply by scaler
                mov mpr, r0             ; update the control registers
                out     OCR1AL, mpr     ; update the control registers
                out     OCR1BL, mpr     ; update the control registers
                skip1:
                ldi mpr, 0b00001111
                out EIMSK, mpr
                out EIFR, mpr
                sei
                ret

full_fast:
                cli
```

```
ldi mpr, 0x00

out EIMSK, mpr

out EIFR, mpr

rcall wait

ldi r20, 0x0F  ; set the speed to 15

out PORTB, r20 ; update the LED's

ldi mpr, 0xFF  ; update the control registers to max speed

out    OCR1AL, mpr    ; update the control registers

out    OCR1BL, mpr    ; update the control registers

ldi mpr, 0b00001111

out EIMSK, mpr

out EIFR, mpr

sei

ret
```