ECE 375 Lab 7

Introduction to AVR Development Tools

**Lab Time: Tuesday 8-10**

Alexander Wood

Zack Divita

# Introduction

Lab 7 focuses on the use of timers and counters. The ATmega128 AVR has 4 timers built in to it, two of which are 8-bit timer/counters and two of them are 16-bit timer/counters. This lab allowed the designers to have the freedom to choose whether to use polling or interrupts to design a control interface for the tekbot platform. This control interface is made up of four buttons that each have a different meaning when pushed. The four buttons will initiate an all stop, increase speed by of its maximum speed, decrease speed by of its maximum speed, or set to maximum speed. This is accomplished using the fast PWM mode of the AVR micro controller.

# Program Overview

This program was designed to use interrupts to allow for commands to be executed. This method was chosen to allow for greater functionality to be programmed into the tekbot at a later date if need be. Since the speed change commands are relatively short, and infrequent when viewed from the perspective of the CPU. This allows for the CPU to be ready to handle other tasks and still accept the speed change request as needed. The interrupt method also made more sense since, the internal timer/counters utilize interrupts to initiate timer/counter operations. Since timer/counter zero and two are to be used, this means we are limited to 8-bit values to control the AVR’s fast PWM mode.

The fast PWM mode allows the AVR micro controller’s timers to control the pulses of power supplied to the TekBot motor. This technique allows us to control the speed of the TekBot based on different increments of an 8-bit value. This increment, controls the duty cycle of the power signal to the tekbot. For the lab we were required to allow for fifteen different speed values starting at no speed, and going to full speed. To accomplish this goal, we divide 255 by 15, this results in an increment of 17 units per increase or decrease in duty cycle.

## Interrupt Vectors

This program utilizes interrupts to control the increase or decrease in the speed of the Tekbot’ s motors. In order to use the external interrupt capability of the AVR board we need to create interrupt vectors for the expected commands. First, the interrupt called when interrupt address $0002 is initiated, the MAXSPEED routine will be called. Next the interrupt address $0004 is assigned to call the subroutine ACCEL. Followed by the interrupt vector from address $0006 to call the DECEL subroutine. Finally, the interrupt vector for address $0008 it configured to call the MINSPEED function. Finally the last values to set are the EICRA

## Initialization Routine

The initialization routine starts with initializing the stack pointer. The initialization routine then sets up both I/O PORTB and PORTD. This includes the configuration of the data direction registers and the port registers initial configuration. Port B is set up for output to the TekBot’s motors as well as set to low initially. Port D is configured as inputs to trigger the interrupts for the speed controls The Port D pins are set up using the tri state. Next the EIMSK is configured to accept the external interrupts from attached button board. The hex value $0F us used to configure interrupts 0 through 3 as inputs. The 8-bit timers are also configured at this point. Both TCCR0 and TCCR2 are configured for fast PWM mode, non-inverting, and no prescalar value. This is how the program will control the speed of the Tekbot. Port B is now given the signal to set the Tekbot up for forward motion for the start and duration f the program. Since the Tekbot should initially be in a nonmoving state the Speed register (r21) is set with the value $00. This value is also stored in the OCR0 and OCR2 registers. This is done to set the initial duty cycle of the PWM generator to 0, since the Tekbot is in the stopped state. Now the EICRA must be set to accept interuts on the falling edge of the clock cycle. This is done using the hex value $AA which will set the appropriate binary value in the appropriate register position. Finally, the speed increment value is loaded into the “incr” register (r20) whifh is the decimal value 17, and he Speed register is set to 0 in decimal, and the last step is setting the SREG interrupt flag to enable.

## Main Routine

The main routine for this program plays little roll. During the entire operation it sits in an idle wait state until an interrupt is made which will change the state of the Tekbot.

## ACCEL Sub-Routine

The ACCEL subroutine is designed to increment the duty cycle of the PWM generator by 1/15 of its total range. This will cause the tekbot motors to increase in speed slightly at each call to the subroutine. This function also needs to ensure that id the duty cycle of the PWM generator is already at max, that nothing will change if the ACCEL subroutine is called again. This is achieved by comparing the current value of the speed register to the hex value $0F, which is the max value of the 8-bit PWM generator. If the compare finds the values to be equal the BREQ condition will be met, and the subroutine will jump to the return command, skipping the state change section of the code.

When the Speed register is not at its max value of 255, the ACCEL function will go into the state change section of the code. First, the current value of the duty cycle of the PWM is copied from the OCR0 register. Now it is incremented by 17, to increase the duty cycle by 1/15 of maximum duty cycle. This value is now copied back into both the OCR0 and OCR2 registers which will implement the increase in the PWM duty cycle. The Speed register value is also incremented to keep track of the current speed which enables the comparison at the beginning of the ACCEL and DECEL routines. Finally, the value in the input register is cleared to allow for setting the new values to be output to the motors. Once the Input register is reconfigured to reflect the new PWM duty cycle, and motor direction, the data is written to the output on Port B. This should make it so the Tekbot is now moving forward at a faster rate than prior to the function being called.

## DECEL Sub-Routine

The DECEL subroutine is almost identical to the ACCEL subroutine with a few changes to allow the function to both reduce the duty cycle by 1/15 of the maximum speed of the PWM generator, and to check to make sure the current speed is greater than zero. If the condition is not met that the speed is 0, the function will skip its state change ability, and just return from the interrupt call.

## MAXSPEED Sub-Routine

The MAXSPEED routine when called simply sets the desired output value of port B to the move forward value. Next the speed register is set to its maximum value, and the motor control output is written to the PORTB outputs. This is followed by the setting of the maximum value in the OCR0 and OCR2 registers, which increases the PWM generator to its maximum duty cycle. Then the subroutine returns from the interrupt leaving the tekbot in the current state until the next interrupt is triggered.

## MINSPEED Sub-Routine

The MINSPEED subroutine is again, almost the same as the MAXSPEED routine with the exception of the function desired result, which is to set the PWM generator to a duty cycle of 0, and stopping the tekbot from moving.

# Additional Questions

1. This new approach to the lab seems to add quite a few complications to the design. Although this is technically possible, it seems like it would take a lot of logic to make it happen correctly. First you would need to calculate how many times a second you would need to toggle the motor enable pins for each interval of speed you would like to support. Then you would need to code subroutines to handle each different speed setting you wanted to support. This would also require frequent interrupts initiated by the timer to change the virtual duty cycle you are creating. The PWM mod of the AVR does have a steep learning curve, but it allows you to easily setup the automatically generated signals which control the speed of the motors. This task uses both 8 bit timers, which can be reduced by the previous method, at the expense of having to code the logic to implement the same functionality.
2. This task would be accomplished in a very similar manner to the process described in question one. The major difference come with the timing mode. In normal mode, you would set a value from which to start counting, and the counter would count to the max value at which point it sets OCF0 and the TOV0 flag would be set. If I needed to emulate the PWM functionality as in question 1, but using CTC mode, I would only use one counter as was explained in the first question. The CTC mode counts from 0 to the value set by the programmer in OCR0, when that value is reached by the counter, setting the TOV0 flag. I would use this to initiate an interrupt which, when called, will toggle the motor enable pins. This process would repeat over the desired time period to emulate the PWM function. The speed would be controlled by separate interrupts connected to the button board. These interrupts would initiate the appropriate command for the button push, which would change the OCR0 value. By doing this you, create the virtual duty cycle of the PWM generator. The change in the OCR0 value would change how long in-between motor enable toggles thus virtually changing the “duty cycle.

# Difficulties

The first major challenge of this lab was learning how to use the timers correctly. There are a lot of registers with specific functions and different configuration modes that can be used to accomplish the same task in many different ways. The additional questions for this lab were a great challenge because of the same challenges faced with the initial counter setup.

# Conclusion

This lab presented a new type of challenge from previous labs. We were given the freedom to determine the best path to accomplish the task of creating speed controls for the tekbot platform. We were asked to use the AVR micro controller’s internal counters to generate a PWM signal which could control the speed of the TekBot motors. This was accomplished by writing four subroutines that executed each of the desired function through the use of interrupts.

# Source Code

Provide a copy of the source code. Here you should use a mono-spaced font and can go down to 8-pt in order to make it fit. Sometimes the conversion from standard ASCII to a word document may mess up the formatting. Make sure to reformate the code so it looks nice and is readable.

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\* Lab7.asm

;\*

;\* Timers/Counters

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\*

;\* Author: Zachary DeVita And Alex Wood

;\* Date: November 8, 2016

;\*

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.include "m128def.inc" ; Include definition file

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Internal Register Definitions and Constants

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.def mpr = r16 ; Multipurpose register

.def incr = r20 ; Register to hold speed increment value

.def Speed = r21 ; register to hold the current speed

.def Input = r22 ; register to hold value that gets

; output to port B

.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

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Start of Code Segment

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.cseg ; beginning of code segment

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Interrupt Vectors

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

.org $0000

rjmp INIT ; reset interrupt

; place instructions in interrupt vectors here, if needed

.org $0002 ; {IRQ0 => pin0, PORTD}

rcall MAXSPEED

reti ; Return from interrupt

.org $0004 ; {IRQ1 => pin1, PORTD}

rcall ACCEL

reti ; Return from interrupt

.org $0006 ; {IRQ0 => pin2, PORTD}

rcall DECEL

reti ; Return from interrupt

.org $0008 ; {IRQ1 => pin3, PORTD}

rcall MINSPEED

reti ; Return from interrupt

.org $0046 ; end of interrupt vectors

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Program Initialization

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INIT:

; Initialize the Stack Pointer

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

; Configure I/O ports

; Initialize Port B for output

ldi mpr, $FF ; Set Port B Data Register

out DDRB, mpr ; for output

ldi mpr, $00 ; Initialize Port B Data Register

out PORTB, mpr ; so all Port B outputs are low

; Initialize Port D for input

ldi mpr, $00 ; Set Port D Data Direction Register

out DDRD, mpr ; for input

ldi mpr, $FF ; Initialize Port D Data Register

out PORTD, mpr ; so all Port D inputs are Tri-State

; Configure External Interrupt Mask

ldi mpr, $0F

out EIMSK, mpr

; Configure 8-bit Timer/Counters

ldi mpr, 0b01101001 ; Activate Fast PWM mode with toggle

out TCCR0, mpr ; (non-inverting), and no prescalar

out TCCR2, mpr ; (non-inverting), and no prescalar

; Set TekBot to Move Forward

ldi mpr, (1<<EngDirR|1<<EngDirL)

out PORTB, mpr ; Send command to motors

; Set initial speed, display on Port B pins 3:0

ldi Speed, $00 ; Set speed initially to zero

out OCR0, Speed

out OCR2, Speed

; Enable global interrupts (if any are used)

ldi mpr, $AA ; set interrupts to falling edge

sts EICRA, mpr

ldi Speed, 0 ; initial value for speed

ldi incr, 17 ; value to increase/decrease speed by

sei ; set global interrupt flag

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Main Program

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

MAIN:

rjmp MAIN ; loop to top of MAIN

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;\* Functions and Subroutines

;\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

;-----------------------------------------------------------

; Func: Accelerat function

; Desc: This function is used when an interupt is recived on

; I/O interupt address $0002. The function will increase the

; duty cycle of the PWM generator by one unit, which is 1/15

; of the range of the duty cycle.

;-----------------------------------------------------------

ACCEL: ; Begin a function with a label

; Execute the function here

cpi Speed, $0F ; check if value is at minimum

breq JUMP

in mpr, OCR0 ; put OCR value into mpr

add mpr, incr ; increment register by 17

out OCR0, mpr ; Set compare value

out OCR2, mpr ; Set compare value

inc Speed ; increment current speed

clr Input ; clear input register

or Input, Speed ; set Input bits for current speed

ori Input, (1<<EngDirR|1<<EngDirL) ; and for engine moving forward

out PORTB, Input ; output the value to PORTB

JUMP:

ret ; End a function with RET

;-----------------------------------------------------------

; Func: Template function header

; Desc: Cut and paste this and fill in the info at the

; beginning of your functions

;-----------------------------------------------------------

DECEL: ; Begin a function with a label

; Execute the function here

cpi Speed, $00 ; check if value is lesser

breq JUMP2

in mpr, OCR0 ; put OCR value into mpr

sub mpr, incr ; decrement register by 17

out OCR0, mpr ; Set compare value

out OCR2, mpr ; Set compare value

dec Speed ; decrement current speed

clr Input ; clear input register

or Input, Speed ; set Input bits for current speed

ori Input, (1<<EngDirR|1<<EngDirL) ; and for engine moving forward

out PORTB, Input ; output the value to PORTB

JUMP2:

ret ; End a function with RET

;-----------------------------------------------------------

; Func: Template function header

; Desc: Cut and paste this and fill in the info at the

; beginning of your functions

;-----------------------------------------------------------

MAXSPEED: ; Begin a function with a label

; Execute the function here

ldi mpr, 0b01101111 ; load mpr with engine on

; & max speed set

ldi Speed, $0F ; set speed to max

out PORTB, mpr ; output mpr to PORTB

ldi mpr, $FF ; load mpr with all bits set

out OCR0, mpr ; Set compare value

out OCR2, mpr ; Set compare value

ret ; End a function with RET

;-----------------------------------------------------------

; Func: Template function header

; Desc: Cut and paste this and fill in the info at the

; beginning of your functions

;-----------------------------------------------------------

MINSPEED: ; Begin a function with a label

; Execute the function here

ldi mpr, 0b01100000 ; load mpr with engine on

; & min speed set

ldi Speed, $00 ; set speed to min

out PORTB, mpr ; output mpr to PORTB

ldi mpr, $00 ; load mpr with no bits set

out OCR0, mpr ; Set compare value

out OCR2, mpr ; Set compare value

ret ; End a function with RET