

## 7.0.1 Lab 4 - A Feedback Controller

### 7.0.1.1 - Purpose

To use the Atmega32 microcontroller for velocity feedback control of a motor.

### 7.0.1.2 - Background/Theory

A basic feedback control system is shown in Figure 7.1. In this system the atmega32 will output a control voltage as a PWM signal. When the output is 0V, the transistor will be off, and act like an open switch. However, when the output is 5V, it will turn the transistor on, and allow current to flow to the motor. (Note: There will be a voltage drop across the transistor, in this case approximately 0.8V.) The motor shaft is connected to a tachometer to measure the motor speed. (Note: In this lab we will use another motor as the tachometer.) The tachometer will produce a voltage proportional to the motor speed.

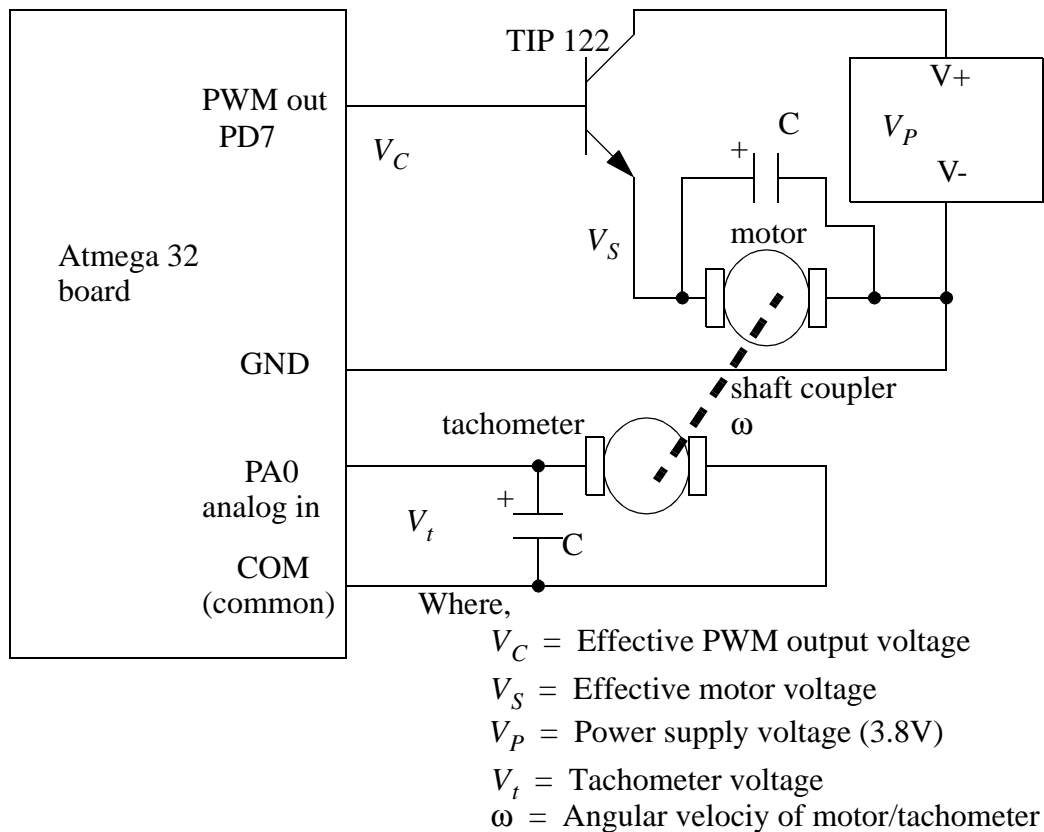
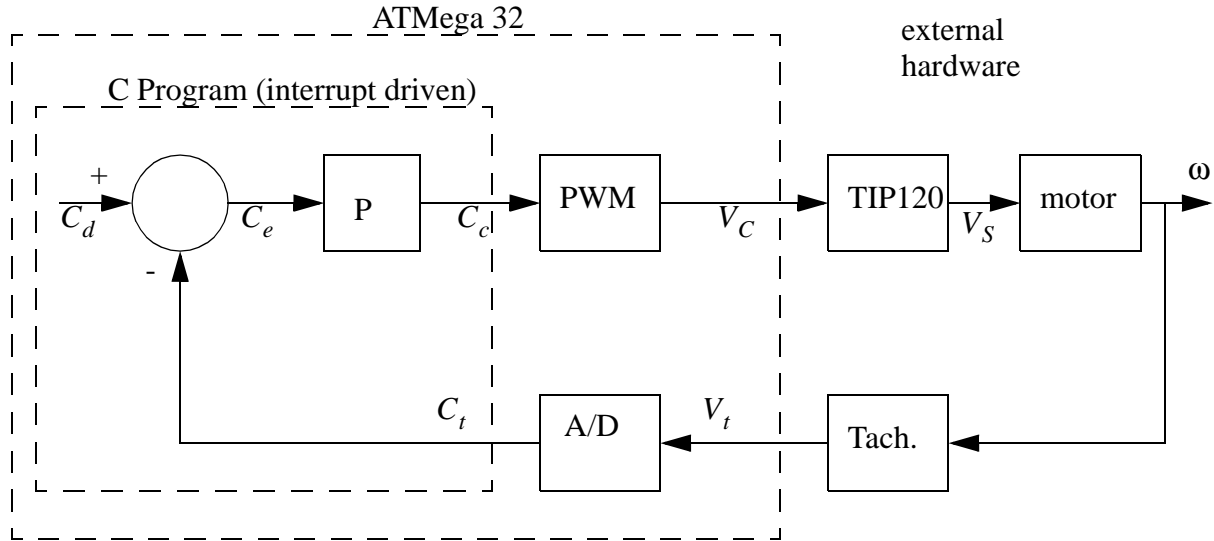


Figure 7.1 Atmega32 based velocity feedback controller

To complete the control system, a program is required. The program reads the motor speed input,  $C_t$ , from the analog voltage,  $V_t$ , and use it to adjust the output,  $C_c$ , to adjust the voltage,  $V_s$ , to control the motor speed. Normally a user may supply a setpoint,  $C_d$ . This setpoint indicates the desired speed. In this program care is required to ensure that the  $C_c$  value remains in the range from 0 to 255 because of the limitation of the PWM functions.



Where,

$C_d$  = Desired tach. speed - input via keyboard (0-1023)

$C_t$  = Actual tach. speed read via A/D (unsigned char, 0-1023)

$C_e = C_d - C_t =$  System error (int)

$C_c = PC_e =$  Control value to output via PWM (unsigned char, 0-255)

$P =$  The controller gain (int)

Figure 7.2 The complete feedback loop

The PWM output to the motor is a square wave with a variable duty cycle. Equation (1) below shows how to convert the PWM value to an effective output voltage. It is worth noting that part of the voltage from the power supply is lost across the transistor, thus reducing the effective voltage to the motor.

$$V_S = (V_P - V_{CE}) \left( \frac{C_C}{C_{MAX}} \right) \quad (1)$$

Where,

$V_S$  = The effective voltage delivered to the motor

$V_P$  = The power supply voltage (use 3.8V)

$V_{CE}$  = The voltage across the transistor when on (use 0.8V)

$C_{MAX}$  = The maximum count in the counter (255)

$C_C$  = The variable counter value (0-255)

To implement this controller we need a better program structure than was used before. This is shown in the following program listing. The `IO_update()` function will be called many times per second by an interrupt subroutine. The `CLK_setup` function initializes the interrupt routines in the processor so that the `'SIGNAL(SIG_OVERFLOW1)'` function will be called once every 10ms. This in turn will call the `IO_update()` function. The main program sets up the various input output devices and then begins a loop where it deals with keyboard IO. Notice that the subroutines to update the inputs and outputs do not appear in the main program loop. It communicates with those routines by changing the value of the global variable 'count'.

## egr345 lab guide - 7.4

```

#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include "sio.c"

// ----- IO Stuff -----

int count = 0; // a count value to be output on port B

void IO_setup(){
    DDRB = 0xFF; // all of port B is outputs
}

void IO_update(){// This routine will run once per interrupt for updates
    PORTB = count;
    // put feedback control stuff here
}

// ----- Clock/interrupt stuff -----

#define CLK_ms10 // set the updates for every 10ms
unsigned int CNT_timer1; // the delay time
volatile unsigned int CLK_ticks = 0; // the current number of ms
volatile unsigned int CLK_seconds = 0; // the current number of seconds

SIGNAL(SIG_OVERFLOW1){ // The interrupt calls this function
    CLK_ticks += CLK_ms;
    if(CLK_ticks >= 1000){ // The number of interrupts between output
changes
        CLK_ticks = CLK_ticks - 1000;
        CLK_seconds++;
        IO_update();
    }
    TCNT1 = CNT_timer1;
}

void CLK_setup(){ // Start the interrupt service routine
    TCCR1A = (0<<COM1A1) | (0<<COM1A0) | (0<<COM1B1) | (0<<COM1B0)
        | (0<<WGM11) | (0<<WGM10) | (0<<FOC1A) | (0<<FOC1B);
    // disable PWM and Compare Output Modes
    TCCR1B = (0<<WGM12) | (0<<WGM13)
        | (0<<ICNC1) | (0<<ICES1)
        | (1<<CS12) | (0<<CS11) | (1<<CS10); // set to clk/1024
    CNT_timer1 = 0xFFFF - CLK_ms * 16; // 16 = 1ms, 160 = 10ms
    TCNT1 = CNT_timer1; // start at the right point
    //SFIOR &= PSR10; // reset the scaling bit
    // use CLK/1024 prescale value
    TIFR&=~(1<<TOV1); // set to use overflow interrupts
    TIMSK = (1<< TOIE1 );// enable TCNT1 overflow
    sei(); // enable interrupts flag
}

```

```

// ----- The main program loop -----

int main(){
    int    c;

    sio_init();
    IO_setup();
    CLK_setup();

    for(;;){
        while((c = input()) == -1){} // wait for a keypress

        if(c == '+'){ // increment the output on port B
            if(++count > 255) count = 255;
            outln("counter incremented");
        } else if(c == '-'){
            if(--count < 0) count = 0;
            outln("counter decremented");
        } else if(c == 'p'){
            outint(count);
        } else if(c == 'h'){
            outln("HELP: +, -, p, q");
        } else if(c == 'q'){
            break;
        }
    }
    sio_cleanup();

    return 1;
}

```

### 7.0.1.3 - Prelab

1. Review the programs from the previous lab.
2. Write a program that will allow keyboard commands to change the PWM, and output the analog input once a second. Please note that this may be noisy and multiple readings may be required to reduce the noise.
3. Write a program that implements the feedback controller described in the background section. Put the feedback loop in an interrupt subroutine, and use the main program for keyboard IO. Don't forget to consider the number limitation of the chosen datatypes.

### 7.0.1.4 - Equipment

computer with an atmega32 compiler  
 atmega32 board  
 2 motors

2 multimeters  
 strobe tachometer  
 TIP 122 Darlington NPN transistor  
 heatsink for TIP 120 transistor  
 external power supply

### 7.0.1.5 - Experimental

1. Build the motor speed controller pictured in Figure 7.3. Use Figure 7.4 when connecting the transistor. Use a heat sink on the transistor to help dissipate heat. Use the PWM program from the previous lab to control the PWM output and test the motor speed control.

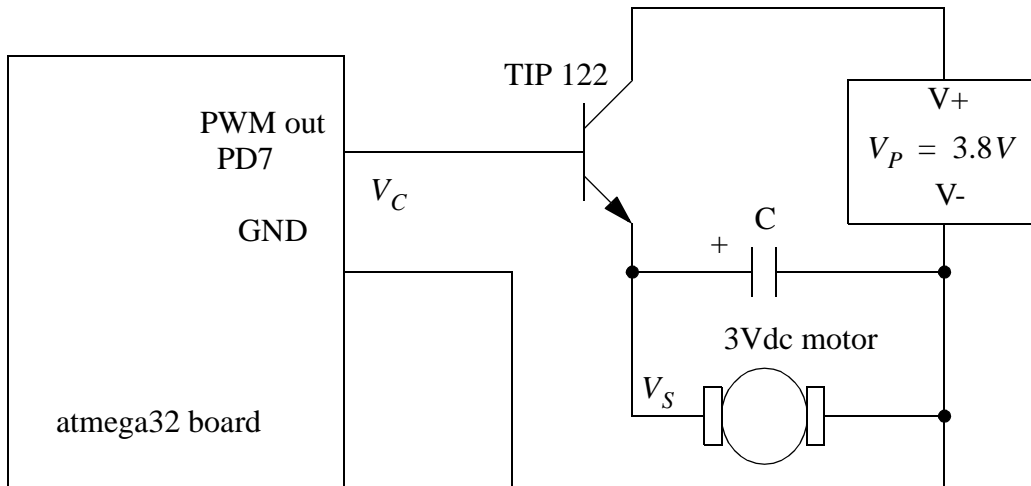


Figure 7.3 PWM control of motor speed

Note: When not running an experiment, turn the motor power supply off for safety and to minimize overheating issues.

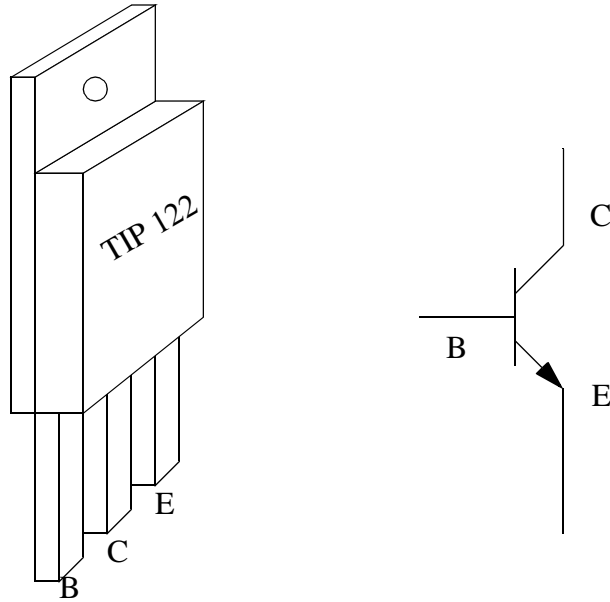


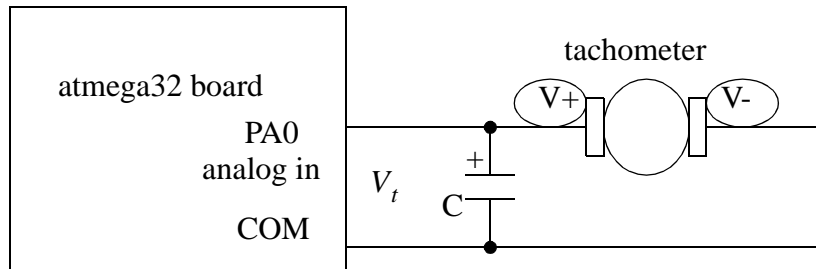
Figure 7.4 TIP 122 NPN Darlington pair transistor

2. Use a second motor as the tachometer (not to be confused with the strobe tachometer, used for calibration). This is done by connecting the motor shafts using a light colored piece of tape. It is also advisable to fix both motors to a common base to reduce vibrations. **DO NOT CONNECT THE TACHOMETER LEADS TO THE ANALOG INPUT YET.**
3. Label both leads of the motor  $V_+$  and  $V_-$ . Connect these leads to a DMM. Connect the  $V_+$  lead to the positive input on the DMM, and  $V_-$  to the common/ground on the DMM.
4. Use the program developed for the prelab to vary the PWM output to control the motor speed. Take the following readings in a tabular form. The motor RPM can be measured using the strobe tachometer as a reference.  $V_t$  is measured from the motor tachometer using the DMM, and a second DMM is used to measure  $V_s$ .

PWM Value (hex)	$V_s$ (V)	$V_t$ (V)	RPM

5. If the voltages for  $V_t$  were negative swap the  $V_+$  and  $V_-$  labels on the motor tachometer. **FAILURE TO DO THIS MAY DAMAGE EQUIPMENT.** Connect

the motor tachometer to the analog input as shown in Figure 7.5. You may also change the signs for  $V_t$  in the table.



*Figure 7.5* Motor tachometer analog input

6. Use the prelab program to vary the PWM output and read the resulting analog input. Add the values to the previous table.
7. Enter and test the feedback control program developed in the prelab with a gain of  $P=2$ . Change the setpoint,  $C_d$ , to different values and measure the resulting speed,  $C_t$ . Record these values in a new table.
8. Repeat the previous step twice for gains of  $P=1$  and  $P=4$ .
9. Graph the results from the three tests on a single graph.