Introduction: Electric motors can be considered a type of transducer that converts electrical energy into mechanical motion, typically in terms of rotation. In robotic and many mechatronic systems motors are also referred to as actuators. Many different motor types and operation methods exist; however, motors are basically divided into AC and DC types each with a number of subtypes. For example: brushed, brushless, and stepper motors are all different implementations of DC motors. This lab will explore the operation, interface, and microprocessor-based control of some motors as commonly used in computer and robotic systems.

I. DC Motors

Objectives:

- Demonstrate the operation of a DC fan motor.
- Show basic processor control of a DC motor.
- Show the application of pulse-width modulation (PWM) power control of a DC motor.

Materials required:

- (1) DC mini-fan
- (1) 3-pin header adapter
- (1) 2N2222 transistor
- (1) 1 kΩ resistor
- (1) 2 kΩ potentiometer

Procedure:

1. From the label on the DC mini-fan, record the following motor specifications:
   Q1. Rated motor voltage = __________, rated motor current = __________.
   The maximum power rating is then = __________.

2. Using a 3-pin header adapter to plug the fan connector into your breadboard, connect the fan’s red and black wires directly to the EVB’s +5V and ground respectively to verify proper operation of the fan.

3. Recall that the maximum output current of the S12’s output port pins is given as 25 mA; note that the required fan motor current well exceeds this. Thus, to interface the fan motor to an output port, a transistor driver circuit will be needed. Because the fan motor current is less than 1 A, the 2N2222 can easily handle this load; however, a larger motor would require a more capable transistor. Construct the interface circuit as shown in figure 1. Note that SW5 is provided on-board the Dragon12 and connects to PH0 with an appropriate pull-up; thus, do not build this part! Refer to the previous digital output lab for a refresher on the transistor if necessary. The fan’s “tach” connection is not used in this lab.

![Fig. 1: DC fan circuit](image-url)
4. Connect an oscilloscope channel to PB0. Using the monitor’s MM command, change PB0 to be an output then change PB0 between high and low to verify proper operation of your circuit.

5. Now conduct the following test to verify your understanding of how to implement programmatic control of the fan. Review Fan1.asm shown in listing 1. Enter, assemble, and download this program to the EVB. Run this program and verify for proper operation.
   Q2. What event causes this program to activate the fan control circuit? _____________________

6. Referring again to figure 1, it should be obvious that with a slight rewiring of the switch, it can directly control the transistor driver and the inclusion of the EVB hardware and S12 software is completely unnecessary. This is an example of a classic Rube Goldberg “overkill” design. However, by having the fan circuit controlled by the processor, we can now enjoy additional capabilities provided by digital computation – like fan RPM control for example!

7. Add a 2 kΩ potentiometer to PAD06 as you did in the A/D part of the previous lab.

8. Enter, assemble, and download Fan2.asm as shown in listing 2. Run this program and vary the pot while observing the PB0 output on the scope and the corresponding effect on the fan.
   Q3. How does the varying duty cycle on the PB0 output affect the fan?

   Q4. Describe the concept of pulse width modulation.

   Q5. What is the PWM frequency of PB0? _____________ You should be able to answer this by observing either the scope or program code.

DISCUSSION: Although PWM is normally used to convey information digitally over a communication link, this application of PWM actually varies the average power to a load (the fan motor). By pulsing the motor power on and off faster than it can physically react to, it “sees” an average power input less than the full 100%. This is hard to observe on the oscilloscope display, but easy to see with a slow-reacting voltmeter...

9. Attach a DMM across the motor connections and observe the motor voltage as you change the PB0 duty cycle by varying the potentiometer. Aha!
   Q6. Describe the relationship between PWM duty cycle and average power delivered to the load.

   Q7. What is the LM331 (http://www.ti.com/product/LM331)?

OBSERVATION: As you should deduce from this last question, even the addition of RPM control is still too simple for a processor-based solution since low-cost devices such as the LM331 provide a much more direct solution. However, in a more sophisticated system with additional IPO (input-processing-output) requirements, processor control of motor RPM is a great solution!

10. You may now disassemble this circuit and return all parts to their respective storage bins.
II. Stepper Motors

Objectives:

- Demonstrate the operation of a unipolar stepper motor.
- Show basic processor control of a unipolar stepper motor.
- Show the application of digital processor-based control of a stepper motor.

Materials required:

(1) PM25L-024 unipolar stepper motor  
NMB step motor data sheet
(1) ULN2003A darlington array IC  
ULN2003A data sheet
(1) 6-pin header pin adapter
(1) 2 kΩ potentiometer

Procedure:

1. Refer to the circuit in figure 2. Because of the multiple coil loads of a stepper motor (4 in this case), we will use the very neat ULN2003A Darlington array driver as a single-chip interface solution. Obtain the ULN200x data sheet (from the resources page of the course website) and review its pin diagram (fig. 2) and its internal schematic diagram (fig. 1). Transfer pin numbers to the circuit diagram below. Note that by connecting the ULN2003’s “Common” pin to V+ we are taking advantage of the kickback (snubber) diodes provided inside the IC.

2. Obtain and review the NMB Permanent Magnet Step Motors datasheet (also on the website). Determine the pinout of our unipolar motor’s 6-pin connector by completing the following chart.

<table>
<thead>
<tr>
<th>Pin:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color:</td>
<td>Brn</td>
<td>Red</td>
<td>Yel</td>
<td>Blk</td>
<td>Grn</td>
<td>Orn</td>
</tr>
<tr>
<td>Function:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3. Build the circuit as given. Affix the stepper motor connector to the breadboard using the handy 6-pin header adapter. For V+, you will need to use an external power supply (don’t forget to common the power supply ground with the EVB ground).

![Stepper motor circuit](image)

**Fig. 2: Stepper motor circuit**

4. Ok, now it’s time to take your circuit for a test drive! Reset the EVB. From the monitor prompt, configure DDRB so that the bottom 4 bits of PORTB are outputs. Open PORTB and verify the outputs are all 0. This turns off the drivers in the ULN2003 which causes none of the coils in the stepper motor to be energized. Turn the motor’s output shaft by hand and notice how “loose” it is. Energize coil A by setting PB3 (only) to 1. Now check out the *holding torque* on the output shaft. (And this is only a small stepper!) Return PORTB back to all zeros to turn off all coils.

5. To properly rotate a stepper motor, the coils must be pulsed in a certain sequence. The typical sequence is A, B, A', B' then repeat. Each coil will cause the shaft to rotate a particular amount according to the number of *poles* within the motor. Did you notice in the NMB data sheet that the
"024" part of our motor’s part number is the number of steps? This is a direct result of the number of poles it has.

Q8. The PM25L-024 stepper has a resolution of $360° \div 24$ steps/revolution = _____ °/step.

6. Since we arranged the coils on PB3~PB0 as A, B, A', B' the required output sequence will be 8,4,2,1. Using the monitor’s memory modify command; try outputting this sequence on PORTB a couple times to achieve ongoing rotation. To achieve opposite rotation, the sequence is simply produced in reverse. Try this! Return PORTB to all zeros when finished.

Q9. Clockwise (CW) rotation is achieved by outputting the sequence: __________ .

   Counter-Clockwise (CCW) rotation is achieved by outputting the sequence: __________ .

7. The sequences in the previous step are actually considered “half power” since only one coil at a time is energized. This is also referred to as single phase excitation. The more common “full power” sequence would energize adjacent pairs of coils for each step in the sequence AB, BA', A'B', B'A which would be C,6,3,9. This is known as dual phase excitation. Go back to PORTB and turn on coil A again as in step 4. While feeling the half-power holding torque again on the output shaft, have your lab partner also turn on coil B by changing PORTB from 8 to C. Wow, that’s what we need – more power! Turn off all coils after you regain composure from all the excitement.

8. The previous two sequences may also be combined into what is called half-stepping or 1-2 phase excitation. By alternating the output sequence between a full-power and half-power pattern, we can double the number of steps per revolution thereby halving the step resolution. Try outputting the following half-step sequence to the motor:

   8 – C – 4 – 6 – 2 – 3 – 1 – 9

Q10. How does a half-step sequence differ from a full-step sequence in terms of effect on the motor?

Q11. When half-stepped, the PM25L-024 stepper has a resolution of _____ °/step.

9. Ok, now it’s the processor’s turn. Enter Stepper1.asm as shown in listing 3 into the editor. Note that three different step pattern tables are provided but all are “turned off” by being commented. Activate the half-power StepTbl1 by uncommenting only that line. You will also need to replace the missing ?? operand with the ATD0CTL5 register value for the correct channel and left justification.

10. Assemble, download and run this version of the program. Vary the setting of the potentiometer while observing the program’s effect on the motor.

   Q12. How does the potentiometer setting affect the operation of the stepper motor?

11. Return to the editor, re-comment StepTbl1 and uncomment the StepTbl2 line to enable the full-power sequence. Run this version and note the effect on the motor. Finally, switch to the 3rd table for half-stepping and observe its effect. You should notice that overall RPMs are now halved but the motor rotation is much smoother. Imagine a pair of these motors with wheels mounted providing the locomotion for a robotic platform.

**DISCUSSION:** Using a microcontroller and software to control sequencing and timing makes stepper motor control quite straightforward. Although other methods such as discrete and FPGA-based control exist, processor-based control provides a cost effective solution especially when part of a larger embedded system, e.g. inkjet printers.

12. Have the last step of this lab checked off by the instructor. Disassemble your circuit and return all components to their proper storage bins.

   Instructor Signoff: ______________
Listing 1: DC fan motor demo program #1

* Fan1.asm: demonstrate control of a DC fan motor
* Control DC fan on PB0 output bit according to "fan control switch" on
* PH0 input: low=fan on, high=fan off.

***************************************************************************
*         <<< EQUATE SECTION >>>
; constants
CR EQU $0D ;ASCII Carriage Return
LF EQU $0A ;ASCII Line Feed
; memory map definitions
DATA EQU $1000 ;where our data can begin on EVB
CODE EQU $2000 ;where our code can begin on EVB
; "system" objects (I/O registers, monitor routines, etc.)
PORTB EQU $0001 ;I/O port B
DDRB EQU $0003 ;DDR for port B
FANQ EQU %00000001 ;PORTB bit of fan control circuit
PORTH EQU $0260 ;I/O port H
SW5 EQU %00000001 ;port H bit number of SW5
DDRJ EQU $026A
; D-Bug12 utility routine vectors
VPUTCHAR EQU $EE86 ;send character in B to output device
***************************************************************************
*         <<< MAIN PROGRAM LOOP >>>
ORG CODE
START movb #2,DDRJ ;enable LED bank
 movb #1,DDRB
 ldx #Msg1 ;point X to initial message
 jsr PUTS ;output it
 Loop1 brset PORTH,SW5,Loop1 ;wait for SW5 press
 bset PORTB,FANQ ;turn fan on
 ldx #Msg2 ;display "fan on" message
 jsr PUTS
 Loop2 bclr PORTH,SW5,Loop2 ;wait for SW5 release
 bcrl PORTB,FANQ ;turn fan off
 ldx #Msg3 ;display "fan off" message
 jsr PUTS
 bra START ;repeat main loop

* string output routine, X points to null-terminated string
PUTS ldab 0,x ;get 1 char from string
 beq PSDONE ;done if terminating null
 jsr PUTFCHAR ;output this char
 inx ;advance X to next char
 bra PUTS ;continue
PSDONE rts ;done, return to caller

* single character (in B) output routine
PUTCHAR pshx ;save caller's X
 ld x VPUTCHAR ;fetch address of monitor rtn
 jsr 0,X ;call monitor rtn.
 pulx ;restore caller's X
 rts

* Constant data
Mag1 FCC "Waiting for Fan switch"
 FCB CR,0
Msg2 FCC "Fan ON"
 FCB CR,0
Msg3 FCC "Fan OFF"
 FCB CR,0
***************************************************************************
END
Listing 2: DC fan motor demo program #2
* Fan2.asm: demonstrate PWM control of a DC fan motor
* PWM control DC fan on PB0 according to "fan speed pot" on PAD06

***************************************************************************
*         <<< EQUATE SECTION >>>
*        ; constants
CR     EQU        $0D       ;ASCII Carriage Return
LF     EQU        $0A       ;ASCII Line Feed
PERIOD EQU        50        ;1/(20 Hz) = 50ms period
***************************************************************************
*         <<< MAIN PROGRAM LOOP >>>
ORG      CODE
START   ldx       #Msg1     ;point X to initial message
jsr       PUTS      ;output it
movb      #2,DDRB   ;enable LED bank
movb      #1,PORTB
movb      #$80,ATD0CTL2 ;turn on ADC
MainLp   movb      #$86,ATD0CTL5 ;start conversion on PAD06
Loop1    tst       ATD0STAT0     ;check SCF on bit 7
bpl       Loop1     ;A/D not done yet
ldd       ATD0DR0   ;fetch result of A/D conversion
ldx       #20
idiv                ;fan on ms = ADR / 5  [0..51ms]
xgdx                ;move quotient to D
jsr       Pulse     ;output computed pulse width
bra       MainLp

* Pulse: send out a pulse on FANQ bit of PORTB
* in: D=ONTIME (on time in ms; should be 0..PERIOD)
* out: output pulse on PB bit, registers preserved except CCR
Pulse    cmpd      #0        ;check requested ONTIME
beq       NoPulse   ;0ms = no pulse
bset      PORTB,FANQ ;turn fan on
tba                 ;copy ONTIME to A
jsr       Pulse     ;output computed pulse width
bra       MainLp
NoPulse  bclr      PORTB,FANQ ;turn fan off
ldaa      #PERIOD   ;calc off time = PERIOD - ONTIME
sba
bls       PlsExit   ;skip off delay if ONTIME>=PERIOD
jsr       DelayN    ;hold for off-time
PlsExit   rts

* DelayN: delay 'N' milliseconds
* in: A = 'N', number of ms to delay
* out: A = 0, all other registers not affected
DelayN    bsr       Delay1    ;go delay 1 millisecond
dbne      A,DelayN  ;dec delay 1 millisecond
rts

* Delay1: delay 1 millisecond via software,
* assumes F=24 MHz (and no interrupts)
* all registers preserved for caller
Delay1    pshx             ;2~  preserve registers used here
ldx       #7996  ;2~  iterations for 1ms.
dbne      X,*      ;3~  7996 loops * 3~/loop = 23988~
pulx      ;3~  recover used registers
rts      ;5~  23988 + 12 = 24000~ = 1ms.

Lab 8: Motor Control - Page 6
* string output routine, X points to null-terminated string

PUTS    ldab      0,x       ;get 1 char from string
beq       PSDONE    ;done if terminating null
jsr       PUTCHAR   ;output this char
inx                 ;advance X to next char
bra       PUTS      ;continue
PSDONE    rts                 ;done, return to caller

* single character (in B) output routine

PUTCHAR   pshx                ;save caller's X
ldx       VPUTCHAR  ;fetch address of monitor rtn
jsr       0,X       ;call monitor rtn.
pulx                ;restore caller's X
rts

* Constant data

Msg1      FCC       "Change potentiometer on PAD06 to vary PWM on PB0."
FCB       CR,0
***************************************************************************
END
Listing 3: Stepper motor demo program

* Stepper1.asm: demonstrate control of a stepper motor
* Control a stepper motor on PB0-PB3 according to a "stepper control" pot
* on PAD06. The pot controls both direction and speed!

***************************************************************************
*         <<< EQUATE SECTION >>>
; constants
CR EQU $0D ;ASCII Carriage Return
LF EQU $0A ;ASCII Line Feed
MINDELAY EQU 3 ;minimum output step delay

; memory map definitions
DATA EQU $1000 ;where our data can begin on EVB
CODE EQU $2000 ;where our code can begin on EVB

; "system" objects (I/O registers, monitor routines, etc.)
PORTB EQU $0001 ;I/O port B
DDRB EQU $0003 ;DDR for port B
ATD0CTL2 EQU $0082 ;ATD Control Reg 2 (ATDPU)
ATD0CTL4 EQU $0084 ;ATD Control Reg 4
ATD0CTL5 EQU $0085 ;ATD Control Reg 5 (start conversion)
ATD0STAT0 EQU $0086 ;ATD Status Reg 0 (SCF)
ATD0DR0 EQU $0090 ;ATD Result Reg 0
SCI0SR1 EQU $00CC ;SCI Status Register 1
DDRJ EQU $026A ;I/O port J DDR

; D-Bug12 utility routine vectors
VPUTCHAR EQU $EE86 ;send character in B to output device

***************************************************************************
*         <<< DATA SECTION >>>

; variable data allocations (RMB)...
ORG DATA
TblIndex RMB 1

***************************************************************************
*         <<< MAIN PROGRAM LOOP >>>
ORG CODE

START clr TblIndex
movb #$80,ATD0CTL2 ;turn on ATD0
movb #$85,ATD0CTL4 ;enable 8-bit mode
movb #$2,DDRJ ;enable PORTB LEDs
movb #$FF,DDRB ;make PB3-PB0 outputs
ldx #Msg1 ;point X to initial message
jsr PUTS ;output it

MainLp ldaa #$?? ;ADC channel number of interest
staa ATD0CTL5 ;start A/D conversion on PAD06
Loop1 tst ATD0STAT0 ;check CCF on bit 7
bpl Loop1 ;A/D not done yet
clra
ldab ATD0DR0 ;fetch result of A/D conversion
bpl IsPos
negb
IsPos addd #MINDELAY ;minimum step delay
jsr DelayN
stt ATD0DR0 ;determine rotation direction
bmi NextStep
incb ;change direction to CCW

NextStep cmpb #128 ;no output step if pot is centered
beq CKey
ldab TblIndex ;lookup next output step
jsr GetNextStep
stab TblIndex

CKey brclr SCI0SR1,$20,MainLp ;continue until RDRF
clr PORTB ;release all coil drivers

*** Subroutines ***

GetNextStep:
pshy ;preserve Y
tsta ;CW or CCW?
tsb $9F ;short jump to goCCW

getStep: cmpb #StepTblLen ;past end of table?
bne goCW

psly ;preserve Y
tsta ;CW or CCW?

*** Subroutines ***

GetNextStep:
pshy ;preserve Y
tsta ;CW or CCW?
tsb $9F ;short jump to goCCW

getStep: cmpb #StepTblLen ;past end of table?
bne goCW

psly ;preserve Y
tsta ;CW or CCW?

*** Subroutines ***

GetNextStep:
pshy ;preserve Y
tsta ;CW or CCW?
tsb $9F ;short jump to goCCW

getStep: cmpb #StepTblLen ;past end of table?
bne goCW

psly ;preserve Y
tsta ;CW or CCW?

*** Subroutines ***

GetNextStep:
pshy ;preserve Y
tsta ;CW or CCW?
tsb $9F ;short jump to goCCW

getStep: cmpb #StepTblLen ;past end of table?
bne goCW

psly ;preserve Y
tsta ;CW or CCW?

*** Subroutines ***

GetNextStep:
pshy ;preserve Y
tsta ;CW or CCW?
tsb $9F ;short jump to goCCW

getStep: cmpb #StepTblLen ;past end of table?
bne goCW

psly ;preserve Y
tsta ;CW or CCW?

*** Subroutines ***

GetNextStep:
pshy ;preserve Y
tsta ;CW or CCW?
tsb $9F ;short jump to goCCW

getStep: cmpb #StepTblLen ;past end of table?
bne goCW

psly ;preserve Y
tsta ;CW or CCW?
bra  getStep  ;continue

goCCW   tstb                  ;at first entry?
bne    goCCW2              ;if not
ldab   #StepTblLen         ;reset to end of table
goCCW2  decb                ;backup to previous index
getStep  ldy       #StepTbl   ;point Y to begin of table
ldaa      B,Y                 ;fetch step pattern
puly
rts       ;and return it

* DelayN: delay 'N' milliseconds
* in:  D = 'N', number of ms to delay
* out: D = 0, all other registers not affected
DelayN   bsr       Delay1    ;go delay 1 millisecond
dbe     D,DelayN  ;dec 'N' & continue till zero
rts

* Delay1: delay 1 millisecond via software,
* assumes F=24 MHz (and no interrupts)
* all registers preserved for caller
Delay1    pshx             ;2~  preserve registers used here
ldx       #7996  ;2~  iterations for 1ms.
dbe     X,*                ;3~  7996 loops * 3~/loop = 23988~
pulx             ;3~  recover used registers
rts       ;5~  23988 + 12 = 24000~ = 1ms.

* string output routine, X points to null-terminated string
PUTS     ldab      0,x       ;get 1 char from string
beq       PSDONE    ;done if terminating null
jsr       PUTCHAR   ;output this char
inx                 ;advance X to next char
bra       PUTS      ;continue
PSDONE    rts                 ;done, return to caller

* single character (in B) output routine
PUTCHAR   pshx                ;save caller's X
ldx       VFPUTCHAR  ;fetch address of monitor rtn
jsr     0,X       ;call monitor rtn.
pulx             ;restore caller's X
rts

* Constant data
Msg1      FCC       "Dial potentiometer on PAD06 for speed, press any key to exit"
FCB     CR,0

* Define motor step pattern table
StepTbl1:
;StepTbl1  FCB       8,4,2,1
;StepTbl2  FCB       $C,6,3,9
;StepTbl3  FCB       8,$C,4,6,2,3,1,9
StepTblLen EQU      *-StepTbl
***************************************************************************
END