Sample Chapter

Building Simulated Aircraft Instruments

John Michael Powell

Mike's Flight Deck Books
Lafayette, CA 94549-0778

Sample Chapter

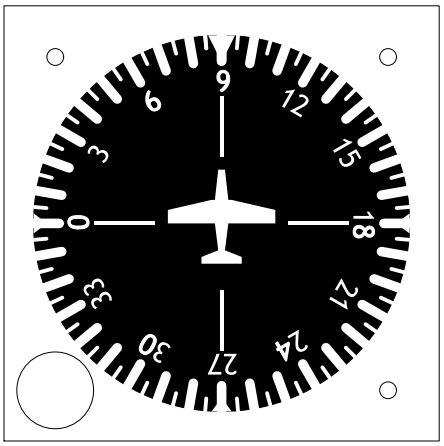
Published by Mike's Flight Deck Books™
www.mikesflightdeckbooks.com
Printed in the United States of America.

The projects described in this book are intended for hobby purposes. Neither the designs nor the suggested construction methods are appropriate for applications in which failure could result in harm to people or damage to property.

These projects have no place in real aircraft.

Table of Contents

Stepping Motor Gyro Compass



This project makes use of two strengths of using a stepping motor in a simulated instrument or gauge. There is substantial torque to work with and the motor can continually rotate without hitting a mechanical stop. This is a good match for the relatively large inertia of the compass rose disk, and the need to let it turn freely if you decide to fly in circles.

Of course this approach will work just as well with a low inertia pointer should you want to make an air speed indicator instead.

In addition to demonstrating the stepping motor in a simulated instrument, this project introduces the optical interrupter. It is used here to sense the zero position of the compass card during power up initialization and execution of the reset command. These will be valuable additions to your instrument simulation toolbox.

Mechanical Overview

The side view in figure 1 shows the relative positions of the major components. The stepping motor is centered immediately behind the faceplate assembly. The compass rose mounts directly on the motor shaft. The rotary encoder mounts in a lower corner of the motor deck and is positioned such that its extended shaft comes through the front bezel where the lower left instrument mounting screw would otherwise be. Immediately above the motor is an optical interrupter. It mounts on the front surface of the deck with its leads pointed toward the circuit deck. The circuit deck is just behind the motor deck and is followed in turn by the rear deck.

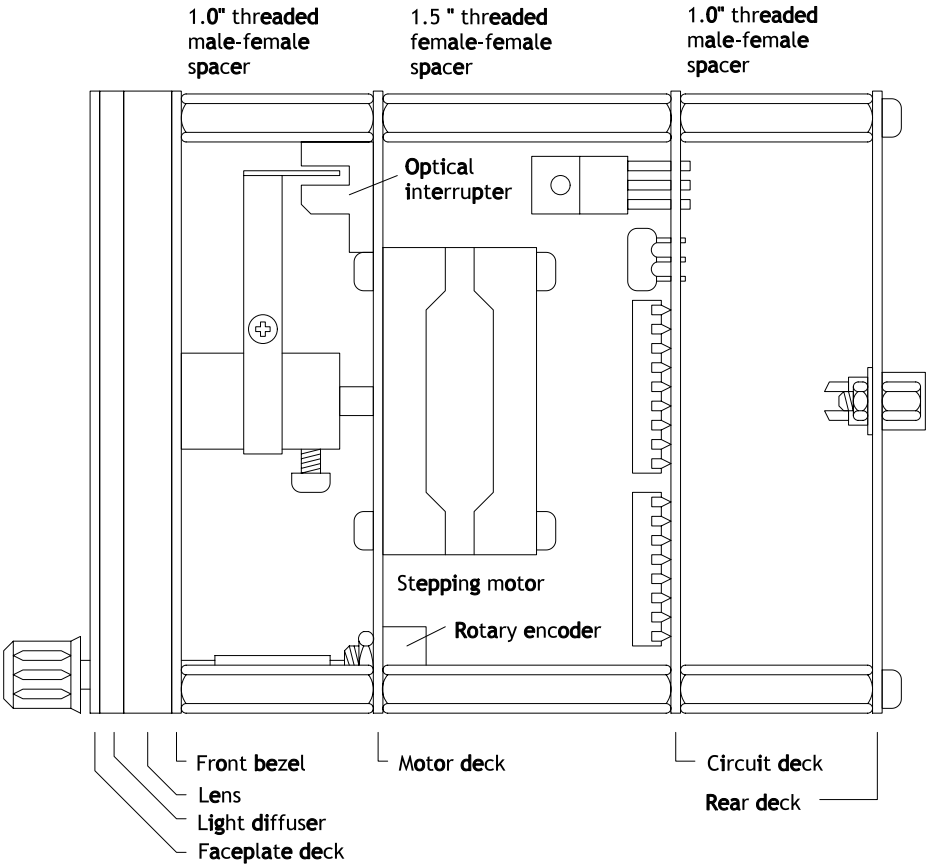


Figure 1. A side view of the simulated directional gyroscope.

The stepping motor has no unique starting position. The motor shaft is in an unknown rotational position when powered up. The flag fastened to the compass rose support

provides the means for the firmware to rotate the compass rose to a known position upon initialization.

Mechanical Construction

The project's mechanical construction is reasonably straightforward. The only metal bending is forming the interrupter flag. Most of the effort goes into cutting flat metal pieces and drilling holes. Tolerances are not stringent. Nonetheless, there are some mounting holes that should line up during final assembly. Good layout technique and the use of a drilling template are recommended to help make this come about. Details of these suggestions are covered in the appendix on precision layout.

Faceplate Assembly

The front bezel, lens, light diffuser and faceplate make up the faceplate assembly. It incorporates internal lighting based on either LEDs or incandescent lamps. Lighting options are covered in the air-core airspeed project chapter, and you are referred there for those particular details.

This assembly has only a few unique aspects relative to the other projects. The most visible difference is the white heading reference lines painted on the inside of the lens. This can be as simple as a single vertical white line bisecting the lens. Alternatively it can be a complex silhouette of an aircraft with the nose pointing to the heading on the compass rose.

The lens markings can be added using a permanent ink marking pen. A white ink pen such as the Sakura Pen Touch™ or equivalent works well. These pens are generally available from arts and crafts stores.

A template is very useful. Transparency material used for overhead projectors is a good material for the template. The edges of the holes act as pen guides. Placing several layers of masking tape on the lower side of the template next to the cuts will hold the template above the lens and help prevent the ink from wicking between the template and lens.

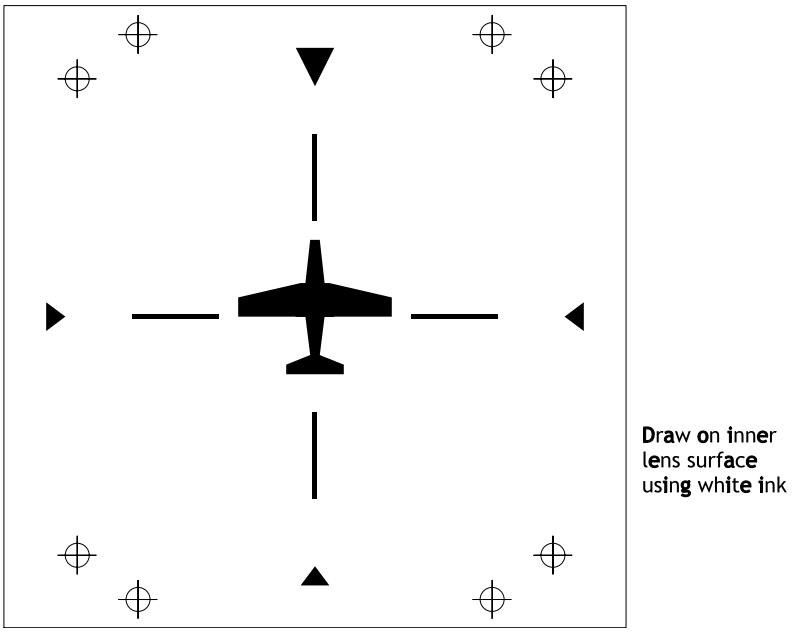


Figure 2. Sample graphic for lens.

Unlike those in other projects, this faceplate deck has no markings on it. It simply has a hole large enough to clear the compass rose support. It's the compass rose that carries the markings.

Compass Rose

The compass rose is an aluminum disk 2.7" in diameter. It's supported on the motor shaft by a short length of ½" diameter aluminum rod. The support is glued with epoxy to the center of the disk. The easiest way to glue these parts together is to assemble the rest of the instrument first. It can then be used to hold the disk and support in proper alignment while the epoxy cures. This approach has the further advantage that it will compensate for small dimensional errors made while fabricating these two parts.

The compass rose graphic is, at least in basic form, simply white radial markings and lettering on a black background. You'll get the nicest results if you generate it in a drawing program and print out an original. If you don't have access to a drawing program you can make a copy of the following figure.

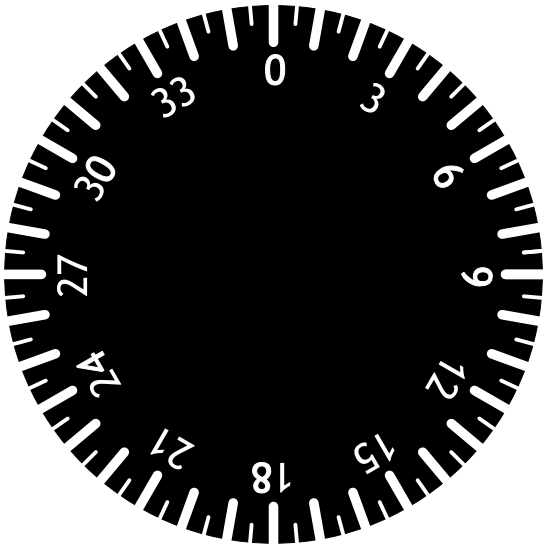


Figure 3. Compass rose graphic.

The graphic can be mounted on the disk with double sided tape. Alternatively, if you plan on printing an original, you might consider printing on self adhesive paper. This is available from many office supply stores. It's the same material labels are made from. It's simply not scored into individual labels.

Interrupter Flag

The interrupter flag is an L-shaped piece of thin sheet metal that wraps around the rod supporting the compass rose. The flag is held in place by a machine screw and nut. The outer end of the flag is bent such that as the compass rose rotates, the flag passes through the optical interrupter gap. The flag is nominally made of 0.020" aluminum, but can be any sufficiently rigid opaque material. A strip cut from a discarded food can would work quite well, and has a decided price advantage over purchased material.

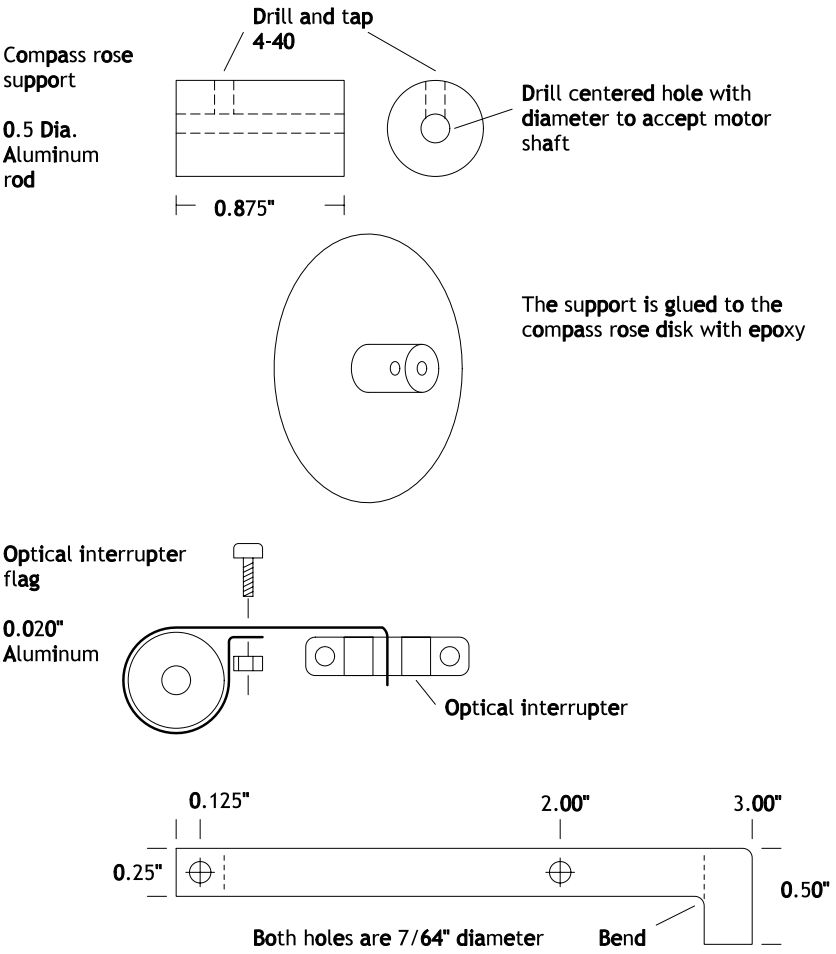


Figure 4. Compass rose and interrupter flag assembly.

Motor Deck

The motor deck is the support for the rotary encoder, the optical interrupter, and the stepping motor. The holes for mounting the motor are positioned for a standard size motor that was commonly used in older floppy and some hard drives. The motor is 1.7 inches in square cross section and roughly ¾ of an inch long. This size turns up on the surplus market frequently as well. If you are uncertain that the motor you plan on using is this size, you might want to measure the location of the mounting holes on your motor before you fabricate the motor deck.

The motor mounts on the rear surface of the deck with the motor shaft pointing forward. There is no particular up or down orientation of the motor. Choose an orientation that positions the motor leads to your preference.

The rotary encoder mounts in a ¼" hole in the lower left corner of the deck. The encoder body has an indexing pin for aligning the encoder and preventing it from turning after it is installed. After drilling the mounting hole in the deck, you will need to cut a small notch as shown below to accept the index pin. The notch should be made 45 degrees from vertical to position the encoder body so that it will clear the nearby threaded spacer.

The shaft of the rotary encoder must be extended to reach through the front bezel assembly. This is done with a split coupling made of a short length of 9/64" ID brass hobby tubing. Make a lengthwise cut through the wall of the tubing so that it has a "C" cross section. Squeeze the tubing so it slides tightly over the encoder shaft. A short length of 1/8" diameter hobby brass rod can be pushed into the other end of the tubing to extend the shaft.

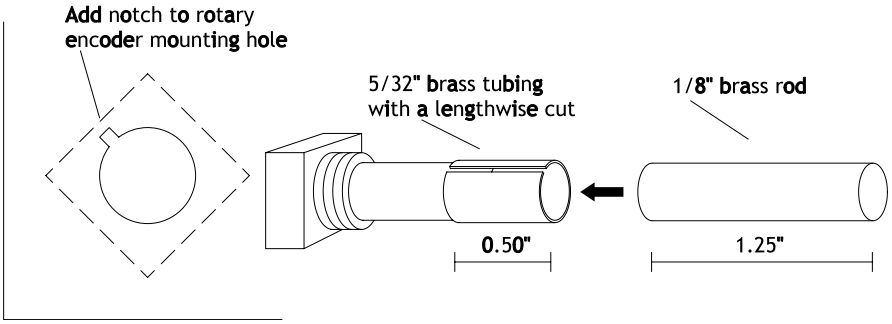


Figure 5. Rotary encoder mounting hole and shaft extension.

The optical interrupter mounts on the front surface of the motor deck just above the motor. The leads go through holes in the deck. The interrupter has two mounting ears, but only the top is used. Use a ½" 4-40 machine screw and nut.

Dimensioned Drawings

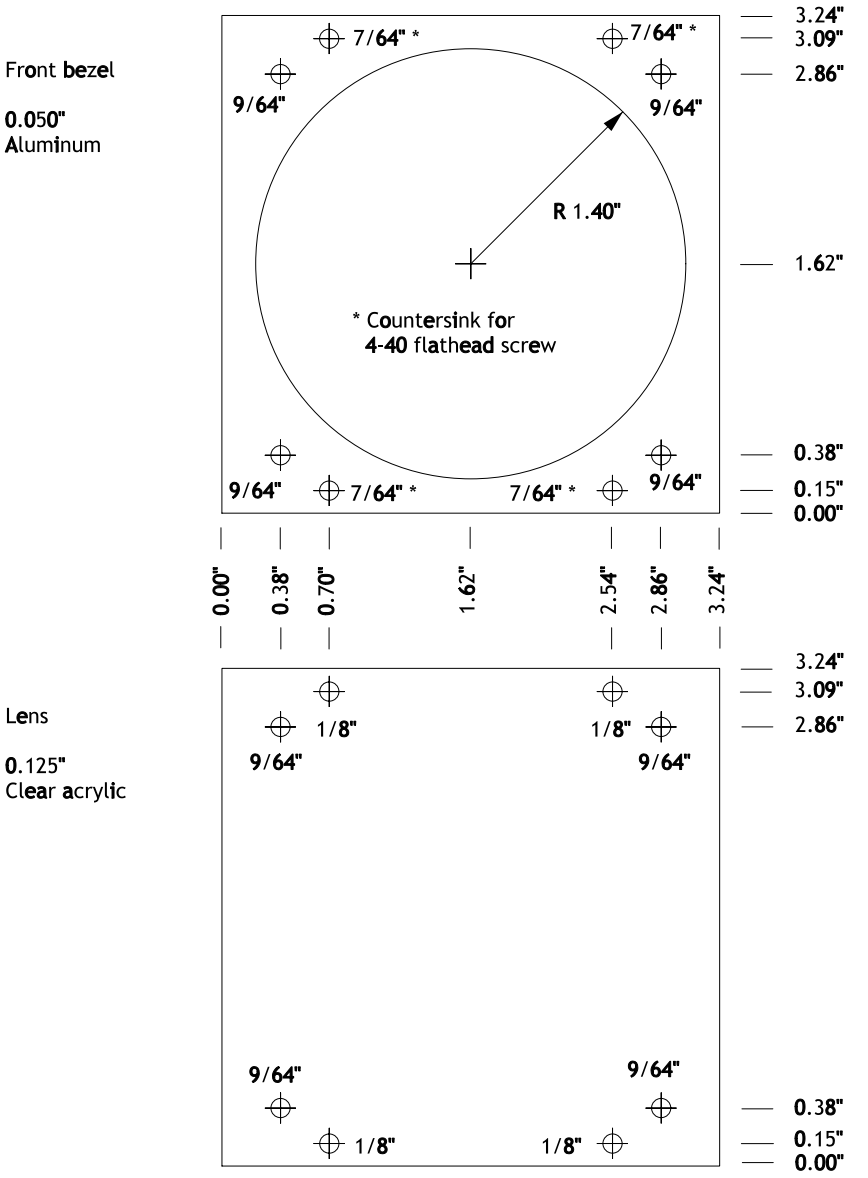


Figure 6. Front bezel and lens.

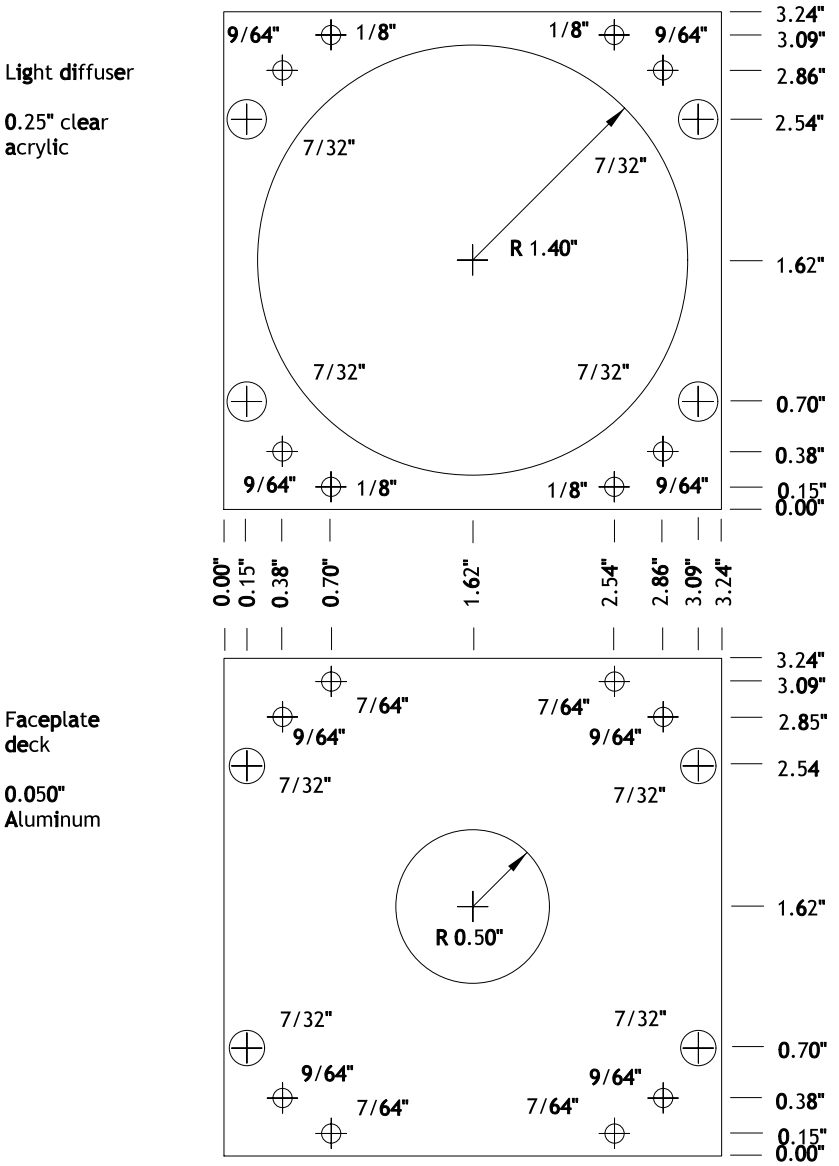


Figure 7. Light diffuser and faceplate deck.

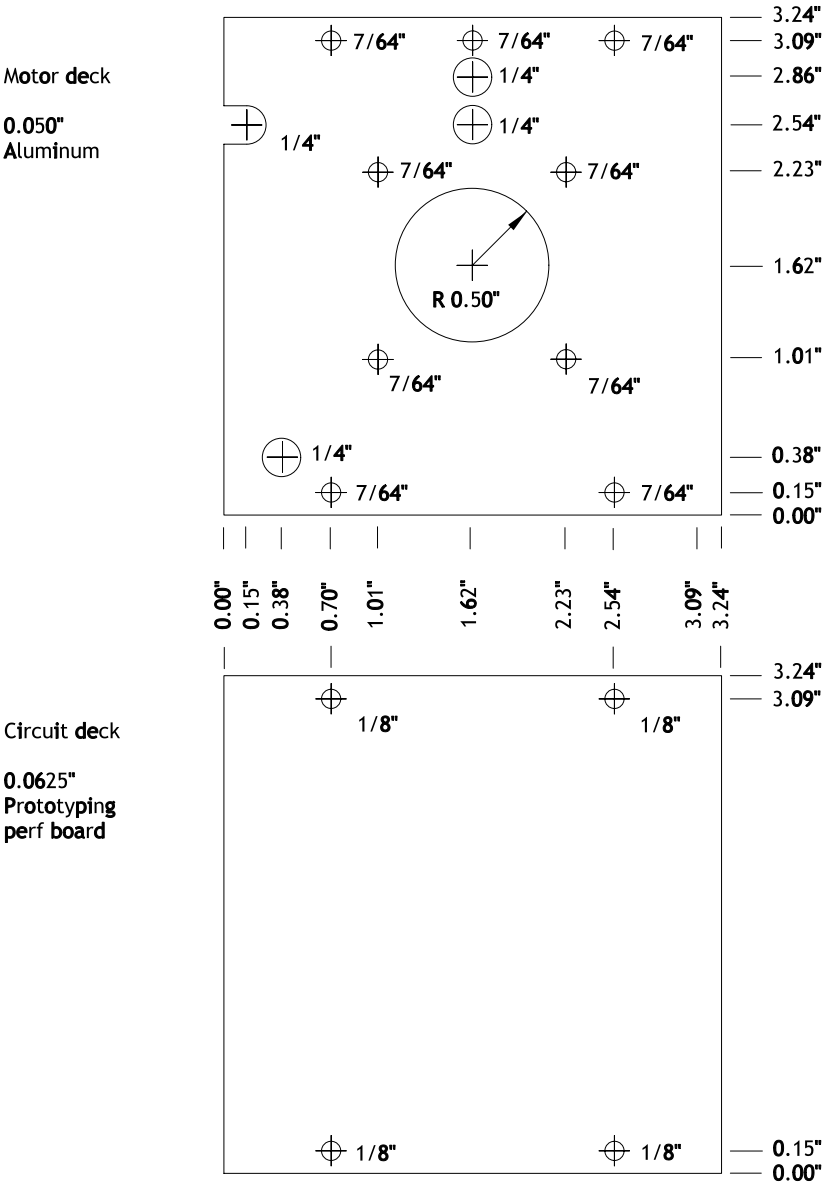


Figure 8. Motor deck and circuit deck.

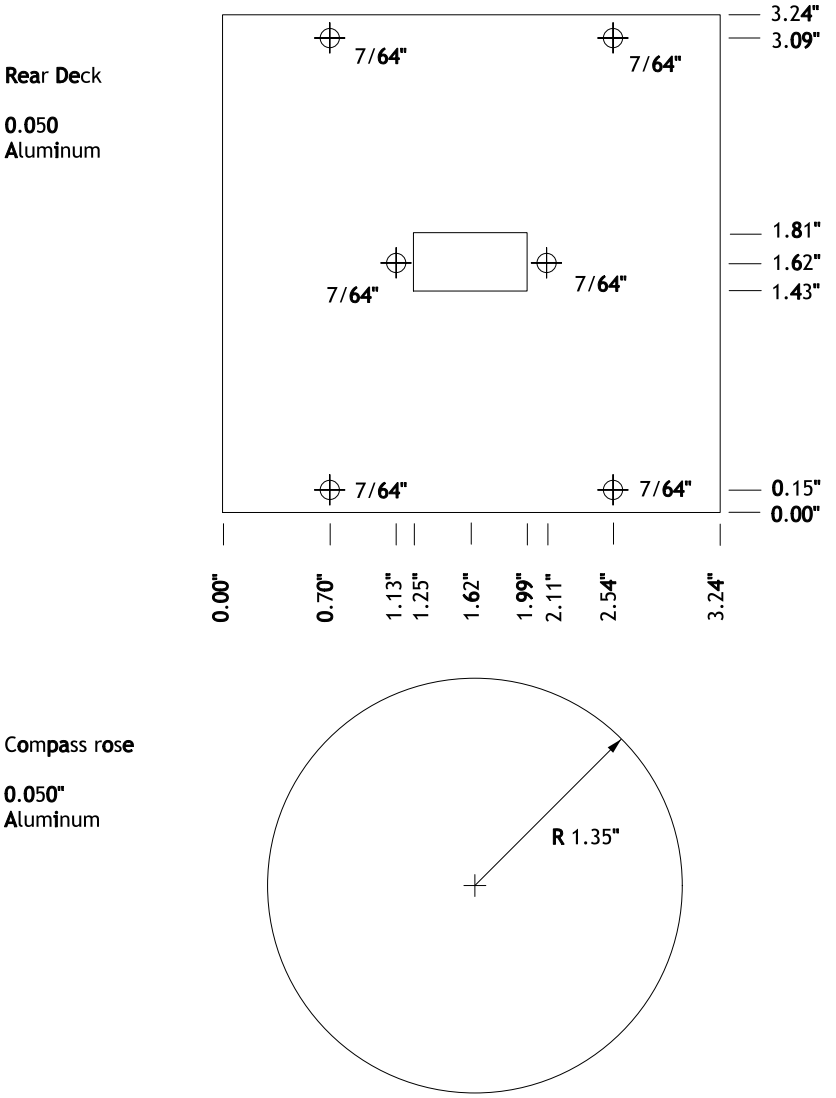


Figure 9. Rear deck and compass rose disk.

Electronics

The circuitry is very similar to that of the VSI project. The motor used here has bipolar windings and can be driven using the same H-bridge chip. The same types of micro controller (PIC16F628) and RS232 transceiver (MAX232) are used. The only differences are the additions of a rotary encoder and an optical interrupter. Because real

directional gyros drift, they have an adjustment knob. The rotary encoder serves that function on this simulated DG. Rather than making a mechanical adjustment however, the rotary encoder interfaces to the micro controller. The micro controller, in turn, directs the stepping motor to make the adjustment.

Stepping motors power up at random rotational positions. They have no inherent means of moving to a known "zero" position. An optical interrupter adds that capability. An optical interrupter is a combination of an LED and a phototransistor. With nothing blocking the light path, the light from the LED biases the phototransistor on. If something blocks the path, the phototransistor turns off. In this project, an arm clamped on the motor shaft moves a flag that can block the path. This allows firmware to sense when the shaft has rotated to its "zero" position.

When a real DG spins up it displays a random heading, so it may seem that a simulated unit would not have the ability to self-initialize. Quite likely, the random nature of the DG is handled within the simulation software on the host computer. The host computer then commands the simulated instrument to display a particular heading. If the simulated instrument cannot self-initialize, it will be out of sync with the software.

Furthermore, the self-initialing capability allows this instrument simulation approach to be used for a variety of instrument and gauge types. By replacing the compass rose disk with a pointer, this approach can provide you with a VSI, a tachometer, airspeed indicator or just about any single pointer type unit.

The voltage regulator that supplies power to the motor you use may be need to be different than the 7805 shown in the schematic. Depending on the actual motor you use, you may need to select a regulator with a different voltage rating. You don't have to supply the full voltage the motor is rated for as it's not necessary for the motor to develop full rated torque. It's only spinning an aluminum disk. Use the lowest voltage that gives reliable operation. This will minimize current demands from your power supply.

Construction Technique for Electronics

This is a fairly benign circuit as far as construction demands go. Follow a few basic guidelines and it should work fine. Printed circuit boards look very nice but are not a necessity. Perf board works fine. Generally, try to position parts to minimize the wiring lengths. Give precedence to parts that generate heat like voltage regulators and driver chips. Place those on the top edge of the board so the waste heat doesn't heat the other components. Run the power and ground connections first. The power bypass capacitors work best when placed close to the power pins of the chips they bypass.

The one area that is most likely to cause you grief is the voltage regulator for the motor. Depending on your motor selection, the regulator may heat up. Heat sink this regulator. Bolting it to the rear deck is one possibility.

If building electronics is new to you, there are a few references in the information resources appendix that may be of interest.

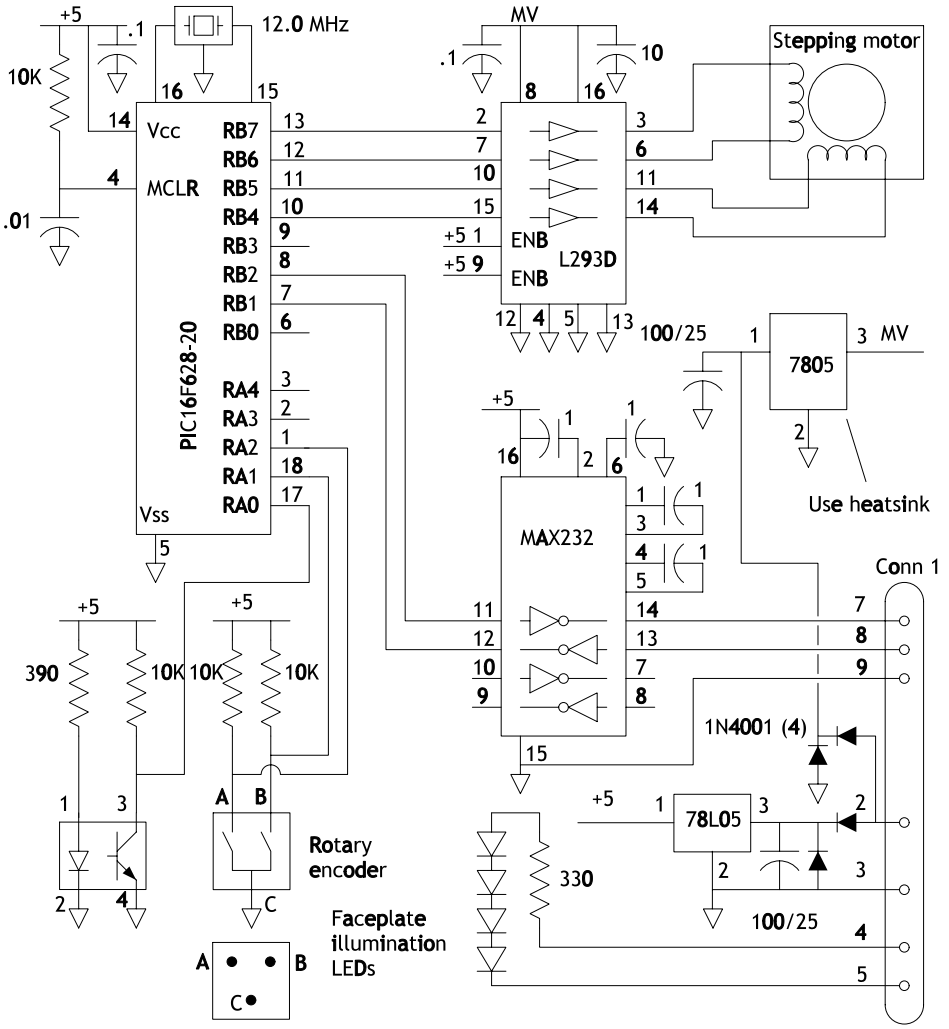


Figure 10. Stepping motor based directional gyroscope schematic.

Firmware

Overview

The firmware's basic functionality is to put a stepping motor under the control of a host computer, to communicate with the host through a standard com port, and to provide a manually adjustable offset to the host commanded motor position. Additionally, it provides the ability to rotate the motor to a known "zero" position during initial power up or when the host issues a reset command.

The firmware is based upon the same framework covered in detail in the VSI chapter. The PIC spends most of its time in a tight polling loop checking flags used to manage communication with the host. A command from the host to rotate the compass disk does not directly move the stepping motor. Instead, the command loads a two byte variable representing the desired motor position. Motor movement is handled by the interrupt service code.

Very little is accomplished by a single pass through the interrupt service code. When initially powered up, each pass will result in a motor half-step toward the compass card zero position. During normal operation, each pass checks for changes in the rotary encoder state, and will move the motor a half-step if a compass card movement command is in process. Large effects accrue from many passes through this code.

One of the PIC's internal timers generates a repeating interrupt that causes the periodic execution of the interrupt service code. It is the interval between interrupts that establishes the stepping speed of the motor and the response time of the rotary encoder.

What is actually a fairly simple approach is somewhat obscured by the volume of assembly code that implements it. The motor shaft can be rotated to any of 800 possible locations, requiring two bytes each to track the current and desired positions. The need to do 16 bit arithmetic on an eight bit micro controller further obscures the approach. Just keep in mind that the underlying approach is a simple one, and don't loose the forest for the trees.

Compass Card Reset

When the PIC is first powered up it executes initialization code that, among other things, sets the flag "fSeekZero". When the interrupt service code is run, fSeekZero indicates that the first order of business is to rotate the compass card to its zero position. Until this is done, the code for normal operation is skipped.

The zero position of the compass card is sensed by an optical interrupter. The firmware will half-step the motor until the interrupter flag just enters the gap. If the interrupter is already in the gap, the motor is first stepped the other direction until the interrupter is

moved out of the gap. If this were not done, positioning could be off by up to the width of the interrupter flag.

Once the compass card has been initially positioned at zero, fSeekZero is cleared, opening the path to normal operation.

This flowchart illustrates the overall process rather than details of the assembly language.

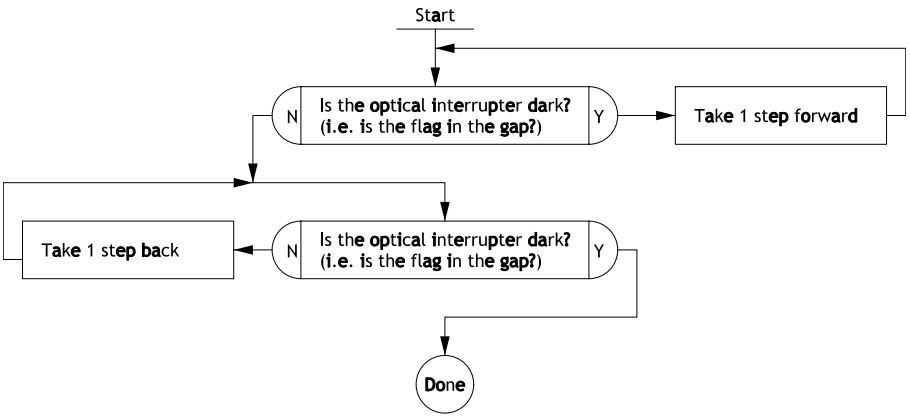


Figure 11. Compass card reset flowchart.

Rotary Encoder Support

The rotary encoder, along with the bulk of the firmware used to read, debounce and decode it, is identical to what is used (and covered in great detail) in the VOR/GS CDI chapter.

Basically, each pass through the interrupt service code checks the encoder outputs for a change. When a change is detected, several more passes through the code are allowed to go by before taking a final reading on the encoder's outputs. This allows the outputs to fully settle and avoids taking action on what might be electrical noise or mechanically bouncing switch contacts.

The encoder change is decoded by combining the new output state with the just previous output state to create a four bit "state transition vector". This vector can reflect all possible changes in the encoder's outputs including those that should not normally occur. The vector is added to the PIC's program counter, effectively resulting in a 16-way branch. The sixteen endpoints of this branch direct firmware execution to the appropriate code. Essentially the decoding function is "hardwired" into the branch structure.

The firmware times the interval since the last encoder change. The interval will be short if the encoder shaft is being turned rapidly. Rapid turning is indicative of the user's desire to quickly change the orientation of the compass card. To make the adjustment function more responsive, rapidly turning the encoder results in disproportionately larger step changes to the compass card's position. Slowly tuning the encoder results in small steps for accurate positioning.

The firmware uses two variables to track both the current and desired positions of the compass card. When the rotary encoder is used to move the compass card, the contents of POSITION, the variable tracking the current position, is increased or decreased. The firmware that actually manages motor movement will note the change and shift the compass card position.

Because motor movement commands from the host write to NEWPOS, the variable holding the desired position, a host sent movement command does not over-write or negate the effects of manual compass card adjustments. Only a reset command will do so.

Motor Movement

This project makes use of a 400 step per revolution stepping motor. The motor is half stepped to boost the number of steps to 800. The motor has bipolar windings and so is driven with an H-bridge. The micro controller supplies the four digital signals necessary to control the H-bridge.

During each pass through the interrupt service routine, the difference between the desired motor position and the current position is calculated. A difference of zero indicates no need for movement and the code executes a return from interrupt.

A non-zero difference is further examined to determine the best direction of movement. This is not simply a case of looking at the sign of the difference. The goal is to move in the direction that minimizes the required number of steps. A movement from 0 to 1 will go in one direction while a movement from 0 to 799 should clearly go the other way even though both are positive differences.

A tentative direction is set by looking at the sign of the difference, with a positive difference implying forward movement. The magnitude of the difference is then compared to the number of half-steps equivalent to half a rotation. If the magnitude is larger, it's shorter to go the other way, and the direction flag is complemented.

With the direction determined, the current motor step variable, MOTORSTEP, is incremented or decremented as appropriate. The lower three bits are extracted and used as an index into a table containing the values needed to generate the next step. This value is written to PORTB, which is connected to the motor H-bridge power driver.

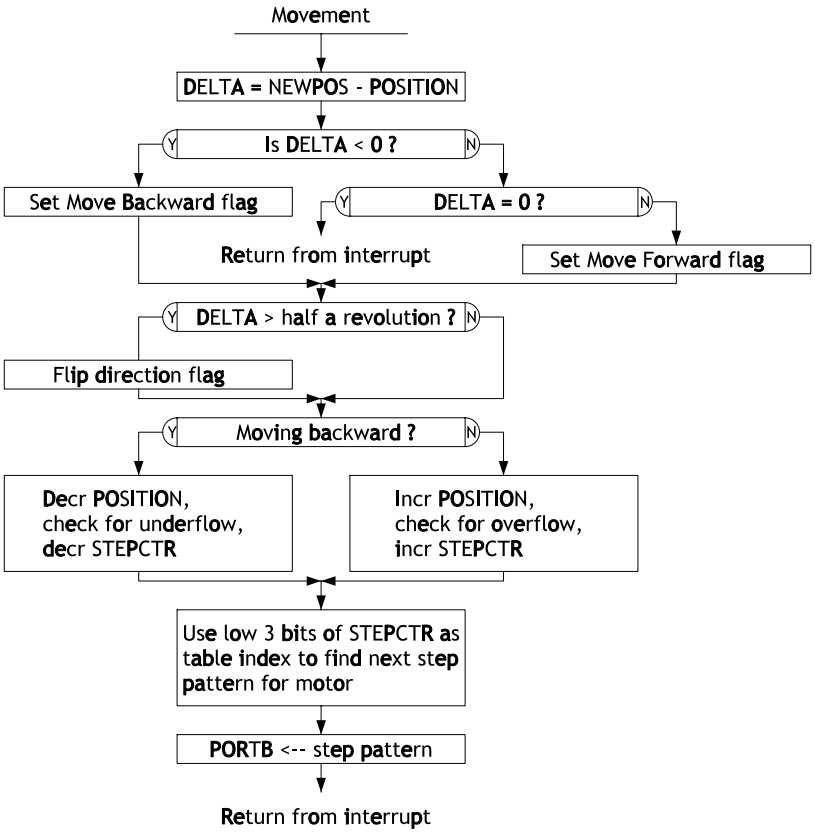


Figure 12. How the motor is stepped.

Command Summary

Reset

Bytes [0:4]	Byte 5
ID	0

Reset forces the firmware to jump to the power on reset code. It will initialize all variables and the timers, and enable interrupts. The stepping motor will rotate to its zero position.

Set Motor Position

Bytes [0:4]	Byte 5	Bytes [6:7]
ID	1	Motor position

Set motor position loads the internal variable commanding the relative motor position. The physical position the motor will rotate to is the sum of the relative motor position

and the offset set by turning the rotary encoder. After a reset command or a power on reset, the offset is zero. Motor position is a two-byte integer ranging from 0 to 799. Byte 6 is the low byte.

Set Motor Position (frame transfer)

Bytes [0:4]	Byte 5	N Bytes	Bytes [N+6:N+7]
ID	2	X	Motor position

Set motor position (frame transfer) has the same effect as the set motor position command. The difference is in the data transfer. A frame transfer sends a vector of data to a group of instruments sharing the same broadcast ID. This is a more efficient use of I/O bandwidth as only a single ID-Command header is used for the group. Each instrument picks its data out of the vector from a unique slot. See command set frame slot.

Query Name

Bytes [0:4]	Byte 5
ID	8

Query name causes the instrument to transmit the 17 bytes from the nonvolatile EEPROM memory containing the instrument's unique ID and the firmware version.

Rename

Bytes [0:4]	Byte 5	Bytes [6:10]
ID	9	New unique ID

Rename writes the five-byte unique ID field stored in nonvolatile EEPROM. It also places a copy in the working registers so the instrument will immediately respond to the new ID.

Set Frame Slot

Bytes [0:4]	Byte 5	Byte 6
ID	10	Slot

Set frame slot writes the value used to index the instrument's data position in the frame data transfer. This value is stored in nonvolatile EEPROM memory and the index stored in working register is initialized on power-on reset and on the reset command.

Firmware Listing

```
title  "2 Phase driver for stepping motor based gauge/instrument"
;
;  This application generates 4 drive signals for an L293D
;      power driver which in turn provides bipolar drive
;                for both phases of a bipolar stepping motor
;                RB7 - Phase A high
;                RB6 - Phase A low
```

```
;               RB5 - Phase B high
;               RB4 - Phase B low
;               RB3 - NC
;               RB2 - RS232C data out
;               RB1 - RS232C data in
;               RB0 - NC
;               RA3:7 - NC
;               RA1:2 - Rotary encoder inputs
;               RA0 - Optical interrupter input
;
;  Hardware Notes:
;   This application runs on a 16F628 executing at 12 MHz
;   _MCLR is tied through a 10K Resistor to Vcc
;
;  Mike Powell February 25, 2004
;
  LIST R=DEC
 ifdef __16F628
  INCLUDE "p16f628.inc"
 endif

;  Registers
 CBLOCK 0x020
Holder0             ; Placeholder for 8 bytes that are used
Holder1             ; in other projects. Having placeholders
Holder2             ; keeps the READ NAME data format the same
Holder3             ; across all the projects. Makes it easier
Holder4             ; for lazy programmers like me to write
Holder5             ; host interface code for these projects
Holder6             ;
Holder7             ;
SkipCt              ; # bytes to skip before picking data
Name2Byte0
Name2Byte1          ; Unique instrument ID is stored
Name2Byte2          ; here
Name2Byte3
Name2Byte4
Name2Byte5
 ENDC

 CBLOCK 0x030
NameBasePtr         ; Points to start of ID1 field
NBoffset            ; Offset to name element with ID1 field
NameByte0
NameByte1           ; Shared Instrument ID is stored here
NameByte2
NameByte3
NameByte4
NameByte5
Name2BasePtr        ; Points to start of ID2 field
NB2offset           ; Offset to name element with ID2 field
```

```
fCMD            ; flag indicating command being processed
fDATA           ; flag indicating serial comm moving data
RXBUF           ; Received data buffer
DAToffset
Count           ; Used to count bytes transferred
 ENDC

 CBLOCK 0x040
BasePtr         ; points to start of data buffer
Data0           ; Buffer area used to collect command
Data1           ; data before executing command
Data2
Data3
Data4
Data5
Data6
Data7
Data8
Data9
DataA
DataB
DataCtr         ; byte counter for data xfers
 ENDC

 CBLOCK 0x050
Pos:2           ; Tracks current position of motor
NewPos:2        ; New position motor is to move toward
Delta:2         ; Stores diff between current and new positions
HalfWay:2       ; Constant # steps = 1 half motor revolution
MaxPos:2        ; Constant # steps = 1 full motor revolution -1
RegPtr          ; Pointer into register (data) memory
EEPtr           ; Pointer into EEPROM memory
fQuery          ; Flag indicating host query in progress
QueryCtr        ; Counter used in servicing host query
SkipCtr         ; number of data bytes skipped in frame xfer
Temp            ; Who knows? it's a temp value!
FMoveBack               ; flag indicating desired motor step direction
fSeekzero       ; flag to force motor position reset
fDebounce       ; flag indicating RE contact debounce in process
DebounceCt      ; counter tracking debounce process progress
MotorStep       ; index into the motor step pattern
REState         ; state variables used in decoding rotary encoder
REStateTemp     ;   state changes
RETime          ;

 ENDC

 ifdef __16F628
 __CONFIG _CP_OFF & _WDT_OFF & _XT_OSC & _PWRTE_ON & _LVP_OFF
 else
 __CONFIG _CP_OFF & _DEBUG_OFF & _XT_OSC & _PWRTE_ON  & _WDT_OFF &
```

```
_LVP_OFF & _WRT_ENABLE_ON & _BODEN_ON & _CPD_OFF
 endif

  PAGE
;  Mainline of  GYRO COMPASS SIM

 org 0
  nop

  goto INITIALIZE

 org 4                          ; ****INTERRUPT ENTRY POINT******

  bcf       PIR1, TMR2IF        ; clear the timer2 interrupt flag
  movf      fSeekzero, w        ;
  btfsc     STATUS, Z           ; position reset in progress?
  goto      OPERATE             ; No: proceed with normal operations

  btfss     fSeekzero, 1        ; in first portion of position reset?
  goto      FINDDARK            ; No: proceed to second portion
  btfsc     PORTA, 0            ; Yes: are position sensors dark? 1 = dark
  goto      INCRSTEP            ; Yes: move forward a step
  bcf       fSeekzero, 1        ; No: done with first portion of reset
  retfie

FINDDARK
  btfss     PORTA, 0            ; are position sensors dark? 1 = dark
  goto      DECRSTEP            ; No: back up another step
  clrf      Pos                 ; Yes: clear POS and flag, ready to go now
  clrf      Pos + 1
  clrf      fSeekzero           ; done with position reset

OPERATE
            ; This segment will look at the rotary encoder bits
            ; and look for a change.  if not a counter will be decremented
            ; to time the interval between state changes.  A slow
            ; rate will result in a fine adjustment movement.  A Fast
            ; rate will result in a larger adjustment step.  A detected
            ; change in state will start a debounce interval before
            ; sampling the new state.  Once debounced the current and new
            ; state will be combined to interpret move forward, move
            ; back or illegal state transition and incr or decr the offset

  btfsc     fDebounce, 0
  goto      DEBOUNCE
  movf      PORTA, w            ; read the two bits of the rotary encoder
  andlw     0x06                ; mask out anything else on the port
  subwf     REState, w          ; compare it to the last recorded state
  btfss     STATUS, Z           ; look at zero flag for equality comparison
  goto      RECHANGE
  decfsz    RETime, f           ; count down RETime to see if slow rotations
```

```
        goto    MOVEMENT
        bsf     fDebounce, 1        ; no change for RETime, so set slow flag
        goto    MOVEMENT
RECHANGE
        bsf     fDebounce, 0        ; There's a change so initiate contact debounce
        movlw   0x03
        movwf   DebounceCt          ; Initialize debounce counter
        movlw   15
        movwf   RETime              ; Initialize "slow" timer
        goto    MOVEMENT

DEBOUNCE
        decfsz  DebounceCt, f
        goto    MOVEMENT            ; Debounce counter not zero yet
        rrf     PORTA, w            ; Read the rotary encoder bits, shift down
        andlw   0x03                ; and mask everything else out
        movwf   REStateTemp         ; and stuff into a temp register
        rlf     REState, w
        andlw   0x0c
        addwf   REStateTemp, w      ; combine stored and new rotary encoder states
        andlw   0x0f                ; make sure no other bits are on
        movwf   Temp
        movlw   HIGH RETABLE
        movwf   PCLATH
        movf    Temp, w
        addwf   PCL, f
RETABLE
        goto    NOCHANGE            ; State <00><00>
        goto    ADJMINUS            ; State <00><01>
        goto    ADJPLUS             ; State <00><10>
        goto    NOCHANGE            ; State <00><11> illegal transition
        goto    ADJPLUS             ; State <01><00>
        goto    NOCHANGE            ; State <01><01>
        goto    NOCHANGE            ; State <01><10> illegal transition
        goto    ADJMINUS            ; State <01><11>
        goto    ADJMINUS            ; State <10><00>
        goto    NOCHANGE            ; State <10><01> illegal transition
        goto    NOCHANGE            ; State <10><10>
        goto    ADJPLUS             ; State <10><11>
        goto    NOCHANGE            ; State <11><00> illegal transition
        goto    ADJPLUS             ; State <11><01>
        goto    ADJMINUS            ; State <11><10>
        goto    NOCHANGE            ; State <11><11>

NOCHANGE
        bcf     fDebounce, 0        ; clear the debounce flag
        movf    PORTA, w            ; Read and load current rotary encoder state
        andlw   0x06
        movwf   REState
        goto    MOVEMENT
```

```
ADJPLUS
  movf    PORTA, w        ; Read and load current rotary encoder state
  andlw   0x06
  movwf   REState
  btfss   fDebounce, 1    ; is this a fast movement?
  goto    FASTPLUS
  movlw   1
  goto    PLUS
FASTPLUS
  movlw   8
PLUS
  clrf    fDebounce
  addwf   Pos, f
  btfss   STATUS, C       ; 16 bit arithmetic here
  goto    CHECKPADJ
  movlw   1
  addwf   Pos + 1, f
CHECKPADJ
  movf    MaxPos, w       ; check if this change has pushed Pos
  subwf   Pos, w          ; beyond the maximum.  if so replace Pos

  movwf   Delta           ; with Pos - MaxPos (both bytes)
  btfss   STATUS, C
  goto    BORROW3
  movf    MaxPos + 1, w
  subwf   Pos + 1, w
  movwf   Delta + 1
  btfsc   STATUS, C
  goto    TOOBIG3         ; Result postive, so Pos is too big
  goto    MOVEMENT
BORROW3
  movlw   1
  addwf   MaxPos + 1, w
  subwf   Pos + 1, w
  movwf   Delta + 1
  btfsc   STATUS, C
  goto    TOOBIG3
  goto    MOVEMENT
TOOBIG3                   ; replace pos with pos - maxpos
  movf    Delta, w
  movwf   Pos
  movf    Delta + 1, w
  Movwf   Pos+ 1
  goto    MOVEMENT

ADJMINUS
  movf    PORTA, w        ; Read and load current rotary encoder state
  andlw   0x06
  movwf   REState
  btfss   fDebounce, 1    ; is this a fast movement?
  goto    FASTMINUS
```

```
  movlw   1
  goto    MINUS
FASTMINUS
  movlw   8
MINUS
  clrf    fDebounce
  subwf   Pos, f
  btfsc   STATUS, C          ; 16 bit arithmetic here
  goto    MOVEMENT
  movlw   1
  subwf   Pos + 1, f
  btfsc   STATUS, C
  goto    MOVEMENT
  movf    MaxPos + 1, w      ; Adjustment unflowed, add MaxPos to bring
  addwf   Pos + 1, f         ; Pos back into range
  movf    MaxPos, w
  addwf   Pos, f
  btfss   STATUS, C
  goto    ADDONE
  movlw   1
  addwf   Pos + 1
ADDONE
  movlw   1
  addwf   Pos, f
  btfss   STATUS, C
  goto    MOVEMENT
  addwf   Pos + 1, f
  goto    MOVEMENT

MOVEMENT
                  ; This segment determines the absolute difference between
                  ; current and new positions this is 16 bit arithmetic
                  ; being performed on an 8 bit machine

  bcf     fMoveBack, 0
  movf    Pos, w             ; first check the low bytes
  subwf   NewPos, w
  movwf   Delta
  btfss   STATUS, C          ; Carry set means positive result
  goto    BORROW1
  movf    Pos + 1, w
  subwf   NewPos + 1, w
  movwf   Delta + 1
  btfss   STATUS, C
  goto    ITSNEG
  goto    ITSPOS
BORROW1
  movlw   1
  addwf   Pos + 1, w
  subwf   NewPos + 1, w
  movwf   Delta + 1
```

```
  btfsc     STATUS, C
  goto      ITSPOS
ITSNEG
  bsf       fMoveBack, 0        ; if the result is negative, complement it
  comf      Delta + 1, f        ; we want the absolute difference of movement
  comf      Delta, f
  movlw     1
  addwf     Delta, f
  btfss     STATUS, C
  goto      ITSPOS
  movlw     1
  addwf     Delta + 1, f

ITSPOS
                                ; the result could be zero, if so we're done
  movf      Delta, f
  btfss     STATUS, Z
  goto      CHECKSIZE
  movf      Delta + 1, f
  btfsc     STATUS, Z
  retfie

CHECKSIZE
  movf      Delta, w
  subwf     HalfWay, w
  btfss     STATUS, C           ; CARRY set means positive result
  goto      BORROW2
  movf      Delta + 1, w
  subwf     HalfWay + 1, w
  btfss     STATUS, C
  goto      TOOBIG
  goto      SIZEOK
BORROW2
  movlw     1
  addwf     Delta + 1, w
  subwf     HalfWay + 1, w
  btfss     STATUS, C
  goto      TOOBIG
  goto      SIZEOK

TOOBIG
  movlw     1
  xorwf     fMoveBack, f
SIZEOK
  btfss     fMoveBack, 0
  goto      FORWARD
BACKWARD
  movf      Pos, f              ; before moving backward see if position is at zero
  btfss     STATUS, Z           ; if so, set position to maximum position
  goto      DECRPOS             ; otherwise decrement position and take a step
  movf      Pos + 1, f
```

```
  btfss     STATUS, Z
  goto      DECRPOS
  movf      MaxPos, w
  movwf     Pos
  movf      MaxPos +1, w
  movwf     Pos + 1
  goto      DECRSTEP
DECRPOS
  movlw     1
  subwf     Pos, f
  btfsc     STATUS, C
  goto      DECRSTEP
  subwf     Pos + 1, f

DECRSTEP
  decf      MotorStep, f
  goto      TAKESTEP

FORWARD
  movf      MaxPos, w           ; see if motor is already at maximum position
  subwf     Pos, w              ; if so, next postion is zero
  btfss     STATUS, Z           ; otherwise increment position and take a step
  goto      INCRPOS
  movf      MaxPos + 1, w
  subwf     Pos + 1, w
  btfss     STATUS, Z
  goto      INCRPOS
  clrf      Pos
  clrf      Pos + 1
  goto      INCRSTEP

INCRPOS
  movlw     1
  addwf     Pos, f
  btfss     STATUS, C
  goto      INCRSTEP
  addwf     Pos + 1, f

INCRSTEP
  incf      MotorStep, f

TAKESTEP
  movf      MotorStep, w
  andlw     0x07                ; Look only at the lower 3 bits
  call      NEXTSTEP            ; get the step pattern for the drivers
  movwf     PORTB               ; and stuff it in the output
  retfie

NEXTSTEP
  movwf     Temp
  movlw     HIGH STEPTABLE
```

```
  movwf   PCLATH
  movf    Temp, w
  addwf   PCL, f              ; add the table offset
STEPTABLE
  retlw   0x090               ; step pattern for step 1
  retlw   0x010               ; step pattern for step 2
  retlw   0x050               ; step pattern for step 3
  retlw   0x040               ; step pattern for step 4
  retlw   0x060               ; step pattern for step 5
  retlw   0x020               ; step pattern for step 6
  retlw   0x0A0               ; step pattern for step 7
  retlw   0x080               ; step pattern for step 0

          ; ******************END OF INTERRUPT SERVICE*************

INITIALIZE
  clrf    Pos                 ;
  clrf    Pos +1              ;
  clrf    NewPos              ;
  clrf    NewPos + 1          ;
  movlw   31                  ; This is the maximumposition for a half
  movwf   MaxPos              ; stepped 400 S/R motor i.e. :799
  movlw   3                   ; Adjust if you use another type motor
  movwf   MaxPos + 1
  movlw   144                 ; This is the number of steps to go half
  movwf   HalfWay             ; way around for a 400S/R motor
  movlw   1                   ; adjust if using another type motor
  movwf   HalfWay + 1
  movlw   3                   ; this will force motor position reset
  movwf   fSeekzero

  movlw   0x41
  movwf   BasePtr
  clrf    DAToffset
  clrf    Data0
  clrf    Data1
  clrf    Data2
  clrf    Data3
  clrf    Data4
  clrf    Data5
  clrf    Data6
  clrf    Data7
  clrf    fQuery

          ;*************************************************
          ;     This portion initializes the position
          ;              in a frame data transfer and
          ;     the PICID2 from values stored in EEPROM
           ;************************************************

  movlw   0x020
```

```
        movwf   Name2BasePtr
        bcf     STATUS, RP0         ; assure we are in block 0
        clrf    NB2offset
INITID2
        movf    Name2BasePtr, w
        movwf   FSR
        movf    NB2offset, w
        addwf   FSR, f              ; after this FSR points to registers
        bsf     STATUS, RP0         ; starting at 0x20
        movwf   EEADR               ; and EEADDR to char in EEPROM
        bsf     EECON1, RD          ; command a read
        movf    EEDATA, w           ; pick up the character
        bcf     STATUS, RP0
        movwf   INDF                ; and do an indirect write to registers
        incf    NB2offset, f
        movlw   15
        subwf   NB2offset, w        ; see if we are done
        btfss   STATUS, Z
        goto    INITID2

        ;********** End of EEPROM reading ******************


        movlw   'Z'                 ; Initialize the ID to ZZZ99
        movwf   NameByte0
        movwf   NameByte1
        movwf   NameByte2
        movlw   '9'
        movwf   NameByte3
        movwf   NameByte4

        movlw   0x032               ; Housekeeping variable set up
        movwf   NameBasePtr
        clrf    NBoffset
        movlw   0x029
        movwf   Name2BasePtr
        clrf    NB2offset
        clrf    fCMD
        clrf    fDATA
        clrf    DAToffset

        ;  initalize PORTB<7:4>  as outputs, PORTA<3:0> as inputs

        movlw   7                   ; turn off the input comparators
        movwf   CMCON               ; or PA inputs won't respond properly
        bsf     STATUS, RP0         ; select page 1 for TRIS access
        movlw   0x0F
        movwf   TRISB ^ 0X080       ;
        movlw   0xFF
        movwf   TRISA ^ 0X080
        bcf     STATUS, RP0         ; back to page 0
```

```
            ; Turn on timer 2

  bsf       STATUS, RP0
  movlw     190
  movwf     PR2 ^ 0x080
  bcf       STATUS, RP0
  movlw     0x0F
  movwf     T2CON               ; prescale = 16, TMR2 on, postscale = 2

            ; set up the USART

  bsf       STATUS, RP0
  movlw     77                  ; for 2400 baud with 12 MHz clock
  movwf     SPBRG ^ 0x080

  movlw     (1 << TXEN)         ; set transmit enable
  movwf     TXSTA ^ 0x080
  bcf       STATUS, RP0
  movlw     (1 << SPEN) | (1 << CREN)
  movwf     RCSTA               ; enable serial port and receive functionality

                                ; turn on interrupts

  bsf       INTCON, GIE         ; enable global interrupts
  bsf       INTCON, PEIE
  bsf       STATUS, RP0
  bsf       PIE1 ^ 0x080, TMR2IE          ; enable interrupts from TMR2
  bcf       STATUS, RP0

            ;*************************************************
            ;  This is the "idle loop" where the PIC waits for
            ;  something interesting to happen
            ;*************************************************

LOOP
  bsf       INTCON, GIE         ; turn interrupts back on

IDLELOOP
  btfsc     PIR1, RCIF
  goto      SERIALIN
  btfsc     fQuery, 7
  goto      QUERY
  goto      IDLELOOP

            ;***************** End of idle loop ***************

SERIALIN
  bcf       INTCON, GIE         ; turn off global interrupts
  movf      RCREG, w            ; pick up the received character
  movwf     RXBUF               ; and stick in the receive buffer
```

```
  bcf     PIR1, RCIF          ; clear the received character flag
  btfsc   fDATA, 0            ; test if handling data yet
  goto    HANDLEDATA
  btfsc   fCMD, 7             ; test if this is a command byte
  goto    HANDLECMD

; This portion checks incoming byte stream for ID match

                              ; first check against name 1
  movf    NameBasePtr, w      ; Generate the address into the ID
  movwf   FSR                 ; and stuff it into the FSR
  movf    NBoffset, w
  addwf   FSR, f
  movf    INDF, w             ; Next ID byte is now in W
  subwf   RXBUF, w
  btfss   STATUS, Z           ; zero if character matches
  goto    NOMATCH1
  incf    NBoffset, f
  movlw   5
  subwf   NBoffset, w         ; check if all characters of ID have
  btfss   STATUS, Z           ; scanned.  If not, we're done for now
  goto    CHECK2
  clrf    NBoffset            ; if we got here, the character stream
  clrf    NB2offset
  movlw   0x0FF               ; in matched the unit ID.  Next byte in
  movwf   fCMD                ; will be a command.
  goto    LOOP

NOMATCH1
  clrf    NBoffset            ; Character doesn't match so no ID1 match
CHECK2
  movf    Name2BasePtr, w     ; generate the address into ID2
  movwf   FSR                 ; and stuff it into the FSR
  movf    NB2offset, w
  addwf   FSR, f
  movf    INDF, w             ; Next ID byte is now in W
  subwf   RXBUF, w
  btfss   STATUS, Z           ; zero if character matches
  goto    NOMATCH2
  incf    NB2offset, f
  movlw   5
  subwf   NB2offset, w        ; check if all characters of ID have
  btfss   STATUS, Z           ; scanned.  If not, we're done for now
  goto    LOOP
  clrf    NBoffset            ; if we got here, the character stream
  clrf    NB2offset
  movlw   0x0FF               ; in matched the unit ID.  Next byte in
  movwf   fCMD                ; will be a command.
  goto    LOOP

NOMATCH2
```

```
  clrf    NB2offset
  goto    LOOP                ; start over

HANDLECMD
  movf    RXBUF, w
  movwf   fCMD                ; Store command
  movlw   0x0FF               ; and set fDATA flag
  movwf   fDATA
  movlw   HIGH GETXFERCNT
  movwf   PCLATH
  movf    fCMD, w
  andlw   0x0f
  call    GETXFERCNT          ; Initialize data transfer count for the
  movwf   DataCtr             ; command
  movf    DataCtr, w
  btfsc   STATUS, Z
  goto    CMDPREP
  clrf    DAToffset
  clrf    SkipCtr
  btfss   DataCtr, 7
  goto    LOOP
  bcf     DataCtr, 7
  movf    SkipCt, w
  movwf   SkipCtr
  goto    LOOP

GETXFERCNT          ; This table returns expected byte count
  movwf   Temp
  movlw   HIGH XFERCOUNT
  movwf   PCLATH
  Movf    Temp, w
  addwf   PCL, f      ;  to receive for each command, plus flag
XFERCOUNT
  retlw   0x00        ; Command 0   in bit7 indicating frame
  retlw   0x02        ; Command 1   transfer
  retlw   0x82        ; Command 2   *** this is a frame transfer***
  retlw   0x00        ; Command 3
  retlw   0x00        ; Command 4
  retlw   0x00        ; Command 5
  retlw   0x00        ; Command 6
  retlw   0x00        ; Command 7
  retlw   0x00        ; Command 8
  retlw   0x05        ; Command 9  5 bytes for rename
  retlw   0x01        ; Command 10 1 byte for skip count value
  retlw   0x00        ; Command 11
  retlw   0x00        ; Command 12
  retlw   0x00        ; Command 13
  retlw   0x00        ; Command 14
  retlw   0x00        ; Command 15

HANDLEDATA                  ; This implements skipping data bytes
```

```
        movf    SkipCtr, f          ; for the frame data transfer
        btfsc   STATUS, Z
        goto    STUFFDATA
        decf    SkipCtr, f
        goto    LOOP

STUFFDATA
        movf    BasePtr, w          ; get the pointer into the display data
        movwf   FSR
        movf    DAToffset, w
        addwf   FSR, f
        movf    RXBUF, w            ; grab the new byte
        movwf   INDF                ; and stuff it in
        incf    DAToffset, f
        decfsz  DataCtr, f          ; All bytes transferred?
        goto    LOOP
CMDPREP
        clrf    DAToffset
        clrf    fDATA
        movlw   HIGH CMDBRANCH
        movwf   PCLATH
        movf    fCMD, w
        andlw   0x0f
        addlw   LOW CMDBRANCH
        btfsc   STATUS, C
        incf    PCLATH, f
        movwf   PCL
CMDBRANCH
        goto    INITIALIZE          ; CMD 0
        goto    SETMOTORPOS         ; CMD 1
        goto    SETMOTORPOS         ; CMD 2
        goto    INITIALIZE          ; CMD 3
        goto    INITIALIZE          ; CMD 4
        goto    INITIALIZE          ; CMD 5
        goto    INITIALIZE          ; CMD 6
        goto    INITIALIZE          ; CMD 7
        goto    QUERYNAME           ; CMD 8
        goto    RENAME              ; CMD 9
        goto    SETFRAMESLOT        ; CMD 10
        goto    INITIALIZE          ; CMD 11
        goto    INITIALIZE          ; CMD 12
        goto    INITIALIZE          ; CMD 13
        goto    INITIALIZE          ; CMD 14
        goto    INITIALIZE          ; CMD 15
        ;*****************************************************
        ;               SETMOTORPOS  checks the new position to see if
        ; it exceeds the step count for a full rotation. If so
        ; it loads the max position value in the new position
        ; variables. If it does not exceed it loads the new values
        ;*****************************************************
```

```
SETMOTORPOS

  movf    Data0, w          ; check low byte
  subwf   MaxPos, w
  btfss   STATUS, C         ; carry set means a positive result
  goto    BORROW1IN         ; carry is clear so must borrow
  movf    Data1, w          ; carry set so no need to borrow
  subwf   MaxPos + 1, w     ; check high byte
  btfss   STATUS, C         ; carry set means positive result
  goto    INTOOBIG          ; carry is clear so result is negative

INPUTOKAY
  movf    Data0, w          ; input not too big so just load
  movwf   NewPos            ; new values are go back to idle loop
  movf    Data1, w
  movwf   NewPos + 1
  goto    LOOP

BORROW1IN
  movf    Data1, w          ; Underflow without carry means too big
  subwf   MaxPos + 1, w
  btfss   STATUS, C         ; carry set means positive result
  goto    INTOOBIG
  movwf   Temp              ; no underflow means much check after borrow
  movlw   1
  subwf   Temp, w
  btfsc   STATUS, C         ; carry clear means negative result
  goto    INPUTOKAY         ; carry set means positive result

INTOOBIG
  movf    MaxPos, w
  movwf   NewPos
  movf    MaxPos+1, w
  movwf   NewPos + 1
  goto    LOOP


        ;***************** End of SETDISPLAYVAL ******************

QUERYNAME
  movlw   8
  movwf   EEPtr             ; Pointer into EEPROM
  movlw   17
  movwf   QueryCtr          ; Number of bytes to transfer
  movlw   0x81
  movwf   fQuery
  clrf    fCMD
  goto    LOOP

SETFRAMESLOT
  movf    Data0, w
```

```
        movwf   SkipCt
        movlw   0x08
        movwf   EEPtr
        movlw   0x28
        movwf   RegPtr
        movlw   0x01
        movwf   Count
        clrf    fCMD
        goto    BURN

RENAME
        movf    Data0, w
        movwf   Name2Byte0
        movf    Data1, w
        movwf   Name2Byte1
        movf    Data2, w
        movwf   Name2Byte2
        movf    Data3, w
        movwf   Name2Byte3
        movf    Data4, w
        movwf   Name2Byte4
        movlw   0x09
        movwf   EEPtr
        movlw   0x29
        movwf   RegPtr
        movlw   0x05
        movwf   Count
        clrf    fCMD

        ;*******************************************************
        ; This will copy a block of data in registers pointed
        ; to by "RegPtr", of size "Count", into EEPROM memory
        ; at a location pointed to by "EEPtr".
        ;*******************************************************

BURN
        movf    RegPtr, w
        movwf   FSR
        movf    INDF, w                 ; pick up a byte from registers
        bsf     STATUS, RP0             ; EEPROM registers are in bank 1
        movwf   EEDATA
        bcf     STATUS, RP0
        movf    EEPtr, w
        bsf     STATUS, RP0
        movwf   EEADR
        bcf     EECON1, WRERR
        bsf     EECON1, WREN            ; enable write
        movlw   0x055                   ; required magic sequence for write
        movwf   EECON2
        movlw   0x0AA
        movwf   EECON2
```

```
 bsf     EECON1, WR
 bcf     EECON1, WREN      ; clear write enable since its going
 bcf     STATUS, RP0
 btfss   PIR1, EEIF        ; poll for EEPROM write completion
 goto    $ - 1
 bcf     PIR1, EEIF        ; clear the done flag
 incf    EEPtr, f
 incf    RegPtr, f
 decfsz  Count, f
 goto    BURN
 clrf    NB2offset
 clrf    fCMD
 clrf    fDATA
 goto    LOOP
         ;****************** End of write to EEPROM ***************

         ;*********************************************************
         ;          QUERY services requests from the host.
         ;          fQuery, 7 indicates an outstanding query.
         ;*********************************************************
QUERY
 bcf     INTCON, GIE       ; turn off global interrupts
 btfss   PIR1, TXIF        ; Is serial transmit buffer ready?
 goto    LOOP              ; No, back to LOOP
 movf    EEPtr, w
 bsf     STATUS, RP0       ;
 movwf   EEADR             ; and EEADDR to char in EEPROM
 bsf     EECON1, RD        ; command a read
 movf    EEDATA, w         ; pick up the character
 bcf     STATUS, RP0
 movwf   TXREG
 incf    EEPtr, f
 decfsz  QueryCtr, f
 goto    LOOP
 clrf    fQuery
 goto    LOOP

         ;****************** End of QUERY ************************

 org     0x02100    ; This is the start of the EEPROM area
 de      0
 de      0
 de      0
 de      0
 de      0
 de      0
 de      0
 de      0
 de      0   ; position in frame data transfer
 de      "DG001 STPR V1.000"
 end
```

Stepping Motor Gyro Compass Bill of Materials

Quantity	Item	Sources, notes, etc.
	.050", .020" sheet aluminum	Aircraft Spruce and Specialty, Enco
	.125" clear acrylic sheet	Local hardware store, Tap Plastics
	.25" clear acrylic sheet	
4	5/8" #4-40 flat head screws	Local hardware store, Digikey, Mouser
4	¼" #4-40 round head screws	
1	Knob of 1/8" shaft	Jameco, Digikey, Mouser
4	1.5" long, .25" hex spacers, female-female 4-40 thread	
8	1.00" long, .25" hex spacers, male-female 4-40 thread	
	1/8" brass hobby rod	Hobby shop, Small Parts
	5/32" brass hobby tubing	
	Quick setting, filled epoxy	Any number of brands will work as long as it has a filler. This makes the epoxy less likely to run and strengthens the cured material.
	White paint, enamel	Hobby shop
	Flat black paint, enamel	
1	400 step/rev bipolar stepping motor	
1	7805 5 volt regulator	This is for the motor. See text.
1	78L05 5 volt regulator	Jameco, Digikey, Mouser, etc.
1	PIC16F628-20 micro controller	
1	MAX232CPE RS-232C xcvr	
1	L293D H-bridge driver	
1	12.0 MHz ceramic resonator with internal capacitors (Digikey X907-ND or eqv.)	
1	H21A1 optical interrupter	
4	1N4001 diode	
1	Rotary encoder, Bourns 3315Y-1-006 or eqv.	
4	High efficiency red LED	
1	330 ohm, ¼ watt resistor	
1	390 ohm, ¼ watt resistor	
4	10K ohm, ¼ watt resistor	
2	100 ufd, 35 volt capacitor	
1	10 ufd, 35 volt capacitor	
4	.1 ufd, 50 volt capacitor	
4	1 ufd, 50 volt capacitor	
1	.01 ufd, 50 volt capacitor	
1	DB9 male panel mounting connector with solder lugs, mounting hardware	
	Perf board, wire, solder, etc.	

Ordering Information

You can order copies of this book by filling out the following form and mailing it along with payment. For on line ordering information check www.mikesflightdeckbooks.com.

Mail form to:

Mike Powell
Mike's Flight Deck Books™
P.O. Box 778
Lafayette, CA, USA 94549-0778

Please send:
____ copies of Building Simulated Aircraft Instruments @ US$39.95 each _____
Add 8.25% sales tax for California delivery addresses _____
Shipping (via air) & Handling _____
Domestic US shipping: US$4.50 per book.
International shipping: US$10.00 per book.
TOTAL ENCLOSED: _____

Funds must be in US dollars drawn on a US bank.
Please make checks out to "Mike Powell".

Ship to:

International shipping requires a street address (not a box number) and a contact telephone number.

If you provide an email address I will notify you when your order is shipped.

Name:_____
Address:_____
_____
_____
City:_____ State:_____
Postal code:_____ Country:_____
Telephone:_____
Email:_____

Duty, VAT, custom fees, etc. are the responsibility of the buyer.