Build Your Own EFIS, Part 2

An introduction to building and programming an Electronic Flight Instrument System.

BY JAMES P. HAUSER

In the first of this series, I presented the minimum of what you may expect from a build-it-yourself EFIS. These articles are condensed from my book titled "Building and Programming Electronic Flight Instrument Systems."

In this installment, I will present the development of a simple attitude display using MS QBASIC (Microsoft Quick Beginners All purpose Instruction Code). This will introduce you to programming graphical displays as well as the general idea of programming. The primary purpose of this installment is to demystify the programming of graphics displays, especially EFIS. Although considerable persistence is required to produce a functional display, there is no great magic involved. You may feel a bit (or a lot) overwhelmed at first if you have no experience in programming. But, if you can use a word processor, you can program your PC in QBASIC. My recommendation is to carefully follow through the steps as presented in this article. At the end, you will be rewarded with a simple attitude indicator working on your PC. It is unlikely that you will be a proficient programmer as a result, but it will be a start.

I should also point out that there are open source programs being prepared which will obviate the need to learn how to program your own EFIS, but it is still worthwhile to go through the exercise. There also may be commercial programs available. But, in addition to being an added cost, commercial programs may not be modified to meet your personal preferences.

I have several reasons for using QBASIC. One, it is readily available on most PC's. Two, it is easy to learn. And third, it has a primitive graphics engine so that it is not necessary that you understand methods such as Bresenham's algorithm, line clipping, rasterization, etc. That being said, QBASIC is not suitable as a programming language for an actual EFIS. Except for the most primitive of displays, it is simply too slow. But, that does not lessen its usefulness as a learning tool. (Once upon a time there was a faster version available, but it has been discontinued. Later versions of MS Visual Basic are a possibility, but I have not made a run time comparison.)

I will provide four examples. The first three will be thoroughly explained. The last example will expect you to explore the QBASIC "Help" to understand the details. It is a truism that the only way to learn programming is to program. You should expect to make errors. Finding and correcting those errors is the best teacher.

Although there are many common errors, typographical errors are sometimes the most difficult to find and deserve special mention. This is because a variable such as X0 (X 'zero') and XO (X 'Oh') look similar but are different variables to QBASIC. This is one

of the drawbacks to QBASIC. It does not require that variable be declared. It simply declares them “on the fly”.

This installment is very much a “hands on” tutorial. Although you are encouraged to give this installment a cursory review before beginning, it is important to be sitting at a PC while trying to understand what follows.

First, you need to locate the QBASIC program and its Help file. I assume in this article that you have access to a Microsoft Windows based PC. There are several ways to proceed. The easiest is to use the Find function. Click “Start”, then “Find”, and then “Files or Folders.” Enter “QBASIC” in the “Named” entry. “Look in” should be the boot drive, most likely “C:”. Users of Windows 3.X should use Explorer to find QBASIC.

If there are no QBASIC files, then the Windows installation CD or disks should be searched. As a last resort, QBASIC may be downloaded free from the Microsoft web site. The following URL should lead you to the correct Microsoft download location.

http://www.johuco.com/snail/snail.html

After locating the QBASIC Application file (QBASIC.exe), I recommend that you copy it and the QBASIC Help File (QBASIC.hlp) to a folder created to work through the examples to follow. **Right** click on the QBASIC files and drag them to the new folder. Select copy from the menu that appears when the right button is released. Alternatively, if disk space is limited, shortcuts can be created.

(A new directory may be created by double clicking on My Computer, double click on a convenient hard drive (ex. C:), then select New Folder from the File menu. Type in a convenient name such as QBEFIS, then press “Enter” twice. This will open the new folder. You may wish to create a shortcut to this folder on the Desktop.)

If more than one version of the Application or the Help file is shown, use the latest version.

In the new folder, right click on the blue QBASIC icon, select “Properties”, then “Screen”. Under “Usage”, click on “Full-screen”. This is not absolutely necessary, but QBASIC appears to behave better in full screen mode.

Now double click on the blue QBASIC icon in the new folder. After QBASIC opens, pressing the Esc key will present the user with a blank window for entering code. Verify that your mouse is working. If not, you will need to use the arrow keys to navigate. In any case, it is unlikely that any mouse scroll wheel will work, as a DOS driver is required for this function.

Our first bit of code is text based for those readers with no programming experience. It is the classic “Hello World!” program. It consists of one line:

```
PRINT “Hello World!”
```

Type this line in the blank window then press F5. A DOS screen should appear with “Hello World!” Press any key to return to the QBASIC programming window. To close QBASIC and return to Windows, select Exit from the File Menu or press Alt, then F, and then X. If you wish to return to Windows without closing QBASIC, press the Alt and Esc keys simultaneously (Alt + Esc).

You are strongly encouraged to explore the QBASIC window. This will lead to more comfort in using this programming environment.

If at any time a program stops running but does not return to the QBASIC window, press Ctrl + C or Ctrl + Break. This should terminate the program and return to QBASIC.

As the next step, modify the “Hello World! Program as follows. This illustrates one of the fundamental techniques of programming, namely looping. In this case, it uses the FOR ... NEXT loop.

```
' This routine prints "Hello World!" to the terminal screen 10 times.

FOR I = 1 TO 10

  PRINT I, "Hello World!"

NEXT I

END
```

The function of the FOR ... NEXT loop should be fairly apparent. The first line is a comment and is not executed by QBASIC.

Now that we have the basics for entering and running a program in QBASIC, we can now begin to build a graphic display of an attitude indicator. First, we need to talk a bit about Screen Modes.

Drawing a line

As most PC users know, there are a variety of screen modes. For example, standard VGA is 640 pixels wide by 480 pixels high while Super VGA is 800x600. A pixel is the smallest element of a screen picture and has one and only one color. As you might expect, the more pixels, the more processing time is required to draw a picture pixel by pixel. So, for our tutorial, we will use a mode with a resolution of 640x480. In QBASIC, Mode 12 has this resolution. However, for a more realistically sized display, we will only use a 320x240 window within the 640x480 screen. This will also speed up the program by reducing the number of pixels that must be plotted.

Start QBASIC and enter the following code:

```
' This routine draws a horizontal line
SCREEN 12                       ' Set Screen Mode 640x480
CLS                             ' Clear the screen
LINE (0, 120)-(319, 120), 9     ' Draw a Blue Horizontal Line
WHILE (INKEY$ = ""): WEND       ‘ Wait for a keystroke to end
END
```

Run this program (F5). A blue line should horizontally bisect the upper left quarter of the screen. If not, correct typing errors and try again. Press the Space Bar to end. Then, press any key to return to the QBASIC code window.

Now is a good time to learn how to save a program. Click on File in the tool bar and select Save (Alt, F, S). Enter a convenient name such as BlueLine and press Enter.

Now that you have a working program, I will explain the code you have entered.

```
' This routine draws a horizontal line
```

This is a comment. QBASIC interprets all text following an apostrophe or REM as a comment. Comments are not processed as instructions. However, comments are very important to the readability of the code. Use them freely.

```
SCREEN 12
```

This line simply sets the display to Screen Mode 12 (16 colors, 640x480). Click on the instruction SCREEN and press F1. A help screen should appear which details all the variations of this instruction and its nuances. More on this command later.

```
CLS
```

This instruction sets all the pixels on the screen to black (or off).

```
LINE (0, 120)-(319, 120), 9
```

This instruction draws a line with the left coordinate at 0, 120 and the right coordinate at 319, 120. The last parameter is the color. In this case, “9” sets the line color to blue. Contrary to what might be expected, the upper left corner of the screen is 0, 0. Thus, X increases to the right and Y increases downward.

```
WHILE (INKEY$ = ""): WEND
```

This is an example of a WHILE...WEND loop. The program will continue to repeat this loop WHILE the value in the parentheses is true. In this case, it provides a convenient pause until the Space Bar (or any key) is depressed. The parentheses are not essential, but they make the code more readable. Here, the loop has been concatenated onto one line by use of the colon ":". (Concatenated means more than one instruction is placed on a line. Each instruction on that line must be separated by a colon ":". Spaces and tabs are ignored.)

```
END
```

This is simply the end or exit command for the program.

Drawing a "Wings Level" Attitude Indicator

It is quite possible to use the LINE instruction as used in BlueLine to draw all the features of an attitude indicator. However, the LINE instruction has options to draw both unfilled and filled rectangles.

Click on LINE in the previous program, and then press F1. The Help screen for this instruction should be displayed. From this screen we see that a B or BF appended to the instruction will produce an unfilled or filled box respectively. We will now use this option to draw a "wings level" attitude indicator. No roll markings are drawn, only pitch markings.

Open the BlueLine program and save as WingsLvl or other convenient name. Modify this program to be as follows:

```
' This routine draws a simple wings level attitude display

SCREEN 12                                          ' Set Screen Mode 640x480

CLS                                                ' Clear the screen

LINE (0, 0)-(319, 120), 9, BF                      ' Draw Sky
WHILE (INKEY$ = ""): WEND

LINE (0, 120)-(319, 239), 2, BF                    ' Draw Earth
WHILE (INKEY$ = ""): WEND

LINE (160 - 25, 120-2)-(160 - 8, 120 + 2), 14, BF  ' Draw Left Wing
WHILE (INKEY$ = ""): WEND

LINE (160 + 25, 120 - 2)-(160 + 8, 120 + 2), 14, BF  ' Draw Right Wing
WHILE (INKEY$ = ""): WEND

CIRCLE (160, 120), 2, 14                           ' Draw Fuselage Outline
WHILE (INKEY$ = ""): WEND

LINE (160 - 1, 120 - 1)-(160 + 1, 120 + 1), 14, BF  ' Fill Fuselage
```

```
WHILE (INKEY$ = ""): WEND

END
```

Save and run this program.

Following each LINE instruction is the instruction we used previously to pause the execution of the program. This allows you to see the result of each individual LINE instruction. Press the space bar to proceed from one LINE instruction to another. The last space bar will terminate the program.

I have also introduced the CIRCLE instruction in this program. The Help screen explains its usage.

Color Palette

Thus far, we have simply provided the numbers for the colors used by the drawing instructions. The QBASIC Help provides the correspondence between the sixteen available colors and the numbers.

GOSUB Procedures

To aid in the organization of subsequent programs, we introduce the GOSUB procedure. This method allows the main program to be relatively free of details. The format of a program with a GOSUB procedure is:

```
GOSUB Name                          ‘ Go to the subroutine Name

END                                 ‘ End of main program

Name:                               ‘ Subroutine Name + “:”
...                                 ‘ Subroutine code here
RETURN                              ‘ Return from subroutine
```

For example, the BlueLine program could be coded as:

```
' This routine draws a horizontal line using GOSUB procedures

GOSUB Initialize

GOSUB DrawLine

GOSUB WaitBar

END


‘ GOSUB Procedures

Initialize:                                 ‘ Procedure label
```

```
        SCREEN 12                       ' Set Screen Mode 640x480
        CLS                             ' Clear the screen
RETURN                                  ‘ Return to main program

DrawLine:                               ‘ Procedure label
        LINE (0, 120)-(319, 120), 9     ' Draw Horizontal Line
RETURN                                  ‘ Return to main program

WaitBar:                                ‘ Procedure label
        WHILE (INKEY$ = ""): WEND       ‘ Wait for space bar (or any key)
RETURN                                  ‘ Return to main program
```

Note that each procedure label must be followed by a colon. All GOSUB procedures referred to in the main code must appear somewhere in the code. (But, after the END statement.) Otherwise, QBASIC will generate an “Label not defined” error. The order of the appearance of the procedures is unimportant.

The GOSUB is similar to the subroutine method to be used in the next example. The subroutine method is more commonly used in other computer languages. The GOSUB method is peculiar to the BASIC family of languages.

Attitude Indicator with Roll and Pitch Marks

Until now, all of our lines have been horizontal. Roll marks introduce a new challenge. They are drawn at an angle. We could, of course, carefully calculate the end points of fixed roll marks and enter those numbers in the LINE instruction. But, since we will also need to know how to rotate the displayed image, we will include a subroutine which will calculate the position of a point after the image is rotated about the origin by some angle. In the interest of brevity, I will not explain the mathematics behind this calculation.

As before, save the previous PitchMrk program as RollMark. Modify to the code below and save the result. Run the program. The keypad provides input for roll and pitch. Increase pitch = “8”, decrease pitch = “2”, left roll = “4”, and right roll = “6”. To terminate, press the Escape key.

When you begin typing the last few lines, beginning with SUB, QBASIC will open a new window. This is a subroutine window. When finished, press F2 and then Enter to return to the main program window.

If you have this article in electronic form, you may copy and paste the code into Notepad, then save as RollMark.bas into your QBASIC folder. When saving, be sure to select

Save as type: All Files(*.*). Otherwise, Notepad will append ".txt" to your file. RollMark.bas may then be opened and run from QBASIC.

```
' This routine draws an attitude display with pitch and roll control
DECLARE SUB Rotation (X1!, Y1!, CosTheta!, SinTheta!, X2!, Y2!)
    GOSUB Initialization
    WHILE (Done = False)
        GOSUB DrawSky
        GOSUB DrawEarth
        GOSUB DrawPitchMarks
        GOSUB DrawAirplane
        GOSUB DrawRollMarks
        GOSUB GetKey
    WEND
END


' GOSUB Procedures

WaitBar:      ' This subroutine waits for the Esc key to be depressed
    WHILE (INKEY$ <> CHR$(27)): WEND
RETURN


Initialization:

    Esc$ = CHR$(27)
    False = &H0:                  True = &HFF
    Done = False
    pi = 4! * ATN(1)
    Xmin = 160:                   Ymin = 120
    Xmax = Xmin + 319:            Ymax = Ymin + 239
```

```
    Xctr = (Xmin + Xmax) / 2:      Yctr = (Ymin + Ymax) / 2

    AngularScale = 4

    Pitch = 0:                     Roll = 0

    UpArrow$ = "2":                DnArrow$ = "8"

    RtArrow$ = "4":                LtArrow$ = "6"

    SCREEN 12                                  ' Set Screen Mode 640x480 (VGA)

    CLS                                        ' Clear the screen

RETURN


DrawSky:

    LINE (Xmin, Ymin)-(Xmax, Ymax), 9, BF              ' Draw Sky

RETURN


DrawEarth:                                             ' Draw Earth

    ' First calculate the endpoints of the border.  For this simple example
    ' we assume that Roll is less than +/- 90 degrees.  (A divide by zero
    ' will occur at +/- 90 degrees.)

    TanRoll = TAN(Roll * pi / 180)

    PitchOffSet = AngularScale * Pitch / COS(Roll * pi / 180)

    YL = -160 * TanRoll:         YR = 160 * TanRoll

    X1 = Xmin:                   Y1 = Yctr + YL + PitchOffSet
    X2 = Xmax:                   Y2 = Yctr + YR + PitchOffSet
    X3 = Xmax:                   Y3 = Ymax
    X4 = Xmin:                   Y4 = Ymax


    ' Now draw a polygon using the points

    LINE (X1, Y1)-(X2, Y2), 0
    LINE (X2, Y2)-(X3, Y3), 0
    LINE (X3, Y3)-(X4, Y4), 0
    LINE (X4, Y4)-(X1, Y1), 0


    ' Select a point within the polygon and fill with color Green.

   SELECT CASE Ymax
```

```
        CASE IS > Y1
                PAINT (Xmin + 1, Ymax - 1), 2, 0

        CASE IS > Y2
                PAINT (Xmax - 1, Ymax - 1), 2, 0

    END SELECT

RETURN


DrawAirplane:

    LINE (Xctr - 25, Yctr - 2)-(Xctr - 8, Yctr + 2), 14, BF          ' Draw Left Wing

    LINE (Xctr + 25, Yctr - 2)-(Xctr + 8, Yctr + 2), 14, BF          ' Draw Right Wing

    CIRCLE (Xctr, Yctr), 2, 14                                       ' Draw Fuselage Outline

     PAINT (Xctr, Yctr), 14, 14                                      ' Fill Fuselage

RETURN


DrawPitchMarks:

    SinAngle = SIN(Roll * pi / 180)
    CosAngle = COS(Roll * pi / 180)

    FOR Mark = -20 TO 20 STEP 5

        PitchLine = -(Mark - Pitch) * AngularScale

        IF ((Mark MOD 10)) THEN HalfWidth = 20 ELSE HalfWidth = 40

        XL = -HalfWidth:          XR = HalfWidth

        YL = PitchLine:           YR = PitchLine

        CALL Rotation(XL, YL, CosAngle, SinAngle, X1, Y1)
        CALL Rotation(XR, YR, CosAngle, SinAngle, X2, Y2)

        XLt = Xctr + X1:          YLt = Yctr + Y1
        XRt = Xctr + X2:          YRt = Yctr + Y2


        ' Check if marks within window

        DrawMark = True
```

```
        IF (XLt < Xmin OR YLt < Ymin) THEN DrawMark = False
        IF (XRt < Xmin OR YRt < Ymin) THEN DrawMark = False
        IF (XLt > Xmax OR YLt > Ymax) THEN DrawMark = False
        IF (XRt > Xmax OR YRt > Ymax) THEN DrawMark = False

        IF (Mark = 0) THEN DrawMark = False


        IF (DrawMark) THEN LINE (XLt, YLt)-(XRt, YRt), 15

    NEXT Mark

RETURN



DrawRollMarks:

    ' Draw fixed pointer

        ' Compute the corners of the triangle

        X1 = Xctr - 7:                  Y1 = Ymin + 18
        X2 = Xctr + 8:                  Y2 = Ymin + 18
        X3 = Xctr:                      Y3 = Ymin + 10

        ' Draw the outline

        LINE (X1, Y1)-(X2, Y2), 14
        LINE (X2, Y2)-(X3, Y3), 14
        LINE (X3, Y3)-(X1, Y1), 14

        ' Fill the triangle

        PAINT (Xctr, Ymin + 15), 14, 14

    ' Draw rotating pointer

        SinAngle = SIN(Roll * pi / 180)
        CosAngle = COS(Roll * pi / 180)

        ' Compute the corners of the triangle

        X = -7:                         Y = Yctr - Ymin
        CALL Rotation(X, Y, CosAngle, SinAngle, X1, Y1)
        X1 = Xctr - X1:                 Y1 = Yctr - Y1

        X = 8:                          Y = Yctr - Ymin
        CALL Rotation(X, Y, CosAngle, SinAngle, X2, Y2)
        X2 = Xctr - X2:                 Y2 = Yctr - Y2

        X = 0:                          Y = Yctr - (Ymin + 9)
        CALL Rotation(X, Y, CosAngle, SinAngle, X3, Y3)
        X3 = Xctr - X3:                 Y3 = Yctr - Y3

        ' Draw the outline
```

```
        LINE (X1, Y1)-(X2, Y2), 15
        LINE (X2, Y2)-(X3, Y3), 15
        LINE (X3, Y3)-(X1, Y1), 15

        ' Fill the triangle

        PAINT ((X1 + X2 + X3) / 3, (Y1 + Y2 + Y3) / 3), 15, 15


    ' Draw the Roll Marks

    XL = 0:                                 YL = Yctr - Ymin - 8:
    XR = 0:                                 YR = Yctr - Ymin - 4

    Angle = 0

    WHILE (Angle < 90)

        IF Angle >= 30 THEN Angle = Angle + 30 ELSE Angle = Angle + 10
        IF Angle = 30 THEN YR = Yctr - Ymin

        SinAngle = SIN((Angle - Roll) * pi / 180)
        CosAngle = COS((Angle - Roll) * pi / 180)

        CALL Rotation(XL, YL, CosAngle, SinAngle, X1, Y1)
        CALL Rotation(XR, YR, CosAngle, SinAngle, X2, Y2)
        LINE (Xctr + X1, Yctr - Y1)-(Xctr + X2, Yctr - Y2), 15

        SinAngle = SIN((-Angle - Roll) * pi / 180)
        CosAngle = COS((-Angle - Roll) * pi / 180)

        CALL Rotation(XL, YL, CosAngle, SinAngle, X1, Y1)
        CALL Rotation(XR, YR, CosAngle, SinAngle, X2, Y2)
        LINE (Xctr + X1, Yctr - Y1)-(Xctr + X2, Yctr - Y2), 15

    WEND

RETURN


GetKey:

    Key$ = ""

    WHILE (Key$ = ""): Key$ = INKEY$:   WEND


    SELECT CASE Key$


      CASE IS = UpArrow$

            Pitch = Pitch + 5
```

```
        CASE IS = DnArrow$
            Pitch = Pitch - 5
        CASE IS = RtArrow$
            Roll = Roll + 5
        CASE IS = LtArrow$
            Roll = Roll - 5
        CASE IS = Esc$
            Done = True
    END SELECT
RETURN
SUB Rotation (X1, Y1, CosTheta, SinTheta, X2, Y2)
    X2 = X1 * CosTheta - Y1 * SinTheta
    Y2 = X1 * SinTheta + Y1 * CosTheta
END SUB
```

The FOR ... NEXT, SUB, IF ... THEN, PAINT, and MOD methods are explained in the QBASIC Help.

Notice that no alphanumeric characters are displayed on this primitive EFIS. This requires substantially more code to arbitrarily place characters on the screen (needed for scroll indicators). In the interest of keeping this article to a reasonable length, I have omitted that capability from the example. The display of alphanumeric characters in graphics is discussed in the book.

Summary

This has been a very brief and incomplete introduction to the programming of EFIS displays. My intent was to intrigue the reader and demystify these displays. Although you may never program an actual display for your aircraft, the software should no longer seem to be magic or inaccessible. As I am fond of saying, it is just putting dots on the screen.

Check back on www.kitplanes.com on June 15 for Part 3, the series conclusion.