PC12 Panels

By Roy Seifert

Introduction

I am an instrument-rated private pilot but can no longer fly due to medical reasons. Without actually stepping into a real plane, the closest I can get to flying is with Microsoft® Flight Simulator. I am running FS 2004 which gives a very good simulation of instrument flight. There are some things it won't do, nor allow me to do, but overall it satisfies my urge to fly in an instrument environment. Plus, it allows me to fly aircraft that in real life I couldn't afford to really learn how to fly. My wonderfully patient wife thinks it's a dumb game because she can't ever see anything happening, but I know I'm cruising at 24,000 feet at 180 knots in pressurized comfort and in total control!

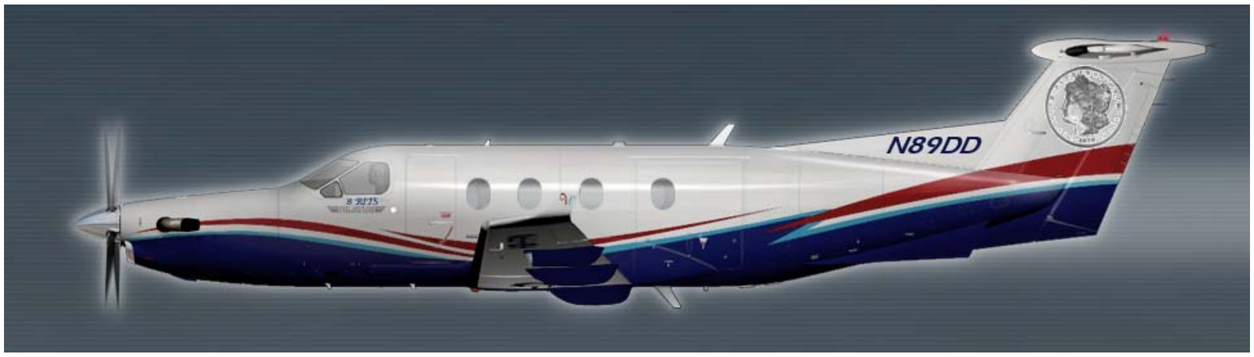


Figure 1: Pilatus PC-12 in "Eight Bits" Livery

I decided early on that since I'm playing make-believe pilot, I would find a virtual aircraft that I would like to buy in real life if I ever won the lottery. I found the single-engine turbo-prop Pilatus PC-12 that performs like a twin, but handles like a single-engine Cessna. For a significantly smaller price than $3.5 million I purchased one from Flight One Software. There are too many positive features about this aircraft simulation to list here, but suffice it to say all the aircraft available from Flight One Software are very realistic and high quality.

To enhance my flight simulator experience I purchased CH Products flight yoke, rudder pedals, and throttle quadrant. Along with their program, CH Control Manager I could configure buttons and axes to perform pretty much anything I wanted.

I didn't want to build a complete cockpit; I just wanted to substitute real switches for mouse clicks to activate avionics and cockpit functions, so I decided to build some switch panels (what my neighbor's son calls a "dashboard".). There are many articles on the Internet regarding building these devices, but I found one that was very informative (can't find the link now) that showed how to build switch panels that interface with Flight Simulator so I decided to try to build simulations of the panels I use most in the PC-12.

PC-12 Panels

When I fly the PC-12 I like to start with a cold cockpit, i.e. engine off, power off, standing outside of the aircraft ready to do a preflight inspection. The panels I use most are:

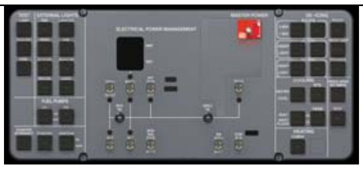
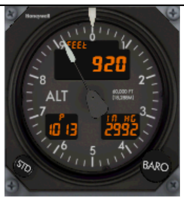
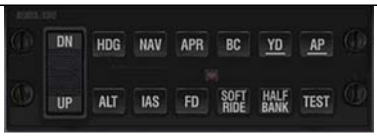
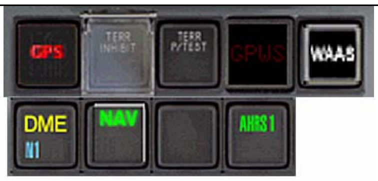
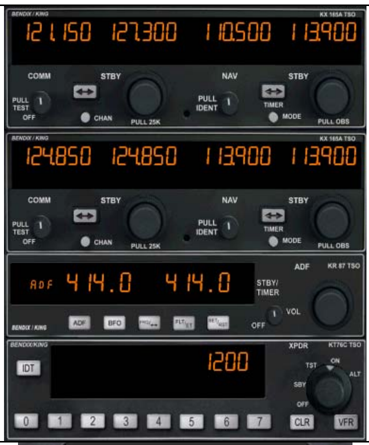
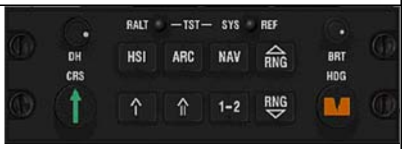
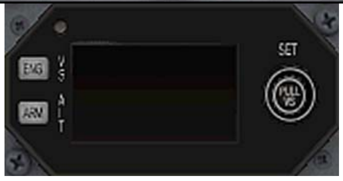
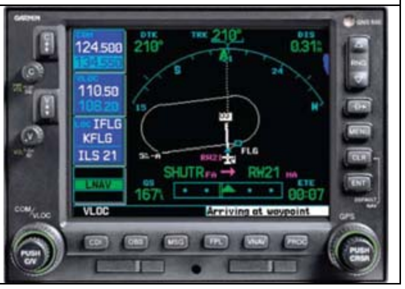
Overhead • 32 push-buttons • 10 toggle switches (only 8 used)		Altimeter • 1 rotary encoder w/switch	
Autopilot • 14 push-buttons		Other panel switches • 8 push-button switches	
Radios • 11 push-buttons • 5 rotary encoders w/switch		EFIS controller • 6 push-button switches • 3 rotary encoders w/switch (only 2 pb switches used)	
Altitude/Vertical speed adjust • 2 push-buttons • 1 rotary encoder w/switch		GPS • 13 push buttons • 1 rotary encoder w/switch	

Table 1: PC-12 Panels and Switches

Of course, I also use the pedestal that contains the throttle, prop, gear, and flap levers. I already have a CH Products throttle quadrant, but maybe someday I'll build a more realistic pedestal.

All of these panels can be duplicated using real switches. The altitude/vertical speed adjustment controller and the radios have digital displays, but these would not function on my units; only the switch functions would be duplicated. Also, some of the switches have corresponding lights or indicators which I will not duplicate on the panels.

Parts Sources

After doing some research I found the parts I needed. Most were available from Mouser Electronics, but I did purchase a few items from Radio Shack:

Source	Description	P/N	Total Qty
www.mouser.com	rotary encoder with switch	652-PEC11-4215F-S24	11
www.mouser.com	push button sw, SPDT mom	612-PS1057A-BLK	87
www.mouser.com	Toggle switch, DPDT, on-on	1055-TA2160-EVX	9
www.mouser.com	Green LED	78-TLHG5400	6
www.mouser.com	120 ohm 1/4watt resistor	660-MF1/4DC1200F	6
www.mouser.com	Knob, black plain	450-BA600	11
www.mouser.com	1N4148 diode	621-1N4148-T	306
www.mouser.com	USB Type B Connector, Vertical	806-KUSBVX-BS1N-B	1
www.mouser.com	220 ohm 1/4watt resistor	660-MF1/4DC2200F	20
www.mouser.com	10K ohm 1/4watt resistor	660-MF1/4DC1002F	8
www.mouser.com	6” x 9” photo-sensitive copper clad board	590-612	1
www.mouser.com	PCB developer solution	590-418-500ML	1
www.mouser.com	8 MHz crystal	73-XT49S800-20	1
www.mouser.com	.22 uF capacitor	594-K224K20X7RF53H5	1
www.mouser.com	22 pF capacitor	594-K220J15C0GF5TL2	1
www.mouser.com	.1 uF capacitor	594-K104Z15Y5VF53L2	1
www.mouser.com	PIC18F2455 Microcontroller	579-PIC18F2455-I/SP	1
www.mouser.com	24 AWG hookup wire, black, 100’	566-9975-100-10	1
www.mouser.com	24 AWG hookup wire, green, 100’	566-9975-100-05	1
www.towerhobbies.com	Plastruct White Sheet Styrene .125" (2)	LXDL98	2
www.radioshack.com	Bus Wire	278-1341	1
www.radioshack.com	25-Position Male Solder D-Sub Connector	210-3240	4
www.radioshack.com	25-Position Female Solder D-Sub Connector	210-3239	4
www.radioshack.com	15-Position Male Solder D-Sub Connector	210-2601	4
www.radioshack.com	15-Position Female Solder D-Sub Connector	210-2496	4
www.radioshack.com	9-Position Male Solder D-Sub Connector	210-2497	1
www.radioshack.com	9-Position Female Solder D-Sub Connector	210-2498	1

Table 2: Parts List

I found a web site called Desktop Aviator which sells flight simulator panels and interface boards. I purchased one of their Super Rotary Encoder interface boards which allowed me to interface 6 rotary encoders or 12 push button/toggle switches, and 8 axes. The Super Rotary Encoder uses a PIC18F2450 microcontroller (MCU) to interface with the switches, encoders, axes, and the PC via a USB connection. A microcontroller is nothing more than a computer in one integrated circuit (IC) package. Since I'm a computer geek I decided to see if I could program my own interface. In fact, it was thanks to Desktop Aviator that I began learning how to program microcontrollers and have developed many projects since this one.

PIC Microcontroller Programming

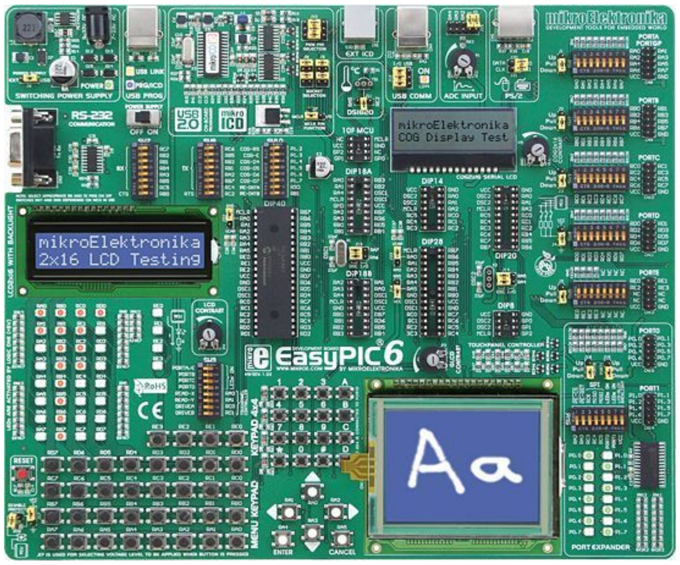


Figure 2: EasyPIC6 Development Board

I first needed to acquire the hardware and software necessary to program a microcontroller. Searching the Internet I found mikroElektronika, a company in Serbia that produces both the hardware and software that I needed to program microcontrollers. The EasyPIC6 development board they manufacture allows me to program many different PIC microcontrollers (PIC stands for peripheral interface controller), and contains the circuits to develop and test LED's, LCD graphical and character displays, switches, one analogue to digital converter (ADC) and various interfaces including USB. It also contains a port expander and access to the microcontroller's ports via header pins. The software that comes with the board contains examples designed for the PIC16F887 microcontroller, and the board comes with one of those microcontrollers. What a great tool for developing different applications! I used this board with its switches, LED's, and USB interface to develop and test my controller.

Compiler

A compiler is a specialized computer program that converts English language statements into the hexadecimal machine code that a microcontroller can read. MikroElektronika allows you to download trial versions of each of their compilers. Assembly language is a bit cumbersome for me, I know nothing of Pascal, I know some C language, but I am very fluent in BASIC, so I decided to purchase their MikroBASIC Pro compiler for PIC.

MikroElektronika also provides libraries and sample programs for learning how to program microcontrollers. These were invaluable for learning how to interface with the USB port.

Microcontroller

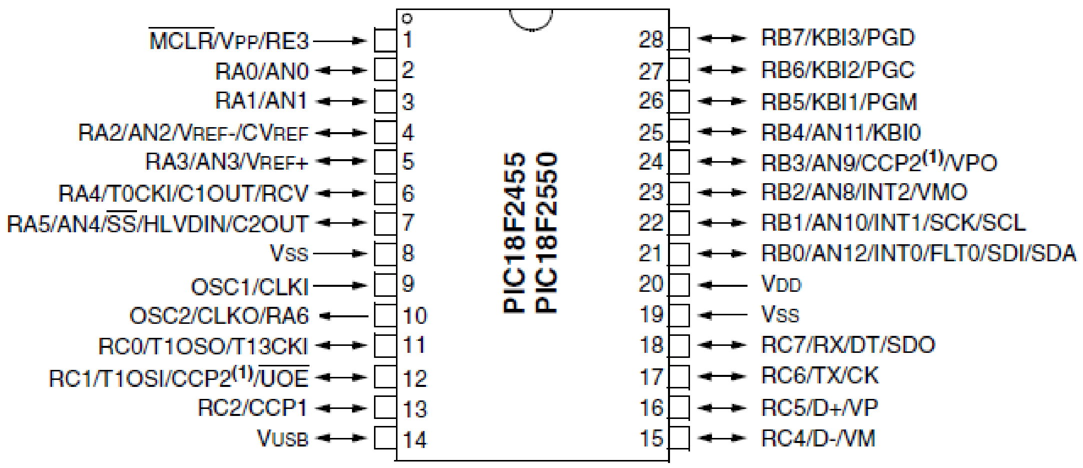


Figure 3: PIC18F2455

As mentioned before, a microcontroller is essentially a computer in a small, integrated circuit package. It has all the capabilities of a larger computer, but with less capacity. A microcontroller is normally programmed to perform one function, rather than many functions.

Microchip Technology Inc., a manufacturer of microcontrollers, publishes a selection guide for choosing the right microcontroller. I chose the PIC18F2455 for the following reasons:

- Small footprint, 28-pin DIP
- Eight pins had weak internal pull-up. These would be used for the switch column inputs. The internal pull-ups meant I didn't need external pull-up or pull-down resistors.
- At least four output pins. These would be used for the switch row outputs.
- An additional eight input pins used to connect toggle switches.
- USB interface capability for communication with the PC and Flight Simulator.

The figure below shows the schematic for the microcontroller circuit. C1 and C2 are required for the 8 MHz crystal. C3 is required for the internal USB 3.3 volt regulator, and C4 is a bypass capacitor required for the microcontroller. The 220-ohm resistors are static protection for the microcontroller. Power, +5 volts and ground, is provided by the USB port.

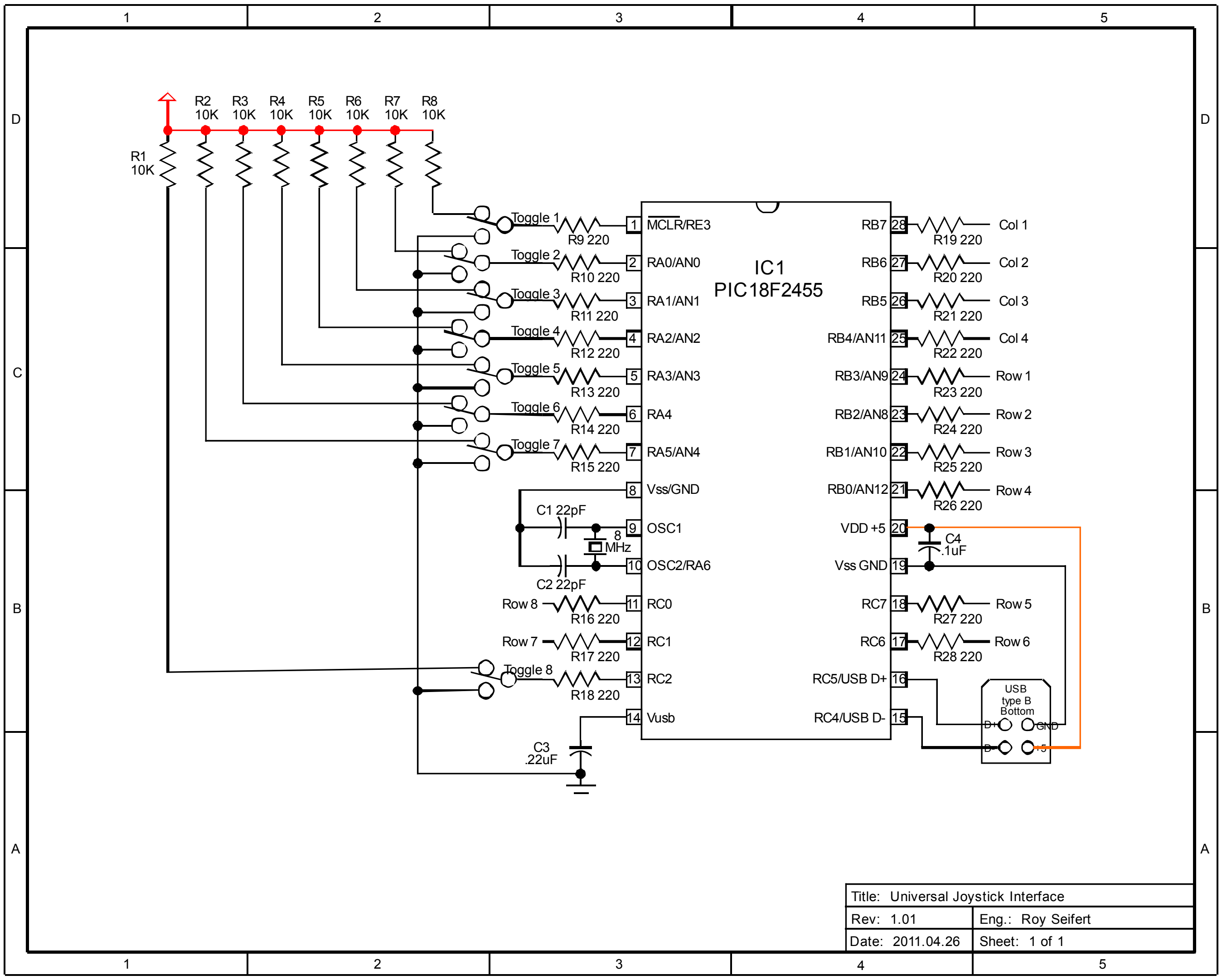


Figure 4: PC-12 Interface Microcontroller Schematic

Switch Matrix

If you have more switches than pins available on the microcontroller, the switches can be arranged in a matrix, sometimes called a keyboard matrix or keypad. Microsoft® Windows® only allows a maximum of 32 switches on a game controller so I used a 4 x 8 matrix.

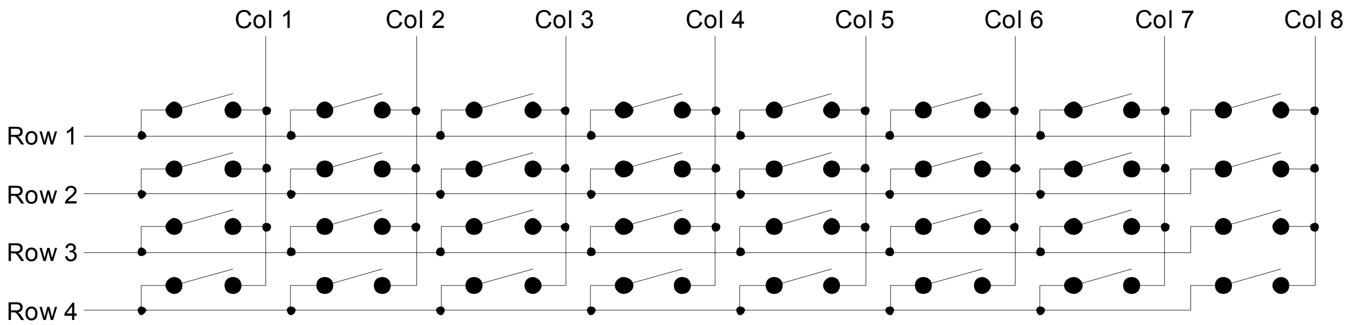


Figure 5: 4 x 8 Switch Matrix

The eight columns are connected to eight input pins of the microcontroller. These pins have an internal pull-up resistor which ensures that if nothing is connected, or the switch is open, it reads a logic 1. I just have to remember to invert the data before sending it to the USB port. The rows are connected to output pins of the microcontroller. Initially all of the row outputs are set high. Each row is then set low and the columns are read looking for a low or 0. Since there are eight bits in a byte, the eight columns are connected to the eight pins of port B so I can read all eight pins at once. Changing each row to 0 then reading the 8 columns is called polling.

My panels use 11 rotary switches which have two outputs; therefore each rotary switch requires 1 row and two column connections. I decided to use rows 1 – 3 for the rotary switches. Since each rotary switch requires two columns, each row can contain 4 rotary switches for a total of 12 which gives me one spare; more about rotary switches later.

The last row, row 4, I am using to read push-button switches. I need to interface with 87 push-button switches, and each rotary switch also has a built-in push-button switch for a total of 98 switches. Since row 4 only allowed me to interface with 8 switches, how could I interface one row with almost 100 switches?

Diode Matrix

The answer was to create a diode matrix. By creating a diode matrix, I could expand the number of push-button connections. Think of the 8 columns as a byte of 8 bits. With 8 bits I can have 256 different combinations. Since the combination of all zero's 0000 0000 would not be used because at least one column has to be turned on, I can therefore connect up to 255 switches. The key here is rather than having one switch drive only one column, I can have one switch drive up to 8 columns. So long as each switch drives a different combination of columns, I can connect up to 255 switches to 8 columns.

Switch	Col 4	Col 3	Col 2	Col 1
1 ●╱ ●				|◀
2 ●╱ ●			|◀	
3 ●╱ ●			|◀	|◀
4 ●╱ ●		|◀		
5 ●╱ ●		|◀		|◀
6 ●╱ ●		|◀	|◀	
7 ●╱ ●		|◀	|◀	|◀
8 ●╱ ●	|◀			
9 ●╱ ●	|◀			|◀
10 ●╱ ●	|◀		|◀	
11 ●╱ ●	|◀		|◀	|◀
12 ●╱ ●	|◀	|◀		
13 ●╱ ●	|◀	|◀		|◀
14 ●╱ ●	|◀	|◀	|◀	
15 ●╱ ●	|◀	|◀	|◀	|◀

Table 3: 1 x 4 Diode Matrix

The table above shows an example of a 1 x 4 matrix. Using 4 columns I can connect up to 15 switches. Where ever you see a diode symbol in the table that is where a switch connects. This is a simple binary progression, or you can think of it as counting from 1 to 15 in binary. You can see that switch 1 only connects to 1 diode on column 1, but switch 7 connects to 3 diodes, and switch 15 connects to four diodes. You need to have the diodes to isolate the switches; otherwise all the switches would be shorted together which would cause all the columns to come on every time any switch was pressed.

This flexibility works because of the programming power of FSUIPC. FSUIPC is a dynamic linked library (.dll) file that resides in the Modules folder of Flight Simulator. It allows hardware and software to interface with Flight Simulator. One of its features is the ability to program push buttons[1], rotary switches, and joystick axes. The ability to program combinations of up to 16 push-buttons to activate one function allows me to expand the number of push-button switches I can use to 65,535 (2^{16} – 1). I don't think even the space shuttle has that many switches! This function only works with the registered version, but it was well worth the money. You can download and register FSUIPC from simMarket.

FSUIPC needs to look at all eight switch lines, otherwise multiple actions could occur whenever a button is pressed. For example, the FSUIPC command `1=P3,0,M1:2,0` says that if joystick #3, switch #0 is pressed, run macro 1, command 2. But any other combination of switches that uses switch 0 will also run this macro. Because one switch is controlling up to 8 switch lines every switch command in FSUIPC must look at all 8 of those lines. However, if you remember from my diode matrix, not all of those 8 switch lines are turned on by a switch, but I must still program FSUIPC to look at all 8 switch lines every time, whether the line is on or off.

```
50=CP(-3,24)(-3,25)(-3,26)(+3,27)(+3,28)(-3,29)(-3,30)3,31,C66617,0;GPS DIRECT
```

I discovered that Windows sees switches 1-8, but FSUIPC sees the switches as 0-7 so I had to program all my switch commands accordingly. FSUIPC sees my MCU as joystick #3. The above switch command says that joystick 3, switch lines 27, 28 and 31 must be on, and switch lines 24, 25, 26, 29, and 30 must be off to perform the GPS Direct To function. By programming the other push-button combinations in this way I only needed one row for all 98 of my push-button switches. You can also see why I needed 306 diodes; 284 for different push-button combinations, and 22 for the 11 rotary encoders.

Hardware

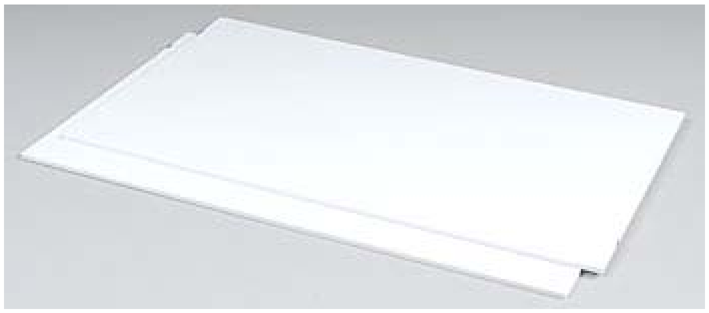


Figure 6: White Plastic Sheets

[1] **FSUIPC for Advanced Users** by Peter L. Dowson pp 19-25

I decided to use a 7” x 12” x 1/8” thick white plastic sheet for my panel fronts mounted on a wooden frame. I cut all four panels from one sheet. Each panel is 7” x 3” which leaves 1/2” on each side for mounting holes. Because the sheet is white, I can paint it black, and then use my hobby CNC mill to engrave the lettering. The paint is removed by the engraving leaving the white plastic showing through so the letters stand out. A layer of automotive clear coat preserves the paint and prevents it from chipping. I also plan to use my CNC mill to mill the switch cutouts.

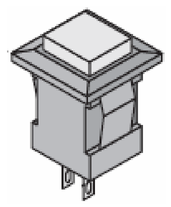


Figure 7: Push Button Switch

The push-button switches I purchased from Mouser Electronics lock into a square cutout and have solder terminals. They are the perfect size for my panels.

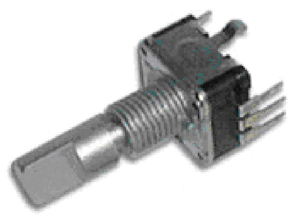


Figure 8: Rotary Encoder with Push-Button Switch

Mouser also sells rotary encoders that contain a push-button switch for a very reasonable price. I needed 11 for this project.

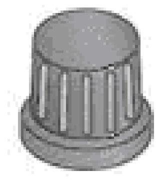


Figure 9: Rotary Encoder Knob

I chose knobs from Mouser Electronics to fit the D-shaft of the rotary encoders. I wanted to paint the front of the knob to correspond to functions, but it had a rubber coating which I found difficult to paint, so I just left them black.

Panel Design

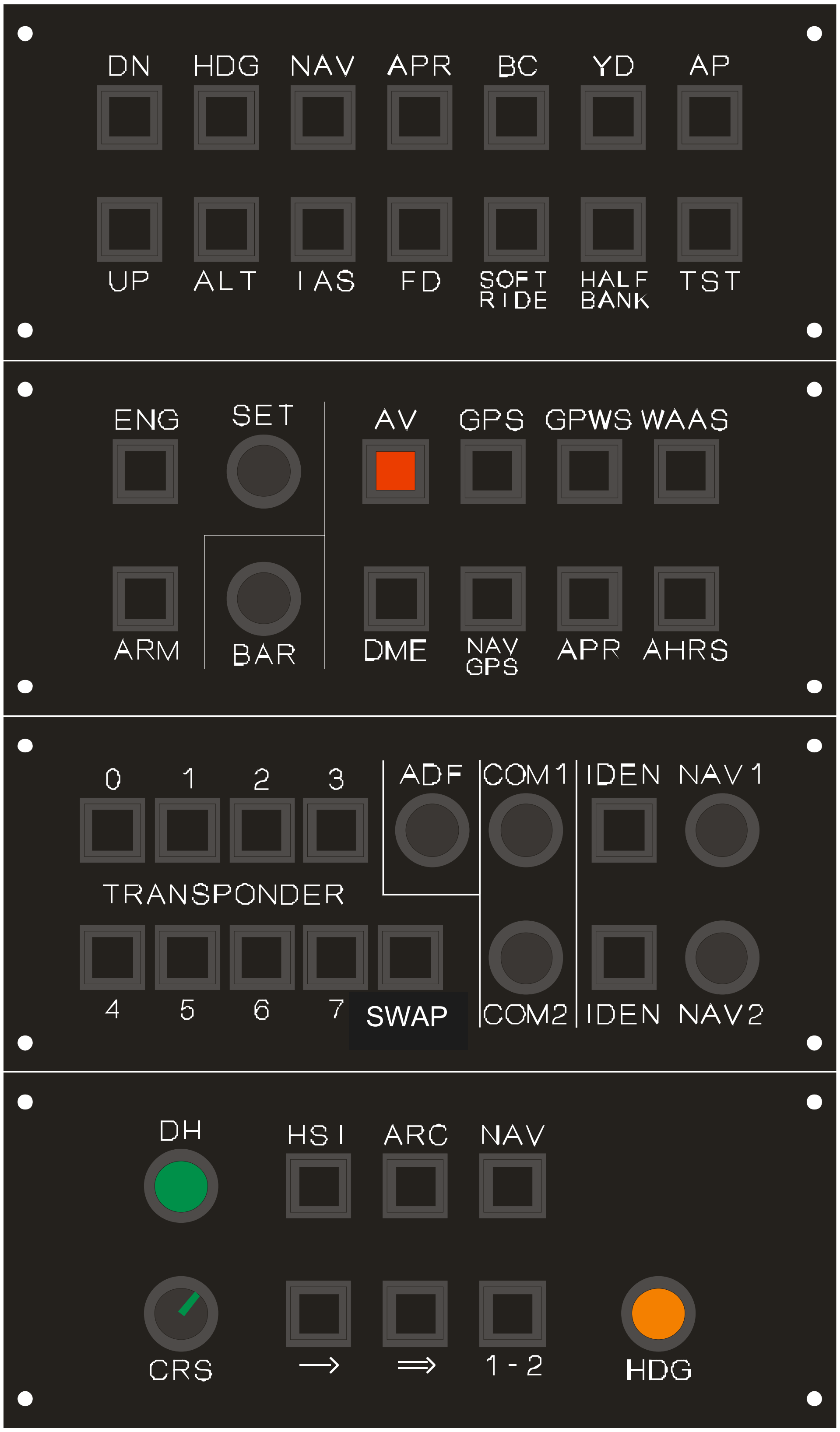


Figure 10: Panel Layout, Left Side

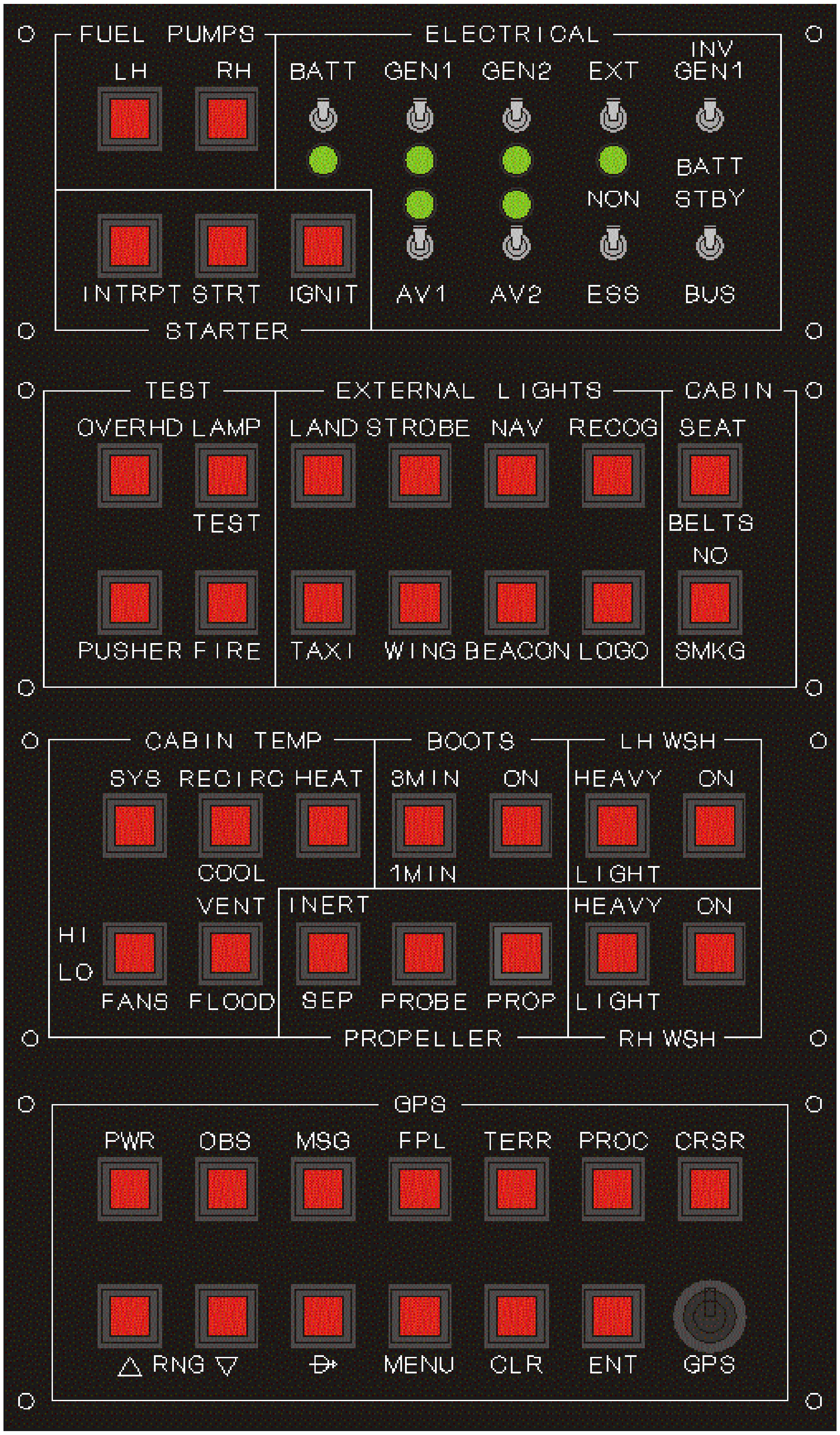


Figure 11: Panel Layout, Right Side

You will notice that my right side layout uses red switches because Mouser had run out of black.

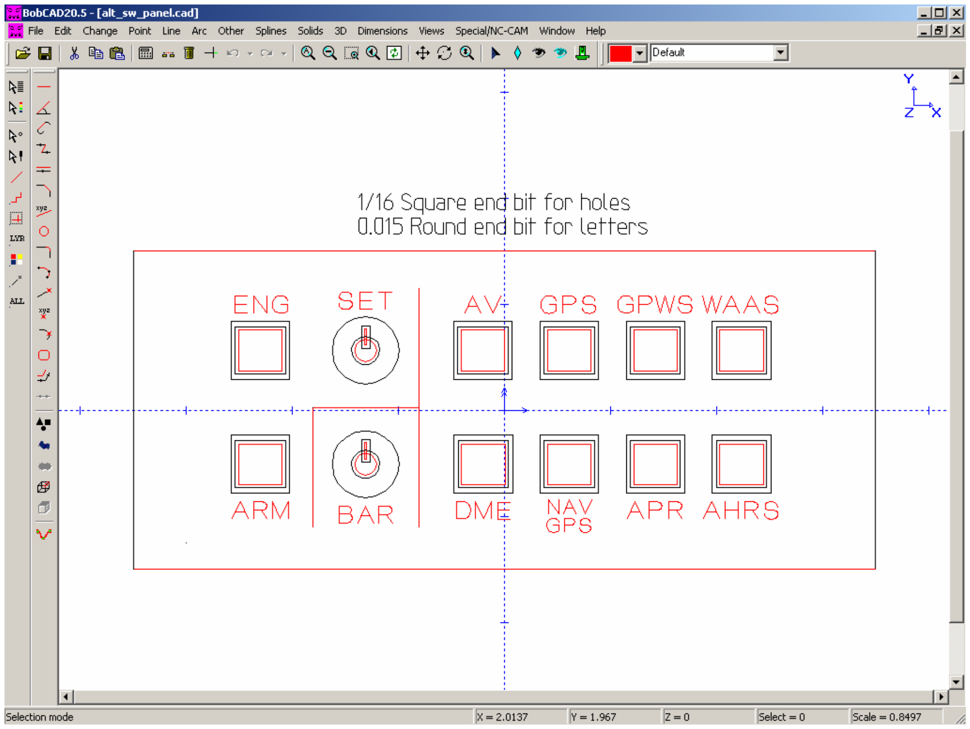

BobCAD20.5 - [alt_sw_panel.cad]
File Edit Change Point Line Arc Other Splines Solids 3D Dimensions Views Special/NC-CAM Window Help
Default
1/16 Square end bit for holes
0.015 Round end bit for letters
ENG
SET
AV
GPS
GPWS
WAAS
ARM
BAR
DME
NAV GPS
APR
AHRS
Selection mode
X = 2.0137
Y = 1.967
Z = 0
Select = 0
Scale = 0.8497

Figure 12: CAD Drawing

I used CorelDraw12 to create the layout of each panel, then exported the layout to BobCAD-CAM v20, my CAD/CAM program. The red lines in the CAD drawing represent a tool path. The rotary switch has a post to prevent the switch from rotating in the panel, so I had to mill a corresponding notch above the mounting hole. I used a 1/16" square end milling bit to cut out the holes, and a 0.015" round end bit to engrave the lettering and lines.

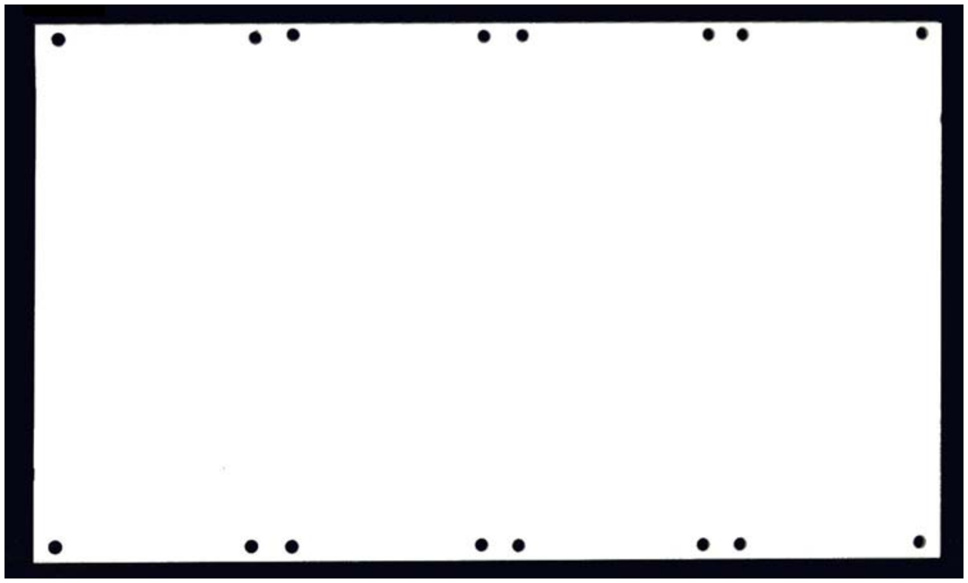


Figure 13: Mounting Holes Drilled

Figure 14: Panel Ready for Milling

First I drilled all the mounting holes, then I cut each panel on my table saw. I put masking tape on each side of the plastic before cutting to prevent the plastic from chipping. I sprayed on a layer of flat black which is the final color of the panels, then I sprayed on a layer of automotive clear-coat to help preserve the paint and keep it from peeling/chipping due to handling.

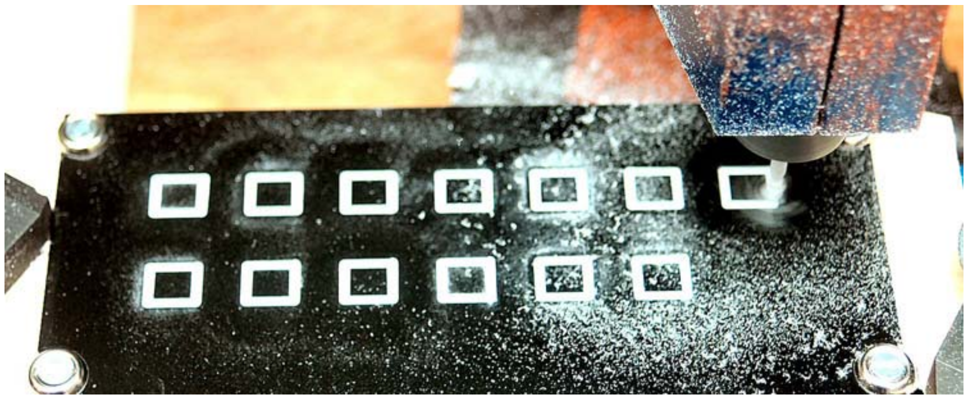


Figure 15: Milling Switch Cutouts

I mounted the plastic sheet onto a flat piece of wood, which I then mounted to the milling table. I programmed my CNC mill to first mill the switch cutouts. Then I installed the 0.015” round end bit and loaded the program to cut the engraving. After I milled the engraving I gave the panel another shot of clear coat.

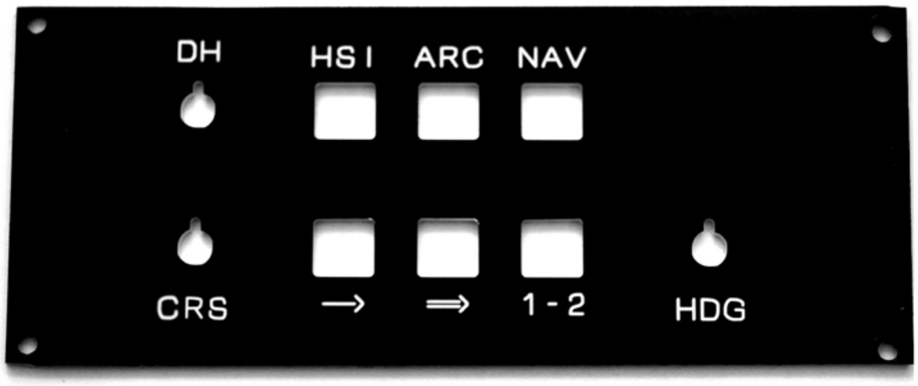


Figure 16: EFIS Panel Ready for Switches

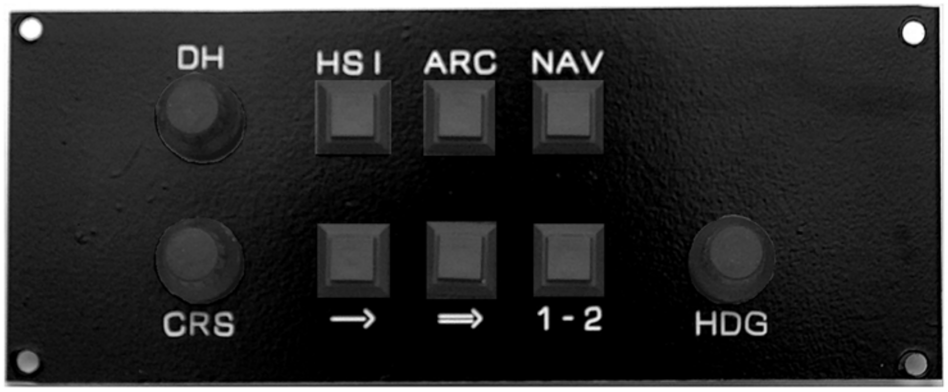


Figure 17: EFIS Panel with Switches Installed

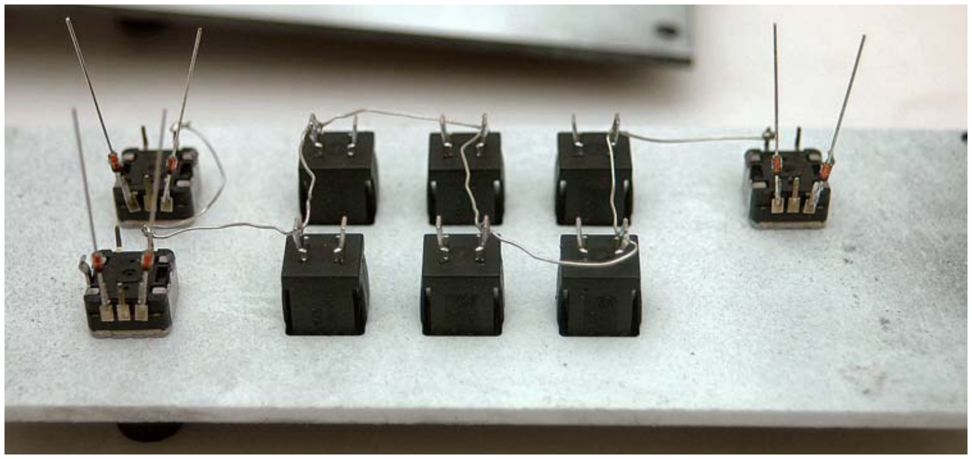


Figure 18: Wiring of Switches

After I installed the switches I wired all the common connectors together using bus wire. The common of all push button switches are tied to row 4 of the switch matrix. Notice in the above photo the diodes connected to the rotary switch connections; these will be connected to the rotary switch connections on rows 1 – 3 of the switch matrix.

Building the Boxes

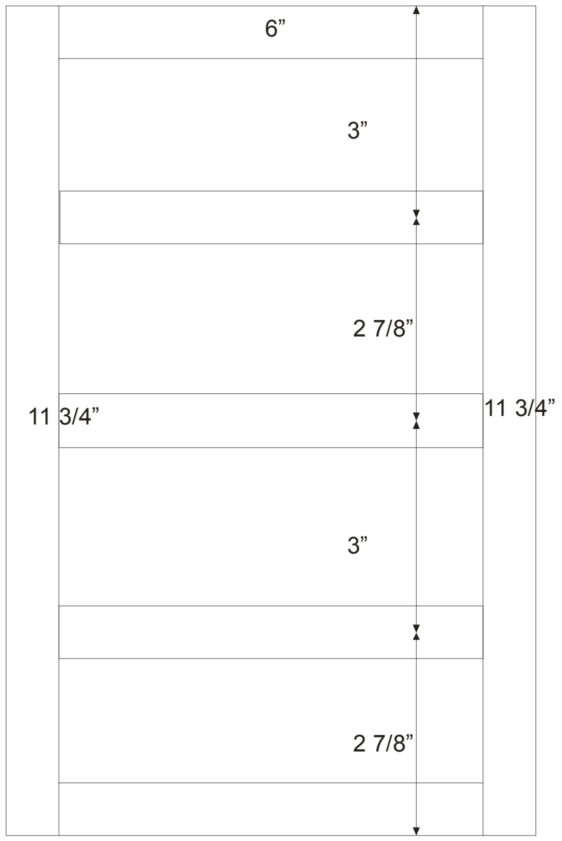


Figure 19: Box Dimensions

I built two boxes of the same dimensions. I purchased an eight-foot piece of 1” x 6” (which was really 3/4“ x 5 1/2“) and an eight-foot piece of 1” x 3” (which was really 3/4“ x 2 1/2“) to build the box. I used 1” x 6” to make the top, bottom, and sides, and used the 1” x 3” to make the center supports. The center supports are not to keep the box rigid, but to prevent the panels from flexing when I press a button.

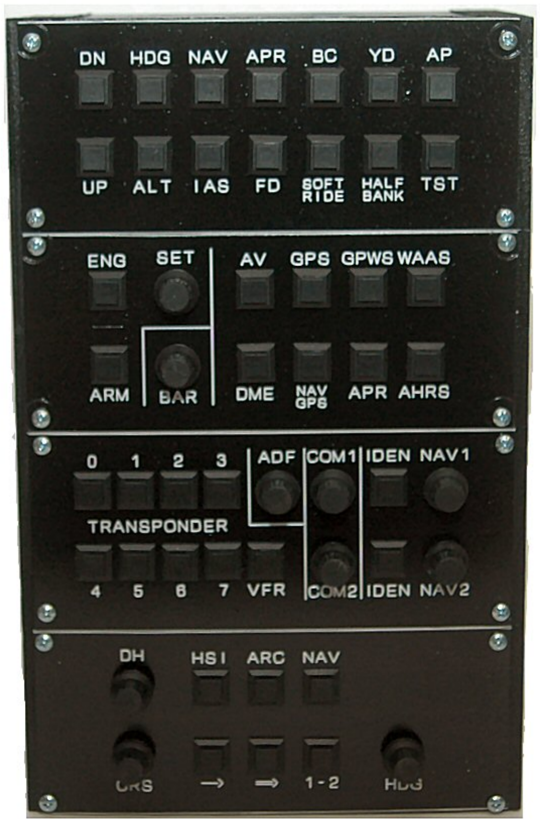


Figure 20: Panels Mounted on the Left Box

When I assembled the box, I used wallboard screws and butted the joints. I didn't do anything fancy like mitered or dovetailed joints. After all, the joints are covered up by the panels. I painted the box flat black, then sprayed it with clear coat. I aligned the panels, drilled mounting holes, then mounted the panels with #6 pan head wood screws. I then connected the two boxes together again using short wallboard screws.

Wiring to the Matrix

I decided to make the panels modular, so rather than wire them directly to the matrix board, I wired them to D sub-miniature connectors which I purchased from Radio Shack. In this way I could easily remove one panel in case I needed to perform troubleshooting or maintenance, or make any changes. The following table shows the connectors I used; refer to Appendix A for the complete wiring table.

Panel	Connector
Auto Pilot	DB-15
Alt/VS/SW	DB-25
Radios	DB25 and DB-9
EFIS	DB25
Fuel/Electrical/Starter	DB-15
Test/Lights/Cabin	DB-15
Temperature/Deice	DB-15
GPS	DB-25

Table 4: Panel Connectors

I wanted to maximize space on my board so I took the switch combination table and sorted it by the number of diode connections, i.e. combinations of only one diode, then two diodes, etc. Each row of diodes on the board contains up to 8 diodes (because there are 8 switch lines/columns). The anode lead of each diode is connected to a switch column, and the cathode lead (the one with the black line) is connected to one contact of a push-button switch. Since each row in the matrix contains 8 diodes, I next organized the table by number of switches that can be contained in one row of 8 diodes. For example, in the table in Appendix A the first 8 switches connect to only one diode so I put those 8 diodes in diode row 1. Next I found four combinations of 2 diodes that would fit in one row, and so on. When I was all finished I needed 40 rows of diodes. There was enough space on the board that I added two additional diode rows. Because I only needed one switch connection on row 40 but it had space for two switches, I had 5 spare switch connections not used.

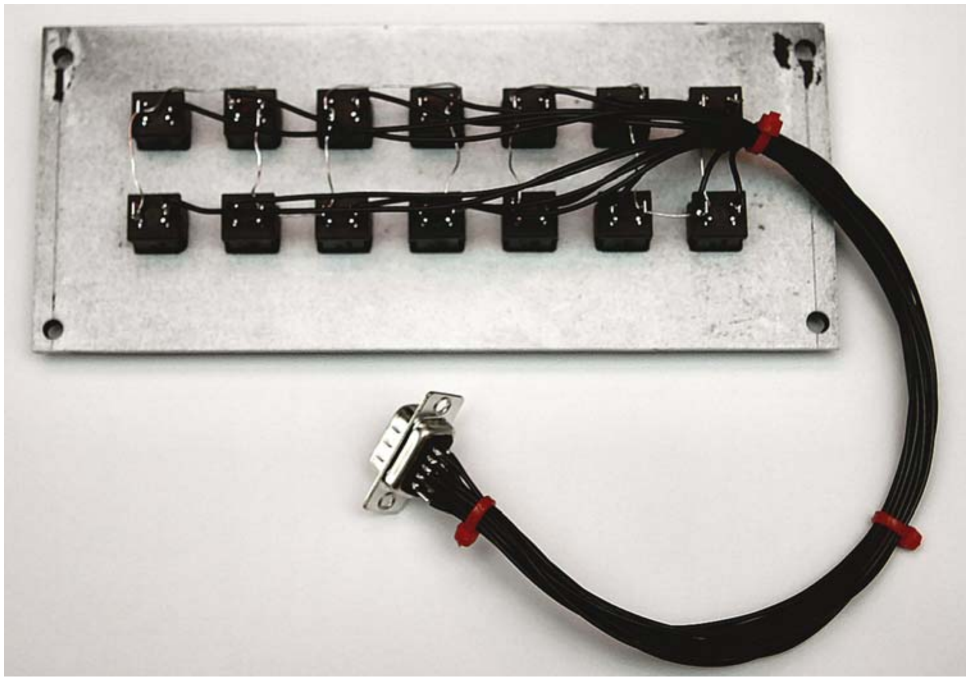


Figure 21: Auto Pilot Panel Wiring Completed

I wired the male connectors to the panels, and the female connectors to the matrix. Because I had multiple DB-15 and DB-25 connectors, I used different colored cable ties to match the appropriate connectors.

Toggle Switches

I discovered the hard way that I needed to treat the toggle switches specially. A push-button switch is a momentary contact switch. It is closed only while being pressed. When I remove my finger from the button contact is broken. A toggle switch is different because once toggled it stays in that position. Because of that I couldn't wire the toggle switch directly to the diode matrix because once toggled it would keep one or more of the column lines low all the time. Therefore the toggle switches needed to be isolated from the diode matrix.

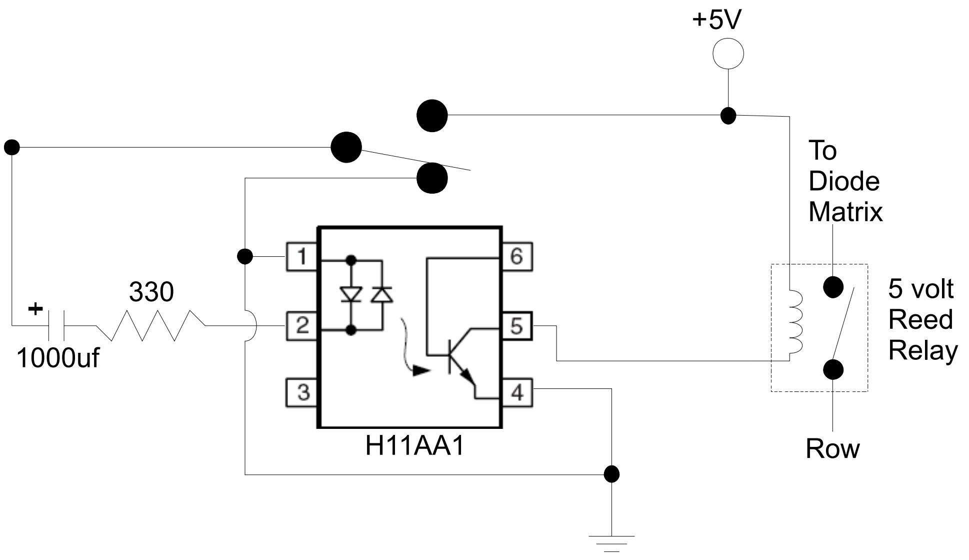


Figure 22: Toggle Switch Pulse Circuit

One way to isolate a toggle switch from the diode matrix is to use an opto-coupler. Plus, in order to function properly with Flight Simulator a toggle switch must send a pulse. The above figure shows the schematic of a sample pulse circuit using a H11AA1 opto-coupler to trigger a reed relay. The reed relay takes the place of the push-button switch. Toggling the switch up applies +5 volts to the positive side of the 1000uF capacitor. For a short amount of time the capacitor is shorted causing +5 volts to be applied to pin 2 which causes the right photo-diode to conduct. This causes the photo-transistor to conduct which causes the relay to close. As the capacitor charges the +5 volts on pin 2 eventually goes away turning off the photo-diode which turns off the photo-transistor which causes the relay to open. The capacitor now has +5 volts on the positive side because it is now fully charged.

When the toggle switch is moved down the positive side of the capacitor is connected to ground which causes the capacitor to discharge. For a short amount of time +5 volts is applied to pin 1 which causes the left photo-diode to conduct and triggers the relay. As the capacitor discharges the +5 volts on pin 1 eventually goes away which causes the left photo-diode to stop conducting and the relay opens. The time to charge or discharge the capacitor is determined by the size of the capacitor and the 330-ohm resistor which also controls the pulse length. I have 9 toggle switches in my electrical panel, but only 8 are connected.

Rather than use the opto-coupler to create a pulse I decided to program the pulse in the MCU. This saved me a few dollars in parts and some time in connecting the switches. I connected the 8 toggle switches to 8 separate inputs on the MCU. Each toggle switch is connected to one pin. I programmed the MCU so that if it sees only one of these toggle switch inputs change it sends a pulse to the PC. This way I get a pulse each time the toggle switch is moved; up or down. This pulse is also used for the rotary switches.

The 8 input pins I'm using for the toggle switches don't have an internal pull-up resistor, so to make the toggle switches work correctly I connected the normally closed (NC) side of each switch to ground, and the normally-open (NO) side of the switch to +5 volts through a 10K-ohm pull up resistor. The common of each toggle switch is connect to an MCU input. I didn't build my circuit board to accommodate this so I had to hard-wire my changes to the board. It would have been better to put the pull-up resistors on the board, but I had to put them on the toggle switches instead.

It turns out I didn't even need to program the pulse in the MCU. FSUIPC automatically sends a pulse to Flight Simulator. However, if I didn't program the pulse in the MCU I would have had to program two lines in FSUIPC for each toggle switch; a pulse when the switch is moved up, and another pulse when the switch is moved down.

Rotary Encoder Switch

A rotary encoder switch is actually two switches in one. It has two switched outputs, A and B, and one common connection. As the switch is turned the A and B outputs change state, but only one output changes at a time, called gray coding. Gray coding for a rotary switch is shown below.

Gray coding for clockwise rotation		
Phase	**A**	**B**
1	1	0
2	1	1
3	0	1
4	0	0

Gray coding for counter-clockwise rotation		
Phase	**A**	**B**
1	0	0
2	0	1
3	1	1
4	1	0

The common pin is connected to a row, and the A-B pins are each connected to a column. To decode direction I basically said if pin A changed state and it was different from pin B the switch moved clockwise. If pin A changed state and it was the same as pin B the switch moved counter-clockwise. Since each rotary switch is connected to two columns, I made the right column pulse if the switch was moving clockwise; I made the left column pulse if the switch was moving counter-clockwise. I also decided to count every fourth change of the rotary switch. That way I couldn't turn the switch faster than the microcontroller could process the switch changes.

Printed Circuit Board Design and Fabrication

I could have hard-wired the diode matrix using a pre-punched circuit board but I decided to make my own board instead. I've had very good luck using photo-sensitive boards.

There are a number of methods for producing PCB's at home. The key to the process is being able to transfer the artwork onto the copper board in preparation for etching. I don't have a laser printer so the iron transfer method wouldn't work for me. Another method is to make a silk-screen stencil, but I didn't want to spend the $20 for two sheets of stencil, and whatever it would cost for the proper paint. Plus, since this method was new to me, I didn't want to spend the extra money to experiment, learn and possibly make mistakes.

I've had good luck in the past with photo-sensitive boards so I decided to go with this method. I purchased a positive photo-sensitive board and the developer solution from Mouser. I probably should have purchased 2 boards in case I messed one up, but I was on a tight budget for this project.

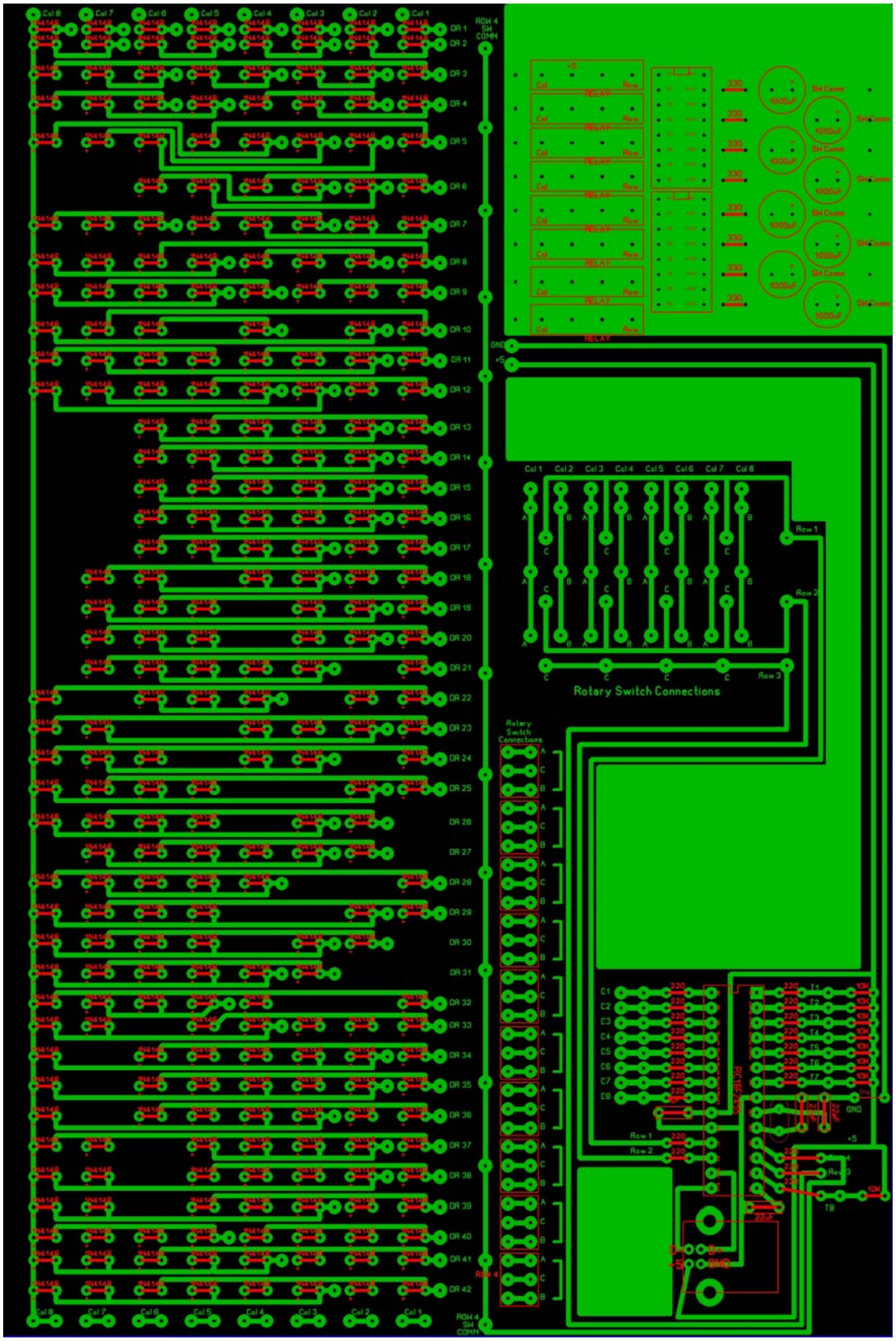


Figure 23: PC-12 Interface Board Corrected Artwork

I found a PCB design program on the Internet called Abacom Sprint Layout 50. The above figure shows my corrected PCB design. While populating and testing the board I found some errors in the artwork tht I correct. The actual size of the board was 6” x 9”. 50% of the board is taken up by the 42 rows of the diode matrix, each row having up to 8 diodes. The lower right corner contains the artwork for the microcontroller, support components, and the USB socket.

The upper right corner is laid out for 8 opto-coupler pulse circuits, but, as mentioned before, I decided to do the pulse with programming instead, so this area is solid copper. In the right middle are the circuits for connecting up to 12 rotary switches in a 3 x 8 matrix. The rotary switches do not connect to the diode matrix; the diode matrix is only for row 4. I used red to indicate the outline of the components to ensure they would fit on the board and not interfere with each other. The large green areas are unused space on the board. Since this space was unused I decided not to remove the copper from those areas; this saves some time and some of the etching chemical.

I printed a mirror image of the PCB etch side on 3M transparency CG3480. This transparency is specifically designed for use with ink-jet printers. My artwork printed heavy and sharp.

Following the instructions that came with the board, I laid the artwork print side down (which is why I had to print a mirror image) onto the sensitized side of the board, attached it with a piece of tape, and covered it with a piece of glass I removed from an old picture frame. I positioned it five inches below a florescent light and exposed it for 10 minutes, then placed the board into the developer solution I diluted with ten parts cold tap water. I constantly wiped the board with a foam brush I purchased from my local hardware store. The development process took about 6 minutes. After the board was developed I washed off the excess developer in running tap water; now the board was ready to etch.

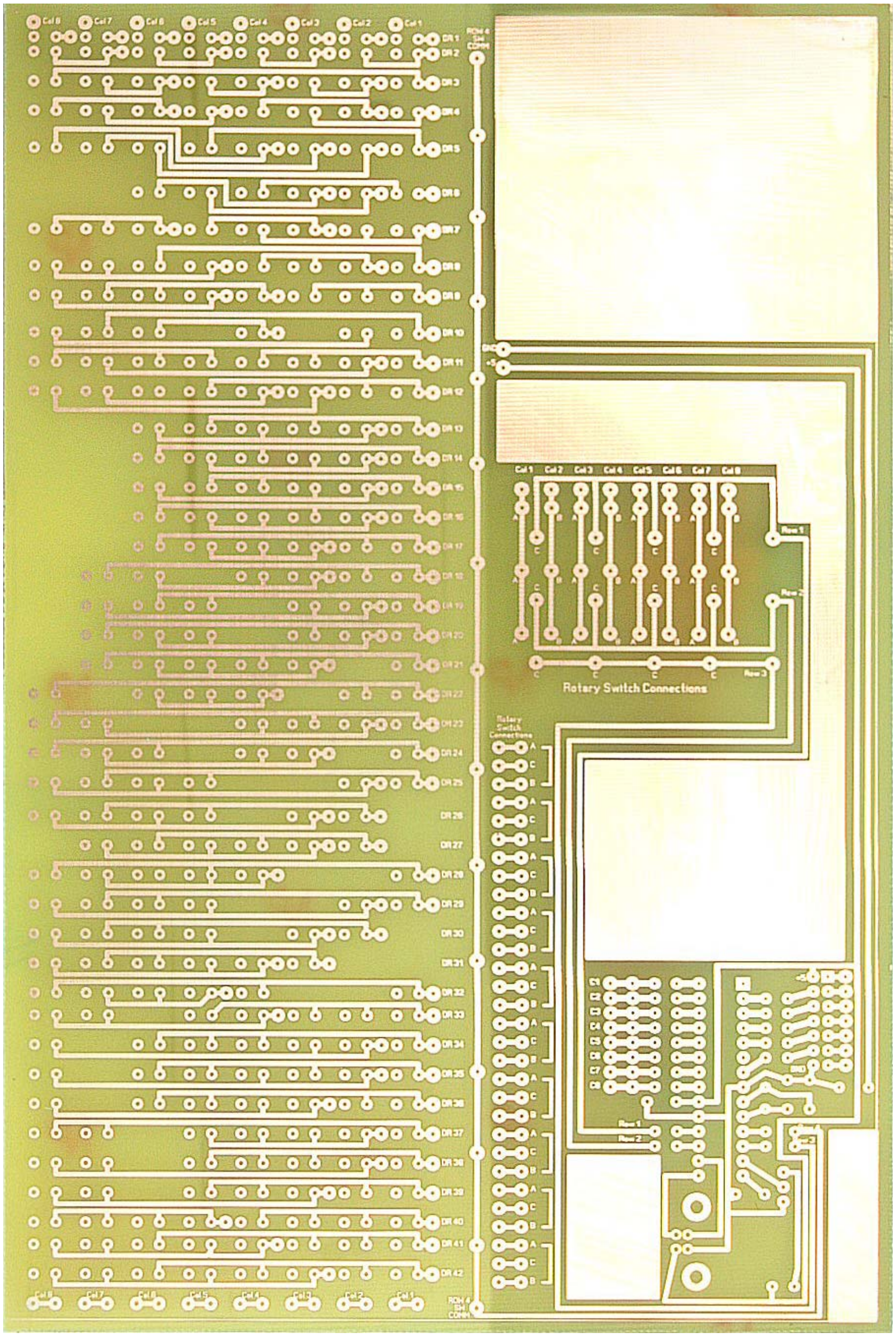


Figure 24: Etched PCB from Original Uncorrected Artwork

I purchased a bottle of PCB Etchant Solution from Radio Shack #276-1535, and using a glass tray, etched my circuit board. This process took about 30 minutes to remove the excess copper. I used a hair dryer to warm the solution and keep it moving over the board which sped up the etching process.

Ok, so now I had to drill the holes for the components. I am fortunate to have a MAXNC 10 CL CNC hobby mill that I use for fabricating parts. Abacom Sprint Layout 50 allows me to export all of the component holes as a plotter file which I import into my CAD/CAM program, BobCAD-CAM V20, and use to drill the holes in the board.

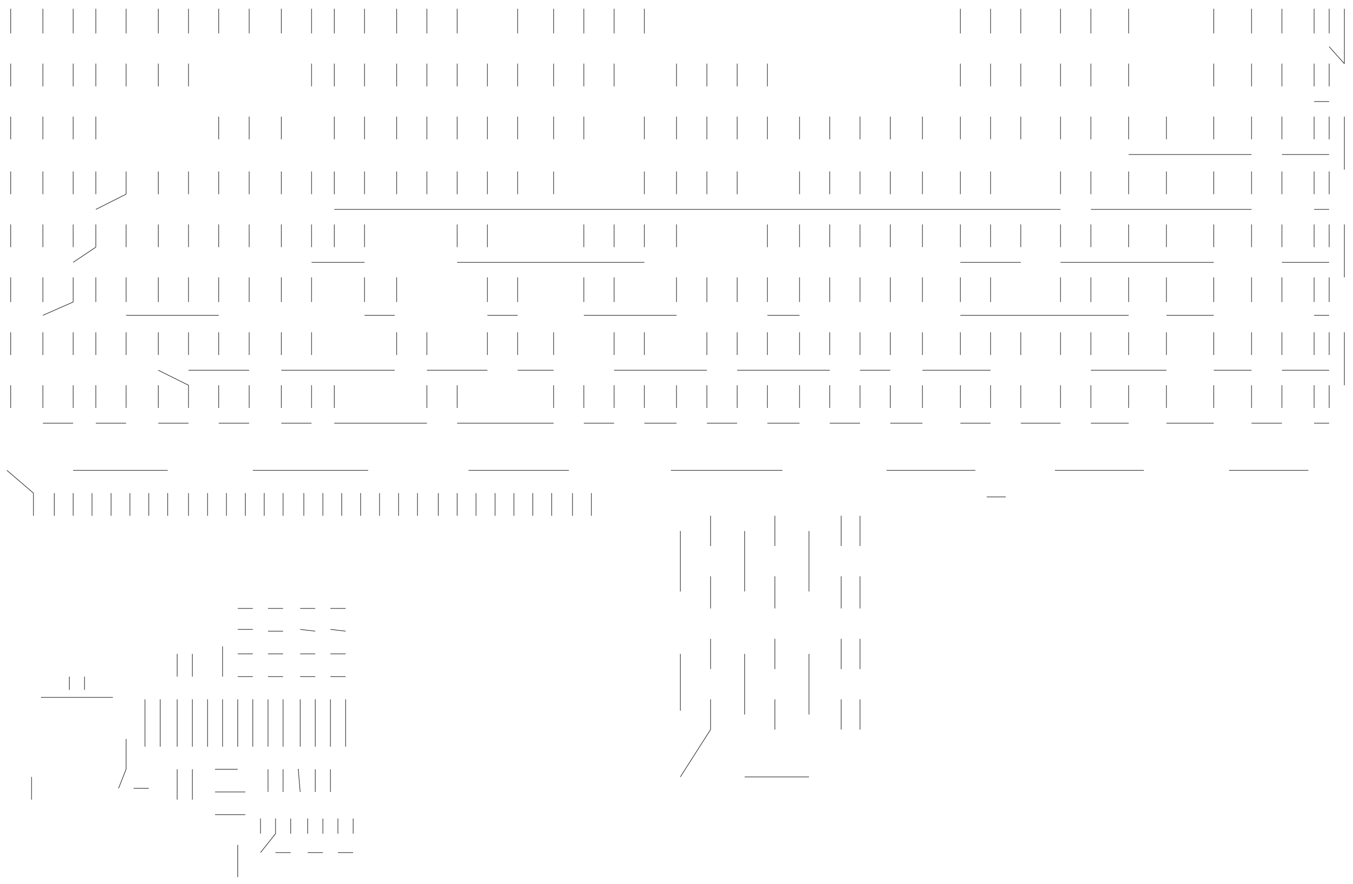


Figure 25: Corel Draw® Plot

It's actually not quite that easy. I first imported the plotter file into Corel Draw® 12. Each of the holes gets imported as a node. I then connected two nodes with a line. BobCAD-CAM V20 doesn't recognize the nodes if I import the plotter file directly, but it does recognize the lines I created with Corel Draw® 12.

]

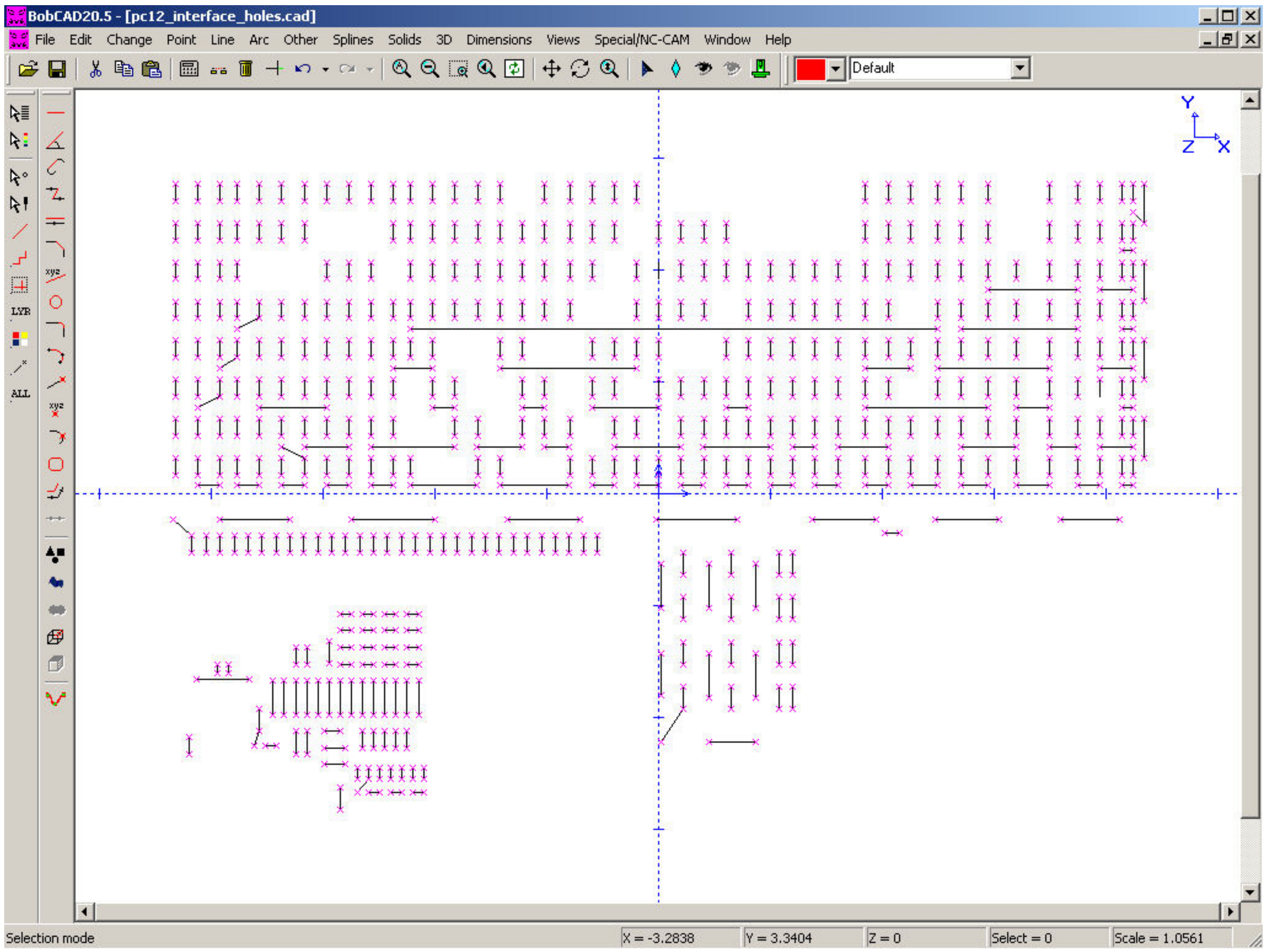

BobCAD20.5 - [pc12_interface_holes.cad]
File Edit Change Point Line Arc Other Splines Solids 3D Dimensions Views Special/NC-CAM Window Help
Default
Selection mode
X = -3.2838
Y = 3.3404
Z = 0
Select = 0
Scale = 1.0561

Figure 26: BobCAD-CAM Drilling

Next I exported the lines to a drawing file, which I then imported into BobCAD-CAM V20. At the end of each line I placed a point; each point became a hole location which I programmed with the CAM portion of the software. In the above figure you can see the points and lines.

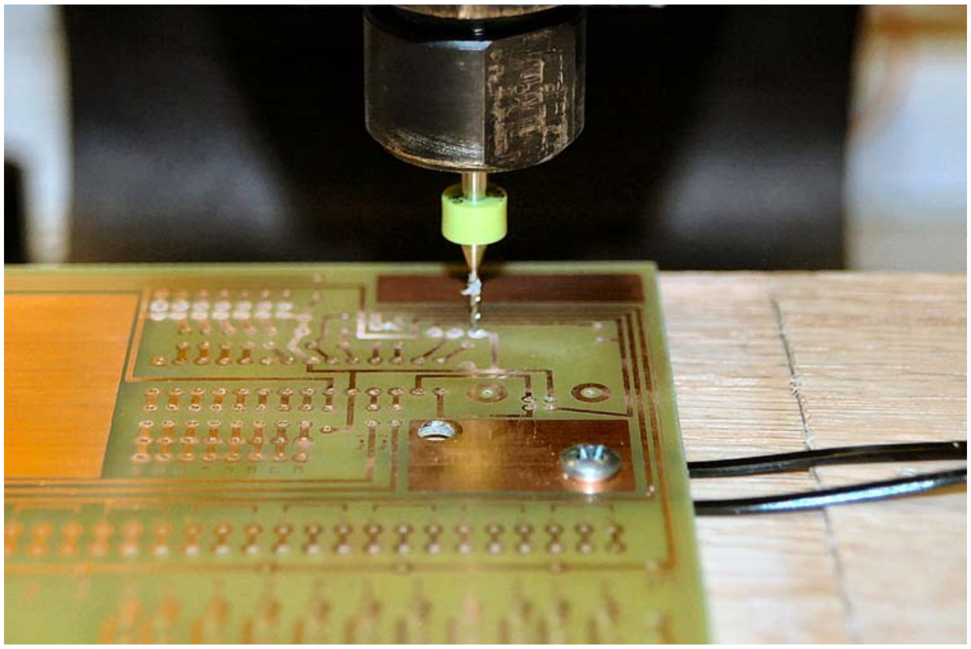


Figure 27: Drilling PCB Holes

I purchased a circuit board maker kit from Drill Bit City that contained the correct size bits I needed to drill the component holes. I drilled holes in the corners of my board and mounted it to a piece of wood using screws, the wood was mounted to the cross-slide table of the mill. Using my MAXCNC 10CL mill and the CAM program I created I drilled the component holes in the circuit board.

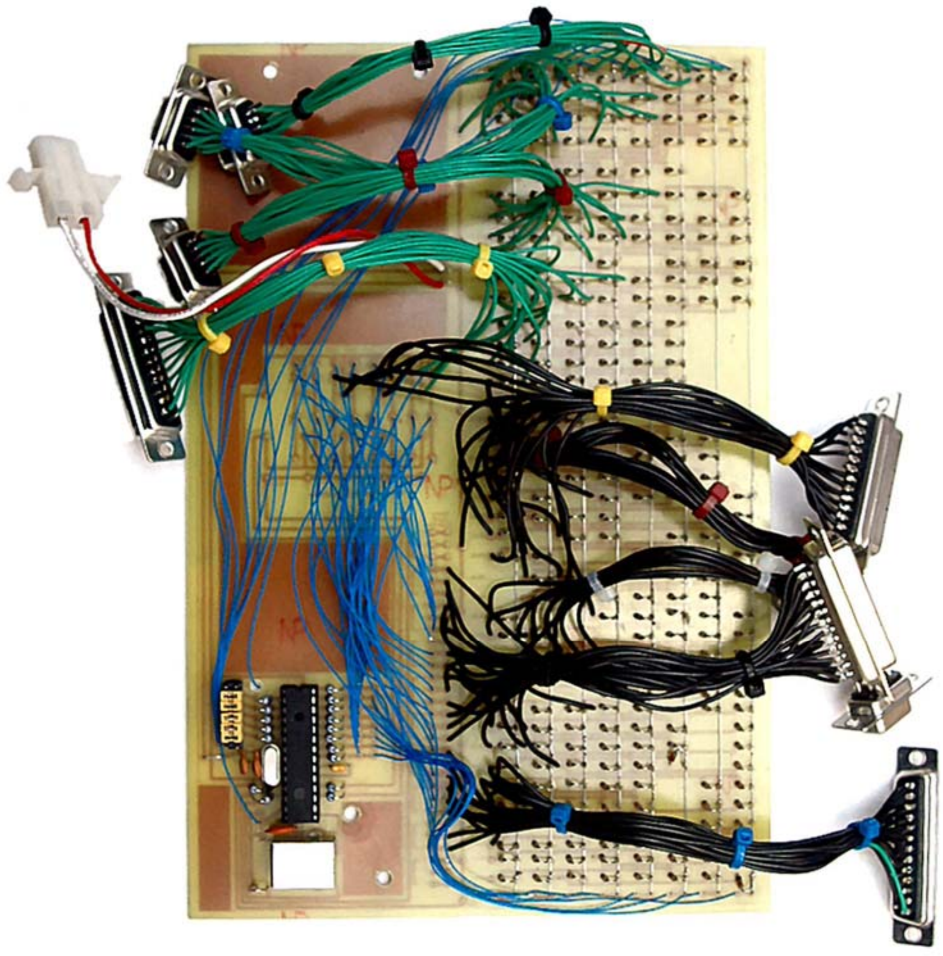


Figure 28: Completed PCB

Once the holes were drilled I populated it with the components I purchased from Mouser. I mounted a 28-pin DIP socket onto the board so the MCU wouldn't be permanently installed in case I needed to reprogram it. As I was populating the board I discovered a number of artwork errors. Plus I didn't accommodate the toggle switches correctly so I had to make some changes to the board using 30-gauge wire.

In the process of testing my panels I found that I had installed one diode backwards. Not bad for soldering 265 diodes!

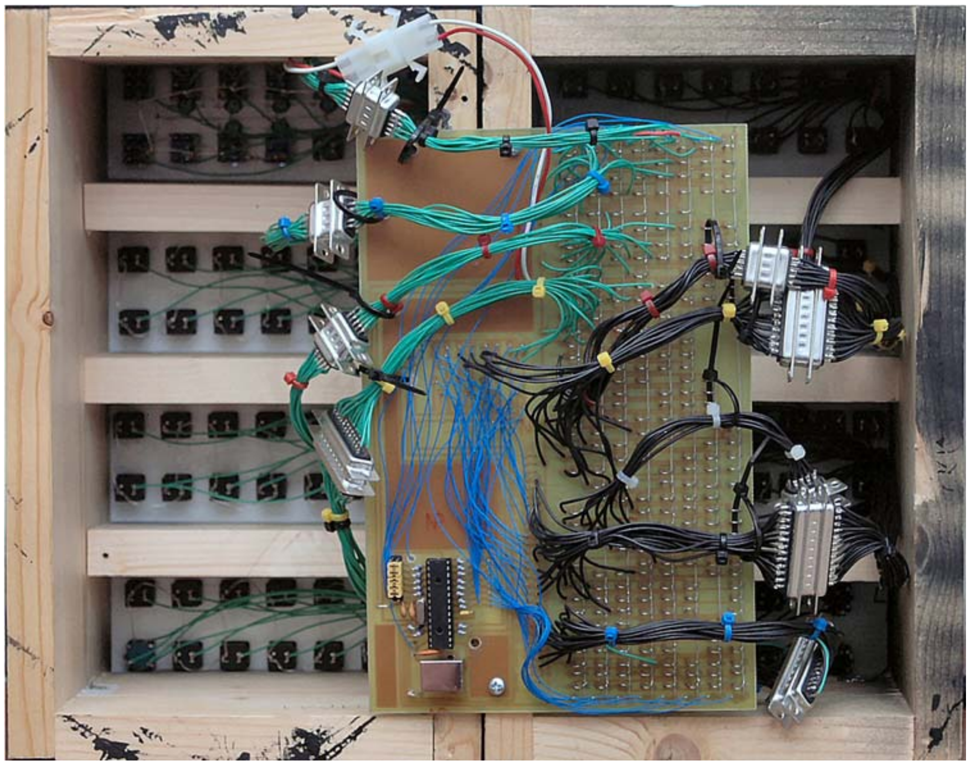
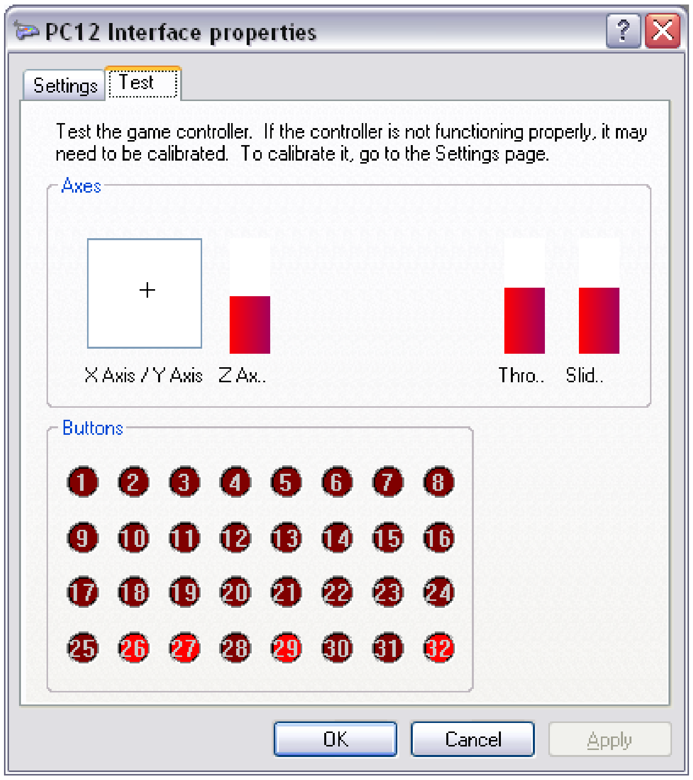


Figure 29: Back of Panels Box

On the right you can see the diode matrix. Each row corresponds to a row as laid out in my wiring table in Appendix A. I used bus wire I purchased from Radio Shack to wire the vertical columns. The anode of each diode is connected to one of these column busses. The D subminiature connectors are wired directly to the diode matrix. In the center of the board are connections for the rotary switches which are then wired to the 3x8 matrix on the left. This matrix is connected to the MCU via etching. Finally, coming from the left side of the MCU are the connections for the 8 toggle switches. I used the large holes in the board for mounting to a jig to drill the component holes, but these are also used to mount the board onto the rear of my box.



Test the game controller. If the controller is not functioning properly, it may need to be calibrated. To calibrate it, go to the Settings page.

Figure 30: Switch Test

After I wired each panel and the corresponding diode matrix I tested each button. I clicked on Control Panel, clicked on Game Controllers, selected the PC12 Interface, then clicked on Properties. Because I also declared 5 axes, X, Y, Z, throttle and slider you can see them appear. The above figure shows the result of pressing a button on one of the panels. This switch activates row 4, columns 2, 3, 5, and 8 which corresponds to switches 26, 27, 29, and 32 as shown. When programming FSUIPC I had to remember that the switches were numbered 0-7, not 1-8. This was just a little confusing and caused some interesting results when I made mistakes with the programming. I had the program set up to generate a pulse so the switches wouldn't actually stay lit as shown in the above figure; instead they would pulse, i.e. stay lit for only a short duration.

Programming

Appendix B contains the BASIC code for my microcontroller. There are two files involved; the USB descriptor and the PC12 interface.

USB Descriptor

This was probably the most difficult for me to get right. The USB descriptor is used during the enumeration process to identify the peripheral device to the host, i.e. identify the microcontroller to the PC. This has to be correct, otherwise the microcontroller will not be recognized and you will get a device error message. What I am building with the microcontroller is a human interface device (HID) which requires a specific type of descriptor. I did a lot of research on the Internet regarding USB HID descriptors. I've listed those references here:

- USB Implementers Forum, Inc. www.usb.org – This is where you can find the USB interface specifications and everything else you ever wanted to know about USB. The most useful thing I found here was the HID Descriptor Tool which allows you to build the report descriptor. It also comes with examples of different types of report descriptors. You can use these report descriptors instead of the generic report descriptor that the mikroBASIC Pro descriptor tool built.
- Amr Bekhit, http://www.helmpcb.com/Electronics/USBJoystick/USBJoystick.aspx - This article explains how to convert an old joystick that used a joystick port to one that uses a USB port using a microcontroller. **This article provided my first breakthrough on creating a valid joystick HID descriptor.**
- USB Made Simple, http://www.usbmadesimple.co.uk/index.html - A series of articles on how USB works. The descriptor explanation was outstanding.
- USB in a Nutshell, http://www.beyondlogic.org/usbnutshell/ - Great article on how USB works, especially good explanation of the descriptor.

Creating a valid USB descriptor was probably the most difficult part of the process. It took me a couple of weeks of research and failure until I got it right. Rather than go into the gory details here, I wrote a little document to help others avoid the problems I ran into called PIC and USB which you can download and read for yourself. The important thing to remember is that in the report descriptor you must declare whole bytes, i.e. exact multiples of 8 bits, otherwise the MCU will not connect and communicate with the PC.

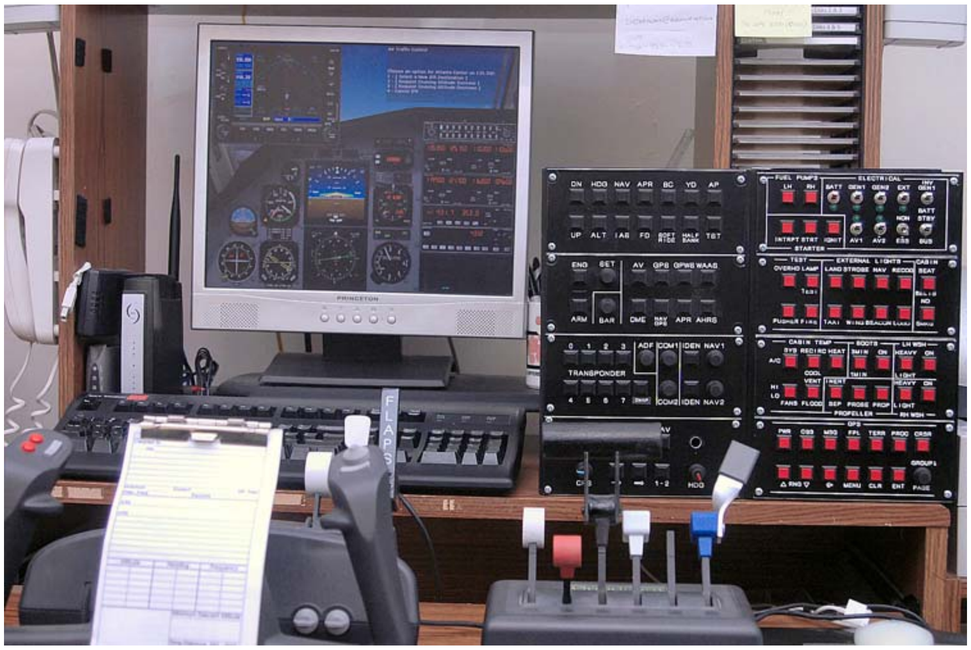


Figure 31: PC12 Panels Inplace

Appendix A: Wiring Table

Panel	Function	D-Sub Connector	3, 24 Col 8	3, 25 Col 7	3, 26 Col 6	3, 27 Col 5	3, 28 Col 4	3, 29 Col 3	3, 30 Col 2	3, 31 Col 1	Diode Row
FUEL/ELEC/START	BATT	15-1	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟩 1	1
FUEL/ELEC/START	GEN1	15-2	🟥	🟥	🟥	🟥	🟥	🟥	🟩 1	🟥	1
FUEL/ELEC/START	GEN2	15-3	🟥	🟥	🟥	🟥	🟥	🟩 1	🟥	🟥	1
FUEL/ELEC/START	EXT	15-4	🟥	🟥	🟥	🟥	🟩 1	🟥	🟥	🟥	1
FUEL/ELEC/START	AV1	15-5	🟥	🟥	🟥	🟩 1	🟥	🟥	🟥	🟥	1
FUEL/ELEC/START	AV2	N/C									
FUEL/ELEC/START	INV	15-7	🟥	🟥	🟩 1	🟥	🟥	🟥	🟥	🟥	1
FUEL/ELEC/START	ESS	15-8	🟥	🟩 1	🟥	🟥	🟥	🟥	🟥	🟥	1
FUEL/ELEC/START	STBY	15-9	🟩 1	🟥	🟥	🟥	🟥	🟥	🟥	🟥	1
FUEL/ELEC/START	LH	15-10	🟥	🟥	🟥	🟥	🟥	🟥	🟩 1	🟩 1	2
FUEL/ELEC/START	RH	15-11	🟥	🟥	🟥	🟥	🟩 1	🟩 1	🟥	🟥	2
FUEL/ELEC/START	INTRPT	N/C									
FUEL/ELEC/START	STRT	15-13	🟥	🟥	🟩 1	🟩 1	🟥	🟥	🟥	🟥	2
FUEL/ELEC/START	IGNIT	15-14	🟩 1	🟩 1	🟥	🟥	🟥	🟥	🟥	🟥	2
FUEL/ELEC/START	SW Common	15-15									
FUEL/ELEC/START	+5	2-1									
FUEL/ELEC/START	GND	2-2									
TEST/LIGHTS/CABIN	OVERHD	15-1	🟥	🟥	🟥	🟥	🟥	🟩 1	🟩 1	🟥	3
TEST/LIGHTS/CABIN	LAMP TEST	15-2	🟥	🟥	🟥	🟩 1	🟩 1	🟥	🟥	🟥	3
TEST/LIGHTS/CABIN	LAND	15-3	🟥	🟩 1	🟩 1	🟥	🟥	🟥	🟥	🟥	3
TEST/LIGHTS/CABIN	STROBE	15-4	🟩 1	🟥	🟥	🟥	🟥	🟥	🟥	🟩 1	3
TEST/LIGHTS/CABIN	NAV	15-5	🟥	🟥	🟥	🟥	🟥	🟩 1	🟥	🟩 1	4
TEST/LIGHTS/CABIN	RECOG	15-6	🟥	🟥	🟥	🟥	🟩 1	🟥	🟩 1	🟥	4
TEST/LIGHTS/CABIN	SEAT BELTS	15-7	🟥	🟩 1	🟥	🟩 1	🟥	🟥	🟥	🟥	4
TEST/LIGHTS/CABIN	PUSHER	15-8	🟩 1	🟥	🟩 1	🟥	🟥	🟥	🟥	🟥	4
TEST/LIGHTS/CABIN	FIRE	15-9	🟥	🟥	🟥	🟩 1	🟥	🟥	🟥	🟩 1	5
TEST/LIGHTS/CABIN	TAXI	15-10	🟥	🟥	🟩 1	🟥	🟥	🟥	🟩 1	🟥	5
TEST/LIGHTS/CABIN	WING	15-11	🟥	🟩 1	🟥	🟥	🟥	🟩 1	🟥	🟥	5
TEST/LIGHTS/CABIN	BEACON	15-12	🟩 1	🟥	🟥	🟥	🟩 1	🟥	🟥	🟥	5
TEST/LIGHTS/CABIN	LOGO	15-13	🟥	🟥	🟥	🟥	🟩 1	🟥	🟥	🟩 1	6
TEST/LIGHTS/CABIN	NO SMKG	15-14	🟥	🟥	🟥	🟩 1	🟥	🟥	🟩 1	🟥	6
TEST/LIGHTS/CABIN	SW Common	15-15									
TEMP/DEICE	SYS	15-1	🟥	🟥	🟩 1	🟥	🟥	🟩 1	🟥	🟥	6
TEMP/DEICE	RECIRC	15-2	🟥	🟥	🟥	🟩 1	🟥	🟩 1	🟥	🟥	7
TEMP/DEICE	HEAT	15-3	🟥	🟥	🟥	🟥	🟩 1	🟥	🟩 1	🟩 1	7
TEMP/DEICE	3MIN	15-4	🟩 1	🟩 1	🟩 1	🟥	🟥	🟥	🟥	🟥	7
TEMP/DEICE	BOOTS ON	15-5	🟥	🟥	🟩 1	🟥	🟥	🟥	🟥	🟩 1	8
TEMP/DEICE	LH HEAVY	15-6	🟥	🟥	🟥	🟥	🟩 1	🟩 1	🟩 1	🟥	8
TEMP/DEICE	LH ON	15-7	🟩 1	🟩 1	🟥	🟩 1	🟥	🟥	🟥	🟥	8
TEMP/DEICE	FANS	15-8	🟩 1	🟥	🟥	🟩 1	🟥	🟥	🟥	🟥	9
TEMP/DEICE	VENT	15-9	🟥	🟥	🟥	🟥	🟥	🟩 1	🟩 1	🟩 1	9
TEMP/DEICE	INERT SEP	15-10	🟥	🟩 1	🟩 1	🟥	🟩 1	🟥	🟥	🟥	9
TEMP/DEICE	PROBE	N/C									
TEMP/DEICE	PROP	15-12	🟥	🟥	🟩 1	🟥	🟩 1	🟥	🟥	🟥	10
TEMP/DEICE	RH HEAVY	15-13	🟥	🟩 1	🟥	🟥	🟥	🟥	🟥	🟩 1	10
TEMP/DEICE	RH ON	15-14	🟩 1	🟥	🟥	🟥	🟥	🟥	🟩 1	🟥	10

TEMP/DEICE	SW Common	15-15									
GPS	PWR	N/C									
GPS	OBS	25-2	🟥	🟩 1	🟥	🟥	🟥	🟥	🟩 1	🟥	11
GPS	MSG	25-3	🟥	🟥	🟥	🟥	🟩 1	🟩 1	🟥	🟩 1	11
GPS	FPL	25-4	🟩 1	🟥	🟩 1	🟩 1	🟥	🟥	🟥	🟥	11
GPS	TERR	25-5	🟥	🟩 1	🟥	🟥	🟩 1	🟥	🟥	🟥	12
GPS	PROC	25-6	🟩 1	🟥	🟥	🟥	🟥	🟩 1	🟥	🟥	12
GPS	CRSR	25-7	🟥	🟥	🟥	🟩 1	🟥	🟥	🟩 1	🟩 1	12
GPS	RNG ▲	25-8	🟥	🟥	🟥	🟩 1	🟥	🟩 1	🟥	🟩 1	13
GPS	RNG ▼	25-9	🟥	🟥	🟩 1	🟥	🟩 1	🟥	🟩 1	🟥	13
GPS	DIRECT	25-10	🟥	🟥	🟥	🟩 1	🟥	🟩 1	🟩 1	🟥	14
GPS	MENU	25-11	🟥	🟥	🟩 1	🟥	🟩 1	🟥	🟥	🟩 1	14
GPS	CLR	25-12	🟥	🟥	🟥	🟩 1	🟩 1	🟥	🟥	🟩 1	15
GPS	ENT	25-13	🟥	🟥	🟩 1	🟥	🟥	🟩 1	🟩 1	🟥	15
GPS	GROUP/PAGE SW	25-14	🟥	🟥	🟥	🟩 1	🟩 1	🟥	🟩 1	🟥	16
GPS	GROUP/PAGE	25-15, 16, 17									
GPS	SW Common	25-25									
AP	DN	15-1	🟥	🟥	🟩 1	🟥	🟥	🟩 1	🟥	🟩 1	16
AP	HDG	15-2	🟥	🟥	🟥	🟩 1	🟩 1	🟩 1	🟥	🟥	17
AP	NAV	15-3	🟥	🟥	🟩 1	🟥	🟥	🟥	🟩 1	🟩 1	17
AP	APR	15-4	🟥	🟥	🟩 1	🟥	🟩 1	🟩 1	🟥	🟥	18
AP	BC	15-5	🟥	🟩 1	🟥	🟥	🟥	🟥	🟩 1	🟩 1	18
AP	YD	15-6	🟥	🟥	🟩 1	🟩 1	🟥	🟥	🟥	🟩 1	19
AP	AP	15-7	🟥	🟩 1	🟥	🟥	🟥	🟩 1	🟩 1	🟥	19
AP	UP	15-8	🟥	🟥	🟩 1	🟩 1	🟥	🟥	🟩 1	🟥	20
AP	ALT	15-9	🟥	🟩 1	🟥	🟥	🟥	🟩 1	🟥	🟩 1	20
AP	IAS	15-10	🟥	🟥	🟩 1	🟩 1	🟥	🟩 1	🟥	🟥	21
AP	FD	15-11	🟥	🟩 1	🟥	🟥	🟩 1	🟥	🟥	🟩 1	21
AP	SOFT	15-12	🟥	🟥	🟩 1	🟩 1	🟩 1	🟥	🟥	🟥	22
AP	HALF	15-13	🟩 1	🟥	🟥	🟥	🟥	🟥	🟩 1	🟩 1	22
AP	TST	15-14	🟥	🟩 1	🟥	🟥	🟩 1	🟥	🟩 1	🟥	23
AP	SW Common	15-15									
ALT/VS/SW	ENG	25-1	🟩 1	🟥	🟥	🟥	🟥	🟩 1	🟥	🟩 1	23
ALT/VS/SW	ARM	25-2	🟥	🟩 1	🟥	🟥	🟩 1	🟩 1	🟥	🟥	24
ALT/VS/SW	SET SW	25-3	🟩 1	🟥	🟩 1	🟥	🟥	🟥	🟥	🟩 1	24
ALT/VS/SW	BAR SW	N/C									
ALT/VS/SW	AV	25-5	🟥	🟩 1	🟥	🟩 1	🟥	🟥	🟥	🟩 1	25
ALT/VS/SW	GPS	25-6	🟩 1	🟥	🟩 1	🟥	🟥	🟥	🟩 1	🟥	25
ALT/VS/SW	GPWS	N/C									
ALT/VS/SW	WAAS	N/C									
ALT/VS/SW	DME	25-9	🟥	🟩 1	🟥	🟩 1	🟥	🟥	🟩 1	🟥	26
ALT/VS/SW	NAV/GPS	25-10	🟩 1	🟥	🟩 1	🟥	🟥	🟩 1	🟥	🟥	26
ALT/VS/SW	APR	25-11	🟥	🟩 1	🟥	🟩 1	🟥	🟩 1	🟥	🟥	27
ALT/VS/SW	AHRS	25-12	🟥	🟩 1	🟥	🟩 1	🟩 1	🟥	🟥	🟥	28
ALT/VS/SW	SET	25-14, 15, 16									
ALT/VS/SW	BAR	25-17, 18, 19									
ALT/VS/SW	SW Common	25-25									
AVIONICS	0	9-1	🟩 1	🟥	🟥	🟩 1	🟥	🟥	🟥	🟩 1	28
AVIONICS	1	9-2	🟥	🟩 1	🟩 1	🟥	🟥	🟥	🟥	🟩 1	29
AVIONICS	2	9-3	🟩 1	🟥	🟥	🟩 1	🟥	🟥	🟩 1	🟥	29
AVIONICS	3	9-4	🟥	🟩 1	🟩 1	🟥	🟥	🟥	🟩 1	🟥	30

AVIONICS	4	9-5	1	🟥	🟥	1	🟥	1	🟥	🟥	30
AVIONICS	5	9-6	🟥	1	1	🟥	🟥	1	🟥	🟥	31
AVIONICS	6	9-7	1	🟥	🟥	1	1	🟥	🟥	🟥	31
AVIONICS	7	9-8	🟥	1	1	1	🟥	🟥	🟥	🟥	32
AVIONICS	SWAP	9-9	1	🟥	🟥	🟥	1	🟥	🟥	1	32
AVIONICS	ADF FAST	25-1	1	1	🟥	🟥	1	🟥	🟥	🟥	33
AVIONICS	COM1 FAST	25-2	🟥	🟥	🟥	1	🟥	1	1	1	33
AVIONICS	COM2 FAST	25-3	1	🟥	🟥	🟥	🟥	1	1	🟥	34
AVIONICS	NAV1 IDEN	25-4	🟥	🟥	1	1	1	🟥	🟥	1	34
AVIONICS	NAV1 FAST	25-5	1	🟥	🟥	🟥	1	🟥	1	🟥	35
AVIONICS	NAV2 IDEN	25-6	🟥	🟥	1	1	🟥	1	🟥	1	35
AVIONICS	NAV2 FAST	25-7	1	🟥	🟥	🟥	1	1	🟥	🟥	36
AVIONICS	ADF	25- 8, 9, 10									
AVIONICS	COM1	25-11, 12, 13									
AVIONICS	COM2	25-14, 15, 16									
AVIONICS	NAV1	25-17, 18, 19									
AVIONICS	NAV2	25-20, 21, 22									
AVIONICS	SW Common	25-25									
EFIS	DH SW	25-1	🟥	🟥	1	1	🟥	🟥	1	1	36
EFIS	HSI	25-2	1	1	🟥	🟥	🟥	🟥	🟥	1	37
EFIS	ARC	25-3	🟥	🟥	🟥	1	1	1	1	🟥	37
EFIS	NAV	25-4	1	1	🟥	🟥	🟥	🟥	1	🟥	38
EFIS	->	25-5	🟥	🟥	🟥	1	1	1	🟥	1	38
EFIS	=>	25-6	1	1	🟥	🟥	🟥	1	🟥	🟥	39
EFIS	1-2	25-7	🟥	🟥	🟥	1	1	🟥	1	1	39
EFIS	HDG FAST	25-8	🟥	🟥	🟥	🟥	1	1	1	1	40
EFIS	CRS FAST	25-13	1	1	1	1	🟥	🟥	🟥	🟥	40
EFIS	DH	25-14, 15, 16									
EFIS	CRS	25-17, 18, 19									
EFIS	HDG	25-20, 21, 22									
EFIS	SW Common	25-25									

Appendix B: Program Code

Descriptor File

```
module PC12dsc

const USB_VENDOR_ID as word = 0x89DD
const USB_PRODUCT_ID as word = 0x0002
const USB_SELF_POWER as char = 0xA0            ' Self powered 0xC0,  0x80 bus powered
'const USB_SELF_POWER as char = 0x80            ' Self powered 0xC0,  0x80 bus powered
const USB_MAX_POWER as char = 50               ' Bus power required in units of 2 mA
const HID_INPUT_REPORT_BYTES as char = 64
const HID_OUTPUT_REPORT_BYTES as char = 64
const EP_IN_INTERVAL as char = 1
const EP_OUT_INTERVAL as char = 1

const USB_INTERRUPT as char = 1
const USB_TRANSFER_TYPE as char = 0x03         '0x03 Interrupt
const USB_HID_EP as char = 1
const USB_HID_RPT_SIZE as char = 85

structure device_descriptor
    dim bLength as char               ' bLength         - Descriptor size in bytes (12h)
    dim bDescriptorType as char       ' bDescriptorType - The constant DEVICE (01h)
    dim bcdUSB as word                ' bcdUSB          - USB specification release number
                                        (BCD)
    dim bDeviceClass as char          ' bDeviceClass    - Class Code
    dim bDeviceSubClass as char       ' bDeviceSubClass - Subclass code
    dim bDeviceProtocol as char       ' bDeviceProtocol - Protocol code
    dim bMaxPacketSize0 as char       ' bMaxPacketSize0 - Maximum packet size for endpoint 0
    dim idVendor as word              ' idVendor        - Vendor ID
    dim idProduct as word             ' idProduct       - Product ID
    dim bcdDevice as word             ' bcdDevice       - Device release number (BCD)
    dim iManufacturer as char         ' iManufacturer   - Index of string descriptor for the
                                        manufacturer
    dim iProduct as char              ' iProduct        - Index of string descriptor for the
                                        product.
    dim iSerialNumber as char         ' iSerialNumber   - Index of string descriptor for the
                                        serial number.
    dim bNumConfigurations as char    ' bNumConfigurations - Number of possible
                                        configurations
end structure

const device_dsc as device_descriptor = (
  0x12,                  ' bLength
  0x01,                  ' bDescriptorType
  0x0200,                ' bcdUSB
  0x00,                  ' bDeviceClass
  0x00,                  ' bDeviceSubClass
  0x00,                  ' bDeviceProtocol
  8,                     ' bMaxPacketSize0
  USB_VENDOR_ID,         ' idVendor
  USB_PRODUCT_ID,        ' idProduct
  0x0003,                ' bcdDevice - Version number
  0x01,                  ' iManufacturer
  0x02,                  ' iProduct
  0x03,                  ' iSerialNumber
  0x01                   ' bNumConfigurations
)

' Configuration 1 Descriptor
const configDescriptor1 as byte[41] = (
    ' Configuration Descriptor
    0x09,                   ' bLength             - Descriptor size in bytes
    0x02,                   ' bDescriptorType     - The constant CONFIGURATION (02h)
    0x29,0x00,              ' wTotalLength        - The number of bytes in the configuration
                                      descriptor and all of its subordinate descriptors
    1,                      ' bNumInterfaces      - Number of interfaces in the configuration
    1,                      ' bConfigurationValue - Identifier for Set Configuration and Get
                                      Configuration requests
    0,                      ' iConfiguration      - Index of string descriptor for the
                                      configuration
    USB_SELF_POWER,         ' bmAttributes        - Self/bus power and remote wakeup settings
    USB_MAX_POWER,          ' bMaxPower           - Bus power required in units of 2 mA

    ' Interface Descriptor
    0x09,                   ' bLength - Descriptor size in bytes (09h)
    0x04,                   ' bDescriptorType - The constant Interface (04h)
    0,                      ' bInterfaceNumber - Number identifying this interface
    0,                      ' bAlternateSetting - A number that identifies a descriptor with
                                      alternate settings for this bInterfaceNumber.
    2,                      ' bNumEndpoint - Number of endpoints supported not counting
                                      endpoint zero
    0x03,                   ' bInterfaceClass - Class code
    0,                      ' bInterfaceSubclass - Subclass code
    0,                      ' bInterfaceProtocol - Protocol code
    0,                      ' iInterface - Interface string index

    ' HID Class-Specific Descriptor
    0x09,                   ' bLength         - Descriptor size in bytes.
    0x21,                   ' bDescriptorType - This descriptor's type: 21h to indicate the
                                      HID class.
    0x01,0x01,              ' bcdHID          - HID specification release number (BCD).
    0x00,                   ' bCountryCode    - Numeric expression identifying the country
                                      for localized hardware (BCD) or 00h.
    1,                      ' bNumDescriptors - Number of subordinate report and physical
                                      descriptors.
    0x22,                   ' bDescriptorType - The type of a class-specific descriptor that
                                      follows
    USB_HID_RPT_SIZE,0x00,  ' wDescriptorLength - Total length of the descriptor identified
                                      above.

    ' Endpoint Descriptor
    0x07,                   ' bLength - Descriptor size in bytes (07h)
    0x05,                   ' bDescriptorType - The constant Endpoint (05h)
    USB_HID_EP or 0x80,     ' bEndpointAddress - Endpoint number and direction
    USB_TRANSFER_TYPE,      ' bmAttributes - Transfer type and supplementary information
    0x40,0x00,              ' wMaxPacketSize - Maximum packet size supported
    EP_IN_INTERVAL,         ' bInterval - Service interval or NAK rate

    ' Endpoint Descriptor
    0x07,                   ' bLength - Descriptor size in bytes (07h)
    0x05,                   ' bDescriptorType - The constant Endpoint (05h)
    USB_HID_EP,             ' bEndpointAddress - Endpoint number and direction
    USB_TRANSFER_TYPE,      ' bmAttributes - Transfer type and supplementary information
    0x40,0x00,              ' wMaxPacketSize - Maximum packet size supported
    EP_OUT_INTERVAL         ' bInterval - Service interval or NAK rate
)

structure hid_report_descriptor
  dim report as char[USB_HID_RPT_SIZE]
end structure

const hid_rpt_desc as hid_report_descriptor = (
(0x05, 0x01,                    ' USAGE_PAGE (Generic Desktop)
 0x15, 0x00,                    ' LOGICAL_MINIMUM (0)
 0x09, 0x04,                    ' USAGE (Joystick)
 0xA1, 0x01,                    ' COLLECTION (Application)
 0x05, 0x01,                    '   USAGE_PAGE (Generic Desktop)
 0x05, 0x09,                    '   USAGE_PAGE (Button)
 0x19, 0x01,                    '   USAGE_MINIMUM (Button 1)
 0x29, 0x20,                    '   USAGE_MAXIMUM (Button 32)
 0x15, 0x00,                    '   LOGICAL_MINIMUM (0)
 0x25, 0x01,                    '   LOGICAL_MAXIMUM (1)
 0x75, 0x01,                    '   REPORT_SIZE (1)
 0x95, 0x20,                    '   REPORT_COUNT (32)
 0x55, 0x00,                    '   UNIT_EXPONENT (0)
 0x65, 0x00,                    '   UNIT (None)
 0x81, 0x02,                    '   INPUT (Data,Var,Abs)
 0x05, 0x01,                    '   USAGE_PAGE (Generic Desktop)
 0x09, 0x01,                    '   USAGE (Pointer)
 0x15, 0x00,                    '   LOGICAL_MINIMUM (0)
 0x26, 0xFF, 0x00,              '   LOGICAL_MAXIMUM (255)
 0x75, 0x08,                    '   REPORT_SIZE (8)
 0xA1, 0x00,                    '   COLLECTION (Physical)
 0x09, 0x30,                    '     USAGE (X)
 0x09, 0x31,                    '     USAGE (Y)
 0x09, 0x32,                    '     USAGE (Z)
 0x95, 0x03,                    '     REPORT_COUNT (3)
 0x81, 0x02,                    '     INPUT (Data Var Abs)
 0xC0,                          '   END_COLLECTION
 0x05,0x02,                     '   USAGE_PAGE (Simulation Controls)
 0x09,0xBB,                     '   USAGE (Throttle)
 0x15, 0x00,                    '   LOGICAL_MINIMUM (0)
 0x26, 0xFF, 0x00,              '   LOGICAL_MAXIMUM (255)
 0x75, 0x08,                    '   REPORT_SIZE (8)
 0x95, 0x01,                    '   REPORT_COUNT (1)
 0x81, 0x02,                    '   INPUT (Data Var Abs)
 0x05, 0x01,                    '   USAGE_PAGE (Generic Desktop)
 0x09, 0x36,                    '   USAGE (Slider)
 0x15, 0x00,                    '   LOGICAL_MINIMUM (0)
 0x26, 0xFF, 0x00,              '   LOGICAL_MAXIMUM (255)
 0x75, 0x08,                    '   REPORT_SIZE (8)
 0x95, 0x01,                    '   REPORT_COUNT (1)
 0x81, 0x02,                    '   INPUT (Data,Var,Abs)
 0xC0)                          ' END_COLLECTION
)

'Language code string descriptor
structure str1
  dim bLength as char
  dim bDscType as char
  dim wString as word[1]
end structure

const strd1 as str1 = (
  4,
  0x03,
  (0x0409)
)

'Manufacturer string descriptor
structure str2
  dim bLength as char
  dim bDscType as char
  dim wString as word[16]
end structure
```

```
const strd2 as str2 = (
  14,           'sizeof this descriptor string
  0x03,
  ("8"," ","B","i","t","s")
)

'Product string descriptor
structure str3
  dim bLength as char
  dim bDscType as char
  dim wString as word[23]
end structure

const strd3 as str3 = (
  30,           'sizeof this descriptor string
  0x03,
  ("P","C","1","2"," ","I","n","t","e","r","f","a","c","e")
)

'Serial number string descriptor
structure str4
  dim bLength as char
  dim bDscType as char
  dim wString as word[10]
end structure

const strd4 as str4 = (
  20,           'sizeof this descriptor string
  0x03,
  ("P","C","1","2","-","0","0","0","1")
)

dim USB_config_dsc_ptr as ^const char[1]

dim USB_string_dsc_ptr as ^const char[4]

sub procedure USB_Init_desc()

implements
  sub procedure USB_Init_desc()
    USB_config_dsc_ptr[0] = @configDescriptor1
    USB_string_dsc_ptr[0] = ^const char(@strd1)
    USB_string_dsc_ptr[1] = ^const char(@strd2)
    USB_string_dsc_ptr[2] = ^const char(@strd3)
    USB_string_dsc_ptr[3] = ^const char(@strd4)
  end sub
end.
```

Program File

I am using three types of switches on my panels; each one has to be handled differently by the microcontroller. First I read rows 1 - 3. These rows are only connected to rotary switches so I first determine which direction the switch moved and toggle the appropriate bit in the USB write buffer. It is possible to leave a rotary switch in a position were contacts A and/or B are closed, i.e. connected to the common pin. Therefore, after a specific duration I reset the bits giving a pulse rather than leaving a bit set all the time.

Row 4 is only connected to momentary contact, push-button switches via the diode matrix previously described. I read row four and put the inverted data into the USB write buffer. When I press a switch the corresponding columns come on and stay on until I remove my finger from the button.

Finally I read the 8 toggle switch inputs and determine which one changed. Depending on which toggle switch changed I sent a pulse for that switch.

```
' *
' * Project name:
'     PC12 Interface
' * Copyright:
'     (c) Roy Seifert, 2011
' * Revision History:
'     20110427:
'       - initial release;
' * Description:
'     This firmware reads a 4 x 8 switch matrix.  The first 24 switches are
'     connected to rotary switches, the last 8 are tied to a diode matrix
'     which allows up to 255 switches to be connected.
'
' * Test configuration:
'     MCU:             PIC18F2455
'                      http://ww1.microchip.com/downloads/en/DeviceDoc/39632D.pdf
'     Dev.Board:       EasyPIC6
'                      http://www.mikroe.com/en/tools/easypic6/
'     Oscillator:      HS 8.000 MHz  (USB osc. is raised with PLL to 48.000MHz)
'     Ext. Modules:    on-board USB-HID
'                      http://www.mikroe.com/pdf/easypic6/easypic6_manual_v100.pdf
'     SW:              mikrobasic  PRO for PIC
'                      http://www.mikroe.com/en/compilers/mikrobasic/pro/pic/
' * NOTES:
'     (*) Be VERY careful about the configuration flags for the 18F2455 - there"s
'       so much place for mistake!
'     - Place jumpers J12 in the right position
'
' RA5, RA3 - RA0 analog inputs
' RB7 - RB0 input with pullup
' RC7 - RC6 poll output
' RC1 - RC0 poll output

program Rotary_Interface

   dim col1          as sbit at PORTB.7  ' Port declarations
   dim col2          as sbit at PORTB.6
   dim col3          as sbit at PORTB.5
   dim col4          as sbit at PORTB.4
   dim col5          as sbit at PORTB.3
   dim col6          as sbit at PORTB.2
   dim col7          as sbit at PORTB.1
   dim col8          as sbit at PORTB.0
   dim row1          as sbit at LATC.7   ' Rotary switch output
   dim row2          as sbit at LATC.6   ' Rotary switch output
   dim row3          as sbit at LATC.1   ' Rotary switch output
   dim row4          as sbit at LATC.0   ' Digital switch output
   dim toggle1       as sbit at PORTE.3  ' Toggle switch inputs
   dim toggle2       as sbit at PORTA.0
   dim toggle3       as sbit at PORTA.1
   dim toggle4       as sbit at PORTA.2
   dim toggle5       as sbit at PORTA.3
   dim toggle6       as sbit at PORTA.4
   dim toggle7       as sbit at PORTA.5
   dim toggle8       as sbit at PORTC.2
   dim adc_in        as word             ' Axis analog input
   dim switches      as byte[5]          ' Switch data
   dim oldswitches   as byte[5]          ' Old switch data
   dim userWR_buffer as byte[64] absolute 0x500         ' USB write buffer
   dim userRD_buffer as byte[64] absolute 0x540         ' USB read buffer
   dim temp          as byte             ' Temporary storage
   dim count         as integer          ' General purpose counter
   dim passcount     as integer          ' Used to clear switches
   dim pulse         as integer          ' Pulse duration
   dim swmoved       as boolean          ' Switch moved flag
   dim togglemoved   as boolean          ' Toggle switch moved flag

'*******************************************************************************
' Main Interrupt Routine
'*******************************************************************************
sub procedure interrupt
   USB_Interrupt_Proc
end sub
'*******************************************************************************

'*******************************************************************************
' Initialization Routine
'*******************************************************************************
sub procedure Init
  '---------------------------------------
  ' Disable interrupts
  '---------------------------------------
   INTCON  = 0x00                     ' Disable GIE, PEIE, TMR0IE, INT0IE, RBIE
   INTCON2 = 0x75                     ' Turn on PORTB pull-ups, RB interrupts high
                                      priority
   INTCON3 = 0xC0                     ' High priority interrupts
   RCON.IPEN = 0                      ' Disable Priority Levels on interrupts
   PIE1 = 0
   PIE2 = 0
   PIR1 = 0
   PIR2 = 0
   ADCON1 = 0x0F                      ' All analog inputs digital
   '---------------------------------------
   ' Ports Configuration
   ' RA5 - RA0 toggle switch inputs
   ' RB7 - RB0 input with pullup
   ' RC7 - RC6 poll output
   ' RC2 - toggle switch input
   ' RC1 - RC0 poll output
   ' RE3 - toggle switch input

   '---------------------------------------
   ' TRIS bits:  1 = input, 0 = output
   TRISA = %00111111                  ' PORTA 7:6 output, 5 input, 4 output, 3:0 input
   TRISB = %11111111                  ' PORTB all input
   TRISC = %00000100                  ' PORTC 7:3 output, 2 input, 1:0 output
```

```
'   TRISD = 0xFF
'   TRISE = 0x07

   LATA = 0x00
   LATB = 0x00
   LATC = 0xC3                         ' All rows high
'   LATD = 0
'   LATE = 0

   pulse = 6                           ' Adjust to change switch pulse duration
   swmoved = false                     ' Reset all flags
   togglemoved = false
   count = 0
end sub

'*******************************************************************************
' Main Program Routine
'*******************************************************************************
main:
   Init

   userWR_buffer[0] = 0                ' Initialize the USB write buffer
   userWR_buffer[1] = 0
   userWR_buffer[2] = 0
   userWR_buffer[3] = 0
   count = 0

   HID_Enable(@userRD_buffer, @userWR_buffer)

   while true

         ' Read axis inputs
         ' Read AN0
         adc_in = adc_read(0)
         adc_in = adc_in/4
         userWR_buffer[4] = adc_in

         ' Read AN1
         adc_in = adc_read(1)
         adc_in = adc_in/4
         userWR_buffer[5] = adc_in

         ' Read AN2
         adc_in = adc_read(2)
         adc_in = adc_in/4
         userWR_buffer[6] = adc_in

         ' Read AN3
         adc_in = adc_read(3)
         adc_in = adc_in/4
         userWR_buffer[7] = adc_in

         ' Read AN4
         adc_in = adc_read(4)
         adc_in = adc_in/4
         userWR_buffer[8] = adc_in

      ' Read switches
      row2 = 0                         ' Read row 2 rotary switches
      delay_us(1500)                   ' Debounce delay
      switches[2] = PORTB
      row2 = 1
      if switches[2] <> oldswitches[2] then
```

```
         count = count + 1
         if count = 2 then
            swmoved = true
'            row2moved = true
            ' Find which input changed
            temp = switches[2] xor oldswitches[2]
            select case temp
               case 0x80
                  if (switches[2].7 <> switches[2].6) then
                     userWR_buffer[1].1 = not userWR_buffer[1].1
                     userWR_buffer[1].0 = 0
                  else
                     userWR_buffer[1].0 = not userWR_buffer[1].0
                     userWR_buffer[1].1 = 0
                  end if
'               case 0x40
'                  if (switches[2].7 = switches[2].6) then
'                     userWR_buffer[1].1 = not userWR_buffer[1].1
'                     userWR_buffer[1].0 = 0
'                  else
'                     userWR_buffer[1].0 = not userWR_buffer[1].0
'                     userWR_buffer[1].1 = 0
'                  end if
               case 0x20
                  if (switches[2].5 <> switches[2].4) then
                     userWR_buffer[1].3 = not userWR_buffer[1].3
                     userWR_buffer[1].2 = 0
                  else
                     userWR_buffer[1].2 = not userWR_buffer[1].2
                     userWR_buffer[1].3 = 0
                  end if
'               case 0x10
'                  if (switches[2].5 = switches[2].4) then
'                     userWR_buffer[1].3 = not userWR_buffer[1].3
'                     userWR_buffer[1].2 = 0
'                  else
'                     userWR_buffer[1].2 = not userWR_buffer[1].2
'                     userWR_buffer[1].3 = 0
'                  end if
               case 0x08
                  if (switches[2].3 <> switches[2].2) then
                     userWR_buffer[1].5 = not userWR_buffer[1].5
                     userWR_buffer[1].4 = 0
                  else
                     userWR_buffer[1].4 = not userWR_buffer[1].4
                     userWR_buffer[1].5 = 0
                  end if
'               case 0x04
'                  if (switches[2].3 = switches[2].2) then
'                     userWR_buffer[1].5 = not userWR_buffer[1].5
'                     userWR_buffer[1].4 = 0
'                  else
'                     userWR_buffer[1].4 = not userWR_buffer[1].4
'                     userWR_buffer[1].5 = 0
'                  end if
               case 0x02
                  if (switches[2].1 <> switches[2].0) then
                     userWR_buffer[1].7 = not userWR_buffer[1].7
                     userWR_buffer[1].6 = 0
                  else
                     userWR_buffer[1].6 = not userWR_buffer[1].6
                     userWR_buffer[1].7 = 0
                  end if
```

```
'               case 0x01
'                  if (switches[2].1 = switches[2].0) then
'                     userWR_buffer[1].7 = not userWR_buffer[1].7
'                     userWR_buffer[1].6 = 0
'                  else
'                     userWR_buffer[1].6 = not userWR_buffer[1].6
'                     userWR_buffer[1].7 = 0
'                  end if
            end select
            passcount = 0
            oldswitches[2] = switches[2]
            count = 0
         end if
      end if

      row3 = 0                           ' Read row 3 rotary switch
      delay_us(1500)                     ' Debounce delay
      switches[3] = PORTB
      row3 = 1
      if switches[3] <> oldswitches[3] then
         count = count + 1
         if count = 2 then
            swmoved = true
'            row3moved = true
            ' Find which input changed
            temp = switches[3] xor oldswitches[3]
            select case temp
               case 0x80
                  if (switches[3].7 <> switches[3].6) then
                     userWR_buffer[2].1 = not userWR_buffer[2].1
                     userWR_buffer[2].0 = 0
                  else
                     userWR_buffer[2].0 = not userWR_buffer[2].0
                     userWR_buffer[2].1 = 0
                  end if
'               case 0x40
'                  if (switches[3].7 = switches[3].6) then
'                     userWR_buffer[2].1 = not userWR_buffer[2].1
'                     userWR_buffer[2].0 = 0
'                  else
'                     userWR_buffer[2].0 = not userWR_buffer[2].0
'                     userWR_buffer[2].1 = 0
'                  end if
               case 0x20
                  if (switches[3].5 <> switches[3].4) then
                     userWR_buffer[2].3 = not userWR_buffer[2].3
                     userWR_buffer[2].2 = 0
                  else
                     userWR_buffer[2].2 = not userWR_buffer[2].2
                     userWR_buffer[2].3 = 0
                  end if
'               case 0x10
'                  if (switches[3].5 = switches[3].4) then
'                     userWR_buffer[2].3 = not userWR_buffer[2].3
'                     userWR_buffer[2].2 = 0
'                  else
'                     userWR_buffer[2].2 = not userWR_buffer[2].2
'                     userWR_buffer[2].3 = 0
'                  end if
               case 0x08
                  if (switches[3].3 <> switches[3].2) then
                     userWR_buffer[2].5 = not userWR_buffer[2].5
                     userWR_buffer[2].4 = 0
                  else
                     userWR_buffer[2].4 = not userWR_buffer[2].4
                     userWR_buffer[2].5 = 0
                  end if
'               case 0x04
'                  if (switches[3].3 = switches[3].2) then
'                     userWR_buffer[2].5 = not userWR_buffer[2].5
'                     userWR_buffer[2].4 = 0
'                  else
'                     userWR_buffer[2].4 = not userWR_buffer[2].4
'                     userWR_buffer[2].5 = 0
'                  end if
               case 0x02
                  if (switches[3].1 <> switches[3].0) then
                     userWR_buffer[2].7 = not userWR_buffer[2].7
                     userWR_buffer[2].6 = 0
                  else
                     userWR_buffer[2].6 = not userWR_buffer[2].6
                     userWR_buffer[2].7 = 0
                  end if
'               case 0x01
'                  if (switches[3].1 = switches[3].0) then
'                     userWR_buffer[2].7 = not userWR_buffer[2].7
'                     userWR_buffer[2].6 = 0
'                  else
'                     userWR_buffer[2].6 = not userWR_buffer[2].6
'                     userWR_buffer[2].7 = 0
'                  end if
            end select
            passcount = 0
            oldswitches[3] = switches[3]
            count = 0
         end if
      end if

      row1 = 0                           ' Read row 1 rotary switches
      delay_us(1500)                     ' Debounce delay
      switches[1] = PORTB                ' Read 8 column bits
      row1 = 1                           ' Set row 1 high
      if switches[1] <> oldswitches[1] then ' Check to see of a switch moved
         count = count + 1
         if count = 2 then
            swmoved = true                  ' Set switch moved flag
            ' Find which input changed
            temp = switches[1] xor oldswitches[1]
            select case temp
               case 0x80                    ' Rotary A input changed
                  if (switches[1].7 <> switches[1].6) then ' Rotary switch moved clockwise
                     userWR_buffer[0].1 = not userWR_buffer[0].1
                     userWR_buffer[0].0 = 0
                  else                                     ' Rotary switch moved counter
                                          clockwise
                     userWR_buffer[0].0 = not userWR_buffer[0].0
                     userWR_buffer[0].1 = 0
                  end if
'               case 0x40                    ' Rotary B input changed
'                  if (switches[1].7 = switches[1].6) then ' Rotary switch moved clockwise
'                     userWR_buffer[0].1 = not userWR_buffer[0].1
'                     userWR_buffer[0].0 = 0
'                  else                                     ' Rotary switch moved counter
                                          clockwise
'                     userWR_buffer[0].0 = not userWR_buffer[0].0
'                     userWR_buffer[0].1 = 0
'                 end if
               case 0x20
                  if (switches[1].5 <> switches[1].4) then ' Rotary switch moved clockwise
                     userWR_buffer[0].3 = not userWR_buffer[0].3
                     userWR_buffer[0].2 = 0
                  else                                     ' Rotary switch moved counter
                                        clockwise
                     userWR_buffer[0].2 = not userWR_buffer[0].2
                     userWR_buffer[0].3 = 0
                  end if
'               case 0x10
'                  if (switches[1].5 = switches[1].4) then ' Rotary switch moved clockwise
'                     userWR_buffer[0].3 = not userWR_buffer[0].3
'                     userWR_buffer[0].2 = 0
'                  else                                    ' Rotary switch moved counter
                                        clockwise
'                     userWR_buffer[0].2 = not userWR_buffer[0].2
'                     userWR_buffer[0].3 = 0
'                  end if
               case 0x08
                  if (switches[1].3 <> switches[1].2) then ' Rotary switch moved clockwise
                     userWR_buffer[0].5 = not userWR_buffer[0].5
                     userWR_buffer[0].4 = 0
                  else                                     ' Rotary switch moved counter
                                        clockwise
                     userWR_buffer[0].4 = not userWR_buffer[0].4
                     userWR_buffer[0].5 = 0
                  end if
'               case 0x04
'                  if (switches[1].3 = switches[1].2) then ' Rotary switch moved clockwise
'                     userWR_buffer[0].5 = not userWR_buffer[0].5
'                     userWR_buffer[0].4 = 0
'                  else                                    ' Rotary switch moved counter
                                        clockwise
'                     userWR_buffer[0].4 = not userWR_buffer[0].4
'                     userWR_buffer[0].5 = 0
'                  end if
               case 0x02
                  if (switches[1].1 <> switches[1].0) then ' Rotary switch moved clockwise
                     userWR_buffer[0].7 = not userWR_buffer[0].7
                     userWR_buffer[0].6 = 0
                  else                                     ' Rotary switch moved counter
                                        clockwise
                     userWR_buffer[0].6 = not userWR_buffer[0].6
                     userWR_buffer[0].7 = 0
                  end if
'               case 0x01
'                  if (switches[1].1 = switches[1].0) then ' Rotary switch moved clockwise
'                     userWR_buffer[0].7 = not userWR_buffer[0].7
'                     userWR_buffer[0].6 = 0
'                  else                                    ' Rotary switch moved counter
                                        clockwise
'                     userWR_buffer[0].6 = not userWR_buffer[0].6
'                     userWR_buffer[0].7 = 0
'                  end if
            end select
            passcount = 0                ' Reset pass counter for switch pulse
            oldswitches[1] = switches[1] ' Prepare to for next pass
            count = 0
         end if
      end if

      row4 = 0                           ' Read row 4 push-button switches
      delay_ms(3)                        ' Debounce delay
      switches[4] = PORTB                ' Read port B
      row4 = 1                           ' Set row 4 high to stop reading switches
      if switches[4] <> oldswitches[4] then ' Check if a switch moved
         swmoved = true
         userWR_buffer[3] = not switches[4] ' Set USB write buffer to switches
         passcount = 0                   ' Reset pass count for switch pulse
         oldswitches[4] = switches[4]    ' Prepare for next pass
      end if

      switches[0].0 = toggle1            ' Read toggle switches
      switches[0].1 = toggle2            ' Read toggle switches
      switches[0].2 = toggle3            ' Read toggle switches
      switches[0].3 = toggle4            ' Read toggle switches
      switches[0].4 = toggle5            ' Read toggle switches
      switches[0].5 = toggle6            ' Read toggle switches
      switches[0].6 = toggle7            ' Read toggle switches
      switches[0].7 = toggle8            ' Read toggle switches
      if switches[0] <> oldswitches[0] then ' Check if a switch moved
         swmoved = true
         temp = oldswitches[0] xor switches[0]
         if temp <> 0 then
            togglemoved = true
         else
             togglemoved = false
         end if
         select case temp
            case 0x80
                userWR_buffer[3].0 = 1
            case 0x40
                userWR_buffer[3].1 = 1
            case 0x20
                userWR_buffer[3].2 = 1
            case 0x10
                userWR_buffer[3].3 = 1
            case 0x08
                userWR_buffer[3].4 = 1
            case 0x04
                userWR_buffer[3].5 = 1
            case 0x02
                userWR_buffer[3].6 = 1
            case 0x01
                userWR_buffer[3].7 = 1
         end select
         passcount = 0                   ' Reset pass count for switch pulse
         oldswitches[0] = switches[0]    ' Prepare for next pass
      end if

      if swmoved = true then             ' A switch moved
         passcount = passcount + 1       ' Increment pass counter for pulse
         if passcount = pulse then       ' Pass counter = pulse duration
            userWR_buffer[0]=0x00        ' Initialize the USB write buffer to all 0
            userWR_buffer[1]=0x00
            userWR_buffer[2]=0x00
            if togglemoved = true then
               userWR_buffer[3]=0x00
               togglemoved = false
            end if
            passcount = 0                ' Reset pass counter
            swmoved = false              ' Reset switch moved flag
         end if
      end if
      HID_Write(@userWR_buffer, 9)       ' Send data onto USB bus

   wend
   HID_Disable()
end.
```

Appendix C: FSUIPC Button Programming

I did some special programming for the rotary switches. Each rotary switch has a momentary contact switch that is activated by pressing the knob. Pressing the switch toggles a flag. Each switch available in FSUIPC has a corresponding flag. However, I can set, reset, or toggle a switch flag whether the real switch exists or not. A flag is identified as J15B3 which is joystick #15 (which doesn't exist), button #3. I use flags as follows:

Rotary Switch	Switch Action	Rotary Function
GPS Group/Page	Toggle J15B0	Change group or page
Alt/VS Hold Set	Run macro #32	Change altitude or vertical speed value
Barometer	None	Change kohlsman value
ADF	Toggle J15B1	Change tenths or tens frequency
COM1	Toggle J15B3	Change decimal or ones frequency
COM2	Toggle J15B4	Change decimal or ones frequency
NAV1	Toggle J15B5	Change decimal or ones frequency
NAV2	Toggle J15B6	Change decimal or ones frequency
DH (decision height adjust)	Run macro #56	Make DH visible on EADI
CRS	Toggle J15B7	Change OBS indicator fast or slow
HDG	Toggle J15B8	Change heading bug fast or slow

Each radio, including the ADF had a standby and active frequency. Moving the rotary switch only changed the standby frequency. I only have one SWAP button on my radio panel, so I programmed FSUIPC that whenever I change a frequency I set a flag for that radio, and reset all the other radio's flags. Depending on which flag was set the swap button swapped that radio's frequency between standby and active.

```
[Buttons]
ButtonRepeat=20,10
PollInterval=25

[Profile.PC12]
1=F1_Pilatus_PC-12 N89DD

[Buttons.PC12]
1=CP(-3,24)(-3,25)(-3,26)(+3,27)(+3,28)(-3,29)(-3,31)3,30,C1005,3840 ;GPSGRP/PGSW toggleflag
       J15B0
2=CP(F-15,0)3,1,C66625,0 ;GPSGroupInc
3=CP(F+15,0)3,1,C66627,0 ;GPSPageInc
4=CP(F-15,0)3,0,C66626,0 ;GPSGroupDec
5=CP(F+15,0)3,0,C66628,0 ;GPSPageDec
6=P3,3,CM1:57,0 ;up
7=P3,2,CM1:58,0 ;dn
8=P3,5,C65883,0 ;Kohlsman inc
9=P3,4,C65884,0 ;Kohlsman dec
```

```
14=P3,6,C1003,3850 ;SetADFmovedflag
15=P3,6,C1004,3851 ;ResetCOM1movedflag
16=P3,6,C1004,3852 ;ResetCOM2movedflag
17=P3,6,C1004,3853 ;ResetNAV1movedflag
18=P3,6,C1004,3854 ;ResetNAV2movedflag
19=P3,7,C1003,3850 ;SetADFmovedflag
20=P3,7,C1004,3851 ;ResetCOM1movedflag
21=P3,7,C1004,3852 ;ResetCOM2movedflag
22=P3,7,C1004,3853 ;ResetNAV1movedflag
23=P3,7,C1004,3854 ;ResetNAV2movedflag
30=CP(-3,24)(-3,25)(-3,26)(+3,27)(-3,28)(+3,29)(+3,30)3,31,C1005,3843 ;COM1FASTtoggleJ15,B3
31=P3,8,C1004,3850 ;ResetADFmovedflag
32=P3,8,C1003,3851 ;SetCOM1movedflag
33=P3,8,C1004,3852 ;ResetCOM2movedflag
34=P3,8,C1004,3853 ;ResetNAV1movedflag
35=P3,8,C1004,3854 ;ResetNAV2movedflag
36=P3,9,C1004,3850 ;ResetADFmovedflag
37=P3,9,C1003,3851 ;SetCOM1movedflag
38=P3,9,C1004,3852 ;ResetCOM2movedflag
39=P3,9,C1004,3853 ;ResetNAV1movedflag
40=P3,9,C1004,3854 ;ResetNAV2movedflag
41=CP(F-15,3)3,8,C66434,0 ;COM1FracDec
42=CP(F-15,3)3,9,C66435,0 ;COM1FracInc
43=CP(F+15,3)3,8,C65636,0 ;COM1WholeDec
44=CP(F+15,3)3,9,C65637,0 ;COM1WholeInc
45=CP(+3,24)(-3,25)(-3,26)(-3,27)(-3,28)(+3,29)(-3,31)3,30,C1005,3844 ;COM2FASTtoggleJ15,B4
46=P3,10,C1004,3850 ;ResetADFmovedflag
47=P3,10,C1004,3851 ;ResetCOM1movedflag
48=P3,10,C1003,3852 ;SetCOM2movedflag
49=P3,10,C1004,3853 ;ResetNAV1movedflag
50=P3,10,C1004,3854 ;ResetNAV2movedflag
51=P3,11,C1004,3850 ;ResetADFmovedflag
52=P3,11,C1004,3851 ;ResetCOM1movedflag
53=P3,11,C1003,3852 ;SetCOM2movedflag
54=P3,11,C1004,3853 ;ResetNAV1movedflag
55=P3,11,C1004,3854 ;ResetNAV2movedflag
56=CP(F-15,4)3,10,C66439,0 ;COM2FracDec
57=CP(F-15,4)3,11,C66441,0 ;COM2FracInc
58=CP(F+15,4)3,10,C66436,0 ;COM2WholeDec
59=CP(F+15,4)3,11,C66437,0 ;COM2WholeInc
60=CP(+3,24)(-3,25)(-3,26)(-3,27)(+3,28)(-3,29)(-3,31)3,30,C1005,3845 ;NAV1FASTtoggleJ15,B5
61=P3,12,C1004,3850 ;ResetADFmovedflag
62=P3,12,C1004,3851 ;ResetCOM1movedflag
63=P3,12,C1004,3852 ;ResetCOM2movedflag
64=P3,12,C1003,3853 ;SetNAV1movedflag
65=P3,12,C1004,3854 ;ResetNAV2movedflag
66=P3,13,C1004,3850 ;ResetADFmovedflag
67=P3,13,C1004,3851 ;ResetCOM1movedflag
68=P3,13,C1004,3852 ;ResetCOM2movedflag
69=P3,13,C1003,3853 ;SetNAV1movedflag
70=P3,13,C1004,3854 ;ResetNAV2movedflag
71=CP(F-15,5)3,12,C66445,0 ;NAV1FracDec
72=CP(F-15,5)3,13,C66446,0 ;NAV1FracInc
73=CP(F+15,5)3,12,C65640,0 ;NAV1WholeDec
74=CP(F+15,5)3,13,C65641,0 ;NAV1WholeInc
75=CP(+3,24)(-3,25)(-3,26)(-3,27)(+3,28)(-3,31)(-3,30)3,29,C1005,3846 ;NAV2FASTtoggleJ15,B6
76=P3,14,C1004,3850 ;ResetADFmovedflag
77=P3,14,C1004,3851 ;ResetCOM1movedflag
78=P3,14,C1004,3852 ;ResetCOM2movedflag
79=P3,14,C1004,3853 ;ResetNAV1movedflag
80=P3,14,C1003,3854 ;SetNAV2movedflag
81=P3,15,C1004,3850 ;ResetADFmovedflag
82=P3,15,C1004,3851 ;ResetCOM1movedflag
```

```
83=P3,15,C1004,3852 ;ResetCOM2movedflag
84=P3,15,C1004,3853 ;ResetNAV1movedflag
85=P3,15,C1003,3854 ;SetNAV2movedflag
86=CP(F-15,6)3,14,C66449,0 ;NAV2FracDec
87=CP(F-15,6)3,15,C66450,0 ;NAV2FracInc
88=CP(F+15,6)3,14,C65644,0 ;NAV2WholeDec
89=CP(F+15,6)3,15,C65645,0 ;NAV2WholeInc
90=CP(-3,24)(-3,25)(+3,26)(+3,27)(-3,28)(-3,29)(+3,30)3,31,CM1:56,0 ;DH Visible
91=P3,17,CM1:54,0 ;DH Inc
92=P3,16,CM1:55,0 ;DH Dec
93=CP(+3,24)(+3,25)(+3,26)(-3,31)(-3,28)(-3,29)(-3,30)3,27,C1005,3847 ;CRSFASTtoggleJ15B7
94=CP(F-15,7)3,19,C65663,0 ;OBS Inc
95=CP(F-15,7)3,18,C65662,0 ;OBS Dec
96=CP(F+15,7)3,19,C66368,0 ;OBS Inc Fast
97=CP(F+15,7)3,18,C1026,0 ;PBS Dec Fast
98=CP(-3,24)(-3,25)(-3,26)(-3,27)(+3,28)(+3,29)(+3,30)3,31,C1005,3848 ;HDGFASTtoggleJ15B8
99=CP(F-15,8)3,21,C65879,0 ;Hdg bug inc
100=CP(F-15,8)3,20,C65880,0 ;Hdg bug dec
101=CP(F+15,8)3,21,C1025,0 ;Hdg bug inc fast
102=CP(F+15,8)3,20,C1024,0 ;Hdg bug dec fast
103=CP(+3,24)(+3,25)(-3,26)(-3,27)(-3,31)(-3,29)(-3,30)3,28,C1005,3841 ;ADFFASTsetJ15,B1
104=;CP(F+15,1)(F-15,2)(+3,24)(+3,25)(-3,26)(-3,27)(-3,31)(-3,29)(-3
105=;CP(F+15,1)(F+15,2)(+3,24)(+3,25)(-3,26)(-3,27)(-3,31)(-3,29)(-3
107=;CP(F-15,1)(F+15,2)(+3,24)(+3,25)(-3,26)(-3,27)(-3,31)(-3,29)(-3
108=CP(F-15,1)3,6,CM1:49,0 ;ADF frac dec carry
109=CP(F-15,1)3,7,CM1:48,0 ;ADF frac inc carry
110=;CP(F+15,1)(F-15,2)3,6,CM1:51,0 ;ADF 1 dec
111=;CP(F+15,1)(F-15,2)3,7,CM1:50,0 ;ADF 1 inc
112=CP(F+15,1)3,6,CM1:53,0 ;ADF 10 dec
113=CP(F+15,1)3,7,CM1:52,0 ;ADF 10 inc
114=;CP(F-15,1)(F+15,2)3,6,CM1:53,0 ;ADF 10 dec
115=;CP(F-15,1)(F+15,2)3,7,CM1:52,0 ;ADF 10 inc
201=CP(-3,24)(-3,25)(-3,26)(-3,27)(-3,28)(-3,29)(-3,30)3,31,C66241,0 ;togglemasterbatt
202=CP(-3,24)(-3,25)(-3,26)(-3,27)(-3,28)(-3,29)(-3,31)3,30,C66242,0 ;togglemasteralt
203=CP(-3,24)(-3,25)(-3,26)(-3,28)(-3,29)(-3,30)(-3,31)3,27,CM1:1,0 ;toggleGen2
204=CP(-3,24)(-3,25)(-3,26)(-3,27)(-3,29)(-3,30)(-3,31)3,28,CM1:2,0 ;toggleExt
205=CP(-3,24)(-3,26)(-3,27)(-3,28)(-3,29)(-3,30)(-3,31)3,25,C66293,0 ;toggleavionicsmaster
206=CP(-3,24)(-3,25)(-3,26)(-3,27)(-3,28)(-3,30)(-3,31)3,29,CM1:3,0 ;toggleInv
207=CP(-3,24)(-3,25)(-3,27)(-3,28)(-3,29)(-3,30)(-3,31)3,26,CM1:4,0 ;toggleESS
208=CP(-3,25)(-3,26)(-3,27)(-3,28)(-3,29)(-3,30)(-3,31)3,24,CM1:5,0 ;toggleStby
209=CP(-3,24)(-3,25)(-3,26)(-3,27)(-3,28)(-3,29)(+3,30)3,31,CM1:6,0 ;togglefuelpumpLH
210=CP(-3,24)(-3,25)(-3,26)(-3,27)(+3,28)(-3,30)(-3,31)3,29,CM1:7,0 ;togglefuelpumpRH
211=CP(-3,24)(-3,25)(+3,26)(-3,28)(-3,29)(-3,30)(-3,31)3,27,CM1:8,0 ;togglestarter
212=CP(+3,24)(-3,26)(-3,27)(-3,28)(-3,29)(-3,30)(-3,31)3,25,CM1:9,0 ;toggleIgnition
213=CP(-3,24)(-3,25)(-3,26)(-3,27)(-3,28)(+3,29)(-3,31)3,30,C65909,0 ;Panel4OVERHD
214=CP(-3,24)(-3,25)(-3,26)(+3,27)(-3,31)(-3,29)(-3,30)3,28,CM1:10,0 ;LAMPTEST
215=CP(-3,24)(+3,25)(-3,31)(-3,27)(-3,28)(-3,29)(-3,30)3,26,C66240,0 ;LAND
216=CP(+3,24)(-3,25)(-3,26)(-3,27)(-3,28)(-3,29)(-3,30)3,31,C65560,0 ;STROBE
217=CP(-3,24)(-3,25)(-3,26)(-3,27)(-3,28)(+3,29)(-3,30)3,31,C66379,0 ;NAV
218=CP(-3,24)(-3,25)(-3,26)(-3,27)(+3,28)(-3,29)(-3,31)3,30,C66377,0 ;RECOG
219=CP(-3,24)(+3,25)(-3,26)(-3,31)(-3,28)(-3,29)(-3,30)3,27,CM1:13,0 ;SEATBELTS
220=CP(+3,24)(-3,25)(-3,31)(-3,27)(-3,28)(-3,29)(-3,30)3,26,CM1:11,0 ;PUSHER
221=CP(-3,24)(-3,25)(-3,26)(+3,27)(-3,28)(-3,29)(-3,30)3,31,CM1:12,0 ;FIRE
222=CP(-3,24)(-3,25)(+3,26)(-3,27)(-3,28)(-3,29)(-3,31)3,30,C65751,0 ;TAXI
223=CP(-3,24)(+3,25)(-3,26)(-3,27)(-3,28)(-3,31)(-3,30)3,29,C66378,0 ;WING
224=CP(+3,24)(-3,25)(-3,26)(-3,27)(-3,31)(-3,29)(-3,30)3,28,C66239,0 ;BEACON
225=CP(-3,24)(-3,25)(-3,26)(-3,27)(+3,28)(-3,29)(-3,30)3,31,C66376,0 ;LOGO
226=CP(-3,24)(-3,25)(-3,26)(+3,27)(-3,28)(-3,29)(-3,31)3,30,CM1:14,0 ;NOSMKG
227=CP(-3,24)(-3,25)(+3,26)(-3,27)(-3,28)(-3,31)(-3,30)3,29,CM1:16,0 ;SYS
228=CP(-3,24)(-3,25)(-3,26)(+3,27)(-3,28)(-3,31)(-3,30)3,29,CM1:17,0 ;RECIRC
229=CP(-3,24)(-3,25)(-3,26)(-3,27)(+3,28)(-3,29)(+3,30)3,31,CM1:15,0 ;HEAT
230=CP(+3,24)(+3,25)(-3,31)(-3,27)(-3,28)(-3,29)(-3,30)3,26,CM1:24,0 ;3MIN
231=CP(-3,24)(-3,25)(+3,26)(-3,27)(-3,28)(-3,29)(-3,30)3,31,C66337,0 ;BOOTSON
```

```
232=CP(-3,24)(-3,25)(-3,26)(-3,27)(+3,28)(+3,29)(-3,31)3,30,CM1:21,0 ;LHHEAVY
233=CP(+3,24)(+3,25)(-3,26)(-3,31)(-3,28)(-3,29)(-3,30)3,27,CM1:20,0 ;LHON
234=CP(+3,24)(-3,25)(-3,26)(-3,31)(-3,28)(-3,29)(-3,30)3,27,CM1:18,0 ;FANS
235=CP(-3,24)(-3,25)(-3,26)(-3,27)(-3,28)(+3,29)(+3,30)3,31,CM1:19,0 ;VENT
236=CP(-3,24)(+3,25)(+3,26)(-3,27)(-3,31)(-3,29)(-3,30)3,28,C66484,0 ;INERTSEP
237=CP(-3,24)(-3,25)(+3,26)(-3,27)(-3,31)(-3,29)(-3,30)3,28,C66338,0 ;PROP
238=CP(-3,24)(+3,25)(-3,26)(-3,27)(-3,28)(-3,29)(-3,30)3,31,CM1:23,0 ;RHHEAVY
239=CP(+3,24)(-3,25)(-3,26)(-3,27)(-3,28)(-3,29)(-3,31)3,30,CM1:22,0 ;RHON
240=CP(-3,24)(+3,25)(-3,26)(-3,27)(-3,28)(-3,29)(-3,31)3,30,C66605,0 ;GPSOBS
241=CP(-3,24)(-3,25)(-3,26)(-3,27)(+3,28)(+3,29)(-3,30)3,31,C66606,0 ;GPSMSG
242=CP(+3,24)(-3,25)(+3,26)(-3,31)(-3,28)(-3,29)(-3,30)3,27,C66609,0 ;GPSFPL
243=CP(-3,24)(+3,25)(-3,26)(-3,27)(-3,31)(-3,29)(-3,30)3,28,C66611,0 ;GPSTERR
244=CP(+3,24)(-3,25)(-3,26)(-3,27)(-3,28)(-3,31)(-3,30)3,29,C66612,0 ;GPSPROC
245=CP(-3,24)(-3,25)(-3,26)(+3,27)(-3,28)(-3,29)(+3,30)3,31,C66624,0 ;GPSCRSR
246=CP(-3,24)(-3,25)(-3,26)(+3,27)(-3,28)(+3,29)(-3,30)3,31,C66616,0 ;GPSRNGUp
247=CP(-3,24)(-3,25)(+3,26)(-3,27)(+3,28)(-3,29)(-3,31)3,30,C66615,0 ;GPSRNGDown
248=CP(-3,24)(-3,25)(-3,26)(+3,27)(-3,28)(+3,29)(-3,31)3,30,C66617,0 ;GPSDIRECT
249=CP(-3,24)(-3,25)(+3,26)(-3,27)(+3,28)(-3,29)(-3,30)3,31,C66618,0 ;GPSMENU
250=CP(-3,24)(-3,25)(-3,26)(+3,27)(+3,28)(-3,29)(-3,30)3,31,C66619,0 ;GPSCLR
251=CP(-3,24)(-3,25)(+3,26)(-3,27)(-3,28)(+3,29)(-3,31)3,30,C66623,0 ;GPSENT
253=CP(-3,24)(-3,25)(+3,26)(-3,27)(-3,28)(+3,29)(-3,30)3,31,CM1:25,0 ;APDN
254=CP(-3,24)(-3,25)(-3,26)(+3,27)(+3,28)(-3,31)(-3,30)3,29,C65725,0 ;APHDG
255=CP(-3,24)(-3,25)(+3,26)(-3,27)(-3,28)(-3,29)(+3,30)3,31,C65729,0 ;APNAV
256=CP(-3,24)(-3,25)(+3,26)(-3,27)(+3,28)(-3,31)(-3,30)3,29,C65724,0 ;APAPR
257=CP(-3,24)(+3,25)(-3,26)(-3,27)(-3,28)(-3,29)(+3,30)3,31,C65728,0 ;APBC
258=CP(-3,24)(-3,25)(+3,26)(+3,27)(-3,28)(-3,29)(-3,30)3,31,C65793,0 ;APYD
259=CP(-3,24)(+3,25)(-3,26)(-3,27)(-3,28)(+3,29)(-3,31)3,30,C65580,0 ;APAP
260=CP(-3,24)(-3,25)(+3,26)(+3,27)(-3,28)(-3,29)(-3,31)3,30,CM1:26,0 ;APUP
261=CP(-3,24)(+3,25)(-3,26)(-3,27)(-3,28)(+3,29)(-3,30)3,31,C65799,0 ;APALT
262=CP(-3,24)(-3,25)(+3,26)(+3,27)(-3,28)(-3,31)(-3,30)3,29,C65859,0 ;APIAS
263=CP(-3,24)(+3,25)(-3,26)(-3,27)(+3,28)(-3,29)(-3,30)3,31,C66288,0 ;APFD
264=CP(-3,24)(-3,25)(+3,26)(+3,27)(-3,31)(-3,29)(-3,30)3,28,CM1:27,0 ;APSOFT
265=CP(+3,24)(-3,25)(-3,26)(-3,27)(-3,28)(-3,29)(+3,30)3,31,CM1:28,0 ;APHALF
266=CP(-3,24)(+3,25)(-3,26)(-3,27)(+3,28)(-3,29)(-3,31)3,30,CM1:29,0 ;APTST
267=CP(+3,24)(-3,25)(-3,26)(-3,27)(-3,28)(+3,29)(-3,30)3,31,CM1:30,0 ;ENG
268=CP(-3,24)(+3,25)(-3,26)(-3,27)(+3,28)(-3,31)(-3,30)3,29,CM1:31,0 ;ARM
269=CP(+3,24)(-3,25)(+3,26)(-3,27)(-3,28)(-3,29)(-3,30)3,31,CM1:32,0 ;SETSW
271=CP(-3,24)(+3,25)(-3,26)(+3,27)(-3,28)(-3,29)(-3,30)3,31,C65911,0 ;AV
272=CP(+3,24)(-3,25)(+3,26)(-3,27)(-3,28)(-3,29)(-3,31)3,30,C65908,0 ;GPS
273=CP(-3,24)(+3,25)(-3,26)(+3,27)(-3,28)(-3,29)(-3,31)3,30,C66286,0 ;DME
274=CP(+3,24)(-3,25)(+3,26)(-3,27)(-3,28)(-3,31)(-3,30)3,29,C66375,0 ;NAV/GPS
275=CP(-3,24)(+3,25)(-3,26)(+3,27)(-3,28)(-3,31)(-3,30)3,29,C65724,0 ;APR
276=CP(-3,24)(+3,25)(-3,26)(+3,27)(+3,28)(-3,29)(-3,30)3,31,CM1:33,0 ;AHRS
277=CP(-3,24)(+3,25)(-3,26)(+3,27)(+3,28)(-3,29)(-3,30)3,31,CM1:34,0 ;0
278=CP(-3,24)(+3,25)(+3,26)(-3,27)(-3,28)(-3,29)(-3,30)3,31,CM1:35,0 ;1
279=CP(+3,24)(-3,25)(-3,26)(+3,27)(-3,28)(-3,29)(-3,31)3,30,CM1:36,0 ;2
280=CP(-3,24)(+3,25)(+3,26)(-3,27)(-3,28)(-3,29)(-3,31)3,30,CM1:37,0 ;3
281=CP(+3,24)(-3,25)(-3,26)(+3,27)(-3,28)(-3,31)(-3,30)3,29,CM1:38,0 ;4
282=CP(-3,24)(+3,25)(+3,26)(-3,27)(-3,28)(-3,31)(-3,30)3,29,CM1:39,0 ;5
283=CP(+3,24)(-3,25)(-3,26)(+3,27)(-3,31)(-3,29)(-3,30)3,28,CM1:40,0 ;6
284=CP(-3,24)(+3,25)(+3,26)(-3,31)(-3,28)(-3,29)(-3,30)3,27,CM1:41,0 ;7
285=CP(F+15,10)(+3,24)(-3,25)(-3,26)(-3,27)(+3,28)(-3,29)(-3,30)3,31,CM1:42,0 ;SWAPADF
286=CP(F+15,11)(+3,24)(-3,25)(-3,26)(-3,27)(+3,28)(-3,29)(-3,30)3,31,C66372,0 ;SWAPCOM1
287=CP(F+15,12)(+3,24)(-3,25)(-3,26)(-3,27)(+3,28)(-3,29)(-3,30)3,31,C66444,0 ;SWAPCOM2
288=CP(F+15,13)(+3,24)(-3,25)(-3,26)(-3,27)(+3,28)(-3,29)(-3,30)3,31,C66448,0 ;SWAPNAV1
289=CP(F+15,14)(+3,24)(-3,25)(-3,26)(-3,27)(+3,28)(-3,29)(-3,30)3,31,C66452,0 ;SWAPNAV2
293=CP(-3,24)(-3,25)(+3,26)(+3,27)(+3,28)(-3,29)(-3,30)3,31,C65842,0 ;NAV1IDEN
295=CP(-3,24)(-3,25)(+3,26)(+3,27)(-3,28)(+3,29)(-3,30)3,31,C65843,0 ;NAV2IDEN
298=CP(+3,24)(+3,25)(-3,26)(-3,27)(-3,28)(-3,29)(-3,30)3,31,CM1:43,0 ;HSI
299=CP(-3,24)(-3,25)(-3,26)(+3,27)(+3,28)(+3,29)(-3,31)3,30,CM1:44,0 ;ARC
300=CP(+3,24)(+3,25)(-3,26)(-3,27)(-3,28)(-3,29)(-3,31)3,30,C66375,0 ;NAV
301=CP(-3,24)(-3,25)(-3,26)(+3,27)(+3,28)(+3,29)(-3,30)3,31,CM1:45,0 ;->
302=CP(+3,24)(+3,25)(-3,26)(-3,27)(-3,28)(-3,31)(-3,30)3,29,CM1:46,0 ;=>
```

```
303=CP(-3,24)(-3,25)(-3,26)(+3,27)(+3,28)(-3,29)(+3,30)3,31,CM1:47,0 ;1-2

[MacroFiles]
1=PC12
```

Appendix D: FSUIPC Macro File

In order to get some of the functions to work I used FSUIPC's macro function. FSUIPC allows you to record mouse clicks that otherwise wouldn't have a Flight Simulator command. I was able to get most of the functions to work using these macros. The ones I couldn't access were:

- Starter Interrupter
- Deicing probes
- GPS power on
- GPWS (no longer used so I didn't wire the switch)
- WAAS (no longer used so I didn't wire the switch)

So literally I have five unused switches on my panels which I could relabel and use for something else.

```
[Macros]
Module="F1PC12.GAU"
1=GEN2=RXd190*X8bcc
2=Ext=RXd360*X8bcc
3=Inv=RXd120*X8bcc
4=Ess=RXd030*X8bcc
5=Stby=RXd240*X8bcc
6=FuelPump_LH=RXd4e0*X83cc
7=FuelPump_RH=RXd670*X8bcc
8=Strt=RXc680*X83cc
9=Ignit=RXddc0*Xa100
10=Lamp_Test=RXdea0*X8bcc
11=Pusher=RXce60*X8bcc
12=Fire=RXcd90*X8bcc
13=Seat_Belts=RXccf0*X68cc
14=NoSmkg=RXcc70*X68cc
15=CabinHeat=RXcf70*X8bcc
16=SYS=RXdb80*X8bcc
17=RECIRC=RXdd50*X8bcc
18=FANS=RXdca0*X8bcc
19=VENT=RXdf80*X8bcc
20=LH_ON=RXdaf0*X8bcc
21=LH_Heavy=RXd9d0*X8bcc
22=RH_ON=RXda60*X8bcc
23=RH_Heavy=RXd920*X8bcc
24=3min=RXd870*X8b00
25=AP DN=RX9370*X55cc
26=AP UP=RX96a0*X55cc
27=AP Soft=RX9350*X8bcc
28=AP Half=RX9330*X8bcc
29=AP Test=RX9310*X8bcc
30=Eng=RX8700*X83cc
31=Arm=RX8890*X8bcc
32=Set=RX88b0*X8bcc
33=AHRS=RX10070*X8bcc
34=0=RX4b90*Xa1cc
35=1=RX4bc0*Xa1cc
36=2=RX4bf0*Xa1cc
37=3=RX4c20*Xa1cc
```

```
38=4=RX4c50*Xa1cc
39=5=RX4c80*Xa1cc
40=6=RX4cb0*Xa1cc
41=7=RX4ce0*Xa1cc
42=ADF Swap=RX5af0*X83cc
43=HSI=RXa0a0*Xc7cc
44=ARC=RXa0b0*Xc7cc
45=->=RXa0c0*Xa1cc
46=>>=RXa0e0*Xa1cc
47=1-2=RXa120*X33cc
48=ADF frac inc car=RX5a90*Xa1cc
49=ADF frac dec car=RX5a70*Xa1cc
50=ADF 1 inc=RX5ab0*Xa1cc
51=ADF 1 Dec=RX5a50*Xa1cc
52=ADF 10 Inc=RX5ad0*Xa1cc
53=ADF 10 Dec=RX5a30*Xa1cc
54=DH Up=RXa130*X83cc
55=DH Down=RXa220*X8300
56=DH Visible=RXa4d0*X8bcc
57=Alt Up=RX8630*Xa1cc
58=Alt Dn=RX8560*Xa1cc
```