Welcome

Mike's Flight Deck is an introduction to home cockpit building, the hobby that takes off where flight simulation game software ends. When staring at a 17" monitor on a crowded desk, and pushing, pulling and twisting a wimpy joystick just doesn't do any more, it's time to build a simulated cockpit or flight deck.

Add some rudder pedals. Use multiple monitors for panoramic scenery. Add a radio stack that displays and controls the frequencies in FS. Enter your course on an FMC. Improve your game attitude with a simulated analog artificial horizon that interfaces to FS...

Get lost in the sky.

What's New:

Getting Switch Action Into a Sim (24 Feb 2007)

Switches don't occupy a huge part of your attention when you're in a sim. True, you tune radios and operate the landing gear, but most of your interaction with the sim takes place through the flight and engine controls, the instruments and the through-the-window view. The magic of having switches in a sim is that they let you remove the mouse and keyboard, two major detractors from the flight illusion.

For the past few weeks I've been working on getting switch action into a simulation computer. There are a number of interfaces you can buy, but of course I'm more interested in DIY solutions. It's easy to use a $7 PIC16F876A to scan an 8 by 8 array of switches and signal changes to the PC through the serial port. It gets a little more involved when you have multiple devices talking to the PC, but not seriously so.

I'm a big fan of the serial com port. There's no question that USB is a technologically more advanced interface, but from a practical, hobbyist point of view it's hard to beat the serial port. It's easy to access from a user program. It's inexpensive to use. It's robust and offers respectable bandwidth.

A shortcoming of the serial port is that it's based on RS-232C, a point to point signaling protocol. You can only hang one (bi-directional) device on a serial port. Sims tend to have lots of devices, and I'm reluctant to stuff my PC with lots of serial port expansion modules. A better solution to this sim-related shortcoming is to use a converter to change the RS-232C protocol to a party line protocol like the full-duplex RS-422.

A small sim might have a collection of random switches (master, avionics power, landing gear, starter, etc.), a nav/comm radio head and a transponder. I could build a single large module including everything that interfaced through a single RS-232C serial port, but using an RS-422 adapter allows me to build separate modules. That's easier and lets me add modules as I build them.

RS-422 provides versatility, but there's a potential problem. With any kind of party line system, the potential exists for more than one party trying to talk at the same time. When this happens data gets lost. There are several ways to deal with this.

One approach is to do nothing preventative. Simply look for an indication that the problem occurred and take corrective action as needed. This is not a bad solution

devices might not need to share the RS-422 connection. We might then need an unacceptably large chunk of resources to take corrective action.

Another approach is to give the host PC the role of traffic cop. It can poll each device periodically. No one talks until asked to speak. Well, this works, but carries an overhead. It uses processor time as well as a portion of the port's output bandwidth. Further, it adds latency. As more devices are added to the group, there is an increased average time before the computer appears to respond to the switch action. For some things, like landing gear, this isn't a problem. For others, like radio tuning, it can be a major annoyance.

A more efficient approach is to require a device to ask permission before sending data. The request doesn't have to go to the PC. It can be handled by a dedicated micro controller using dedicated request/grant links. A requesting device can be granted permission usually within tens of micro seconds, so latency is minimal.

I have these projects designed. The firmware assembles. Next step is to build.

FSX Data Output with SimConnect (4 Feb 2007)

SimConnect is a utility that ships with the deluxe version of Microsoft Flight Simulator X. It allows programs to interact with FSX in a great many ways. The SimConnect programming interface consists of several dozen functions. A few of those functions relate specifically to getting data into and out of FSX. This is a great boon to us flight deck builders. (Thank you, Microsoft.)

The FSX Deluxe installation DVD includes an SDK for SimConnect that contains documentation and code examples. But while SimConnect is documented, it's not necessarily explained with the casual programmer in mind. A fair amount of background knowledge is (not unreasonably) assumed. Learning how to use SimConnect can require a bit of exploration and research.

That's what I'm up to these days, exploring and researching.

In particular, I've recently been looking into how SimConnect tells your program that new data is available.

By way of background, you get data from FSX through SimConnect by telling SimConnect the names of the FSX data you're interested in then telling it under what conditions to collect that data. However, SimConnect doesn't actually send you the data until you ask for it.

The question is, when should you ask for the data?

Most of the examples in the SimConnect SDK using polling. There is a loop that continually calls SimConnect_CallDispatch(). This is similar to a kid in the back seat of your car asking "Arewethereyet?Arewethereyet?Arewethereyet? Arewethereyet?..."

It's not actually that bad. The CallDispatch() call is followed by a Sleep(1) statement. When the operating system gives this piece of code its time slice, one call to CallDispatch() goes out and the code releases the remainder of its time slice. The "1" parameter tells the operating system not to wake it up for at least 1 millisecond. If you've set up the data collection to happen at the end of each FS frame and you're getting 20 frames per second, you're making 50 calls to get the data sent.

SimConnect do much, you're simply setting a flag that tells SimConnect you're ready to receive data. Still, the call does result in a context swap, and maybe you have a lot of separate connections to SimConnect ALL beating on it for data. You might then begin to incur some serious waste.

The alternative to polling is to have SimConnect tell your program when it has the data ready. There are two ways to do this. One makes use of the Windows messaging system. When you first open the SimConnect connection you can give SimConnect a handle to your window and a message to send to you. When the data structure you requested is ready, SimConnect sends your window procedure the message. You then call CallDispatch() to collect the data. Of course you have to have a window to use the messaging system. You can set up a bare bones window for this purpose, but if you wouldn't otherwise use a window you can use an event instead of the Windows messaging system.

An event is a flag that is set up by a program and maintained by the operating system. It is accessed through a handle and can be set and cleared. It's used to synchronize execution between different threads or programs. You create an event using the CreateEvent() call. If you pass the event handle to SimConnect when you first open the SimConnect connection, SimConnect will set the event when data is available. You can use the Windows function WaitForSingleObject() in your program to synchronize its execution with the availability of data.

I'm finally getting to a point that I can almost write the "Interfacing with MSFS" chapter of my upcoming book, Building A Recreational Flight Simulator. Not surprisingly, SimConnect figures prominently in it.

Scenery Display & FSX Data Output with SimConnect (25 Jan 2007)

After my visit with Tap Plastics a few weeks ago I got interested in the optics behind scenery display systems again. It's an interest that pops up fairly frequently, but I usually don't have the time to make much progress. This time I used two new tools.

The first, geometrical ray tracing, isn't really new, it's just new to me. Based on a few reasonable simplifications to how mirrors and lenses deflect light, it allows you to gain a basic understanding of how an optical system behaves. Armed with this deceptively simple tool, I turned my sights on a basic mirror collimated display. Lo and Behold, things began to make sense. I had read the words before concerning what a collimated display did, but I was finally able to tie it back to a couple of proprioceptive depth clues.

I also took a closer look at using Fresnel lenses in front of monitors. I had always been confused by the range of comments posted by supporters and detractors. Why the difference, and what was really going on? Well, I think I have a better grasp on that as well. A lens (Fresnel or otherwise) can be used to enlarge the apparent size of a monitor. If you position the lens to magnify the image a great deal, you also produce significant (and to my taste, unacceptable) geometric and chromatic distortion, particularly around the edges. Moderate magnification results in some arguably acceptable distortion. With moderate magnification, you can blend the images from adjacent monitors into a continuous panoramic view. With a single monitor, a lens allows you move the monitor away from your point of view and maintain the same apparent image size. For some people, having the

The second tool is new. It's Google's Beta patent search. There is an incredible wealth of information in patents. Yes, I know it's been possible to search patents on line for several years. IMHO, the user interfaces have been a real pain in the backside. Google makes it easy. Okay, I'll admit it would be nicer if the patents weren't presented as image data, but I can live with it. Anywho, a few searches on "simulator display", "collimated display", etc. and all sorts of interesting material pops up. With a little patience you get a good look at the history and technology behind simulator display systems.

Of course, as I had put all this effort into the topic, I dropped my foray into modern day programming self-education and cranked out a draft of the chapter on simulator display systems.

I did mange to spend a little time with FSX. (I emphasize the word little.) I got a very basic data output example limping along. Mostly, it pointed out a shortcoming in another project with the result that I had to do a bit more micro controller assembly debugging.

Acrylic Mirrors, FSX Data Output & The REAL Programming Problem with Memory Leaks (5 Jan 2007)

First of all, the best of New Years to everyone.

Acrylic Mirrors

I just came back for a very informative and pleasant trip to my local Tap Plastics. I had some questions about heat forming mirrored acrylic. I specifically wanted to know if the mirror backing would come off when heated and, if not, how far it could be bent before it did. The person I asked said he didn't know but would find out. I watched as he heated a fairly large "scrap" of mirrored acrylic and proceeded to bend it back and forth with no separation of the mirror backing at all! I was very impressed with both the mirrored acrylic AND with the level of customer service of Tap Plastics.

Why the interest in plastic mirrors? Well, curved mirrors are central to many panoramic display systems. Professional flight simulators make use of flexible mylar membrane mirrors that are held in shape by a partial vacuum behind them. These are simple in concept but quite difficult in the execution. I recall heat forming of acrylic as junior high school shop projects (a very long time ago). I wondered if it would be possible to make a curved mirror suitable for sim use by thermo-forming mirrored acrylic. Hence the trip to Tap Plastics.

I don't have a definitive answer as yet, but so far it looks feasible.

Getting Data From FSX

I continue to explore SimConnect. The SDK contains a number of code samples that highlight various SimConnect functions. A very helpful sample is the "Request Data" example. It waits for a flight to start then asks for a few bits of data on the aircraft. It works great and has been easy to modify to continually extract a variety of data.

I've also been re-creating the basic test code I used when developing the prototype instruments for the book Building Simulated Aircraft Instrumentation. I originally wrote the code using VC++ version 5. My attempt to recompile it using VC++ .NET Express Edition didn't work too well. (Crashed and burned, in fact.) I

So, I now have the two halves to the FSX data output demo program that goes into book 2. Just have to glue them together. It's always a hoot when the pieces come together and things start working the way they're supposed to.

Programming Memory Leaks

Memory leaks are generally considered to be programming problems associated with improperly releasing computer memory. This occurs when a program dynamically asks for and is allocated additional memory, but loses track of when it is no longer needed. A program that runs long enough can accumulate enough memory that it squeezes other applications out, causes itself to be shut down and/or crashes the operating system.

There is, however, a more insidious memory leak, a biological leak.

It happens, oh, perhaps in your late fifties, when you realize you're working through your fourth, fifth, sixth(?) intro programming book. The stuff goes in, but some of it leaks out! Getting old sucks, but I guess they're right. It's better than the alternative!

FSX SDK & Multiple Monitors under WinXP (28 Dec 2006)

Learning SimConnect

I'm drifting into software mode. I plan on including code samples in my upcoming book that show how to interface simulator hardware to Microsoft FSX. So, I've begun working through the code examples in the SimConnect SDK.

In the beginning, this is really an exercise in properly configuring the coding environment. The fact that the code examples compiled and worked for ACES team members is no guarantee that they will work for any random hobbyist.

Like me, for instance.

I don't have the experience of an ACES team member, a building full of talented and helpful people to call upon or anywhere near the software development tools they have. But, hey, that's all part of the challenge and fun!

The SimConnect SDK specifically says that the express version of Visual Studio and VC++ can be used to compile and run the code examples. What is doesn't say is that the express version is not all that's needed. When first trying to compile the "open-close" example, it didn't get past the first include directive. Seems the express version of VS doesn't come with the headers for building windows apps.

These are trivial issues for those with even moderate programming experience. They are rather more challenging for those just beginning. If you're just starting you will need to:

1. Install the express edition of Visual Studio and VC++.
2. Install the Windows platform SDK (free from Microsoft)
3. Add a path description in Visual Studio's configuration that points to the folder holding the simconnect.lib file.
4. Don't forget to put a copy of simconnect.h in you code project folder.

Fortunately, there are some good resources for us beginners. If you're into C#, visit NotASenator's Blog for a growing list of tutorials. For general questions, you can

Multiple Monitors

I'm slowly exploring multi-monitor configurations for use in flight sim. I'd like two or three monitors for scenery display and a fourth for a soft instrument panel. I intend to experiment with Fresnel lenses, and find out just how far I can push a single processor sim system. The SLI video card configurations look attractive, but I'd like some first hand experience before talking them up in the book.

I've got a GeForce 6800XT video card. It's a dual headed beastie that gives reasonable bang for the buck. It's capable of SLI pairing, though I've not yet bought a second card. For a while I ran a pair of Samsung SyncMaster 740N LCD monitors (with analog inputs) on the single 6800XT. All-in-all, not too bad. Then my daughter left for college, taking monitor number 2 with her.

Newegg had a sale on 19 inch LCD monitors, so I bought a Samsung SyncMaster 941BW. It's a wide screen model with 1440x900 resolution. Unfortunately I couldn't get it to play well with the 740N in dual monitor mode. When Newegg had another sale I bought a second 941BW.

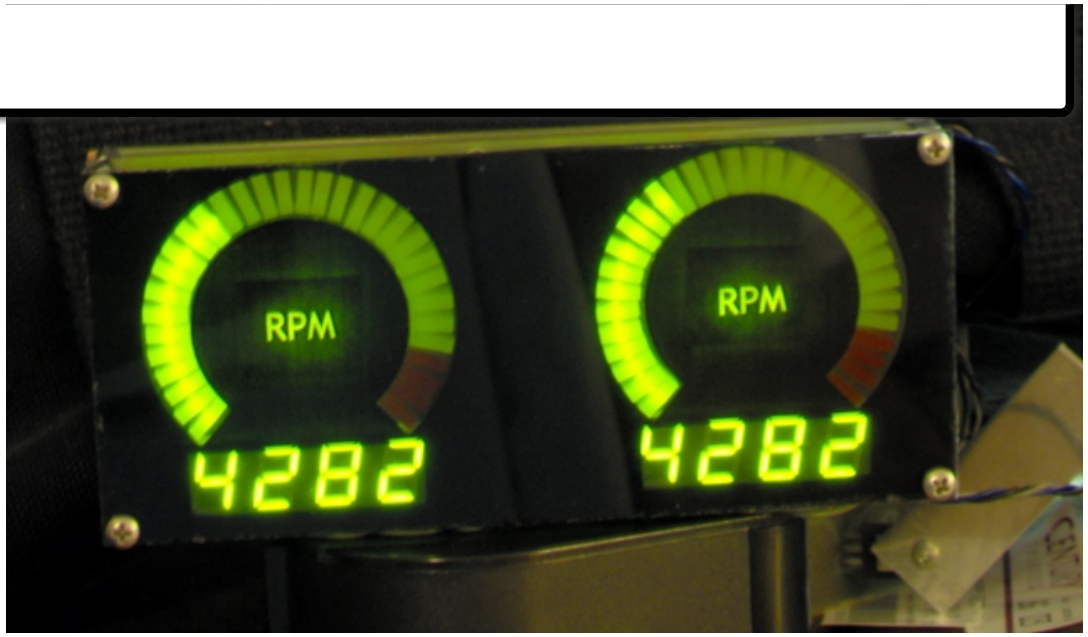
I've had on heck of a time getting the two 941BWs set up in a multiple monitor configuration. Just plugging them both into the video card caused Windows to blue screen before it could finish booting. After updating drivers, I was able to have them both plugged in and boot successfully, but still got the BSOD when I tried to enable the second monitor. I finally got a dual monitor config working by using analog inputs on the monitors.

This offends me because the video card has dual DVI outputs. The monitors have both analog and digital inputs. It's digital data into the video card, digital data out. LCDs screens are digital. I should be able to use a digital connection. It really, really annoys me that I can't and don't understand why.

The electronic gauge project is working (4 Dec, 2006)

The prototype for dual LED-based electronic gauge project is turning on its LEDs and digits under computer control. Most of the debugging issues were the result of writing firmware for a PIC16F628 and expecting it to run on a PIC16F876A. It's the same instruction set, but the 876A has a few extra control registers and its control bits are not always in the same place as in the 628. Well, "RTFM", as they say.

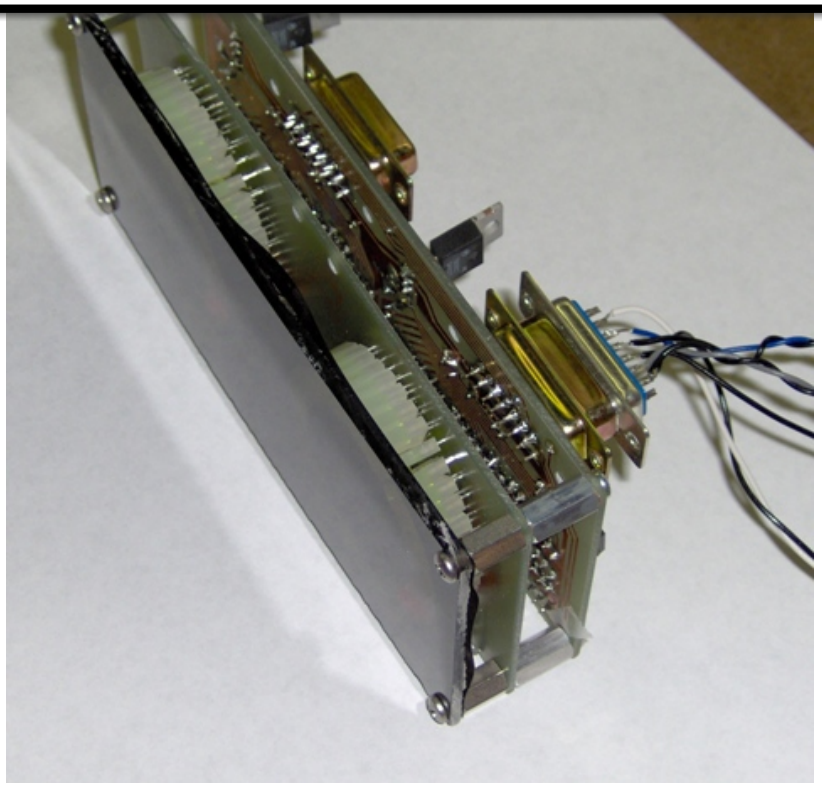
With apologies for the poor photos, here's what it sort of looks like. The pictures don't provide the same contrast you get when viewing them for real. The fuzziness, however, accurately depicts what I see after working on projects for too long.



Both sides are displaying the same numbers because I haven't gotten around to giving the two sides different instrument IDs.

Here's a shot showing the project pretty much buttoned up. Guess I should put a couple of small heat sinks on the voltage regulators. They get a bit warm to the touch . I'll do my standard piece of scrap metal trick.

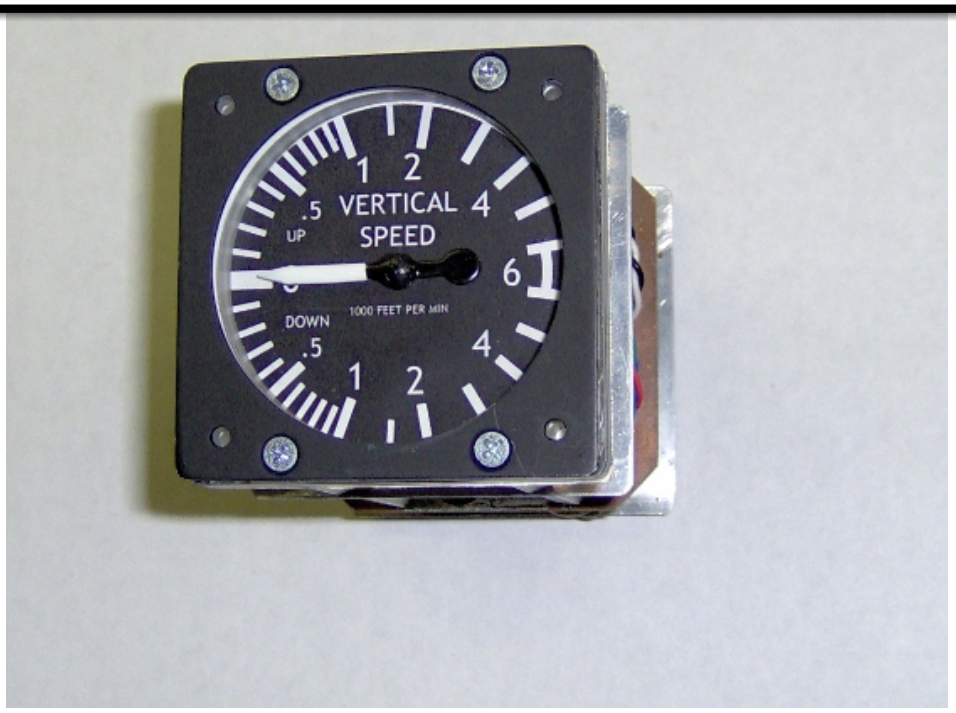
The two connectors are data in and data out. The project uses an RS-422 multi-drop connection that can be daisy-chained through a fair number of devices.



Revisiting the DIY Instruments chapter and the electronic gauge project (28 Nov, 2006)

I have a strong draft of the DIY Instruments chapter. It overviews the use of stepping motors, air-core movements, RC servos and LEDs for making instruments. It's a big chapter because instruments have an important role in building and supporting the flight simulation illusion.

There are two projects. The first is a updated stepping motor based VSI. It uses a PIC16F648A micro controller and communicates through a multi-drop RS-422 serial connection. Using a very simple RS-232C to RS-422 converter, a reasonably large cluster of devices can hang off a single PC serial com port. (Plans for just such a converter are included in another chapter.)



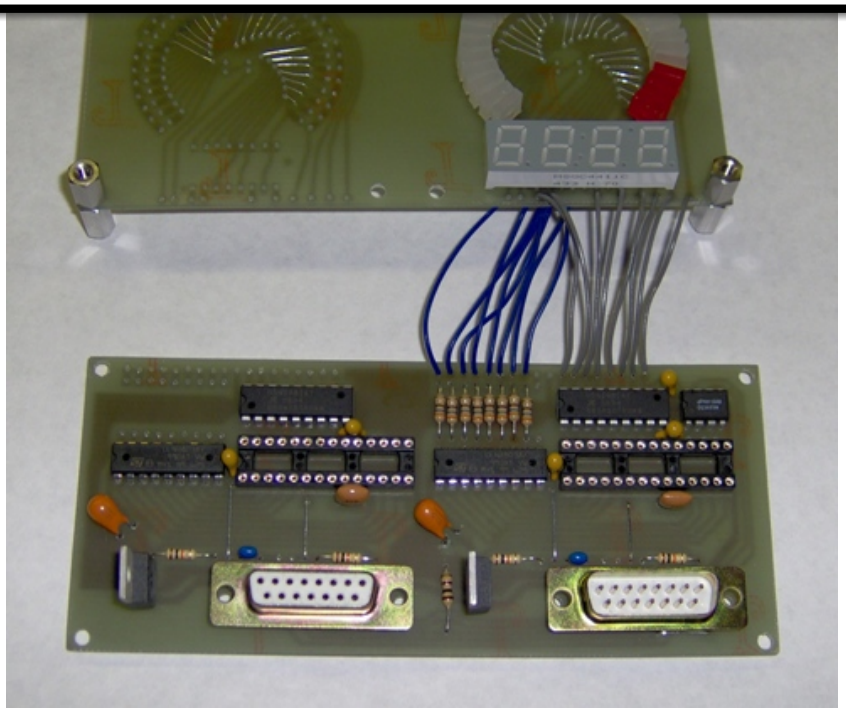
The second project is an electronic gauge patterned after the gauges that were developed to replace their electro-mechanical cousins. This project has a 270 degree arc of 32 LEDs that mimics the motion of a physical pointer. Just below the arc is a 4-digit, 7-segment display that gives a precise reading. The project description includes a set of circuit board layouts for a pair of side-by-side gauges. They share the I/O connectors and data receiver, but otherwise are separate gauges. You could stack half a dozen of these dual modules on an instrument panel for your engine gauges. Bussing the power connections and the received serial data line would give you a cluster of gauges that needs only a single cable connecting it to your sim.

The first version of the LED gauge was based on a 3" instrument. I made the boards for the prototype, but decided there was too much room between the LEDs. I didn't have enough I/O pins on the micro controller to add more LEDs, So I decided to try reducing the gauge to a 2 inch diameter arc. This turned the redesign into a major pain, but finally seems to be working out.

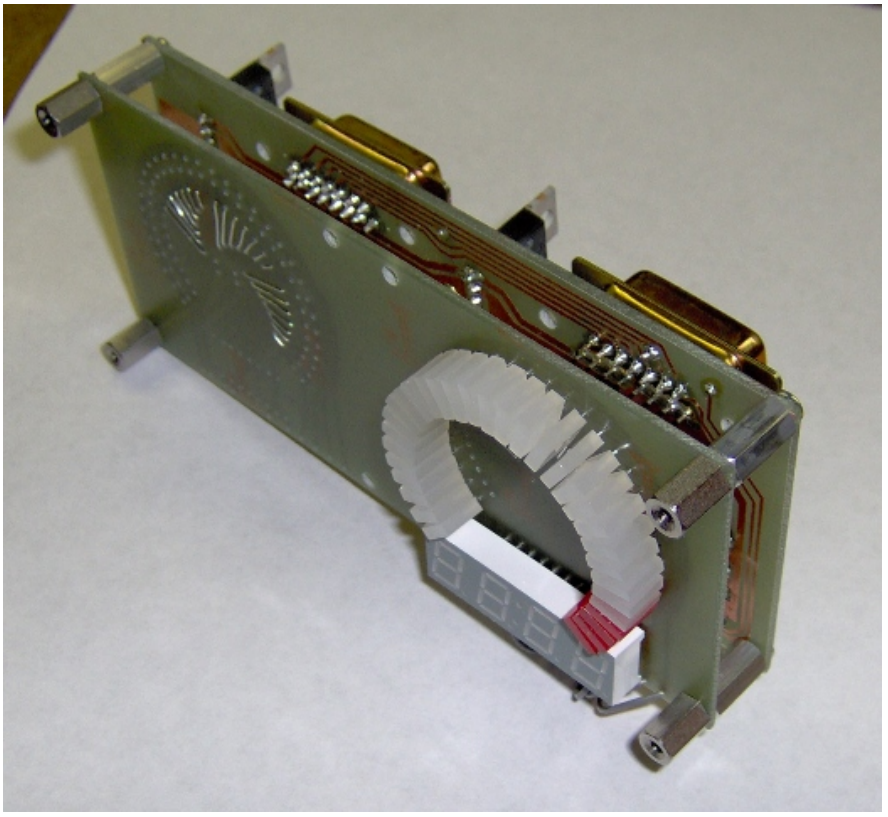
It would be much easier (for me) if I used double sided circuit boards, but I am restricting the board designs to single sided. The rationale is that it's much easier to make a single sided board at home. If the people who buy the book want to use double sided boards, they will most likely use a commercial board prototyping service. This will require them developing a board layout in an electronic format. If they're going to do that, it's a small step to convert to a double sided design. However, if I present an optimized, double sided design in the book, it can be a real pain to convert to single sided for the guy who wants to build the boards at home.

Here are a few pictures of the 75% built prototype. (I won't add the rest of the LEDs until I get the firmware updated. No point in frying any more LEDs than necessary.)

24 Jun 2002 - 19 Sep 2017 2006



The logic board mounts behind the LED board. This assembly will mount to a clear plastic front piece. There will be a thin transparency just behind the plastic front. The transparency will have the gauge legend printed on it and will mask everything but the LEDs and display.



The 7-segment display and the bulk of the LEDs emit green light. There are four red LEDs at the end of the arc to indicate an overrange.

I had the firmware pretty well together for the earlier, 3 inch version. With the size reduction I did some major I/O pin reassignments to the micro controller. This means all the port write commands must be changed.

(Hmmm, been a long time since I powered up the stepping motor VSI. Guess I should re-assemble the code, making sure it works with the current free version of MPLab, and make sure it flashes the PIC properly. Oh, it'll work, np, right? right???)

Go to Old Stuff for previous rants.

Something, somewhere in this site was tweaked, changed, added, deleted or otherwise updated on 02/24/07.