

LESSONS FROM THE TRENCHES

George Martin

Structured Design

Part 2: Putting Theory into Practice



As I stated last month, using Nassi Schneiderman flowcharts helps me develop a more structured design. And, I find that the more a design is structured, the more manageable it becomes.

I hope you had a chance to look at the references from Part 1. If you did, you probably noticed that I left out case, or switch, constructions. I thought it would be too much to include, as I already had enough to cover. But when I started writing the second part of this series, wouldn't you know that one of the first constructs I needed was best shown as a case construction. So, allow me to back up a little and cover the case and switch constructs.

Depending on a value of a variable, different tasks may need to be performed. Let's say that the keyboard's input of 0 to 9 is to perform 10 different functions. Listing 1 is an example of what you could code until all the cases are covered.

Most languages have a switch or case statement. And in C, it goes something like what you see in Listing 2.

Revisiting a former project by Jeff Bachiochi, George continues with his use of Nassi Schneiderman flowcharts to develop a structured, manageable design. George originally intended to include case and switch constructs, so he picks up with that this month. So, allow him to put this structured vehicle in reverse and back things up a bit. The end of the road holds the tool to help isolate problems..

The Nassi Schneiderman construction can be seen in Figure 1.

I only diagrammed the 0, 1, and 2 processes and added a default process. But, you can see that this construction has only one input and one output. And depending on the switch variable, only one set of processes are followed.

DIGGING IN

OK now, let's do a real project. Actually, I don't have a project to lay out using the Nassi Schneiderman charts. The best flowchart would be your next one or the one that's giving you the most problems. However, I do have a suggestion. Jeff Bachiochi gave me permission to redo a project that he published in *Circuit Cellar* 126.

Imagine that one of his designs has gone into production, and you're in charge of the production line. It's a high-volume line, and the goal is a six-sigma failure rate (3.4 defects in 1,000,000 operations) off the assembly line. Therefore, you need to understand the software and investigate all possible sources of error in the design and figure out how to eliminate them. There may not be any errors in the design, but remember that the goal is six sigma, so you need to chase away even the most remote possibility.

There are really three flowcharts in Jeff's figure (see Figure 2). The lines and boxes on the left two columns do not interconnect with the other two

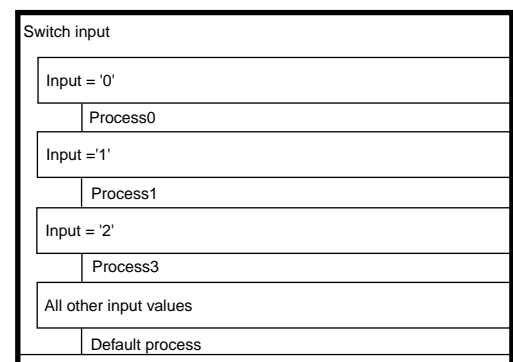


Figure 1—Here you can see the switch, or case, construction set up as a Nassi Schneiderman chart.

self-contained charts found in the remaining columns.

My conversion of the left two columns into a structured design using Nassi Schneiderman flowcharts is shown in Figure 3. Note that I've added some general routines that are not in the original flowchart. They are probably in the code, but if not, then it means that there are some hardware and variables that have not

been initialized. A good place to start is with the six-sigma goal.

Also, notice that I didn't include any of the second column. Those procedures are buried in the ProcessModem routine. A good rule of thumb is that, if you can't fit the design on one page, it's too complicated. And, the routine that writes a message just didn't fit this logic.

MAKING CHANGES

Jeff's original flowchart did not meet the requirements of a structured design. I could not convert the original diagram into an NS diagram without some changes. If I changed the logic, I would defeat the purpose and premise of this article. You've got to ensure that the original work is as error-free as possible. So, when converting the original design into a

structured design, I needed to add flags (or markers) to let me navigate through the program flow. For example, in an unstructured design you can use a goto statement. In a structured design, you need a flag rather than the goto statement to control the program flow.

I added a couple of flags for bookkeeping in all this logic. The Done flag keeps the program running forever. The InitFlag gets into the initialize routine. The StartOverFlag is used to force the code to start over without initializing. And, the GPSAlarm and GPSResult variables are the results of interrogating the hardware.

While constructing this chart, I questioned what would happen if the ReadGPSResult() routine returned something other than 1, 2, or 3. Perhaps that's some defensive coding I should investigate. What do you think?

Remember, you're aiming for a six-sigma failure rate. Do you see how the NS diagrams represent the flowcharts? How the logic has become flattened, and how it's straightforward to trace the program flow? How I added flags to remember and control software operations?

It took me approximately six to eight hours to reshape Jeff's work. So, it takes

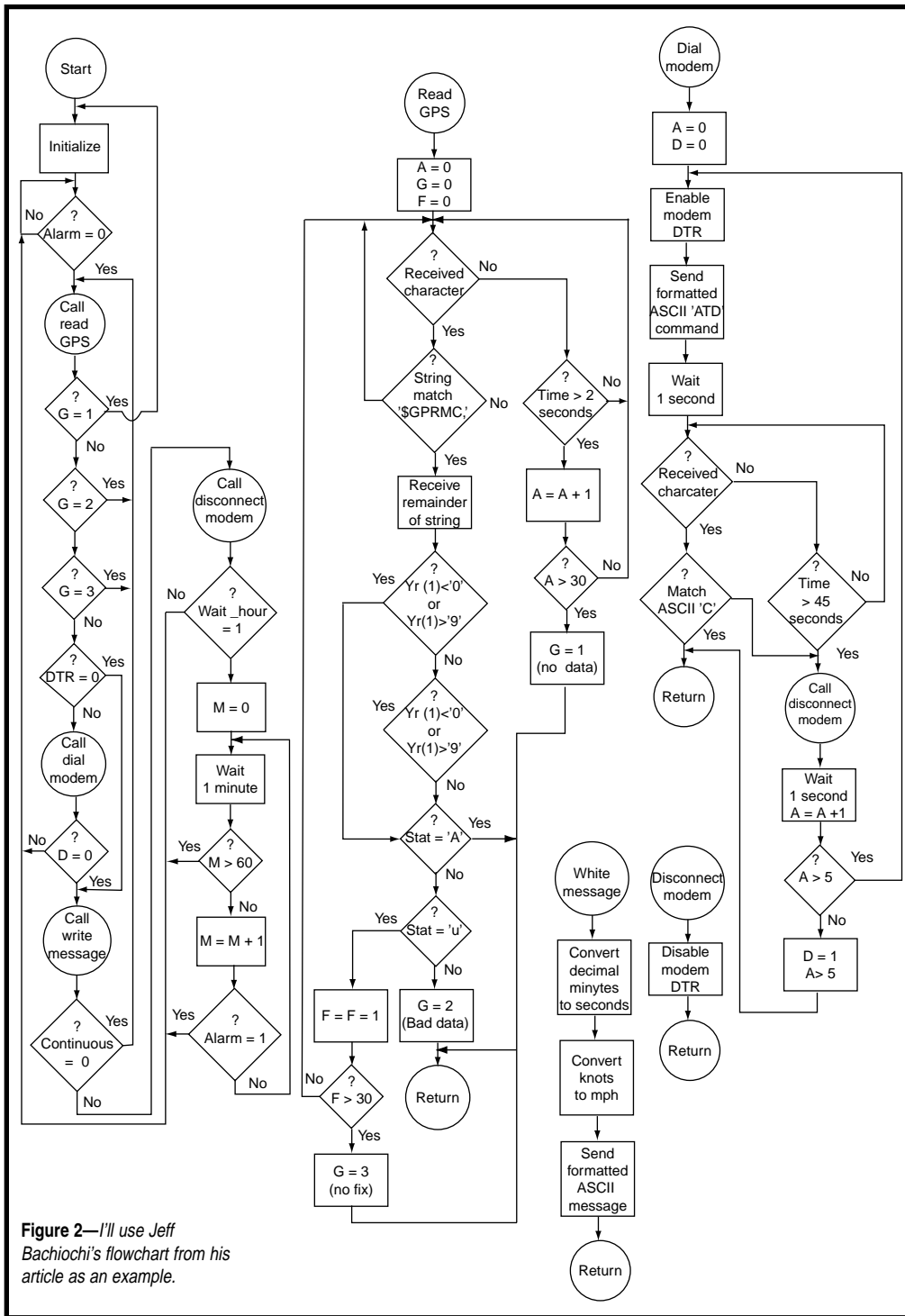


Figure 2—'I'll use Jeff Bachiochi's flowchart from his article as an example.

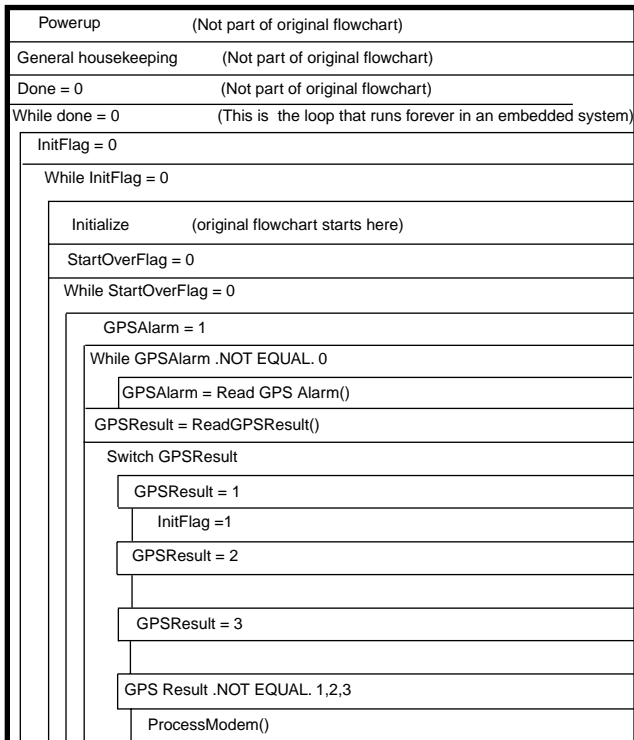


Figure 3—The process's GPSResult can be seen here.

some effort to structure the design. Is this NS representation clearer than the original? Perhaps it is, or maybe it's just another tool to help isolate problems. Especially those once-a-week type of problems.

TRY IT OUT

Let's continue with ProcessModem() (see Figure 4). This routine is taken from the last line of Figure 3. Again, structured code will let you hide the details of implementation until you need to look into them.

That's all there is to it. Try using NS representation on the rest of the flowcharts in Jeff's article. It will take

Listing 1

```
If Input = '0' then do Process0
Else If Input = '1' then do Process1
Else If Input = '2' then do Process2
Else If Input = '3' then do Process3
Else If Input = '4' then do Process4
Else If Input = '5' then do Process5
```

Listing 2

```
Switch (Input) {
Case '0':
Process0;
Break;
Case '1':
Process1;
Break;
... And so on...
}
```

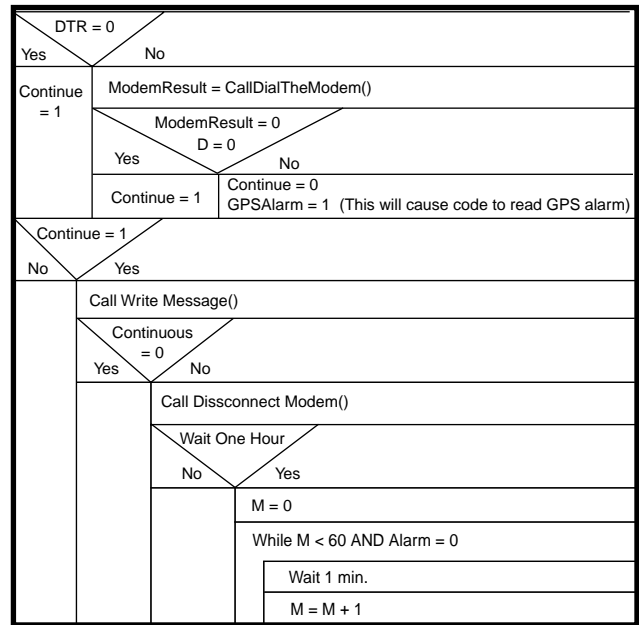


Figure 4—The last line of Figure 3 can be broken down to show the ProcessModem() routine.

some effort, so e-mail me if you get stuck. I'm sure you will have to add flags and flatten out the design. But hey, that's why we get paid the big bucks, right? 📧

George Martin began his career in the aerospace industry in 1969. After five years at a real job, he set out on his own and cofounded a design and manufacturing firm. Typical systems that George designs include servo-motion control, graphical input and output, data acquisition, and remote control. George is a charter member of the Ciarcia Design Works Team and most recently, he's been working on the people-tracking system for Bill Gates' new house. You can reach him at george.martin@worldnet.att.net

REFERENCE

J. Bachiochi, "Where's Waldo?—Pinpointing Location by Interfacing to a GPS Receiver," *Circuit Cellar* 126, January 2001.