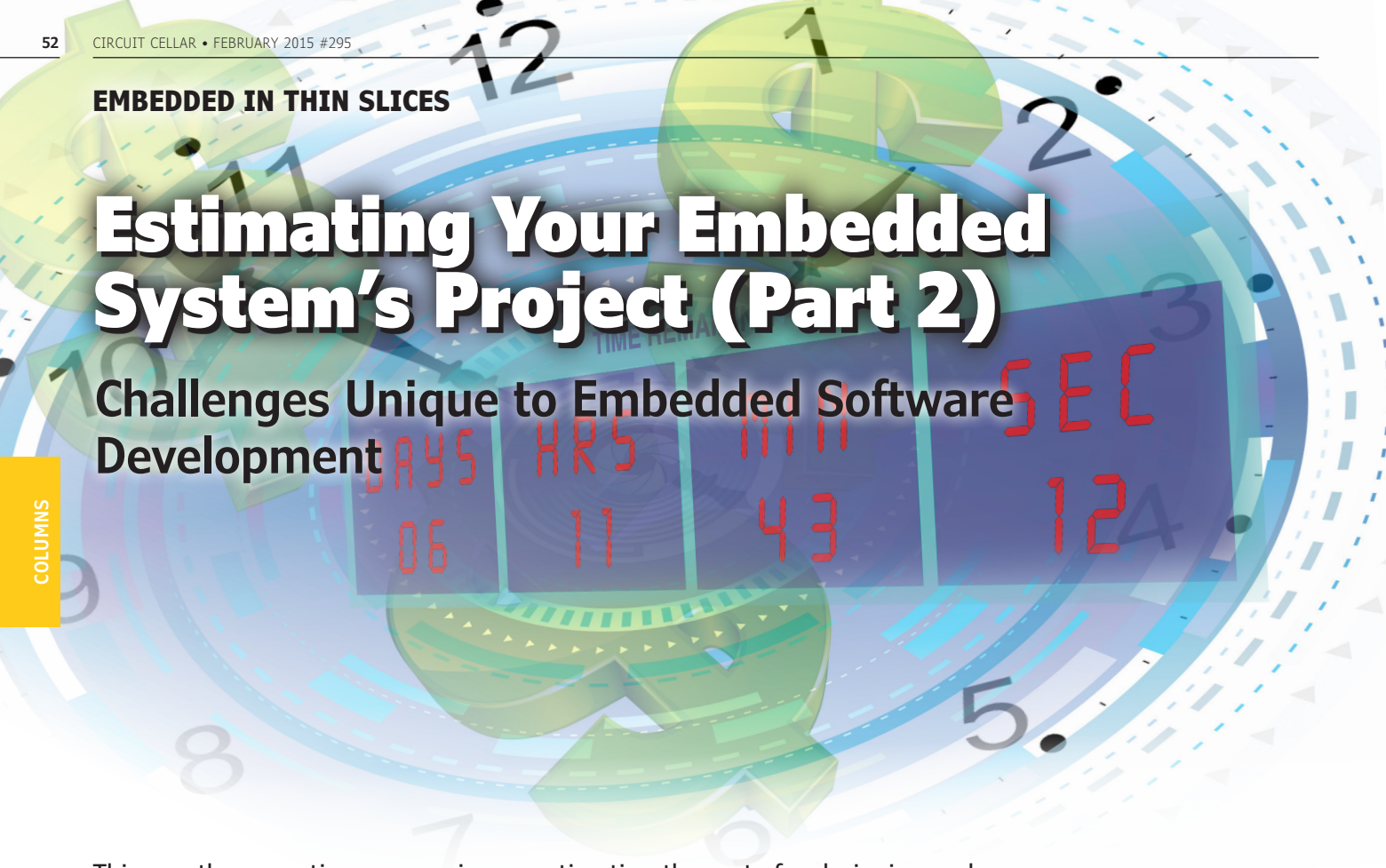
EMBEDDED IN THIN SLICES

Estimating Your Embedded System's Project (Part 2)

Challenges Unique to Embedded Software Development



COLUMNS

This month we continue our series on estimating the costs for designing and developing your embedded systems project. Bob covers the issues unique to estimating an embedded systems project.

By Bob Japenga (US)

Recently, we were asked to estimate the cost to develop a system that would interface with every device of a particular type manufactured in a particular country. Our job was to design a system to extract data from these devices. There are 10 international standards applicable to these devices in this country. These standards define the protocol for accessing this data. The data is available on one of three possible hardware data busses. Some of the data available on these busses is in the public domain and some is only available from the manufacturer. There are over 150 different types of these devices sold each year in this country. Each year, another 150 new or similar types are sold. The specification is perfectly clear. By the way, can you have the estimate to me by Monday? Hmmm! I saw an advertisement the other day that said a company's one-day seminar would teach me to accurately estimate firmware schedules. Maybe if I took that class on Friday, I could get the accurate schedule by the end of business on Monday.

How can one estimate something like this? Here is axiom number one: Don't believe anyone who tells you he can teach you to accurately estimate your firmware schedule in a one-day seminar. Or in a one week seminar. Or even after 10 years of doing it every day. Accurate? No! But we can get better. And the best way I know how is by first defining the problems. That has been the focus of these first to articles in our series.

Last time, we looked at the general problem of estimating software development costs. This month we will look at the challenges that are unique to embedded software development. Certainly there are things that make embedded software more challenging to develop than other types of software. But what makes embedded software that much harder to estimate?

BIGGER SURFACE AREA

Recently, I reviewed last month's article with our team and asked the question: Why is estimating embedded systems more difficult than estimating other kinds of software? One engineer said, "The surface area is much bigger." What he was saying is that all of the standard problems with estimating just got

multiplied. Let's just review what we said last time and see how some of these issues are more complicated for embedded systems.

UNCLEAR REQUIREMENTS

The accuracy of our software estimates can only be as good as our understanding of the requirements. This difficulty is multiplied with embedded systems because of the complexity of the interfaces. In addition, there are a lot of requirements that only become clear after you implement. The datasheet of a small microprocessor we use on one project is 1,400 pages long. There are just a lot more requirements that can be unclear or misunderstood. We approved a rework once to one of our designs that required the manufacturer to add a wire to one end of a capacitor. After the first few thousand were shipped, the capacitors started shorting (especially problematic for bypass capacitors). Buried in the capacitor's datasheet was the requirement to not touch the capacitor with a soldiering iron. The rework needed to be performed with a hot air process. It was very clear on page 78 of the capacitor's datasheet!

The specifications can also be wrong. Many times errata come out after you have started your design. We once missed an errata in an 800-page microprocessor datasheet that said, "Oh, by the way, this device has a 256-MB address range but can only address 16 MB of NOR flash!"

THAT ELUSIVE BUG

Embedded real-time systems and systems with concurrency make debugging much more difficult. That we can plan for. But those elusive bugs that take two weeks in non-embedded systems can take two months on embedded systems because your tools are not as powerful and the complexity of the design is that much greater.

HIDDEN COMPLEXITY

The scale of complexity is greatly multiplied in embedded systems. We are supposed to write software that interfaces with other very complex devices. Take this simple requirement from a datasheet of chip we interface with: "To reset the chip, hold RESET_N low for 300–500 ms." On the surface that seems straightforward. But what is hidden and not written in the manual is that if the RESET_N is held low for more than 1,000 ms, the chip powers down and will not start when the RESET_N line is brought high. If for some reason your function that releases RESET_N gets delayed, the chip would not become operational as you expected.This requirement of raising RESET_N becomes a hard deadline that you might not expect to be as such.These kinds of hidden complexities are legion in embedded systems.

PROGRAMMER EFFICIENCY

Two years ago, I sat with one of the best embedded designers I know. He was running out of real time on a project. The problems were so complex that it took two of us with a combined experience of 60 years of designing embedded systems to figure out what was going on and how to fix it. Where a less-efficient programmer might be four times less efficient than your best designer, in an embedded environment that same programmer might be 10 times less efficient.

OPTIMISM & HUBRIS

A couple months ago, one of our customers asked us to add a splash screen and a progress bar to the start of a device. One of our best designers saw that u-boot had hooks for sending an image to our display. Linux had a progress bar app (psplash) that worked with our display. (If you want to have an open-source progress bar for Linux (psplash), check out the Yocto project's distribution http://git.yoctoproject.org/cgit/cgit.cgi/psplash/tree/.) The system was built on a BeagleBone architecture so others must have done this before. The on-line community support for this architecture is huge. We knew we could get lots of help. In addition, we have done similar projects in about four days without this kind of support. We know how to do this. We can deliver this fully tested in four days. (The BeagleBone open source reference design is showing up in the designs of a number of companies. You can find more about it at http://beagleboard.org/.)

At the end of four days, we found that the hooks in u-boot didn't work. No one in the online community knew how to make them work. At the end of two weeks, we discovered that the u-boot image was inverted from what the Linux driver was expecting. At the end of four weeks, we discovered that the progress bar did not play well with this particular display. At the end of six weeks, we discovered that the customer did not provide us with the right code base to start with. We were optimistic. Embedded systems will amplify the negative effects of your optimism and hubris enough to put you out of business. (The u-boot open-source universal bootloader software has been at the start of every Linux project we have designed.You can find more about it at www.denx.de/wiki/U-Boot/WebHome.)

CUSTOMER SCHEDULE CREEP

Customer schedule creep is a specific instance of "The Other Guy" problem we

ABOUT THE AUTHOR

Bob Japenga has been designing embedded systems since 1973. In 1988, along with his best friend, he started MicroTools, which specializes in creating a variety of real-time embedded systems. With a combined embedded systems experience base of more than 200 years, they love to tackle impossible problems together. Bob has been awarded 11 patents in many areas of embedded systems and motion control. You can reach him at rjapenga@microtoolsinc.com.

talked about last time. But it has a unique feature to it. We are now six months behind schedule releasing a new version of embedded software for a product we designed for a customer. One of the driving factors in the delay is that the customer still doesn't have their portion of their web server operational. Every day it slips, our team has to work on other things instead of completing the testing. Each day the team might spend a half hour coordinating with the customer. None of this 60 hours was estimated. The inefficiency of this schedule creep is even more costly. Fred Brooks in *The Mythical Man-Month* puts it this way: "Disaster is due to termites, not tornadoes."

Some of this is common to non-embedded software. But embedded software by its very nature is embedded in stuff. And often, stuff that is being designed in parallel. As a minimum, it must talk with hardware that is often not completely designed. It may also talk with other machines that are being developed in parallel. How well those are designed and when they are delivered can be a multiplier in the schedule and cost of an embedded system.

PARTNER QUALITY

Another instance of "The Other Guy" problem is with your partners. Some of the partners we interface with are the hardware we run on, the busses we communicate on, the networks we connect to, the other devices we talk to, the hardware designer who designed our board, the hardware layout team that laid out the PCB, and the hardware build team that actually built the board. How well they do their job has a direct bearing on how much it will cost you to develop your embedded system.

Let me share two examples. We have a supplier who builds our printed circuit boards and assembles them during our development stage. We love this supplier because their work is impeccable. Sometimes our customers require us to get the boards built someplace else or by them. Invariably, parts are put in backwards. Ball Grid Arrays (BGA) parts are not X-rayed to verify their connections. Flow soldering techniques cause modules to reflow and not re-center on their footprint. When we get the boards, it might take us two to three days more to debug and troubleshoot these problems because of the supplier. Remember that we are checking out a new design which can have flaws in it as well. How does one estimate for that extra two to three days? You don't know the quality of that supplier until you have used them.

Another problem we have is with other hardware designers. When we design the boards, we know the quality factor of our designers. They may not be perfect, but they are a known quantity. We know by experience how long it will take to integrate the boards designed by our own people because we have metrics and experience. But what if you are designing embedded software that runs on a board that is designed by "the other guy?" Our experience shows that it can ruin a schedule in two ways. The first is the extra time it takes to "bring the board up" because there are more errors in the design than you are used to. This can easily add several weeks to a schedule. But often we find that it takes more turns of the board than it normally takes you to get an operational board. During that extra two to three weeks, your team is much less efficient. Do you assign them to a new project? That is not practical. So the software team becomes less efficient. They work on "cleaning up the code" and "doing some documentation." Sounds good, but these are schedule killers. And for estimating, the problem is: how do you know the quality factor in advance?

TESTING DIFFICULTIES

Embedded systems are much more difficult to test than conventional software systems. That additional difficulty can be planned for and the estimate adjusted to take that into account. The problem comes when we don't think through these difficulties when we estimate the project. We developed a tiny embedded device that was implanted into a human body. This device communicated to the outside world via infrared. The device sent 8 bytes every millisecond. We accurately estimated the time it would take to design the hardware and the software necessary to accomplish these requirements. However, when it came to test it, we did not have a means to easily do that. There were no off-the-shelf tools to read the IrDA and provide an integrity check to it. How does one know that all 8,000 bytes are correct every second?

A special test tool was needed to display and analyze that it was meeting its requirements. But special test tools take time and money to design. They can drastically expand the effort required to design and develop an embedded system.

Another thing that can affect our ability to estimate embedded system is the time delay inherent in many designs between making a change, testing the change and reprogramming the device. When the time delay is very small (as in non-embedded systems), iterative designs can be created much more quickly. Where this impacts our estimates is that we often don't know what the time delay is and exactly how it will impact the schedule. For example, let's imagine that over the course of the project you make 1,200 changes to your software requiring a compile and load. If the compile and load time takes 70 s compared to 10 s, this can add three extra days to your project. Often, during the time we estimate, we don't know with that precision the compile and load time.

FACE THE IMPOSSIBLE

The surface area of complexity in estimating embedded systems is many times more complex than designing non-embedded software. Knowing what some of the problems are can help us get better at this impossible task. Next time, we will look at how we can address these problems and get a little better. If you have some other suggestions about the problems in estimating embedded software systems and how you deal with them, drop me a line. This is a field in which I need constant improvement. And of course, I only improve in thin slices.