The Embedded Muse 43

Editor: Jack Ganssle (jack@ganssle.com) January 14, 2000

Editor's Notes

In response to many requests I've (finally!) archived all of The Embedded Muses, from issue 1 to this one, at http://www.ganssle.com in .PDF format. I've also rescanned the old Signetics Write Only Memory datasheet (it's now much easier to read); find it at http://www.ganssle.com/misc/wom.html.

If you'd like me to present the full-day "Better Firmware Faster" seminar at your company, please drop me a note sooner rather than later, as there's only a limited number of these slots.

Finally, are you sick of the toolchain you're using? Or perhaps you think the product is the best thing ever produced. Why not share your experiences with the embedded community? A big problem we all face is selecting reasonable tools. The vendor hype may be right on target… or it might be hype. Wouldn't it be nice if there were a repository of real world user experiences with various tools, that prospective buyers could check out before investing time and money into a product?

I've decided to try and collect real user experiences with various toolchains. If you're using a particular compiler, editor, debugger, or other tool let me know what you think of the product and the company. Give me the product version number and approximate date, and a synopsis of the good and the bad points of your experience. I'll post the results (if any!) to my web site. Let me know if I can include your email address so others can correspond directly with you to get more info.

Use the following questions as a guideline for replies:

- Vendor name:
- Tool name:
- Tool type (compiler, emulator, etc):
- Tool version number:
- Approximate date you used this product:
- Target CPU:
- Host platform (PC, UNIX, etc):
- Your email address (optional):
- May I include your email address in the listing, so potential users can contact you directly?:

- Overall rating of the product on a scale of 1 to 9, 1 being unusable, 5 adequate, and 9 being about perfect:
- Overall rating of the vendor's tech support on a scale of 1 to 9, 1 being they never answered you calls/emails, 5 adequate support, and 9 being astonishing support at every step of the way:
- Did you manage to complete your project using this tool?
- List the product's best features:
- List the product's worst features:
- General comments. Please do feel free to elaborate:

I reserve the right to edit all submissions for clarity and politeness.

Metastability - Conclusion

The volume of email I've gotten about metastability has been gratifying; it shows a lot of us worry about this issue.

One theme that a number of correspondents mentioned was the virtue of gray scale sensors – encoders, for example – since during any code change only a single bit transitions. A metastable state means that you can get only the correct current value of the sensor or the previous value, which is likely a reasonable second choice.

Stephen Deasy wrote:
The real problem is, I believe, most apparent when you consider an example such as a state machine with multiple flip-flops, say four. If you don't properly observe set up times, the problem is not so much which states the flip-flops take or how long they take to switch, it is that some (flip-flops) will change state based on the new (asynchronous) data and some won't at the time of the clock
transition for various reasons (different length propagation delay paths, etcetera. The result is that the system or device will be in the wrong actual state for conditions and may not recover.

I have actually seen it work and work reliably to bring asynchronous data into a synchronous state machine. However, in order for this to be true the following conditions were true:

1. One and only one asynchronous input.
2. The flip-flop assignments were made, and the state machine design was done such that one and only one flip-flop can change at any one time (Gray coded sequence).
3. Any time the asynchronous input transitions it will remain in that state long enough for at least two synchronous clocks to occur.

My theory is that this works since, because of the Gray code, one and only one flip-flop is allowed to change for any state of the system. At the time of the clock, the asynchronous data will either cause a transition at that one flip-flop or it won't. Thus it will either remain in the current state or transition to the new one. If it didn't make it for this clock, it will definitely make it for the next. Thus the transition can delay one clock period, but it cannot end up in a wrong or out-of-sequence state.

Hans J Weedon wrote with more detail about the problem:
Digital logic is constructed from real-world devices, and like all amplifying devices they have limitations in gain and bandwidth. The devices may be bipolar transistors, FETs or even tunnel diodes. All the amplifying devices have a device limitation called gain-bandwidth product. This gain-bandwidth product represents the frequency at which the device has a gain of one.

As far as the amplifying device is concerned, it has no knowledge of which circuit it is connected into, and only does what the laws of the physics of the device is telling it to do. It is only us the design engineers using the device that observe mysterious circuit behavior. To the device itself there are no mysteries, it does exactly what it is supposed to do.

As a thought experiment, let us hook two devices up to be a linear amplifier with negative feedback and a gain of one. When this amplifier is amplifying a step input, the output settles according to an exponential function towards the final output. An exponential function never settles completely, but at some time the difference between the final settling and the actual vale is small enough to be immaterial. If we wait long enough, the error will be small enough to be ignored.

Every amplifying device has some random, thermally generated noise associated with it. As a practical engineer we can state that when the final value of a settled signal is less than the noise at the output of the device, any further settling is wasted time because the difference between final value and the settled value has "drowned" in noise.

This seems to have nothing to do with metastability in flip-flops, but stay with me, this is just a way of helping us understand the basics of metastability.

Back to the settling amplifier.

When we write the equation for the voltage waveform at the output of the amplifier, the dominant pole of the amplifier will describe an exponential function on the form $e^{(-t/tau)}$ where e is the base of the natural logarithm and t is time and tau an expression of the gain-bandwidth product of the amplifier.

It may not be obvious to all, but the minus in front of t is there because the amplifier has negative feedback. If we hook the amplifier up with positive feedback, the amplifier will settle according to the expression $e^{(t/tau)}$. In other words the amplifier will "fly to the rails" no matter how small an input. An amplifier with positive feedback is a flip-flop. The flip-flop acts, for small signals, exactly like an amplifier, except that time appears to be running in reverse. The amplifier cascades from a settled state to the final state at the "rails".

With a little bit of "hand-waving" and some intuition, we can calculate how long a metastable state can be expected to last.

It is a little bit easier to estimate the parameters of a CMOS flip-flop than for a bipolar flip-flop, especially when Shottky saturation protection is used. For CMOS the delay time from CLK to Q is some number of nanoseconds, at least in former days, the device manufacturers would give us a device schematic of a flip-flop. From that schematic we can tell how many devices there are between the input and the output.

Take for instant a CD4013B dual Data master slave flip-flop, it has a delay of 65ns CLK to Q and about 5 devices in the loop. 65/5 =13 , and we can "guesstimate" 13ns delay per stage. A rough guess of the gain-bandwidth product of this device as a linear amplifier would be 1/(2*pi*13ns)=12MHz. The Noise density of CMOS amplifiers is from experience no better than 10nV/rootHz. giving us a noise voltage at the input of about 10E-9*SQR(12E6) = 35 micro volts thermal noise.

A CMOS device has a switching threshold of 50% of the rail voltage. The specs I used above were from 10V operating conditions, and the transition from metastability will be from 5V to zero or 5V to 10V, a 5V transition.

Ln(5/35E-6)=11.8 That means that the metastability of the CD4013B can be expected to last for 11.8*13ns = 153ns. This estimation assumes that the probability of an error is 50% at the calculated time. for lower probability a larger factor must be used.

As a general rule of thumb, the metastability of a flip-flop will last about 2-5 times the propagation delay from CLK to Q out.

Texas Instruments has good and complete papers on the issues of metastability in flip-flops, SDYA006 is one of many. This paper lists the measured performance of several families of IC technologies. Among other things it shows that 74S series of flip-flops are among the worst offenders in the area of metastability.

The ultimate in resistance to metastability, in my experience, is the fast tunnel diode. When done properly, a tunnel diode will have less than 1ns of metastability. That is one

of the reasons that the triggers used for High Energy physics electronics around particle accelerators were done with tunnel diode discriminators.

How do you design circuits that has minimum sensitivity to metastability?

In my opinion you use 74F type circuits with a clock rate less than 80MHz and you use a two stage synchronization. The asynchronous data is clocked into one flip-flop and is reclocked into a second flip-flop. If the first flop doesn't get it right, the second flop will. Above 80MHz: ECL logic will work up to about 250MHz beyond that ECLIPS logic will work to about 500MHz beyond that, I do not know. Metastability was a big problem in very high speed sloped and successive approximation A/D converters.

Thought for the Week

Mutant Marsupials Take Up Arms Against Australian Air Force

The reuse of some object-oriented code has caused tactical headaches for Australia's armed forces. As virtual reality simulators assume larger roles in helicopter combat training, programmers have gone to great lengths to increase the realism of their scenarios, including detailed landscapes and - in the case of the Northern Territory's Operation Phoenix- herds of kangaroos (since disturbed animals might well give away a helicopter's position).

The head of the Defense Science & Technology Organization's Land Operations/Simulation division reportedly instructed developers to model the local marsupials' movements and reactions to helicopters. Being efficient programmers, they just re-appropriated some code originally used to model infantry detachment reactions under the same stimuli, changed the mapped icon from a soldier to a kangaroo, and increased the figures' speed of movement.

Eager to demonstrate their flying skills for some visiting American pilots, the hotshot Aussies "buzzed" the virtual kangaroos in low flight during a simulation. The kangaroos scattered, as predicted, and the visiting Americans nodded appreciatively... then did a double-take as the kangaroos reappeared from behind a hill and launched a barrage of Stinger missiles at the hapless helicopter. (Apparently the programmers had forgotten to remove that part of the infantry coding.)

The lesson?

Objects are defined with certain attributes, and any new object defined in terms of an old one inherits all the attributes. The embarrassed programmers had learned to be careful

when reusing object-oriented code, and the Yanks left with a newfound respect for Australian wildlife.

Simulator supervisors report that pilots from that point onward have strictly avoided kangaroos, just as they were meant to.

- Reportedly from June 15, 1999 Defense Science and Technology Organization Lecture Series, Melbourne, Australia, and staff reports

About The Embedded Muse

The Embedded Muse is an occasional newsletter sent via email by Jack Ganssle. Send complaints, comments, and contributions to him at jack@ganssle.com.

To subscribe, send a message to majordomo@ganssle.com, with the words "subscribe embedded *your-email-address*" in the body. To unsubscribe, change the message to "unsubscribe embedded *your-email-address*".

The Embedded Muse is supported by The Ganssle Group, whose mission is to help embedded folks get better products to market faster. We offer seminars at your site offering hard-hitting ideas - and action - you can take now to ***improve firmware quality and decrease development time***. Contact us at info@ganssle.com for more information.