The Embedded Muse 163

Editor: Jack Ganssle (jack@ganssle.com) Aug. 11, 2008

EDITOR: Jack Ganssle, jack@ganssle.com

CONTENTS:

Editor's Notes

Did you know it IS possible to create accurate schedules? Or that most projects consume 50% of the development time in debug and test, and that it's not hard to slash that number drastically? Or that we know how to manage the quantitative relationship between complexity and bugs? Learn this and far more at my Better Firmware Faster class, presented at YOUR facility. See http://www.ganssle.com/classes.htm .

Todd Lawall wrote a simulator for one of my debounce routines (which is here: http://www.ganssle.com/debouncing.pdf). His code is in Perl, and can be found at: http://www.ganssle.com/misc/debounce-sim.pl .

James Morrison posed an interesting question – does anyone have suggestions? He wrote: We often need to debug serial busses (I2C, SPI, UART) in embedded systems. I was wondering if any of your readers could recommend a device that could be used as either a monitor or as a master/slave device in such a serial network. Monitoring is a very useful function. But often during debug you want to actually source or sink a command in order to track down the problem. A device that would do both functions would be very useful. I know TotalPhase has both the Aardvark and the Beagle for these protocols, but I'm

looking for something that combines both into one device. And of course, low-cost and high-quality.

Hacking HP

I've long been a fan of HP calculators, since long ago days when my dad gave me an HP-45 which replaced my slide rule. I think it was a second-hand machine, but at the time a new one cost $395, equivalent to over $1500 today.

HP continues to innovate in calculators, but the new HP-20B financial machine has been opened up. A reader who wishes to remain anonymous says: $40 gets you an Atmel AT91SAM7L128 CoB (arm7tdmi, 128k flash, 6k ram) bonded to a board with a 400 segment LCD, 37 keys, battery holder, nice case, etc. Best part: HP has already released an SDK of sorts, with schematics, sample code, simulator, etc. The designer left pads for JTAG, optional 32kHz rock, '232 level translator, and some pads for GPIO and ADC inputs. I've reverse engineered most of it and have been hacking for three weeks now, and I'm having a blast with it.

Here are some links:
http://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/forum.cgi?read=139415

http://www.edn.com/blog/980000298/post/820030882.html

http://www.dev-monkey.com/blogs/jon_titus.php?mid=321

http://hpwiki.fatcity.com/doku.php?id=20b:repurposing_project

http://h10010.www1.hp.com/wwpc/us/en/sm/WF05a/215348-215348-64232-20036-215349-3732534.html

Datasheets

Sick of incomplete and erroneous datasheets? They've been a problem ever since I became an engineer. Bob Paddock was musing about the poor quality of datasheets: I always thought Analog Devices had good quality until today. I looked at several of their data sheets today and all of them were mostly "TBD". Useless. There are better things to spend our collective time on.

This is the worst of the lot on AD's web site today for the ADA4939-1:

"Supply Current (max) 37.699999999999996mA"

http://www.analog.com/en/amplifiers-and-comparators/differential-amplifiers/ada4939-1/products/product.html

I can just hear the production manager screaming "We are going to lose this *REALLY BIG* order if it is not under 38mA!".

Then we have this one from Atmel, who has needed a proof reader for their data sheets for years: "In automotive applications, distributed voltages are very disturbed." http://www.atmel.com/dyn/resources/prod_documents/doc7728.pdf

What are your favorite funny or odd datasheets?

More on Multicore

My comments about multicore generated a lot of dialog. Ray Van De Walker wrote: Many years ago, I worked on a multiprocessor database machine. The costs were constrained, so we had to use a shared bus. The 68010 CPUs were tried on cascaded clock cycles, with controlled bus access windows and caching. The best performance was two CPUs on alternating bus phases:

CPU 1: 100%, 2:198%, 3:205%, 4:207%...

Our computer scientist group wanted to investigate banyan switching networks, in which each CPU has a dedicated bus to a piece of RAM, and the buses can either cross or go straight. Simulations of these gave very good performance indeed. Modern serial busses might even make it affordable.

Steve Leibson commented: I just got your latest Muse and see that you've invoked Leibson's Law for multicore processors. Funny thing about that. A little more than a week ago, I returned from the MPSOC '08 conference held in Maastricht, Netherlands. I wrote 11 blogs on the presentations under Leibson's Law, including my own presentation on green computing. (See Steve's blog here: http://www.edn.com/index.asp?layout=blog&blog_id=980000298).

You and I are in complete agreement with respect to the use of SMP architectures for embedded systems. People are out there looking for the magic bullet that converts one, big, honking C program defining the entire product into a bunch of smaller binary images all happily playing on their own real or virtualized processor. Maybe someday. Not today. I've never seen such a big monolithic program written. It just doesn't work that way.

At the same time, I firmly contend that every one of your readers has the capacity to design embedded systems with dozens of processors in them. Ask any of your readers with systems-design experience to draw a block diagram of some complex system and they can. Fairly easily, I'd wager. They'll quickly draw dozens of blocks: an audio block, a video block, perhaps a DMA controller, some filter blocks, a UART, possibly a network or WLAN interface, USB port, etc. They'll also intuitively know how these blocks fit together and, lo, they probably won't drop all of these blocks on a shared bus. More likely, they'll represent the actual data flow in the system by hooking blocks together with links that mimic that data flow. How can I predict this? I'm no fortune teller. It's simply the way we've been designing board-level embedded systems since the microprocessor first appeared in 1971.

Now tell the system designer that many of the blocks in the system will be implemented with processors. Note that nothing's changed functionally. We've merely decided to use a firmware-controlled state machine (aka a microprocessor) to implement some of the blocks in the system. Magically, the system architecture will start to distort to reflect the designer's notions of how processors should be used in a system. For example, data links that were formerly separate will merge into buses. Why? Why would you choose to use multiple processors to distribute the data-processing load and then route the raw data and processed results on a limited, narrow, shared resource called a bus? Because processors "use" buses? That's a pretty lame excuse in my opinion. It's a matter of mindset.

There are other more suitable communications methods in many cases such as direct point-to-point parallel or serial ports, FIFO queue interfaces, and even networks on chip. Many system designers fail to take full advantage of the massive connectivity possible on nanometer chips because they're blissfully unaware of the resource. They design systems as though they're still using 20-mil traces on FR-4 printed-circuit boards. On chips, things work a whole lot different.

Further, when you treat each processor as its own self-enclosed entity, the multicore programming "problem" vaporizes. Oh yeah. The audio-processing code goes on this processor. The video code goes on this processor, or, if the video processing is sufficiently challenging, the video stream-processing code runs on video processor A and the pixel- or macroblock-processing code runs on processor B. No one writes the audio and video code as one big block of code all mushed together because that doesn't make any sense. In the multicore world, a bunch of cooperating, communicating programs running on independent processors makes a lot of sense. It exploits the centuries-old, tired and true engineering tool "divide and conquer," which has worked on all complex engineering projects from bridges and road systems to buildings to advanced jet aircraft and ICBMs, to Mars-exploration robots. Despite the last two examples, this just isn't rocket science.

Note that these various processors used in these clearly non-SMP systems need not (probably should not) all be the same processor. A processor that's optimized to encode and decode digital audio probably has different registers and execution units than a processor optimized to process video macroblocks. However, the design team's life will be a lot easier if all the processors come from a common architecture and share software-development tools. (Yes, that's a thinly veiled plug for Tensilica's approach. Sorry, it just makes sense.)

Gary Maxwell wrote: I've been working in the telecom industry in one sector or another since 1989, and multi-processor designs have been the norm since I've been involved. I consider multi-proc and multicore to be variations of the same theme, each with their advantages/disadvantages. With multi-processor designs, you're responsible for the overall design, but you have flexibility in specifying the processors and designing the data path and memory model. With multicore, your design work is less, but you are corralled into the vendor's idea of what processing resources you need, and your data path and pinout options are obviously limited.

Most telephony and network processor chips have been multicore for a long time, providing a general purpose CPU and one or more DSP cores. I've had the most experience with Motorola/Freescale's MPC855/860 line, with a PowerPC core and a big coprocessing element to handle HDLC, Ethermet (MII), UART, SPI, I2C, and other functions. In the end, using the HDLC against a full-span T1 (1.5 kbits/sec) was impossible, primarily because the data paths and controllers between the PowerPC, Coprocessor, and memory were too slow. Impressive MIPS numbers advertised by Mot/Freescale vanished into an ever-increasing number of wait states and bus contention issues. And because all of this went on under the chip, there was nothing we could do to tweak the design.

So, to emphasize your point, I would critically analyze any multicore/multi-processor

design, especially with regard to the data paths and the memory controller.

Free Book on Inspections

Tom Harris submitted a review that was a bit different than mine about the free book from SmartBear software. But first, he mentions that the open source tool Codestriker provides some of the functionality as Code Collaborator. On with his review:

We have read the book and "drank the Kool-Aid", and have implemented it here. Some developers love it, while others feel that it really slows down the code review / release process. Using CC is kind of like those folks who like to use Instant Messenger or email to replace real conversation with their co-worker who sits one or two cubes away.

I can see the value in it if developers are geographically dispersed, and I also see the value in collecting metrics and automating the "evidence trail" that a review has taken place.

But nothing can replace the value of two or more developers sitting down at the SAME computer at the SAME time having an in-depth face to face review with healthy discussion. We have found far more bugs with traditional code reviews than using Code Collaborator, which shows you the differences between a few affected files. You don't have the ability to see anything in other non-changed modules that have critical information useful during the review (like how functions in other modules are implemented, variable declarations, #defines, etc). You don't have the benefit of reviewing the code in a real "I.D.E." where you can bounce around between files and "hover" over variables to see their types. People start looking only at the "trees" instead of the "forest", and not how each tree fits into that forest.

Code Collaborator also has also delayed the time it takes to get reviews done, because of it's "email volleyball" nature. Ever got into a drawn-out email discussion where you finally realize it's so much more effective communication to simply walk over and actually TALK to the person or group face to face? In CC review, when you have a question about something, rather than simply discussing it with the author in real time and getting a quick answer, you have to submit comments / bugs via CC, which then launches this laborious on-line Q & A process where the review gets slowly batted between the author and reviewer's court over a period of hours or days. And when a bug is fixed, you have to resubmit the changed files, and start part of the process over, and every reviewer has to re-review it. This tends to make the author only want to use ONE reviewer for the sake of time. It's so much quicker to make the bug fix as soon as the bug is found in a normal review, with reviewers present and interacting in real time. In our office we need to get software releases out "Quickly with Quality". We've also seen that CC does not always "play well" with Microsoft's Visual Source Safe.

In summary I do think it's a great tool to review small "delta" changes to the code when developers are separated in time or space, or when code review "artifacts" are required, but nothing can replace the value of face to face review.

Regarding conducting good code reviews, there are all kinds of good "checklists" out there, just do a search on Google "Code Review Checklist". You just can't do many of the things in these checklists with CC.

I sent Tom's review to Jason Cohen of SmartBear for a response:

Thanks, this is great feedback. I actually agree with almost everything he says, and I wonder if perhaps his company is simply mis-applying Code Collaborator to their process, which is not the fault of the tool.

"Some developers love it, while others feel it slows down the ... process."

You can probably say the same thing about choice of text editors, version control, code style, unit testing, pair programming.....

"But nothing can replace the value of ... developers sitting down at the SAME computer at the SAME time having an in-depth face to face discussion."

I completely agree!

If you're using Code Collaborator to 100% replace other forms of discussion and code review, you have a problem. Like any tool, it's well-suited for certain things and should be avoided if it's a bad fit.
Most code reviews do not require in-depth back-and-forth discussion. In that case, if there's not a lot of discussion, Code Collaborator is still probably better than interrupting another developer -- breaking her out of the zone, a real cost! -- and better than e.g. collecting the diffs yourself and tracking brief conversations by email or the hassle of setting up meetings with four people.

But as soon as you get deep into something you should put down the keyboard and walk on over. No doubt about it!

I would like to submit for the reader's consideration that if the process at his company requires that you use Code Collaborator for 100% of the process of a code review, it is the process that is unreasonable and not the tool.

"It is so much quicker to make the bug fix as soon as the bug is found in a normal review, with reviewers interacting in real time."

Actually you can do that in Code Collaborator. It's a common misconception (which is our own fault) that the reviewers have to mark a review "finished" before the developer can upload fixes, but this is not the case. You can upload fixes any time, and in fact that's exactly how we operate at Smart Bear. As the reviewers are finding things, authors are typically seeing that, making the 60-second changes, and uploading the new files.

"CC does not always play well with Visual Source Safe."

Agreed. Now that Microsoft has moved on to TFS, between that and SourceGear's Vault we have almost no Source Safe customers anymore, and that means support suffers.

I'm not proud of this fact, but I have to agree with the reader that Source Safe support is not up to the same level as our other version control integrations.

Jobs!

Let me know if you're hiring firmware or embedded designers. No recruiters please, and I reserve the right to edit ads to fit the format and intents of this newsletter.

Netrino has multiple open positions for embedded software developers and FPGA and board-level electronics designers, in Maryland and California. Join the Embedded Systems Experts(tm) in a fun work environment that offers challenge and variety. Learn more at http://www.netrino.com/Embedded-Systems/Embedded-Software-Jobs

Goodrich is hiring! We have Embedded SW openings at our facility in Burnsville, MN (Minneapolis suburb). To apply, go to www.goodrich.com and the careers section. You can search for engineering jobs in Burnsville, MN to narrow down the choices. Here are the listings:

1) We need a Software Engineer who will be working on a focused, cross-functional team of engineers developing aircraft avionic products with embedded 'C' software for commercial and military applications. The Software Engineer will be responsible for real time embedded development of avionics software. The software will be developed, designed and tested according to the requirements of DO-178B Level A.

The Software Engineering tasks incumbent of this position include:
1. Software Design Engineering Tasks that comply with RTCA DO-178B Level A software

2. Oversight of Software Verification Engineering tasks that comply with RTCA DO-178B Level A including, but not limited to:
3. SW reviews and meetings with internal and external customers
4. Interfacing with FAA, JAA, Transport Canada or other Certification Authority

Qualified candidates will have a minimum of a BS with 3 to 10 years of experience.

2) We're also looking for a Senior Software Engineer to develop software for various technology and product development projects. The ideal candidate would have expertise in requirements development, design, coding, hardware integration and test. Competencies important for this position are: Drive for Results, Problem Solving, Interpersonal Savvy, Commitment to Continuous Improvement, and Learning on the Fly.

Essential Job Duties of the position:
Programming in C, C++, Matlab/Simulink
Capable of integrating software with hardware
Understand product function to help develop requirements.
Effective communication with project team members.

BS in Computer Engineering, Electrical Engineering, or related field.
MS preferred. At least five years experience. pplicant must qualify for access to U.S. export-controlled technology. Non-immigrant visas will not be sponsored for this position. Applicant must be a U.S. citizen or permanent resident, or designated refugee or asylee under U.S. law.

National Optronics, Charlottesville Virginia, builds machines and measurement devices used by opticians to process lenses for use in eyeglasses. We are seeking a senior embedded systems developer with meaningful experience in some or all of the following areas:

- Motion control (servo and stepper motors)
- RTOS (recently, NUCLEUS and MicroDigital SMX)
- GUI implementation (via PEG or VC++)
- Basic image processing and scene analysis
- Integrating Ethernet, USB, WiFi, etc.
- Interrupt handlers, device drivers, some asm experience (e.g. startup code)
- Some Windows experience helpful.

We use a range of platforms, from embedded x86 SBCs to boards we design, recently using Hitachi H8 and ColdFire MCF532x series. We program mainly in C with some C++, but are open to new ideas. Your work would be about 70% new development and

30% enhancement/repair of existing products. A BS or MS in EE or CS is desired. Charlottesville is a great town for music, arts, sports, hiking, camping, Two hours brings you to DC if you crave the big city. Send your resume in confidence to Hank Schutz hschutz@nationaloptronics.com.

Joke for the Week

Hardware and software people sometimes have trouble speaking the same language. Phil Matthews came up with a solution:

We had an experience recently where a software engineer couldn't solder a 10k resistor on to a circuit board. In fact he flatly refused to as he said "I'm a software engineer, not a hardware engineer!". It took a bit of explaining, but we got there in the end.

Take your soldering iron object and set the temperature property to about 300deg. If you don't have this build tool, then create an instance of the soldering iron class by borrowing one from a hardware engineer. Remember, the soldering iron object goes on the heap of test equipment. Derive an instance of a solder object about 100mm long from the solder reel class, preferably one with 3-core flux property. Create instances of resistor classes and wire classes. You may have to expose the copper interface of the wire by stripping the insulation. Now that you have all your components and build tools on your desktop hardware development environment, it is time to statically bind the components to an instance of a printed circuit board. Apply the hot end of the soldering iron (the end that doesn't have the cord coming out) to the component and then feed the solder in. The solder will inherit the heat property from the iron and melt. Once the solder has flowed into the joint remove the iron, stand back and admire the instance of assembled circuit board class. This new class inherits all the properties and methods of the components that were added to the base class of blank circuit board.

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.

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.