# The Embedded Muse 164

Editor: Jack Ganssle (jack@ganssle.com) September 1, 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 .

Nancy Van Schooenderwoert coaches teams about agile software development. She has written a paper, "The Four Pillars of Agile Adoption", about how to avoid some perils in the adoption and use of agile approaches. I found it fascinating: http://www.leanagilepartners.com/publications.html .

Boy, did I ever screw up. In the last issue I attributed a review of SmartBear's book on Code Inspections to Tom Harris. Actually, Tom's review is on-line at Amazon.com, and the review in the Muse was from a reader who wishes to remain anonymous. Sorry, Tom.

Are you using, or even considering, FOSS? If so do check out the Software Freedom Law Center's advice, embodied in their new paper "A Practical Guide to GPL Compliance." It's here: http://www.softwarefreedom.org/resources/2008/compliance-guide.html .

**********
This issue of The Embedded Muse is sponsored by Netrino.

The question "What is the difference between a mutex and a semaphore?" is short and easily phrased. Answering it properly is more challenging. Fortunately, the key differences between a mutex and a semaphore are among many RTOS usage issues addressed clearly and concisely in the free articles at http://www.netrino.com/Embedded-Systems/How-To-Muse
**********

# Free Books

My parents came of age during the Great Depression. Consequently my siblings and I were raised in a strict, never-waste-anything environment that I've tried to inflict on my own kids.

But despite trying to be minimally consumptive, we sure do acquire a lot of stuff. Books are my bane; Amazon's One-Click ordering makes it oh-so-easy to get the latest intriguing tome. Compounding that, a lot of publishers send me free computer books hoping to get a review. Some I save; some get passed along to friends, and too often I seem to acquire multiple copies.

There's a stack of technical books here that I no longer need. It seems sort of unethical to eBay those that were sent as freebies, and I can't remember which were free and which weren't. So, you can have them.

They're all free and I'll pay for shipping. If you want one, please send marybeth@ganssle.com an email. She'll probably get inundated, so here are a few rules to throttle the packets:
- First come, first serve
- Please include a shipping address (no PO Boxes)
- One book per person – let Marybeth know which one you'd like.
- I ask you to write a review of the book after you read it, so others can benefit. It'll run in the Muse, and with the other book reviews on ganssle.com, with appropriate credit to the reviewer.
- Please don't feel offended when we've run out!

Though the books are in good condition, some have annotations I made while reading them. Here's the list:

MicroC/OS-II, The Real-Time Kernel (second edition) by Jean Labrosse. This is one of the best introductions to using an RTOS, and, I'm told, is the best-selling book about embedded systems ever. Two copies available (you can pry the third copy out of my cold dead hands!)

Practical Software Estimation by M. A. Parthasarathy. I thought this work was very superficial and devoid of hard-hitting advice, but you may feel differently.

Embedded Linux Primer by Christopher Hallinan. A good and useful intro to porting Linux to your board.

Real Time UML Workshop for Embedded Systems by Bruce Powel Douglass. A worthwhile workbook to help developers get started on UML.

Logical Design of Digital Computers by Montgomery Phister, Jr. This 1958 volume is still surprisingly relevant, though the circuit diagrams use vacuum tubes!

The Propeller Development Kit from the good folks at Parallax. This is a book and board. Use it to get familiar with their very cool 8 core propeller chip.

CMMI for Outsourcing by Hofmann, Yedlin, Mishler and Kushner. Too many authors yields a dilute message that's more common sense than insightful.

Embedded Systems Design Using the Rabbit 3000 Microprocessor by Kamal Hyder and Bob Perrin. A really good reference for those using the Rabbit parts.

MISRA C – Guidelines for the Use of the C Language in Vehicle-Based Software. A great firmware standard, but this is the older, 1998, version.

The Devil's DP Dictionary by Stan Kelly-Bootle. An irreverent and amusing dictionary of computer terms. Though containing nothing of value it's a ton of fun.

# Response to the Joke of the Week

In the last issue I ran an idea from Phil Matthews about using OO paradigms to help software people solder. Rob Wehrli appreciated the humor but wanted to clarify the OO issues: This is an exemplification of WHY HW people often DO NOT understand OO. The new circuit board class does NOT inherit the properties and methods of the components that were added to the base class. This is a CLASSIC misuse of inheritance

that is very often perpetrated by HW folks...partly because languages like C++ (and nearly all others) can not differentiate between good and "bad" inheritance. The new class aggregates the functionality as instances of those representative classes that it uses to perform its work. This is often called composition or, obviously, aggregation. To say that the base circuit board class inherits its properties (the preferred term in OO-speak is "attributes") and methods (preferred is "operations") is simply wrong. Here's why. The circuit board class does not become a type of "solder" or "IC" or "discrete component" or anything else that was "added" to it. It is still a circuit board. It won't become a type of LCD controller if one decides to place a pad on a mechanical drawing and makes the necessary etchings to route traces to other devices such as a ribbon cable connector with the appropriate pin-out for a TFT LCD panel with integrated 4-wire resistive touch key and LED front light. The circuit board class is not a type of object populated on it, it is still a circuit board. By using inheritance to say that a circuit board is a kind of "solder joint" or that it is a kind of resistor (with a 10K value attribute) even sounds a bit suspicious, doesn't it?

Over the years I've tried to explain the notion of good inheritance in the face of situations where C++ will gladly let one declare:

```
class Rock;
class Bird : public Rock {};
```

...a Bird is a kind of Rock. I use these two because birds use small pieces of rock in their gizzards in place of teeth. This is a classic use of the external object by the "host" object. Here is another:

```
class Battery;
class Calculator : public Battery {};
```

A calculator is not a kind of a battery. The calculator probably needs the battery for its as-designed operational objectives but (the litmus test of good inheritance) the calculator can not be used anywhere its base can be used. That is, we can not instantiate a calculator class inside of a flashlight class and invoke its On operation for the purposes of lighting the darkened area. In fact, most real-world instances of calculator objects won't actually display in the dark, which isn't very useful to the average category of likely user. The goal of OO is to model the SW after the real world, at least within the constraints of the language and the skill of its programmers.

So, while the joke was certainly funny and aimed at putting HW skills in SW terms, it falls short and helps perpetuate the "mechanical myth" that if the compiler (or development environment) allows it to work, therefore it must be right. Mechanically, C++ will happily allow the following:

```
#include <iostream>
```

```
class Window
{
public:
    Window()
    {
        std::cout << "I'm a Window..." << std::endl;
    }
    void open()
    {
        std::cout << "Window is now open!" << std::endl;
    };
    void close()
    {
        std::cout << "Window is now closed!" << std::endl;
    };
};
class Building : public Window
{
public:
    Building() : Window()
    {
        std::cout << "I'm a Building..." << std::endl;
    }
};
class Unsuspecting
{
public:
    Unsuspecting()
    {
        std::cout << "I'm an Unsuspecting..." << std::endl;
    }
    void hit()
    {
        std::cout << "Ouch!  Who hit me?!" << std::endl;
    };
};
class WaterBalloon
{
public:
    WaterBalloon()
    {
        std::cout << "I'm a WaterBallon." << std::endl;
    }
    void toss()
    {
        std::cout << "...sailing as a result of being toss[ed]!" <<
std::endl;
    }
    void fall()
    {
        std::cout << "...falling!" << std::endl;
    }
```

```
};

class Villian : public Building, public WaterBalloon, public
Unsuspecting
{};

int main()
{
    Villian v;

    v.open();
    v.toss();
    v.fall();
    v.hit();
    v.close();

    return 0;
}

Output:

I'm a Window...
I'm a Building...
I'm a WaterBallon.
I'm an Unsuspecting...
Window is now open!
...sailing as a result of being toss[ed]!
...falling!
Ouch!  Who hit me?!
Window is now closed!
```

...so when is it the right time for a "Villian" to be a "Window" or a "Building" or a "WaterBalloon" or an "Unsuspecting?" However, if the objective was to instantiate the objects, have them perform some tasks, one can argue that the job was completed. Does that make it "right" or "good" code?

# Debugging Busses

James Morrison asked about debugging serial busses like I2C, SPI, and RS-232. Lots of people replied with suggestions:

Edward Gibbins wrote: Interestingly I am placing an order here: http://www.pctestinstruments.com .

David Bevin sent: I don't think this answers James Morrison's question but I remember using a HP 4952A Protocol Analyser for monitoring and simulating RS232 comms.

There's a picture of one here: http://www.atecorp.com/Equipment/HP/4952A.asp

Harry Jones contributed: You could take a look at these tools: http://www.mcc-us.com/I2C_Tools.htm .

Joy D. St. Amant suggested: A company named Willies Computer Software makes serial communications tools. Check out there products at: http://www.wcscnet.com/Default.htm . I have used their SI Scope; it is a good product. I believe they have an upgrade to the basic product which may be useful to Mr. Morrison.

Rijo Varkey uses Tracii http://www.telos.info/traciixl/ for I2C which is a bit expensive and also TotalPhase's tools. Recently we came across USB scopes but not used http://www.usbee.com/busbee.html. If someone has already used this tool let know the feedback.

Greg Harris wrote: For I2C bus monitoring/debugging/testing I have found the Corelis CAS-1000-I2C/E to be very useful. It is not cheap (but they are having a 40% discount right now), but it is very versatile.

Ray Keefe sent: One very inexpensive way to do I2C and SPI debug is the RockyLogic ANT8. http://www.rockylogic.com/ It is the logic state analyser which has direct decoding of both I2C and SPI. We have found it to be easy and straightforward to use. So you can use it to monitor the transactions provided they aren't really long.

You can purchase it from EasySync http://easysync-ltd.com/ although I noted it is marked as discontinued.

RockyLogic also does the Ant18e which can debug address/data busses and the like and provides faster sampling and a greater sample depth. See http://www.rockylogic.com/products/ant18e.html

Bill Brasch had a suggestion: I recently came across this 8-bit USB logic probe from Saleae that has I2C, SPI, RS-232 decoding. Low cost $149. I have not personality used it, but it looks promising. Check it out: http://www.saleae.com/logic/features/?q=i8&gclid=CPHgkeyDjJUCFQObFQod5SV .

Jay D. Hall uses: http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en028600 with great success. Hope it helps him!

Dave Clark wrote: The USBEE-AX and BX (http://usbee.com/usbeeax-standardtestpod.aspx) is a brilliant tool for analyzing USB and other serial protocols. We

have the AX-Plus, but the next one up the range seems to have an I2C controller as well. I could not have even attempted a recent USB project without it. Imagine sampling a 8 signals at 24MHz with 1,500,000,000 sample depth and then decoding the result by hand? Colleagues have also used it for I2C as well as other protocols.

For serial debugging, I use Telix, which has an excellent script language which can be used to decode protocols (I have written scripts for Multi-Drop Bus vending protocol and BACTA gaming protocols). The scripts can also be used to simulate a slave or master device. It appears to be a free download (http://www.telix.com/delta/deltacom/tfw/index.html) but I cannot be sure of this. We purchased it about 10 years ago and have upgraded to 1.15D.

Christer Berg sent: You may want to look at Frontline's SerialTest and other products from same company. Excellent decoding of most industrial protocols.

Martin Zacho wrote: I would use a FPGA dev kit (from your preferred manufacturer) and implement the relevant parts in it. The cost would be below $100,- and some spare time ;-)

Bill Knight suggested: Have James Morrison check out RealTerm http://realterm.sourceforge.net for debugging serial port busses.

Randy Glenn sent: In your most recent Embedded Muse, you passed on a question from James Morrison regarding an inexpensive I2C / SPI / UART debugging tool. The Microchip PICKIT2 programmer can handle all of the above (to some extent, at least) and comes in at $35. It can even act as a 3-channel 1MHz logic analyzer - not great, but still not bad for $35. Oh, and it programs and does ICD for microcontrollers, too. More info at http://www.microchip.com/pickit2.

I should probably also include a link to the PICKIT 2 Serial Analyzer page, http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en028600, where the firmware and software to do the I2C and SPI stuff lives.

Ben Sweet contributed: Regarding James Morrison's question about debugging network interfaces. He might consider the Netway tool from Smart Engineering Tools, Inc.: http://www.smtools.com/

Paul Carpenter wrote: I have never really come across a completely multifunction device for doing insertion of commands and for most systems (I2C and SPI like) the hardware alone dictates, that adding another master device will require changes to hardware or

jumpering out the bus to go via your pseudo master. Back to the problems of two drivers at same time on an SPI bus is a hardware issue, and blowing devices. In many cases you would have to have a device to pass all traffic, add extra, or takeover the bus.

However for monitoring many digital scopes these days have the ability to do the monitoring. The problem of course is in the matching with SPI the clock and data phases, along with maximum data rate.

I am reminded of something I did in a PLD the other year for testing ASICs, device being tested was its own I2C master, to continuously communicate with an I2C EEPROM containing calibration data. However I quickly discovered the amount of reads and writes required to test about 500 ASICs would exceed the EEPROM write cycles spec. Also the data sets that had to be stored in the EEPROM for device testing was in fact very simple (bit manipulation algorithms from the address and mode), some places were just binary inversions of symmetrical places, and some parts had to be read/write registers. By creating an SPI register in a PLD, I was able to compress all the function control to be configured for another block of the PLD to then operate as a simulator of the I2C EEPROM. This included encoded bits in the control register to ensure the next access would deliberately contain bit errors to force error conditions as part of the test process.

One advantage of this was that alongside digital scope measurement of the signals to check frequency, levels etc.. we could determine if the ASIC was in a fault condition and driving the I2C bus at about 2 MHz, which the emulator happily coped with

Considering the various speeds, clock relationships, let alone data protocols need to operate, especially serial memory devices. It may be easier to derive a mixture of scope and PLD/FPGA that suits your application each time.

# Datasheets

Sick of incomplete and erroneous datasheets? They've been a problem ever since I became an engineer. Readers have been sending in funny/odd/annoying excerpts from datasheets for our agony or amusement.

Tom Mosher and others suggested the real classic, the Signetics Write-only-Memory. That's on-line here: http://www.national.com/rap/files/datasheet.pdf

Paul Carpenter wrote: Aaarrrgh..... those who may know of the older H8/3048, the problems with driving the flash write enable pin are legendary, until they bought out the 'B' version 5V only pins.

Basic problem is the spread of information of modes, electrical characteristics, special

functions through 800+ page manuals (PDF). So parts of the microcontroller are described in 20 different places. Often repeating some of the information.

Most parts of the document, even the absolute maximum ratings, tell you that the pin can withstand 13V (must not peak above this).

Hidden away on page 600 and something is the ONLY note that is only a few lines long - "..(Vpp) must be applied after the rise Vcc when the microcontroller is in a stable state. Shut off Vpp before Vcc..."

In other words or absolute max rating of Vpp pins is "8V above Vcc" NOT 13V.

This is not described in the programming examples or how to setup the device electrically for the modes.

Luis G. Uribe sent this: The following is not about a datasheet; it is the Motorola (now Freescale) CPU08RM.pdf Reference Manual, http://www.freescale.com/files/microcontrollers/doc/ref_manual/CPU08RM.pdf . The last section is written to include programming examples to show the new instructions -- those added over the ancient HC05, and their applications. There you can read (page 183):

Code Examples:

```
TAP Transfer Accumulator to Condition Code Register TAP
*
* NOTE: The TAP instruction was added to improve testability of
* the CPU08, and so few practical applications of the
* instruction exist.
*
```

In my opinion, the author of this infamous paragraph has no idea of what he is talking about. This instruction was NOT added to improve testability; it is one of the most important Op-Codes in the entire HC08 instruction set, and one that deserves a full example on using it!

For example, to write some generic routine, one that could be used inside an ISR, if you need to disable interrupts, later you can not simply enable them: You must restore interrupts to the state they where at the beginning, something like this:

```
   TPA  ; Transfer CCR to Acc (TPA is TAPs twin opcode)
   PSHA ; ..Save Acc (CCR) into stack
;...
;...Later you may recover the interrupt state, as follows:
   PULA ; Pop (er, pull) Acc
   TAP  ; ..Transfer Acc to CCR
```

```
        ; ..(saved flags, including I Flag)
;...
```

By the way, TAP stands for: Transfer Accumulator to Processor Status Word. You know, they began calling the flags, PSW and, then, change the name to Condition Code Register (CCR) but, the instruction codes, remain using the old nomenclature (TAP, TPA).

## 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

## Joke for the Week

Definition of a security professional: A hacker with a mortgage.

## 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.