The Embedded Muse 105

Editor: Jack Ganssle (jack@ganssle.com) Nov. 4, 2004

EDITOR: Jack Ganssle, jack@ganssle.com

CONTENTS:

Editor's Notes

When? December 10th. Where? Las Vegas. That's where I'll hold the next Better Firmware Faster seminar. This is the only non-vendor class that shows practical, hard-hitting ways to get your products out much faster with fewer bugs. 80% of systems get delivered late, often hopelessly bug-ridden. It *is* possible to do better – much better. See http://www.ganssle.com/classes.htm for more details including cheap fly-in options, and information about free hotel rooms.

I often do this seminar on-site, for companies with a dozen or more embedded folks who'd like to learn more efficient ways to build firmware. See http://www.ganssle.com/onsite.htm.

I'll be at Electronica/The Embedded Systems Conference in Munich from November 9 to the 11th, presenting three talks. If you're there, stop by and say "hi!"

More on Watchdogs

Several thousands people a month download my paper about building and using watchdog timers (http://www.ganssle.com/watchdogs.pdf). This is obviously a subject of much interest to developers.

EM Microelectronic has a new series of watchdog timers and power supervisory chips that look pretty decent. The devices are "windowing" watchdogs, which force the developer to issue tickles no faster than some programmed rate, as well as no less often than another frequency. This, of course, limits the chances that a crashed program will erroneously issue successful tickles.

See http://www.emmicroelectronic.com/DetailNews.asp?IdNews=78 for more information.

Free x86 Compilers

Wilton Helm sent along this tidbit. The Watcom x86 compiler has been released under a GPL type license with open source. They have had a reputation as being one of the more efficient compilers, the user interface is fairly good, and a debugger, including remote capability, is included. The major advantage over Borland is that the source code is available. See http://www.openwatcom.com.

Borland has long provided Turbo C for free. This compiler is still used by a lot of 186 developers. Find more at http://bdn.borland.com/article/0,1410,20841,00.html

Core Memory

"Core dump? What does *that* mean?" a teenaged Unix-wanna-be inquired. I looked up, and thought (not for the first time) that young folks sure keep the mad dash of technological change in perspective. Their questions often require more thought and introspection than those of their elders

I pulled out one of my holy relics - a 3 pound, 13,000 bit core array acquired from a surplus shop in 1971. A few days after high school graduation I hitchhiked with a pal to Boston (those were kinder, gentler days) to find treasures in the disorganized depths of a surplus shop.

Was it our long hair? Maybe the fact that we were warned three times to get off the New Jersey Turnpike had something to do with it. Somehow Gary and I found ourselves in a New Jersey jail cell, busted for hitchhiking. The police, expecting to find a stash of drugs in our backpacks, were surprised to discover instead my 13,000 bits of core.

"What's this?" the chief growled. I timidly tried to convince him it was computer memory. These were the days when computers cost millions and were tended by an army of white robed technicians, not hitchhiking hippie-freaks. The cops looked dubious, but could find nothing to dispute my story. They eventually let us go, me still clutching the memory.

A few months later I started college. The school had an Univac 1108, a $10 million mainframe that serviced all of the university's students. We upgraded the machine, adding 256k words (36 bits each) of additional core memory, at a cost of $500,000. The core featured a 750 nsec cycle time and was the size of several refrigerators. For months, until Univac's engineers got the bugs out, leaning on the box was sure to cause bit dropouts. This almost always crashed the system and drove users into an uproar, especially during finals week when projects were due. The machine serviced some 500 users simultaneously, yet had only 768k words of memory, much less than in one of today's laptops. A six foot long drum memory stored something like 50 megabytes. The 1108's claim to fame was instruction execution in under a microsecond, a claim that pales next to the Pentium's sub-nanosecond time… for a price that approaches zero dollars.

No one dreamed of personal computers back then. Sure, a few stories circulated about the surplus deal of the century, where a clever person picked up an old 7094 for a pittance. Few could afford the electricity, let alone the astronomical air conditioning costs required for running such a monster.

IBM's 7094 was a wildly successful mainframe. My school had one of those as well. Core was so expensive that the entire system had but a pathetic 64k - with no disk storage. Programs wrote temporary information to tape, keeping a room full of drives whirring constantly. There was no security to speak of. One of our great delights was running a program that searched for teachers' programs on the input tape, always with the hope of finding grades that could be, ah, "debugged". Eventually the university wised up and bought an additional Univac with no student accounts or dial-in lines, which ran all of the grading and billing software.

Coming back to the present, I showed off the core matrix. You know how teenagers are - she didn't believe a word I said. "Come on, that thing only stores 13k bits? It's bigger than my whole computer," she noted, disdainfully, as if I were a stone-age man showing off my cave painting technology to a Photoshop whiz.

I realized that 10 of the cores covered an area about the same as a 256 Mbit DRAM die. How things have changed!

Each core is a ferrite bead, perhaps the size of a small letter "o". Four wires run through the center of each core, four wires tediously strung, by hand, by Asian women who no doubt worked for a pittance.

Cores are tiny magnets, each remembering just one bit of information. The trick is to flip the magnetic field of the cores - one direction is a "one"; the opposite field indicates a "zero".

As we know from basic electromagnetics, a changing voltage creates a magnetic field, just as a change in a magnetic field induces a voltage. The wires running inside of the ferrite beads create the fields that flip the direction of magnetization to store a zero or a one. The wires also sense the magnetic field so the computer can read the stored data.

Two of the wires organize the core into an X-Y matrix. The core plane is an array of vertical and horizontal wires with a bead at each intersecting node. Run 50% of the power needed to flip a bit down each wire - at the intersection there's all that's needed to flip just that one bit. What a simple addressing scheme! As the bit changes state, it induces a positive or negative pulse in a third wire that runs through all of the cores in a plane. Sensitive amplifiers convert the positive or negative signal to a corresponding zero or one.

Since the amplifiers detect nothing unless the core changes state, reads are destructive. You've got to toggle the bit, and then write the data back in, on each and every read cycle. It sounds terribly primitive till one thinks about the awful things we do to keep modern DRAM alive.

Before microprocessors quite caught on, the instrumentation company where I worked embedded Data General Nova minicomputers into products. The Nova used core arranged in a 32k x 16 array.

Since core is non-volatile, remembering even when there's no power applied, we regularly left the Nova's boot loader in a small section of core. My fingers are still callused from flipping those toggle switches tens of thousands of times, jamming the binary boot loader into core each time a program crashed so badly it overwrote these instructions.

As we worked through these reliability issues, my boss - who was the best digital designer I've ever met - told how some military and space projects actually employed core as logic devices. In a former job he designed systems composed of strands of core strung together in odd patterns to create computational elements. I remember being just as incredulous at his stories as the teenager is at mine.

Unfortunately, Unix continues to litter our drives with sendmail.core, httpd.core, and other binary images, each a snapshot of the death throes of a program. Perhaps twenty years on, my young friend will regale a new generation with her stories of the bad old days of computers, before brain implants supplanted keyboards, when people were not yet integrated into the global data processing network.

(There's a picture of my core at www.ganssle.com/misc/core.htm .

http://www.science.uva.nl/faculteit/museum/CoreMemory.html has good core memory information, as does http://en.wikipedia.org/wiki/Core_memory .

Computing CRCs

The CRC (Cyclic Redundancy Check) is a sort of sophisticated checksum commonly used for testing the reliability of data communications. Every sector on a disk has a CRC of the data stored, so the operating system will be aware of data dropouts. A CRC doesn't identify which byte is in error, but it pretty much guarantees that you'll be alerted to at least single bit errors. Serial data, transmitted over SDLC links and the like, use the CRC as well.

All CRCs are binary polynomials that are divided into the data stream. Unfortunately, CRC discussions are usually filled with lots of math proving the inherent robustness of the technique; the math obscures the simplicity of writing CRC-based code, and so discourages designers from using this method, unless the CRC is automatically calculated by a fancy peripheral chip.

The most common CRC polynomial is the CCITT form used by IBM's SDLC protocol. It is of the form:

X**16 + X**12 + X**5 + 1

This means that the input data stream (say, the data being written to the disk drive) is exclusive ORed into a 16 bit shift register that has feedback terms at the bit locations with coefficients in the formula, one bit at a time. Each register bit is a function of the CRC to that point, the input data, and the feedback taps.

In many systems data is transferred as a serial bit stream. SDLC is inherently serial. Disks all write a single bit at a time to the medium. Modem applications are also bit oriented. Unfortunately, serial bit streams are a mess to work with in software.

Tanenbaum, in "Computer Networks", claims the accuracy of the SDLC polynomial will guarantee detection of all single and double bit errors, all errors with an odd number of error bits, all burst errors of length 16 or less, 99.9969% of all 17 bit error bursts, and 99.984% of all possible longer error bursts. That's pretty impressive!

Unless you can test an algorithm, it's pretty much worthless. The following table has test values, for a quick evaluation of your code. It assumes the entire string of data comes in; that is, after the first 0, the output is 0f87, after the second the output is 0f0b8, etc.

Input data	Output CRC
preset	FFFF

0 0F87
0 F0B8
0 3933
0 0321
0 3088
77 0F48
CF F0B8

(Note that feeding one's complement of CRC into the equation yields F0B8)

crc_value is a global value. Be sure to initialize it to ffff before calling this function for the first CRC in a block.

```
unsigned int crc(char ch)
{
  for(i=0; i<8; ++i)
    {
     if((crc_value&1) XOR (ch&1))
        {crc_value=(crc_value>1)XOR 0x8408;}
         else crc_value=crc_value>1;
     ch=ch>1;
    }
}
```

There's much more info about CRCs at http://www.ross.net/crc/ and http://www.repairfaq.org/filipg/LINK/F_crc_v3.html

Jobs!

Let me know if you're hiring firmware or embedded designers. I'll continue to run notices for embedded developers as long as the job situation stays in the dumper. No recruiters please.

Analog Devices currently has several active job opportunities in our Digital Media Technology Center for the following system engineering positions. The general descriptions are shown, though we have multiple, similar openings for each position at various experience and responsibility levels. The job location is nominally Norwood, MA, near Boston, but long term flexibility is possible.

DSP System Engineer: Responsibilities: Design and debug system software on digital media chipsets. Design/code/test hardware drivers, memory interfaces and real-time operating systems. Deliver highly optimized, robust environments for the integrations of software libraries. Deliver a fully productized (and testable) digital media device.

Design/write/debug the environment for a software based a/v codec chipset and products based on this chipset. Debug system issues which arise during product integration.

Linux System Engineer: Responsibilities: Implement and debug Linux Kernel, Device drivers and system software on digital media chipsets. Code/test hardware drivers, memory interfaces and interactions with real-time operating systems. Deliver highly optimized, robust environments for the integration of software libraries. Deliver a fully productized (and testable) Linux based media device. Write/debug the environment for a software based a/v codec chipset and products based on this chipset. Debug system issues which arise during product integration.

Gentex Corporation in Zeeland, MI has an opening for an Embedded Software Engineer. They develop and manufacture advanced electro-optical products for the automotive industry in C using 8 and 16 bit microprocessors. They are looking for someone with experience working in a small groups to develop embedded products. View www.gentex.com for more information or send a resume to debdv@gentex.com.

Digital Audio Corporation (http://www.dacaudio.com) is hiring and senior level Embedded Hardware and Software Development Engineer. We're a small but stable company made up mostly of engineers. We are looking for an engineer experienced in embedded hardware and software (developing DSP and microcontroller hardware and firmware). We're looking for someone who would enjoy the challenges and rewards of working on all aspects of a design project: from concept to final product. For more information, see the detailed job description at (http://www.dacaudio.com/employment.html). Email resumes to employment@dacaudio.com.

Plantronics, the world leader in communications headsets, is seeking Embedded Software Engineers for our Core Technology Development group who have Firmware development and Wireless, Bluetooth, DECT or 802.11 experience. To apply go to Plantronics website at http://www.plantronics.com/careers. Click on "Embedded Software Engineers" to complete a short questionnaire that takes 2-5 minutes. Plantronics is an AA/EEO Employer.

Joke for the Week

For many years molecular biologists have been mystified by the fact that very little of an organism's DNA seems to serve any useful function.

The reason why only 30% of human DNA performs any useful function is that the rest of it is comments.

Once we decode a typical human genome, we see that the contents begin as follows:

```
/* HUMAN_DNA.H
*
* Human Genome
* Version 2.1
*
* (C) God
*/

/* Revision history:
*
* 0000-00-01 00:00  1.0  Adam.
* 0000-00-02 10:00  1.1  Eve.
* 0000-00-03 02:11  1.2  Added male version. A bit messy --
*                        will require a rewrite later on
* 0017-03-12 03:14  1.3  Added extra sex drive to male.h;
*                        took code from elephant-dna.c
* 0145-10-03 16:33  1.4  Removed tail.
* 1115-00-31 17:20  1.5  Shortened forearms, expanded brain case.
* 2091-08-20 13:56  1.6  Opposable thumbs added to hand() routine.
* 2501-04-09 14:04  1.7  Minor cosmetic improvements -- skin color
*                        darker to match my own image.
* 2909-07-12 02:21  1.8  Dentition inadequate; added
*                        extra 'wisdom' teeth.
*                        Must remember to make mouth bigger
*                        to compensate.
* 4501-12-31 14:18  1.9  Increase average height.
* 6004-11-04 16:11  2.0  Made forefinger narrower to fit hole
*                        in center of CD.
*/

/* Standard definitions */

#define SEX male
#define HEIGHT 1.84
#define MASS 68

/* Include inherited traits from parent DNA files.
*
* Files must be pre-processed with MENDEL program to provide proper
* inheritance features.
*/

#include "mother.h"
#include "father.h"

#ifndef FATHER
#warn("Father unknown -- guessing\n")
#include "love_child.h"
#endif

/* Initialization bootstrap routine -- called before DNA duplication.
* Allocates buffers and sets up protein file pointers
```

```
  */
  DNA *zygote_initialize(Sperm *, Ovum *);

  /* MAIN INITIALIZATION CODE
  *
  * Returns structures containing pre-processed phenotypes for
  * the organism to display at birth.
  *
  * Will be improved later to make output less ugly.
  */
  Characteristic *lookup_phenotype(Identifier *i);


===
```

...and so on...

[Note that God programs in C, uses three-space tabs and /* */ style comments]

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.