The Embedded Muse 144

Editor: Jack Ganssle (jack@ganssle.com) April 9th, 2007

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 all this and much more at my Better Firmware Faster class, presented at your facility. See http://www.ganssle.com/brochure-onsite.pdf.

Or, come to **Boston May 4**, where I'll present this class at the Sheraton Braintree hotel. Registration and other info here: http://www.ganssle.com/classes.htm . You'll earn 0.7 Continuing Education Units, learn a lot, and hopefully have a bit of fun, too.

I will be presenting this seminar in Toronto on April 20 for the local IEEE section. Both IEEE members and non-members are welcome. The details are posted here: http://toronto.ieee.ca/education/firmware0407 .

The topic of clear writing continues to elicit email. Steve Karg wrote: "Here is a website called Plain English Campaign. It claims it has been "Fighting for crystal-clear communication since 1979":

http://www.plainenglish.co.uk/ . I highly recommend the General Guide, "How to write in plain English": http://www.plainenglish.co.uk/guides.htm . I printed the Summary and taped it to the top of my computer monitor."

Tools and Tips

Michael Bulgrien likes Beyond Compare: "http://www.scootersoftware.com/ubbthreads/postlist.php?Cat=0&Board=BC3News - Beyond Compare 3 Professional will be released later this year. Although BC2 has always been a favorite of mine, BC3 will add great new features including a full-screen edit mode, syntax coloring, 3-way merges, dynamic recompare, superb source control integration, and much more."

Steve Karg wrote about Kicad: "There is a project called Kicad which is "open source (GPL) software for the creation of electronic schematic diagrams and printed circuit board artwork." It is cross platform (i.e. Windows/Linux) and free. The example layouts they include are pretty slick, especially the 3D view. Of course, you have to be able to create the 3D decals/parts to be able to do that, but the concept is sound and the program works. The home page is at: http://www.lis.inpg.fr/realise_au_lis/kicad/ . There is a fairly active users group at: http://tech.groups.yahoo.com/group/kicad-users/ "

Book Review

Andrew Kirby has submitted the first book review from the recent Great Book Giveaway. He looks at the first edition of Mike Barr's book; note that there's an update which is really a complete rewrite out now.

Review of Programming Embedded Systems in C and C++, Michael Barr, O'Reilly

"How do embedded systems differ from say PC programming?" is a question I normally ask in interviews for embedded positions. The respondent could do much worse than quoting some of the major points from Barr's book. Weighing in at around 140-ish pages before appendices it is an easy-read introduction to the nuts and bolts of simple embedded systems.

The book is probably best read with access to an embedded system to "play along" with (Barr uses the 80188EB as example hardware) but as long as one is familiar with C and willing to read up on x86 assembly to fully understand what is going on it is not critical.

If I have one small gripe it is the propagation of some of the most notoriously bad embedded practices without additional commentary. The most glaring is the use of a while loop to act as a 1-second delay. While it is perfectly understandable to use this to get some lights blinking at 1Hz it would have been good to see some follow-up discussion about why it is preferable to use some real form of timing mechanism (hardware or software).

The book is definitely targeted at the embedded novice. An experienced embedded engineer will find themselves flicking through it very quickly. However, thanks to Jack's generous give-away I now have a copy to pass out to new hires. I don't expect to get asked about the difference between flash and EEPROM again!

Baudot & Corrections

Several readers sent corrections and information about Baudot and teletypewriters. Robert Schwalb wrote: "In the Embedded Muse 143 you made mention of "noisy teletypes". As a former employee of the former Teletype Corporation, I just want to correct what is a common but erroneous use of that name.

"The name "Teletype" is not a generic term. It's a trademark and service mark of the Teletype Corporation for its line of teletypewriters. For example, that teletypewriter you might have used in college was probably a Teletype Model 33. The noisy news room machines were probably Teletype Model 28s.

"The Teletype Corporation had a long and venerable history, and made many important contributions to the field of data communications.

"There's more about Teletype at these URL:
http://www.kekatos.com/teletype/
http://en.wikipedia.org/wiki/Teletype "

If you're at all interested in the history of this field, check out the first link, which has a lot of the early history of teletypewriters.

Eric Smith sent: "One minor nit-pick with your latest newsletter. The 5-level code used by Teletype machines and other teleprinters was *not* Baudot. The Baudot code was actually never used for teleprinters. The actual code used in the US was an apparently unnamed US variant of the ITA2 code (International Telegraph Alphabet #2).

“Baudot code was modified to become Murray Code, and further modified by the CCITT to become ITA2. The US 5-level code is ITA2 with changes to the non-numeric characters in the FIGURES set:

code	ITA2	US
00101	'	BELL
01001	WRU	$
01011	BELL	'
01101	undefined	!
10001	+	"
10100	undefined	# (or motor off)
11010	undefined	&
11110	=	;

“For complete details, see the Annotated History of Character Codes by Tom Jennings: http://wps.com/projects/codes/ .”

Steve Sadler told me that Baudot coding schemes are still used in 911 operations. “It's mandated by NENA (911 National Emergency Number Association www.nena.org). Here's the bit from the NENA Recommend Generic Standards for E9-1-1 PSAP Equipment:

“5.18 TDD/TTY Compatibility

“The regulation implementing Title II of the American with Disabilities Act (ADA) mandates telephone emergency services to provide direct access for people who use Telecommunications Device for the Deaf/Teletypewriter (TDD/TTY) technologies. Therefore, every answering position shall be equipped with TDD/TTY's.

“The TDD/TTY caller shall have direct access to PSAP emergency lines in the same manner as a voice call. "Direct access means that emergency telephone services can directly receive calls from rs without relying on an outside relay service or third party services." When operating in a TDD/TTY mode, each position shall retain all system features available from the existing 9-1-1 system.

“Note: "At present, telephone emergency services must only be compatible with the Baudot format. Until it can be proven that communications in another format can operate in a reliable and compatible manner in a given telephone emergency environment, public entity would not be required to provide direct access to computer modems using formats other than Baudot." ADA Title II Technical Assistance Manual.”

Dilip Warrier and others wrote about Baudot's use to support the disabled: "You will be pleased to know that Baudot codes are alive and well and still used in TTY (phone cabilities for the hearing and speaking disabled). In fact, we work on cellular phone technology that has to support the TTY standard for the US and constantly work with Baudot codes."

Bit Banging

All of this discussion of Baudot and teletypewriters reminds me of the bad old days of microprocessors when hardware was expensive. Did you know that at one time some developers saved the cost of a UART by processing serial streams completely in software? I wrote about an approach to this – called "bit banging" – over 15 years ago. Here's an extract of that article.

Sometimes it's nice (or vital) to add features to a product for engineering or manufacturing reasons only. In a perfect world we'd all add extensive self-testing capability to every embedded system. A software monitor, invoked by some secret switch combination, can be invaluable for sensor field testing. It might be nice to add some sort of logging feature to a simple system to log long term analog circuit drift. Lots of applications can benefit from the operator interaction that is really only possible when a terminal link exists.

All of these dreams need an unused serial port. With the proliferation of high integration CPUs, serial ports aren't quite as scarce as a few years ago. Still, it is surprising how an application can eat up every available resource, including all of the ports.

Perhaps, we lean a little too heavily on complicated UARTs and other peripheral chips everyone takes for granted. For low speed communications a UART really is not a necessary part of the hardware. An old technique with the nearly scatological name of "bit banging" lets you easily use a pair of parallel I/O pins as a serial port.

But first, a bit of background...

RS-232, as has been extended for microcomputer communications, defines signal levels, transfer parameters, and cabling for serial communications over short (under about 50 feet) distances. Of course, different vendors implement various aspects of the standard in different ways, so devices hardly ever work together without some frantic wire swapping.

RS-232 communications takes place one bit at a time. Each of the 8 bits of a byte is sequentially sent out over a single pair of wires.

All communications takes place at a baud rate agreed on by both the driver and receiver. A 9600 rate means that each bit of the character stream takes 1/9600 second to transmit.

When the link is idle (no data being sent) it is in the Marking state (the line is more negative than -3 volts). The Start bit, which puts the line into the Spacing state (more positive than +3 volts) for one bit period, is sent first and serves to announce that a character is on the way. The receiver senses the start bit and sets itself up to read the incoming serialized byte.

Data bits follow Start. The least significant (data bit 0) goes first. One at a time, the other bits follow, each being given exactly one bit time on the link.

After the entire character has been transferred the line goes to the marking state for the length of the stop bit - one or two bit times depending on the protocol agreed to by the communicating devices. Stop bits look like an idle line. They give the UART time to recover before the next character starts.

A logic zero data bit is transmitted as a spacing line condition (positive voltage); ones go as marking bits (negative).

So, to send the character "A" (hex 41), the line assumes the following levels:

Marking	(<-3 volts)	line idle
Spacing	(> +3 volts)	Start bit
Marking	(<-3 volts)	Data bit 0
Spacing	(>+3 volts)	Data bit 1
Spacing	(>+3 volts)	Data bit 2
Spacing	(>+3 volts)	Data bit 3
Spacing	(>+3 volts)	Data bit 4
Spacing	(>+3 volts)	Data bit 5
Marking	(<-3 volts)	Data bit 6
Spacing	(>+3 volts)	Data bit 7
Marking	(<-3 volts)	Stop bits
Marking	(<-3 volts)	line idle

The RS-232 standard defines pinouts for Data Communications equipment (DCE) and Data Terminal Equipment (DTE). Terminals are DTE. Computers seem to be DTE or DCE depending on the whim of the designer. The IBM PC is DTE.

Pin Number		Direction	Pin Name
DB-9	DB-25		
5	7	Ground	
	1	Frame ground (not used on 9 pin connectors)	
3	2	DTE to DCE	Transmitted data from DTE
2	3	DCE to DTE	Received data from DCE

8	5	DCE to DTE	Clear to send (DCE ready)
7	4	DTE to DCE	Request to send (DTE ready)
6	6	DCE to DTE	Data set ready
4	20	DTE to DCE	Data terminal ready

It takes a bit of hardware to convert the 8 bits of parallel character data to a serial stream, insert start/stop bits, and shift the data out. Receiving is harder, since reliable communication mandates that the bit is sampled in the middle of a bit cell. In addition, some sort of interface circuit must shift the computer's +5 and ground levels to RS-232's bizarre plus and minus voltages.

Most systems use a UART to convert parallel bytes of data from the program into a serial stream of bits and vice versa. Any UART handles a lot of other RS-232 interface chores like double buffering the data, automatic handshaking, etc. A UART is almost a tiny external co-processor that sends a character, or that accepts input data and reassembles it, signaling the CPU only when the task is done. The UART is by far the most complex part of a serial port.

You can replace the UART entirely with software if the demands placed on the port aren't too severe. A software UART replacement tediously serializes and deserializes data bytes, and so demands the full attention of the CPU. A “bit banger" software UART won't automatically assemble characters for you while the code is off doing something else; if the code isn't listening when a character comes, data will be lost.

This is not a problem for maintenance ports or back doors into a system. Usually these are invoked rarely, and drive the system into a funny mode which just replies to commands from a terminal.

A surprising number of systems that use RS-232 as a primary communications interface also use bit bangers. Where interrupts are not a problem, and where multitasking doesn't exist, a simple bit banger can be a fairly efficient main comm link.

Why is interrupt service a problem? As will be seen shortly, all of the character's timing is derived from the execution of software loops. During transmission, if the code goes off to service an interrupt the length of a bit time will be shifted. DMA can cause a similar problem, especially if the DMA timing varies.

The following code shows the three subroutines needed to transmit, receive, and initialize the baud rate. This code is written in Z80 mnemonics, but will run equally well on the Z80, 64180, 8085 and NSC800 processors. It is easily adaptable to other processors, but be sure to balance the timing between subroutines.

The routines transmit and receive data through two parallel lines you must supply. It's not too hard to come up with one input and one input bit on most systems; reserve them early in the design for a serial port "back door".

BRID, the baud rate detection routine, loops for the user to type a space character. After BRID detects a start bit it waits for Start to go away (i.e., for the line to return to a logic 0) and then counts loop iterations during the six zero periods before the logic one (space is hex 20) occurs. This count is then transformed into a bit time, the basis of all timing in the transmit and receive routines.

The space character is particularly good to establish bit time, as it does give a very long (6 bit periods) period between the start bit and a one. An '@' might be marginally better, since its hex 40 code has 7 bits before the 1. Using '@' would require some adjustment of the timing calculations in the code.

Routine COUT sends a character by toggling the serial line high (a start bit), delaying for one bit time, and then sending data bits one at a time. It loops for a bit period between each data bit to insure that the character's timing is correct. The routine transmits two stop bits at the end of the character.

Receiving is trickier. When routine CIN detects a start bit it delays for half a bit time to the start's center. This improves timing margins - we always sample the data stream in the center of each bit cell. Why? The line could be a little noisy, or capacitive effects may smear the exact starting edge of the bit.

The code then delays one bit time and reads data bit 0. It repeats the delay and inputs the rest of the character, shifting it into a register as it goes to reverse COUT's parallel-to-serial translation.

The principle works with any processor. Balancing execution times between the three subroutines is the most difficult part of a software UART. Intel's "Using the Intel 8085 Serial I/O Lines" application note (AP-29) is the best reference for the math behind the computations. It was published in August of 1977, and at that time carried the publication number 9800684A.

Notice that all of the operations are independent of baud rate and processor clock speeds. That is, nothing in this code knows either of these parameters. Instead, BRID just measures bit time in terms of counts through a loop, an arbitrary, relative measurement that is indeed a function of both the CPU's speed (i.e., number of loop iterations per second) and the bit rate. If you know your processor's clock rate you could dispense with the BRID routine altogether; just compute (or better, measure) the bit time and plug it into the code.

I find that on a Z80 or 64180 the code will support 9600 baud transmission if the processor runs at more than about 6 Mhz. A very slow clock will necessitate using reduced baud rates. This is due to the loop count computed in routine BRID that forms the basic unit of timing. If the count falls below 1, as it will with very fast baud rates or slow processors, then the routines' counts will be meaningless.

```
;
;  BRID - Determine the baud rate of the terminal. This routine
; actually finds the proper divisors BITTIM and HALFBT to run CIN
; and COUT properly.
;
;   The routine expects a space. It looks at the 6 zeroes in the
; 20h stream from the serial port and counts time from the start
; bit to the first 1.
;
;  serial_port is the port address of the input data. data_bit
; is the bit mask.
;
brid:
      in    a,(serial_port)
      and   data_bit
      jp    z,brid            ; loop till serial not busy
bri1: in    a,(serial_port)
      and   data_bit
      jp    nz,bri1     ; loop till start bit comes
      ld    hl,-7       ; bit count
bri3: ld    e,3
bri4: dec   e           ; 42 machine cycle loop
      jp    nz,bri4
      nop               ; balance cycle counts
      inc   hl          ; inc counter every 98 cycles
                        ; while serial line is low
      in    a,(serial_port)
      and   data_bit
      jp    z,bri3            ; loop while serial line low
      push  hl          ; save count for halfbt computation
      inc   h
      inc   l           ; add 101h w/o doing internal carry
      ld    (bittim),hl ; save bit time
      pop   hl          ; restore count
      or    a           ; clear carry
      ld    a,h         ; compute hl/2
      rra
      ld    h,a
      ld    a,l
      rra
      ld    l,a         ; hl=count/2
      ld    (halfbt),hl
      ret
;
; Output the character in C
;
```

```
;  Bittime has the delay time per bit, and is computed as:
;
;  <HL>' = ((freq in Hz/baudrate) - 98 )/14
;  BITTIM = <HL>'+101H  (with no internal carry prop between bytes)
;
; and OUT to serial_high sets the serial line high; an OUT
; to serial_low sets it low, regardless of the contents set to the
; port.
;
cout: ld    b,11        ; # bits to send
                        ; (start, 8 data, 2 stop)
      xor   a           ; clear carry for start bit
co1:  jp    nc,cc1      ; if carry, will set line high
      out   (serial_high),a ; set serial line high
      jp    cc2
cc1:  out   (serial_low),a; set serial line low
      jp    cc2         ; idle; balance # cycles with those
                        ; from setting output high
cc2:  ld    hl,(bittim) ; time per bit
co2:  dec   l
      jp    nz,co2      ; idle for one bit time
      dec   h
      jp    nz,co2      ; idle for one bit time
      scf               ; set carry high for next bit
      ld    a,c         ; a=character
      rra               ; shift it into the carry
      ld    c,a
      dec   b           ; --bit count
      jp    nz,co1      ; send entire character
      ret
;
;  CIN - input a character to C.
;
;  HALFBT is the time for a half bit transition on the serial input
; line. It is calculated as follows:
;   (BITTIM-101h)/2 +101h
;
cin:  ld    b,9         ; bit count (start + 8 data)
ci1:  in    a,(serial_port) ; read serial line
      and   data_bit    ; isolate serial bit
      jp    nz,ci1      ; wait till serial data comes
      ld    hl,(halfbt) ; get 1/2 bit time
ci2:  dec   l
      jp    nz,ci2      ; wait till middle of start bit
      dec   h
      jp    nz,ci2
ci3:  ld    hl,(bittim) ; bit time
ci4:  dec   l
      jp    nz,ci4      ; now wait one entire bit time
      dec   h
      jp    nz,ci4
      in    a,(serial_port) ; read serial character
      and   data_bit    ; isolate serial data
      jp    z,ci6       ; j if data is 0
      inc   a           ; now register A=serial data
ci6:  rra               ; rotate it into carry
      dec   b           ; dec bit count
      jp    z,ci5       ; j if last bit
      ld    a,c         ; this is where we assemble char
      rra               ; rotate it into the character from carry
      ld    c,a
      nop               ; delay so timing matches that in output
                        ; routine
      jp    ci3         ; do next bit
ci5:  ret
```

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.

Schweitzer Engineering Laboratories (SEL) seeks a professional, innovative and meticulous individual for our Software Engineer position. If you are looking for an opportunity to design new products using the latest development tools working on a dynamic team in a results oriented environment, then this may be the position for you! SEL has a reputation for quality, reliability, integrity, and service. SEL is located in Eastern Washington where you'll enjoy an unmatched quality of life. Please submit your resume through our website at www.selinc.com/careers.

BC Tech is looking for a qualified candidate for the position of Embedded Systems Engineer in Santa Cruz, CA, with a proven track record of designing and developing a variety of medical devices and equipment from concept through prototype and into production.

The candidate shall be experienced in embedded systems engineer while working in an ISO 13485/FDA regulated environment. This position requires a high level of people skills as well as excellent communication skills. The candidate must possess at least 5 years of experience in the following:

* Experience with embedded systems is a must.
* Design of environmental control systems, software and hardware design and prototype development
* Software programming in C and other high-level software languages and assembly languages.
* Analog circuit design: ADC,DAC high speed OP-Amps, Electro-optical
* Digital circuit design: FPGA's, DSP, VHDL, microprocessor.

* Experience with board level debugging skills using logic analyzers, in-circuit emulators (ICE)

The candidate will have a B.S. in Electrical Engineering, Computer Engineering is desired, with a M.S. preferred. Equivalent experience will be considered. Email resumes to jobs@bctechinc.com.

Dynojet Research in Belgrade Montana has an immediate opening for a FIRMWARE DESIGN ENGINEER and a SOFTWARE PROGRAMMER in our Belgrade, Montana facility. Responsibilities for the firmware design engineer include firmware programming for control systems in motorsports products. Candidates must have demonstrated firmware design skills in C and Assembly language and have good documentation exp. Should be familiar with Motorola (Freescale) HC08, HC12 & DSP microcontrollers as well as ARM technology. Demonstrated knowledge in fuel injection & internal combustion engine operation is a plus. B.S. in E.E. or C.S. preferred, but not reqd. Demonstrated firmware work experience is preferred. Must have the ability to prioritize & meet deadlines, have good interpersonal and teamwork skills; strong organization & planning skills.

Responsibilities for the software programmer include data collection/acquisition in motor sports products, developing data analysis tools, graphical user interface & PC based equipment. Work independently & collaboratively on new and existing projects. Understanding of C++ and OO Design required. CS degree or equiv exp. reqd. Exp with ASP & SQL a plus.

Entry Level to Experienced Firmware and Software Engineers Encouraged to Apply! Fax resume to 406-388-6523 or email Dianne@Dynojet.com.

Joke for the Week

.
Dave Clark sent: "How does a project get to be a year late?

“One day at a time!"

From 'The Mythical Man-Month’ by Frederick P Brooks, Jr.

Which reminds me of the classic: “How did you go bankrupt?”

“Slowly at first… then all at once.”

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*". ". BUT - please use YOUR email address in place of "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.