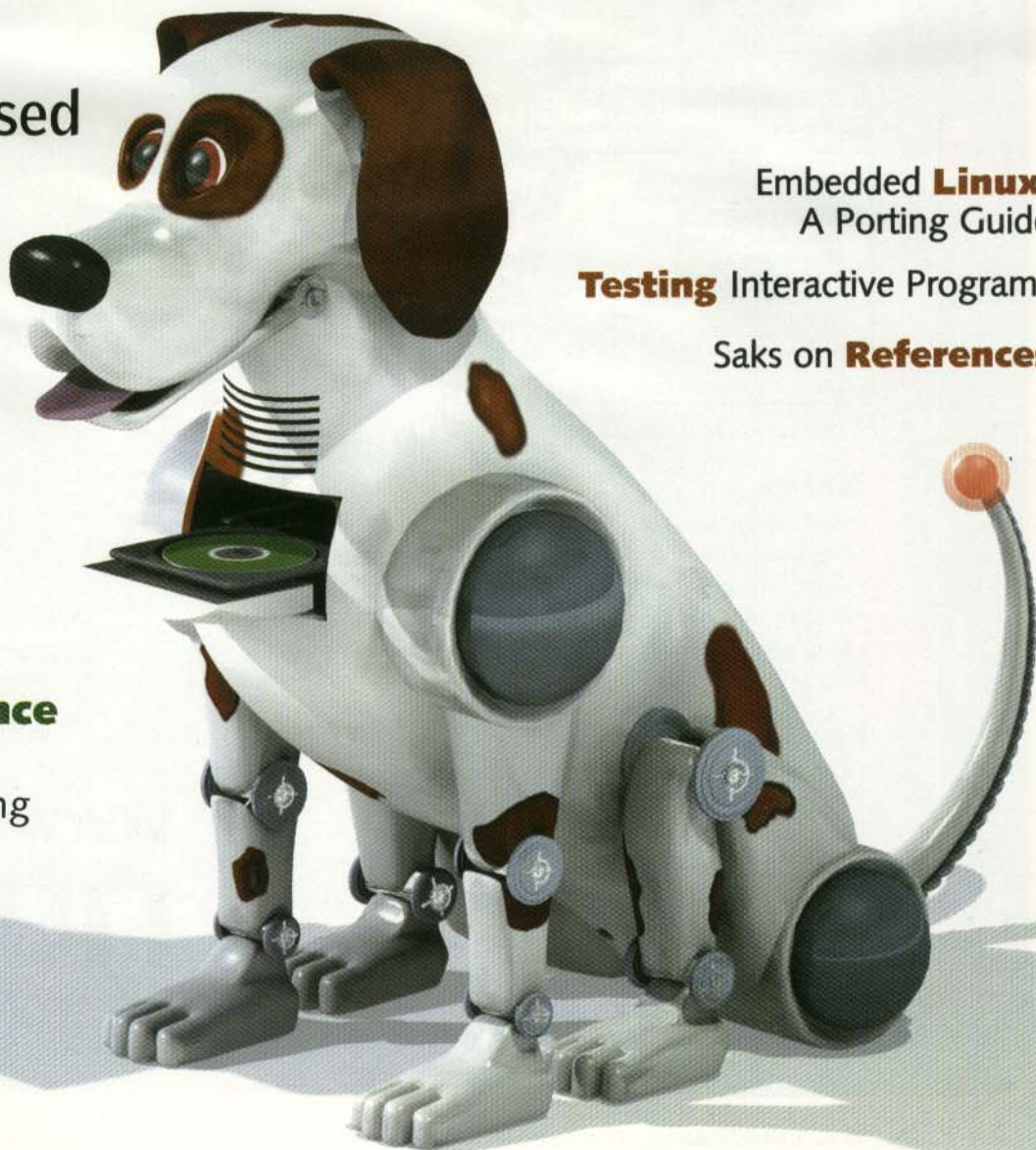


Embedded Systems[®]

P R O G R A M M I N G

Give the Dog a Chip

Processor-based
Toys



Embedded **Linux**:
A Porting Guide

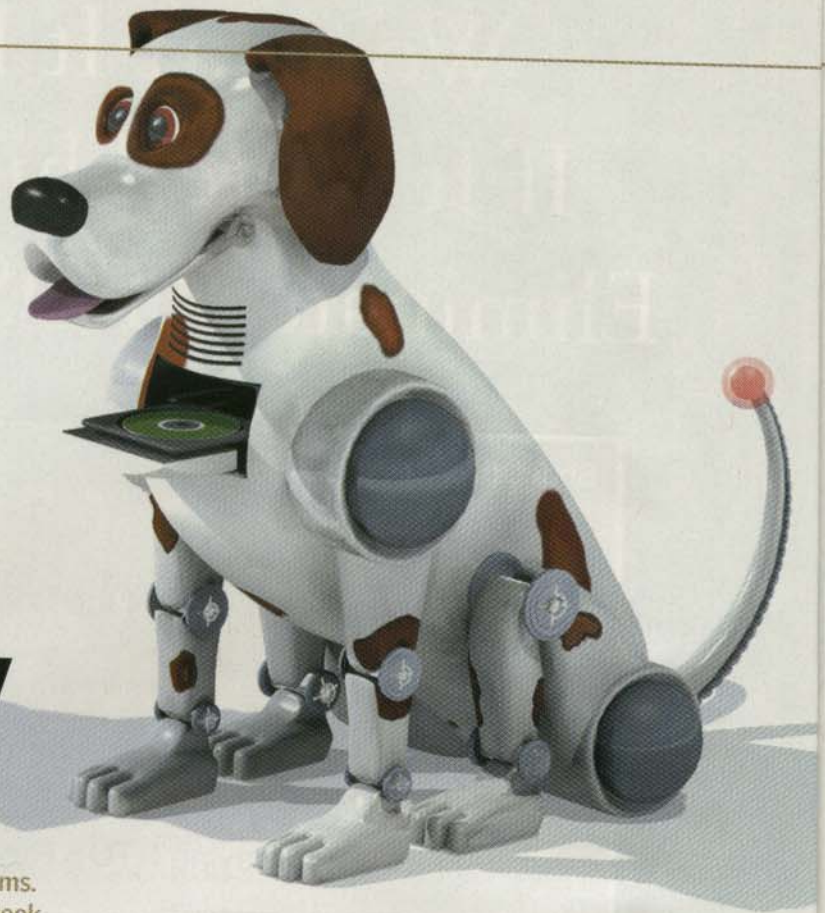
Testing Interactive Programs

Saks on **References**

**Internet Appliance
Design:**

Real-Time Networking

1-Wire Protocol



Toy Story

Increasingly, children's toys are embedded systems. The engineering behind them demands a closer look.

Walk through any toy store (preferably unaccompanied by your children) and you'll see a huge range of talking and musical toy products: a cat that purrs when you pet it, a doll that says "I love you, Mommy" when you hug it, and so on. Engineering these toys is a special science, surprisingly complex for such simple products, and it makes an interesting contrast from designing more expensive systems. In this article, I will describe the engineering behind a simple speaking doll to highlight the particular problems and requirements of the toy industry.

For brevity's sake, I will skip the "idea design" phase of a toy project, and start where the engineering begins, with the project specification already in hand. This is typically in the form of a play-pattern flow chart detailing the behavior of the toy. Our fictitious doll's play pattern is detailed as follows:

- The toy will be a baby doll with a normally open switch in the abdomen (to detect hugging) and a gravity switch (to detect if the doll is standing or lying down)

- The gravity switch will be open while the doll is upright
- If the abdomen switch is pressed while the doll is in the upright position, the doll will play one of the following sounds: giggle, long laugh, or a happy-sounding nonsense sound. It will cycle through these sounds, playing each in turn as the abdomen switch is pressed repeatedly
- If the abdomen switch is pressed while the doll is lying down, the doll will cry and then fall silent
- When the doll is shifted from the lying to the standing position, it will yawn
- When the doll is shifted from the standing to the lying position, it will snore briefly and then fall silent

Sounds

The first real engineering step is to record the sounds and transport them to the chip development system. Although we'll be downsampling a lot later, starting with high-quality samples is a must. The recording sessions are undertaken in an acoustically "dead" studio environment using 44kHz 16-bit sampling, recorded directly to hard disk. It is usual to hire professional voice talent for this, rather than having the project engineer or office staff record the sound. It's

best to do several takes of each word or phrase, so that you have a range of sounds to choose from without having to call the voice talent back in.

For projects that involve music from electronic instruments, it is preferable to record the artist's efforts in MIDI format rather than an audio stream; MIDI allows you to change instruments and timing, and to remove polyphonic elements as necessary when fine-tuning the performance for the target hardware.

Sometimes, the audio engineer must "factor" sampled sounds to reduce ROM space usage by merging redundant sounds. The standard example of this is in a counting toy that must count from one to twenty. Instead of having one separate sample for each number, the toy will contain samples for the numbers from one to 12, and the number 20, then the fragments "thir-" (for 13), "fif-" (for 15), and "eigh-" (for 18), in addition to the suffix "-teen". Note that we can use the "four," "six," "seven," and "nine" sounds both as standalone numbers and as prefix sounds for their "teen" versions.

The degree to which we need to factor depends on how much space is available. For example, we could try to factor 31 into "thir-" (we can use that for 13 and 30), "-ty" (we can use that for all the "-ty" words), and "one." Factoring is exhausting work that involves cutting and pasting fragments of syllables, enhancing plosives, testing various combinations of concatenated sounds to make sure they sound like coherent words, and so on. It can take a couple of weeks of full-time effort to get a realistic-sounding result for just a few words, so try to avoid factoring further than necessary.

The extreme case of factoring is to create an allophonic speech synthesiz-

er chip like the old Votrax SC-01 or General Instruments SPO256A-AL2. These chips contain—actually, synthesize—a library of speech fragments from which you can assemble words. For example, you could make "dog" from the fragments "d," "ah," and "g"; however, it's very hard to make this type of speech sound realistic. The project we are discussing here is much too simple to merit this kind of effort.

Microcontroller selection

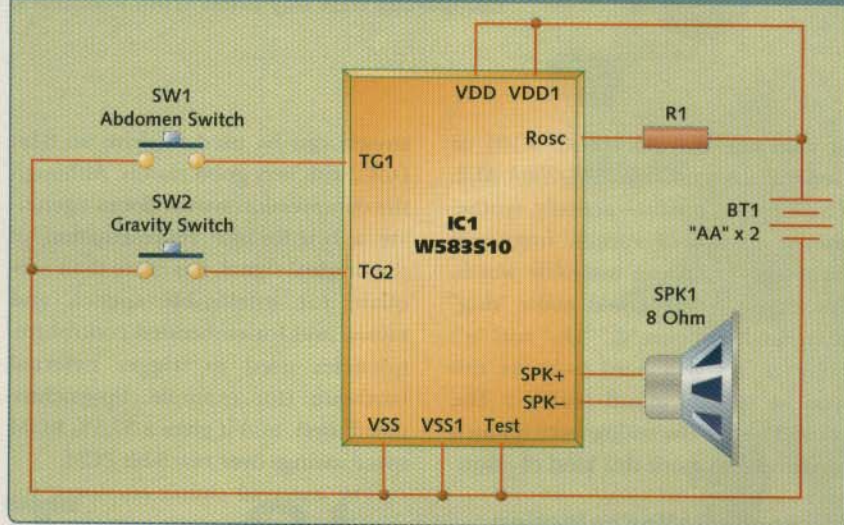
We now need to choose a microcontroller for this creature. Every penny counts in toy production, so an idea that requires too much expensive horsepower will never make it past the toy companies that buy ideas from inventors. As a result, almost all speaking toys are based around 4-bit speech micros such as the enormously popular PowerSpeech line from Winbond. These consist of a simple processor core with a few input and output pins, a couple of registers, and an enormous, serially accessed ROM containing a short program and a large chunk of sampled sound.

Encoding technology in speech micros has come a long way. Early chips used a simple 4-bit PCM encoding to squeeze the longest possible recording time out of the shortest possible ROM space. Especially at low sample rates, this sounds terrible because of quantization errors. Modern chips use 4-bit or 5-bit adaptive differential pulse code modulation (ADPCM) encoding. This is a predictive encoding system that exploits the fact that consecutive samples in a digitized representation of an analog signal will be correlated. Each sample word fetched is arithmetically combined with the current output state of the decoder to generate the next output state. Using ADPCM, a 5-bit input

stream can be used to drive an 8-bit DAC, with very good results. Although the compression system doesn't generate a byte-for-byte representation of the original signal, it is more than adequate for intelligible speech and music, and for embedded control frequencies used to trigger external hardware (for example, lip-synchronized toys), and it gives a 37.5% ROM space savings over raw 8-bit PCM.

The external circuit employed for most toy applications is very simple: a battery, an external resistor to configure the chip's timebase, a small speaker, and possibly a transistor to drive the speaker. To give you an idea of just how cheap these chips are, a masked, unpackaged die with enough ROM space for two minutes of full-quality audio is around \$1.20 in production quantities. Most toys use much smaller chips, around the \$0.15 to \$0.30 price point. For our application, we will choose the W583S10, which is the simplest of the PowerSpeech line. At the nominal 6.4kHz sample rate recommended by Winbond, this chip can store just 10 seconds of audio data; it has four input pins and five output pins. It even has the capability (with the addition of an external IR LED and receiver module) to encode and decode infrared serial transmissions. We won't be using that functionality; our simple circuit is shown in Figure 1. Note that the value of R_1 is dependent on the final sample frequency used. Normally the designer would employ a resistor substitution box to experiment with different values for this part.

As you might guess, the instruction set of these chips is not very rich. They typically have instructions to load a register with a constant, increment/decrement a register, set the state of the output pins, jump unconditional,

FIGURE 1 Schematic of a doll circuit

jump unconditional through a register, and make various conditional jumps based on input pin state. These chips typically have no RAM at all except for a few registers. There is no stack and the only way to implement a subroutine is to store a return address in a register. If you're programming these devices, ignore everything your computer science professor taught you; the only way to write a program is to use a forest of GOTOs and, for any non-trivial project, usually a lot of almost-redundant code.

Power consumption

Although battery-buying parents might not believe it, power consumption is another key design priority. The majority of speaking toys are shipped from the factory with batteries pre-installed, and the batteries are expected to last for their normal shelf life while inside the toy. Many toys lack power switches, and a large proportion of those toys have a "try-me" mode so that people browsing in the store can press a button and see or hear what the toy can do. For this reason, it is vital that we turn off all portions of the chip we don't plan to use, and we must make sure to power down the chip as completely as possible when it finishes any play sequence.

One very important power issue stems from the use of ADPCM encoding. If we simply stop the micro after playing a sound, the output DAC will assume that we plan to play more sample data later, so it will retain the last output state reached after the last sample word. This will almost certainly equate to a non-zero output voltage (probably close to half of the maximum voltage), which is pure waste current going into the loudspeaker. We could turn off the DAC after each sample, but that would cause a nasty click. For this reason, the chip manufacturer provides assembler macros (actually, short audio samples) to ramp the DAC output from 0V up to the halfway point and back again. We ramp down when we finish playing a sample, to conserve power, and we ramp up again just before playing a new sample.

To help us achieve the goal of extended battery life, the program is largely interrupt-driven. At power-up, and for each event on an input pin for which interrupts have been enabled, the micro begins execution at a known point. When a STOP instruction is encountered, the micro powers down almost completely (maximum current drain 1μA for the W583S10). The batteries will last for close to their shelf life in this configuration.

Code

Listing 1 shows some example assembly pseudocode for our doll. This code won't actually assemble for any known speech chip, but in syntax it is almost identical to the Winbond PowerSpeech series language; I have simply chosen some easy-to-understand symbol names rather than using Winbond-specific equate names. Also note that input pins are referred to as "triggers" and output pins, though we aren't using any here, are referred to as "stops."

Of course, a real toy's behavior would probably be more complicated than what we described here. For example, we would most likely insert a half-second pause before the yawn and snore sounds, to make the timing "feel" more realistic. Depending on the application, we might also want to disable interrupts while playing back the sounds, so that the toy would finish playing the sound before recognizing further inputs. Cosmetic issues of that sort are best resolved by building and testing the code, and require a good deal of experience and personal judgement. In addition, we might want a "try-me" mode where the doll will respond only to presses on the abdomen switch (this will avoid battery wastage during shipping). This would normally be accomplished by having a plastic pull-tab holding a pair of contacts apart, keeping the gravity switch out of circuit, until the purchaser removes the tab.

Sounds, revisited

Once the code's behavior is satisfactory, we can tweak and optimize the sound files. This tweaking process involves low-pass and/or high-pass filtering to eliminate undesirable resonances and improve intelligibility. A careful engineer will also boost quiet plosives or fricatives so that they may be more easily distinguished (this is a necessary step for spelling-type games and many other educational products). For toys that have lip movements synchronized with their speech,

a mouth control signal is embedded in the audio file at this time. All the editing and filtering work is done with the sound bites in 16-bit 44.1kHz WAV files; these are then downsampled to the desired sample rate for the final chip (typically in the 6kHz to 8kHz range), and the chip vendor toolchain imports and converts these files to the micro's internal ADPCM format. The exact sample rate will be chosen to maximize utilization of available ROM space; once we have set our ROM budget by selecting a microcontroller, we choose the highest possible sample rate that will still allow the compiler to fit all of our sounds into the device.

Speaking of the compiler, you shouldn't expect a high-quality development system for these 10-cent chips. The best they get is mediocre, and most of the vendor-supplied software is downright awful; peculiar bugs and unexplained behaviors abound. Even


trivial programs can be held up by the compiler refusing to process a particular sound file unless its length is slightly increased or decreased, strange errors being reported due to too much or too little whitespace in the code, and other frustrating problems.

Testing in this stage is preferably carried out using a prototype sample of the toy's real housing. Where possible, we should try to specify the exact type of loudspeaker to be used as well, because it has a very significant effect on the final sound quality. The cheap toy speech chips are capable of excellent fidelity when driving good speakers in a well-designed enclosure; toy designs rarely approach the optimal, but the size, type, and/or positioning of a loudspeaker can be the difference between a product that sells and a product that stays on the shelves because nobody can understand what it's saying.

Up to this point, we have been working with the micro "in vitro," on the chip vendor's development board, using flash memory to store our program under development. The final product is practically always a mask-ROM COB (chip-on-board) part, where the chip die is bonded directly to the PCB, leads are soldered from it to PCB traces, and the chip is protected with a dollop of resin. Since our code is now complete, with the approval of the toy company we send the final object code out for masking and bonding.

By the way, it's easy for me to say "with the approval of the toy company," but in practice, political issues frequently arise at this point. When executives of a toy company finally see a supposedly complete version of the product, with an emulator board inside a real toy body, they often have complaints. Laughing toys sound too "satanic" (yes, really—lawsuits have been filed about this), witty toys are "too risqué," toys are too scary because of sudden loud sounds or pronounced movements, motors or gears are too noisy, the toy company thinks the voice should sound more friendly, and countless similar frustrations appear. This phase can be particularly annoying when the toy contains a licensed voice (such as a children's movie character); the owner of the character's rights will veto anything that doesn't sound sufficiently like the original, or which doesn't fit in with the character's public image.

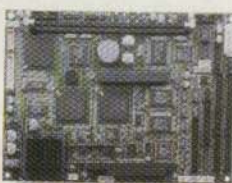
With these issues resolved, the engineer's involvement usually concludes with fine-tuning of production ROMed, COBed samples. For example, the analog characteristics of the masked microcontroller parts are slightly different from the flash-based development system, so the value of the timing resistor is chosen based on final testing of a few production samples. Occasionally, this testing phase uncovers other problems such as code bugs that need to be worked around, if possible, by patching the



Fanless SBC in an EBX Platform

PCM-9550F Features 8" x 5.75"

- Intel low power Pentium® MMX™ processor
- Supports Video-in and TV-out (PCM-9550FM)
- 8 digital inputs and 8 digital outputs
- Supports XGA & 36-bit LCD
- 3D audio & 100 Mbps Ethernet
- One PC/104+ & one mini PCI socket (Type III)



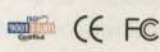
- **EBX Form Factor**
- **No Cooling Fan**
- **Digital I/O**
- **Long-term Supply**

ADVANTECH

Embedded Computing

Advantech Technologies, Inc.
Tel: 949-789-7178 Fax: 949-789-7179
E-mail: EPCInfo@advantech.com

1-800-866-6008
www.advantech.com/epc



circuit in some way. Once the "really final" production samples are approved, our work is done and we can sit back and dream of large royalty checks.

Design options

Of course, not all toy designs use these speech microcontrollers. Three major categories of microcontrollers are used in toys: the speech micros described pre-

viously, single or multi-voice "squeaker" tune-playing chips of the type found in musical greeting cards, and chips with various LCD controllers in them (the latter are used mainly in handheld games). Various manufacturers also make hybrid parts that are essentially one of their 4-bit speech microcontrollers squeezed onto the same piece of silicon as a second, more powerful microcontroller that controls a dot-matrix LCD and tells the speech micro when to make sounds. Apparently, the primary application of these latter chips is the toy "computers" made by VTech, which you will see on the shelves of almost any toy store. (A few maverick items on the market, like the Sony Aibo robot dog, use more powerful micros, but these devices are priced an order of magnitude or two above the normal toy category, so they don't count.) As an interesting aside, Furby is also a complex maverick: he contains several separate electronic subassemblies, and a large number of discrete components.

Working in toy design is at least as much fun as playing with toys, and there's a special pleasure to be derived from walking through a store and hearing children cajoling their parents into buying a product that contains your code. Unfortunately, most toy design work has left the U.S., so there aren't many positions available now. But if you enjoy fun projects that make the most of very constrained systems then you can't get a better assignment than toy design. You might be surprised at the amount of engineering effort behind that \$10 gurgling baby doll, but I bet you'll never look at one the same way again.

esp

Lewin A.R.W. Edwards is an Australian-born embedded engineer now working for Digi-Frame in Port Chester, NY. He has developed hardware, firmware, and device drivers around a variety of different processors. He's worked in the fields of digital imaging, encryption and security, and—no surprise—in the electronic toy industry. He can be contacted by e-mail at larwe@larwe.com.

LISTING 1 Pseudo-code for a simple toy

```
; --- Power-on reset entry point
; --- Enable interrupts on falling edge of triggers 1 (tummy switch)
; --- and 2 (gravity switch), and rising edge of trigger 2.
POR:    LD      INT_ENABLE, TG1_LO OR TG2_LO OR TG2_HI
        LD      RD, GIGGLE ; Register 0 is used to store the next sound
                                ; to be played when the tummy switch is pressed.
        STOP      ; Power down and wait for interrupt.

; --- Micro jumps here when trigger 1 detects a falling edge (tummy switch is pressed).
; --- Note: Switch debouncing is performed in hardware by the microcontroller.
; --- The input pins also have internal pull-ups, so no external parts are
; --- required.
TG1_LO: [rampin] ; This is a vendor-supplied macro that ramps
                ; the ADPCM decoder output up to its center
                ; point
        JMP      RD, TG2_HI ; If the doll is upright (TG2 = high), play
                                ; one of the three tummy behavior sounds below.
        [cry] ; Otherwise, cry
        [rampout] ; Ramp the decoder output back to 0V
        STOP

; --- Tummy behavior #1
GIGGLE: [giggle] ; Play the giggle sound
        LD      RD, LAUGH ; Next time around, we will laugh
        [rampout] ; Ramp the decoder output back to 0V
        STOP

; --- Tummy behavior #2
LAUGH: [laugh] ; Play the laughing sound
        LD      RD, NONSENSE ; Next time around, we will talk nonsense
        [rampout] ; Ramp the decoder output back to 0V
        STOP

; --- Tummy behavior #3
NONSENSE: [nonsense] ; Play a happy-sounding baby-talk phrase
        LD      RD, GIGGLE ; Loop back around to the first sound in the series
        [rampout] ; Ramp the decoder output back to 0V
        STOP

; --- Micro jumps here when trigger 2 detects a falling edge (doll is laid down).
TG2_LO: [rampin] ; Ramp the decoder output to its center point
        [snore] ; Snore for a little while
        [rampout] ; Ramp the decoder output back to 0V
        STOP

; --- Micro jumps here when trigger 1 detects a rising edge (doll is stood up).
TG1_HI: [rampin] ; Ramp the decoder output to its center point
        [yawn] ; Good morning! Yawn.
        [rampout] ; Ramp the decoder output back to 0V
        STOP
```