Philips ARM Design Contest 2005
First Prize

ARM-Based Modern Answering Machine

The next-generation answering machine has arrived. No more shoddy voice quality or limits on storage space. Bernard's Internet-connected system is perfect for the home and office.

Just a few months before the end of the twentieth century, I bought my last telephone answering machine. Although it was a nice-looking brand-name system, I discovered a number of problems as soon as I started using it at home. The first problem was that the machine used an aggressive voice compression algorithm to reduce the size of the voice memory. This compression had a nasty effect on the quality of the recorded voice content. Another problem was that the machine had a cumbersome user interface that used voice prompts. I could check messages remotely, but I had to deal with the horrible voice prompts. The configuration was kept only if the answering machine was fitted with a fresh 9-V battery. Finally, I found the machine's memory to be extremely limited. As a result, I had to check my messages frequently when I was traveling.

I built The Active Mansion Telephone Answering Machine (TAM-TAM) to solve all of these problems (see Photo 1). I didn't replace the original answering machine's poorly designed user interface with a better one. I avoided the issue altogether by editing a configuration file on a PC featuring a clear display, a keyboard, and text editing software. The configuration file is stored on a flash memory card that also stores all of my incoming messages. The system runs a small web server that displays a list of pending messages. I can retrieve the messages via the Internet.

In this article, I'll explain how to build a similar answering system around a Philips LPC2138 microcontroller. Before I describe the circuitry, let's take a look at the system's basic modes of operation.

MODES OF OPERATION

I customized the TAM-TAM system shown in Photo 1 for the Smith family. Featuring only four main modes of operation, the system was fairly easy to develop.

In Idle mode, the system waits for a phone call. The ring detection routine is sophisticated enough to reject short rings and rings of the wrong frequency. All of these parameters are extracted from the core software. Magic numbers are avoided throughout the software.

When the first ring burst is detected, the system enters Smart Answering Machine mode (see Figure 1). First, the caller ID detection routine is activated. An interrupt service routine (ISR) handles the demodulation of the signal, software UART, and checksum calculation. The background task waits for the detection to be completed. A timeout allows the operation to be aborted if the detection is missed or if the checksum is wrong. After a preprogrammed number of rings, the system goes off the hook. Another key step (sometimes mishandled) involves correctly counting the rings. A ring burst on Monday followed by another ring burst on Tuesday isn't counted as two rings!

Photo 1—*The TAM-TAM is a compact system. The wireless bridge provides Internet access. The simple front panel features only four LEDs and four push buttons (the green circles). The flash memory card is sticking out of the front panel. Ethernet and phone line connectors are located on the back panel.*

After going off hook, the machine plays the first WAV file: "Hi, this is the Smith family. Please press 1 for Jim, 2 for Julie, 3 for the kids, or 0 for a general message." This statement invites the caller to send a DTMF key to direct the message to the correct virtual answering machine. If a DTMF digit isn't detected, a generic message is played after a programmable timeout. Otherwise, a more spe-

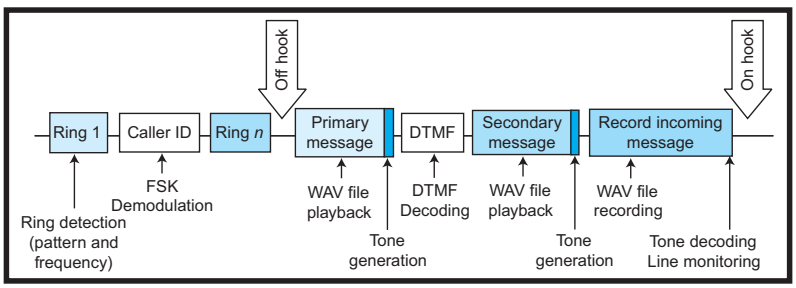


Figure 1—*It's important to understand the different phases of the answering machine mode and the DSP functions that are required in each phase.*

cific message can be played. If the caller presses the 2 button, the system plays Julie's recording: "Hi, this is Julie. I am currently visiting my mother. Please leave me a message."

General messages and the ones recorded after a digit timeout are stored in mailbox 0. Messages are recorded until one of three things happens: the maximum message duration, which is programmable, is reached (in which case a short tone will inform the caller that the system is about to hang up); someone picks up the local handset (thus aborting the message); or a remote hang-up is detected (when a short interruption of the line current is sensed or when a dial tone is detected). After a message is recorded, the recipient is notified by e-mail and his or her assigned green LED (0, 1, 2, or 3) blinks slowly. This indicates that at least one message is pending.

A user like Julie can retrieve her messages when the system is in Local Retrieval mode. The front panel is easy to use. She can press her button and retrieve her messages one after the other. After a message, a short tone sounds. She now has a 3-s window of time to save or erase the message. If she presses a key and holds it for 1 s, the most recently played message will be erased from the queue.

I think Internet Message Retrieval mode is the most interesting. By just typing the IP address of the home network to which the TAM-TAM is connected, Julie can retrieve her messages. This retrieval procedure has an inherent problem because ISPs don't want users without a professional account to run web servers on their network. This is why ISPs often rotate the IP address assigned to such networks and block incoming connection to port 80. I designed the TAM-TAM with the web server running on port 8000 to bypass this restriction.

In addition, I subscribed to a service that enables me to name my web server and access it on its own port for free. The TAM-TAM's own web server running on port 8000 can be accessed from anywhere by typing tamtam.hostredirect.com (hostredirect.com is just an example here). The IP address is maintained current in the DNS servers thanks to a small utility running on my PC. The company that offers this service details how this utility can be implemented on another platform, so theoretically it could be ported to the TAM-TAM.

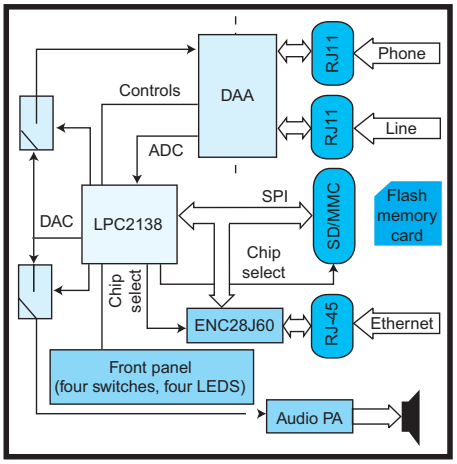


Figure 2—*Study the connections among the different subsystems. The LPC2138 microcontroller and its SPI bus form the TAM-TAM's core.*

SYSTEM OVERVIEW

The TAM-TAM is built around an LPC2138 microcontroller (see Figure 2). The microprocessor shares its SPI bus with an SD/MMC card reader and a Microchip Technology ENC28J60 Ethernet controller. (I used a Kingston Technology Elite Pro SD/MMC flash card, but you can insert any card in the reader.) Two chip select lines direct the SPI bus transfers to the flash memory card or to the Ethernet interface (see Photo 2).

Individual GPIOs control the LEDs and the front panel switches. The integrated ADC that's directly connected to the line interface is used for recording from the telephone line. The DAC is used for local playback on the speaker or remote playback on the telephone line. You may download a complete schematic from the *Circuit Cellar* FTP site.

I initially developed some of the subsystems with a Keil MCB2130 development board. I developed a PCB when I was confident that the project would be successful.

FOUR-PIN ETHERNET

I was fortunate enough to obtain an early sample of the ENC28J60 Ethernet chip. It's an interesting device. The ENC28J60 makes it extremely easy to add Ethernet connectivity to a small processor as long as the processor carries a SPI. I chose this interface because it requires only four pins: SPI IN, SPI OUT, CLOCK, and CHIP SELECT.

The ENC28J60 includes two configurable LED drivers. I used the default configuration, which is a Carrier Detect indication on one LED and a Traffic Indicator on the other. I mounted the LEDs on the PCB for debugging purposes, but they aren't shown outside the box. The chip requires only two additional components: a decoupling capacitor for the internal regulator and a bias resistor.

I was initially concerned about the process of developing a new driver for a new part. But the task was facilitated because the chip performed exactly as documented despite all the vendor warnings that came with an early sample.

LINE INTERFACE

The key component in an apparatus connected to a telephone line is the data access arrangement (DAA). The DAA in the TAM-TAM was derived from the application note "Low Cost Telephone Line Interface (DAA, FXO)." Two optocouplers were added to the design from the Midcom application note. One is in the line to the transformer. It detects the short interruption of the line current that occurs when a caller hangs up the phone. This prevents the TAM-TAM from recording silence. The other optocoupler detects when the local phone is taken off hook. This process also stops the recording process. (You don't want to record someone who hasn't been forewarned!)

Another change from the system described in the application note is the addition of an AC bypass of the solid-state switch and the current sense optocoupler. The telephone company sends the caller ID information while the phone is still on hook. Therefore,

Photo 2—*The PCB I designed integrates all of the components. Different subsystems are designed around the LPC2138 microcontroller. J1 mates with a debug board that includes a level translator and two DE9 connectors.*

an AC path through the transformer must be created even when current isn't flowing through the DAA.

The DAA also includes a two- to four-wire converter. This block converts the balanced telephone line into the unbalanced ADC input and unbalanced DAC output. The ADC input is used for recording messages from the telephone line. The telephone line (playback of messages to the caller) and the audio amplifier (local playback of recorded messages) share the DAC. Two analog switches direct answering machine prompts to the phone line and local playback to the speaker.

A wall plug delivers 5 V, which a linear regulator reduces to the 3.3 V needed by the different ICs. The 5 V is used for the audio amplifier to reduce the strain on the regulator and increase the available audio power.

USER INTERFACE

I don't like poorly designed user interfaces, so the TAM-TAM doesn't have configuration buttons or a display. Think about it. Typically, you must configure such a product only once. Whereas a manufacturer has to deal with a microphone and its associated circuitry (hardware an end user won't use more than once or twice), you can configure the TAM-TAM with a PC by editing and saving a simple text file.

Like all configuration files, the TAM-TAM's configuration file (tam.txt) must be stored in the root directory. You may download a complete configuration file from the *Circuit Cellar* FTP site. You can use any text editor to modify the sample file. You can develop a simple front end in Visual Basic that will create the tam.txt file, generate the sound files, and verify that all the files referenced in the configuration file exist in the flash memory card's root directory.

The other aspect of the configuration process is the creation of WAV files (e.g., the answering machine's greeting message). WAV files are also recorded on a PC. The simplest recording application on a Windows-based PC is the Sound Recorder program, which has a limited number of default profiles. The TAM-TAM uses one of them. Standard rates range from 8 to 44.1 kHz. There are 8- and 16-bit formats for one to two channels.

Because the goal is to play and record files on a telephone line, there's no point in using a rate higher than 8 kHz. The phone lines are limited to frequencies below 4 kHz. Phone companies carry voice signals at 8,000 samples per second. They also use nonlinear coding, which provides roughly the same dynamic range as a 13-bit linear coding would provide.

Finally, note that stereo isn't played on phone lines, so mono is good enough. I selected 8 kHz with 16 bits and one channel. The Sound Recorder program shows this as:

8.000 kHz, 16 bit, mono 15 kb/sec

You'd think that someone at Microsoft would realize that 16 bits are equal to 2 bytes. If you're sending 2 bytes 8,000 times per second, then your recording rate is 16 KBps, not 15 KBps! Despite this small glitch, there's no need to worry. The format is correct.

You can perform more complex editing with more sophisticated tools. Adding music is a popular option. Doing it digitally is better than playing an old tape on a boom box near the microphone of your twentieth-century answering machine!

WAV FILES

The TAM-TAM can play and record WAV files in the same format as a PC. Tracking the WAV file format can be another interesting project because of all the conflicting information on the Internet. My first attempt to record and then interpret WAV file content on a PC running Windows XP failed. There were differences between the format I found on the Internet and the actual data from a hex dump of a WAV file recorded on a PC. This problem was certainly caused by the fact that Microsoft has improved or added parameters to the older format.

The good news is that when you understand the philosophy of the format, it becomes easier to interpret the data instead of relying on a given piece of information at a given location. In some cases, the format interpreted in Playback mode might be slightly different from the format generated in Record mode. This doesn't affect the fact that both types of files are valid WAV files that you can play and record on the TAM-TAM or a PC.

The WAV file format is a hierarchy that starts with the four letters RIFF. The rest of the hierarchy can be navigated because each level announces its name and the length of the data that follows. When a name isn't understood or expected, it can be skipped.

During playback, the WAV files are loosely analyzed. The file interpreter navigates through the file structure to extract a key value: the number of samples to be played. This value indicates how many bytes (not samples) are present in the data section of the file.

The TAM-TAM makes no attempt to fix a wrong sampling rate value by performing quantization or interpolation. Nor does it do anything fancy with files that may have been compressed. On the other end, the parser correctly analyzes files that have been created with different versions of Windows or with different programs in the same operating system. The key is to not rely on a parameter to be present at a fixed offset. You must navigate the file structure while skipping the parameters that aren't understood.

FAT

The article "Portable FAT Library for MCU Applications," by Ivan Sham, William Hue, and Pete Rizun (*Circuit Cellar* 176, March 2005), is a great starting point for learning how to implement the FAT16 on an SD/MMC card. Their implementation is a perfect match for this project because the goal here is to allow the TAM-TAM and a PC to exchange files. Therefore, the files and the way they're stored must be compatible. The FAT16 file system is almost obsolete (at least from Microsoft's point of view), but it's still supported in Windows XP.

The code was designed for a different microcontroller family, but it was relatively easy to update for this project. I modified the hardware-dependent library to account for the LPC2138 SPI. An initial implementation on the Keil development boards was successful, and I was quickly able to read and write files.

I added a few functions to the original code, which doesn't have an option for reading, modifying, or writing files. The original code enables you to read or write only. This is a problem when recording WAV files because information

about the file length is located in the header at the beginning of the file. But this information isn't available until the end of the recording. I addressed this problem by allowing the file to be open in Write Only mode to write the incoming WAV data. Then, it's rewound to the beginning of the file to write the header information.

I also added a function to delete existing files. Without the delete function, there's no way to remove an existing file from the file system. Lastly, I created a file rename function.

VISUAL SIMULATION

One of my goals for this project was to experiment with the implementation of DSP features in a generic RISC processor. The starting point is a sample interrupt, which is actually a timer interrupt in this case. If higher accuracy and low jitter are required, the interrupt should actually come from the ADC, and the ADC value should be buffered until it's read by the interrupt routine. But this feature isn't available on the LPC2138. The same timer interrupt is also used to output samples for voice playback and tone generation.

The DSP functions are often tricky and difficult to debug. Professionals use tools like MATLAB to develop and tune their algorithms. Amateurs generally don't have access to such simulation tools. I tried a few different options.

One problem with demodulation is that you might be trying to catch a signal that shows itself and disappears quickly. On the other hand, a simulator allows you to generate millions of these furtive signals and verify that the algorithm works well in the presence of noise or any other hindrance. I developed and debugged the DTMF demodulator using an old version of Microsoft Visual C/C++. Although I couldn't use that version to develop modern applications to run on Windows XP, it was an excellent tool for developing generic C code. It also has a step function that can be used to track down nasty bugs. This was a nice feature because I didn't have a step function on my LPC2138 platform.

To debug the DTMF detector, I generated some DTMF samples and then sent them to my demodulation routine. Until I got the routine to correctly extract the right DTMF tones, I invested my time on the Windows platform rather than the ARM-based platform. After the DTMF detector was debugged in Windows, it worked the first time it was ported to the LPC2138! You may download the Visual C/C++ source code from the *Circuit Cellar* FTP site.

Because generic RISC processors like the LPC2138 don't have a saturation function, it's also important to ensure that the input values or the size of the variable are correctly scaled to prevent saturation. Refer to the "DP Versus RISC" sidebar for more information on the subject.

I started developing and debugging the FSK demodulator in an Excel spreadsheet. Excel isn't necessarily a perfect DSP development tool, but it has a complete set of arithmetic functions and graphics tools.

DTMF DEMODULATION

A DTMF tone consists of two tones sent simultaneously. The first tone is selected from a group of four low frequencies. The second tone is selected from a group of four high frequencies. This gives 16 possible configurations. The DTMF demodulator uses the Goertzel algorithm. The difference here is that the algorithm is written in C and doesn't use a single line of assembly code. The implementation of the algorithm itself was borrowed from an older implementation in the Asterisk Linux-based PBX.

The Goertzel algorithm performs a simple discrete Fourier transform (DFT). The key to the algorithm involves

DSP Versus RISC

There are a few fundamental differences between DSP and RISC processors. One difference has to do with arithmetic. In the analog domain, saturation, or clipping, isn't recommended. But it generally comes with a design when, for example, an op-amp is driven high with an input signal. In the digital domain, saturation should be prevented because it causes distortion of the signal being analyzed. But some saturation is better than overflow or wrap-around. Generally speaking, a RISC processor will not saturate, but a DSP will. This is an important feature if you want to do signal processing.

Let's take a look at an example. Consider a 16-bit processor working with unsigned numbers. The minimum value that can be represented is 0 (0x0000), and the maximum is 65535 (0xFFFF). Compute:

$$out = 2 \times x$$

where x is an input value (or an intermediate value in a series of calculations). With a generic processor, you're in trouble when x is greater than 32767.

If $x = 33000$ (0x80E8), the result is $out = 66000$ (0x101D0). Because this value can't be represented with 16 bits, the processor will truncate the value:

$$out = 2 \times 333000 = 464(0x01D0)$$

From that point on, all the calculations will be off.

On the other end, a DSP (or an arithmetic unit with saturation) will saturate the value to its maximum (or minimum) capability:

$$out = 2 \times 333000 = 65535(0xFFFF)$$

In the first case, looking at *out*, it would be wrong to assume that x is a small value. With saturation, the *out* is still incorrect, although it accurately shows that the input is a large number. Trends in the signal can be tracked with saturation. If the saturation isn't severe (affecting only a few samples), the signal might be demodulated correctly.

Generic RISC processors like the Philips LPC2138 don't have a saturation function, so it's important to ensure that the input values or the size of the variable are scaled correctly to prevent overflow. This problem can be avoided with a thorough simulation process.

finding the number of samples required to be accumulated before running the detection loop. This is the DTMF_NPOINTS parameter. There are numerous parameters affecting the selection of this parameter. The larger the value, the longer it will take to perform the detection and the narrower the frequency detection. The algorithm performs its analysis on discrete frequencies (also called "bins") that are located at:

$$f_i = i \times \left(\frac{\text{SAMPLINGRATE}}{\text{DTMF\_NPOINTS}} \right)$$

where i is an integer. For the detection to work well, f_i must match the DTMF frequencies as closely as possible; otherwise, the energy will bleed from one bin into another.

Tests and simulations have shown that a value of 205 gives the best result at a sampling rate of 8,000 Hz. A value of 205 gives a window of samples of 25.6 ms. In order to prevent the false detection of DTMF tones, the algorithm looks for two consecutive positive detections in adjacent windows. This means that the minimum tone duration is 51.2 ms, but it can take a little longer if the tone was started in the middle of a sampling window. After the samples are accumulated in the interrupt, the detection is performed quickly with the Goertzel routine (see Listing 1). At the end of the routine, the result variable has a bit set for each filter that has energy above a preprogrammed threshold. The detection routine finally checks that there is only a single bit set in each nibble representing the low and high frequency groups.

My algorithm is simple but reliable. Also, the time slice during which the TAM-TAM expects a DTMF digit is short.

TONE DETECTION

Tone detection enables the system to detect when a caller hangs up. After a while, the phone company sends a dial tone, which in the U.S. is the combination of frequencies 350 and 440 Hz. In some countries, a cadenced busy tone is sent that would require a slightly different implementation with cadence detection.

The DTMF detection works well. Because the buffering system was already in place, I used the same algorithm for tone detection. I increased the detection time to about 1 s (40 buffers of 205 samples). The detection process has to run concurrently with the wave recording. Because only two frequencies have to be detected, the processing is four times lighter than what's required for DTMF. Speech can easily trigger one of the two filters. It could trigger both filters, but because the detection has to happen without a single miss for 1 s, this event is unlikely.

While recording a WAV file, at the end of the interrupt routine that outputs the samples, an ugly goto instruction sends the program flow to the DTMF processing where the samples are gathered in packets of 205. The buffers are then sent to the tone detector by the background processing.

FSK SECRETS

An FSK demodulator demodulates the caller ID information from the telephone line. A simple edge detector followed by a counter can be used to demodulate FSK even on the tiniest 8-bit microcontroller. These demodulators perform well when fed with a perfect signal, but they show limited performance in real (noisy) applications. I couldn't run such a demodulator on my DSL-equipped telephone line. I needed a high-performance FSK detector.

Interestingly, it's hard to find information about demodulating FSK without using an edge detector. One semiconductor company that shall remain nameless even turned the source code from an application note into data statements (.word xxxxh) in the heart of an FSK demodulator in an obvious effort to hide the secret of FSK demodulation. After searching the 'Net for some time, I finally found the technical note, "FSK: Signals and Demodulations," in which author Bob Watson describes a filter-type approach to the demodulation of FSK signals.

I also found a practical implementation in the ham radio world: a data transmission system called packet radio can use FSK modulation. A radio link is generally a poor media for data transmission. Noise, distortion, and phase shift are common impairments that affect the signal. The receiver must be robust enough to take care of these impairments.

The FSK system used by ham radio operators is similar to the Bell 202 standard telephone companies use to carry caller ID information. The signaling speed is 1,200 bps. The frequency for a 0 is 2,200 Hz. The frequency for a 1 is 1,200 Hz.

Listing 1—*In this Goertzel routine, the loop calculates the energy in each of eight filters (the four low plus the four high frequencies).*

```
/* Process the 8 frequency filters */
for (k = 0; k < NDTMFCOEFF; k++)
{
    /* Goertzel processing */
    sk = sk1 = sk2 = 0;
    for (n = 0; n < DTMF_NPOINTS; n++)
    {
        sk = DTMFsamples[n] + ((DTMFCoeff[k] * sk1) >> 15) - sk2;
        sk2 = sk1;
        sk1 = sk;
    }
    /* Prevent overflows */
    sk  >>= 1;
    sk2 >>= 1;
    /* compute |X(k)|**2 */
    power = ((sk * sk) >> AMP_BITS) -
            ((((DTMFCoeff[k] * sk) >> 15) * sk2) >> AMP_BITS) +
            ((sk2 * sk2) >> AMP_BITS);
    result = result >> 1;
    if (power > DTMF_TRESH)
    {
        result += 0x80;
    }
}
```

There are also some differences. For example, packet radio is synchronous, and caller ID is asynchronous. That doesn't really affect the signal processing though. A practical implementation of the secretive FSK demodulation involves two filters tuned to the 0 and 1 frequencies.

The demodulator uses four tables. The `coeffloi[]` and `coeffloq[]` tables are initialized with the cosine and sine components of eight samples at low frequency (1,200 Hz). `coeffhii[]` and `coeffhiq[]` have eight samples at high frequency (2,220 Hz). Every time a sample is retrieved from the ADC, the low- and high-frequency filters are run with the last eight samples. This process is detailed in Listing 2.

At the end of the filter loop, `outloi` and `outloq` represent the phase and amplitude of the 1,200-Hz component of the input signal. `outhii` and `outhiq` represent the phase and amplitude of the 2,200-Hz component. I'm not interested in the phase information, so I can just calculate the total energy in each filter by taking the sum of the squared I and Q component (see Figure 3). I can then subtract the energy detected in the low filter from the energy detected in the high filter. If the result is positive, then a high frequency (2,200 Hz, bit 0) is assumed. If it's negative, a low frequency (1,200 Hz, bit 1) is assumed.

Listing 2—*Every time a sample is received, it's put into circular buffer `rxsamples[]`. When the routine is entered, `rxptr` points to the oldest sample in the buffer. The secret of the FSK demodulator lies in the following few lines of C code.*

```
for (i=0; i<NPERBAUD; i++)
{
    sample = rxsamples[(rxptr+i) % NPERBAUD];
    outloi += ( sample * coeffloi[i] );
    outloq += ( sample * coeffloq[i] );
    outhii += ( sample * coeffhii[i] );
    outhiq += ( sample * coeffhiq[i] );
}
out =    (outhii>>15) * (outhii>>15) + (outhiq>>15) * (outhiq>>15)
       - (outloi>>15) * (outloi>>15) - (outloq>>15) * (outloq>>15);
```

GEAR SHIFT

All of the TAM-TAM's DSP functions are performed at an 8,000-Hz sampling rate except for the FSK demodulator. If the same sampling rate had been used for the FSK demodulation, I would have had to sample the bits every 6.66 samples (8,000/1,200). This would have meant doing it every seven samples most of the time and every six samples some of the time. That's feasible, but the process is easier if the sampling rate is an integer multiple of the data rate. For that reason, the caller ID demodulation runs at a 9,600-Hz sampling rate, which is exactly eight times 1,200 Hz. (Note that 7,200 Hz would work as well.)

To avoid potential problems, the sampling rate is changed the first time an interrupt occurs after the DSP handler is configured for FSK demodulation. This change of gear doesn't interfere with other operations. There is no requirement for other DSP functions while demodulating FSK.

CALLER ID

Extracting the caller ID information is easy when the FSK demodulation is in place. The information is transmitted between the first and second ring. The caller ID frames carry the day and time of a call as well as the caller's name and phone number. The day and time are used to automatically set up a real-time clock as soon a phone call is received.

The caller's phone number and name, if available, are associated with the WAV file. This information is displayed when the TAM-TAM is accessed through a web browser.

TCP/IP STACK

This project required a TCP/IP stack. I didn't have a lot of time on my hands, so I accepted the first attractive offer. I selected Adam Dunkels's uIP embedded TCP/IP stack (www.sics.se/~adam/uip/). It wasn't the best choice, but it functioned well.

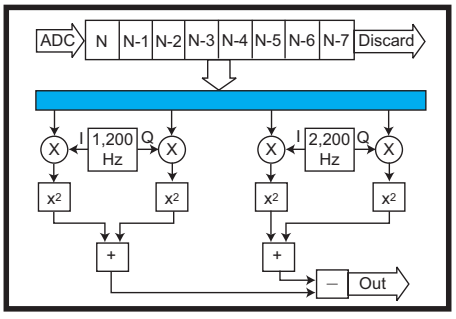


Figure 3—*Check out the FSK demodulator. The last eight received samples are processed through four independent filters. The I and Q components for each frequency are then squared and added together. The filter with the highest energy wins!*

The stack is targeted for processors that are generally much smaller than the LPC2138. It carries some short cuts that wouldn't be necessary for a 32-bit RISC processor such as the LPC2138.

Porting the stack was a straightforward task, except for the process of developing the Ethernet driver for the ENC28J60. Nevertheless, I quickly performed ARP resolution and a ping, and then established a TCP/IP connection in both Receive and Transmit modes.

The uIP stack came with example applications, but I had to modify them for the TAM-TAM. The SMTP mailer was the easiest to integrate. The TAM-TAM can be configured to automatically send an e-mail to a mailbox or to a cellular provider's SMS gateway. The body of a message contains the following sort of information:

Received at 14:14 on 22/10
From: 860-875-2199
Name: Circuit Cellar Inc.

The name and phone numbers are extracted from the caller ID frame. The time comes from the clock, which is automatically set with any valid caller ID frame that carries the date and time information. The information is available even when the caller ID is missed.

WEB SERVER

The uIP stack relies on a memory-based file system. Because the TAM-TAM includes a flash memory file system, I had to modify the web server to use the existing FAT system. The process ended up being more involved than I had initially anticipated.

The way in which web browsers access a web page is interesting. For example, when downloading a file, the web browser attempts to download the beginning of the file by looking for the Content-Length (size) header. It then aborts the file download and restarts it immediately. Although this process is easy to handle with a memory-based file system, it has to be handled carefully with the FAT system. The Content-Length header is added to the original web server, and it requires the addition of a FAT function to discover the size of a given file.

The web page I ended up displaying was simple. It shows a list of messages with the time, name, and phone number. Such messages are automatically downloaded and played when a user clicks on a hyperlink. This is one aspect of the TAM-TAM project where some work would improve the quality of the prototype.

The file streaming process can be slower than real time, and Windows doesn't always cache enough data to take this into account. The problem occurs because the queue isn't always full. An open file is read only when the file system is starving for data.

IMPROVEMENTS

It would be presumptuous to consider the TAM-TAM a finished product. Just making sure that all of the error conditions in the file system are handled correctly would be a huge task.

I would like to add a password on the web server login page and a DNS query. Currently, the SMTP mail server's IP address must be entered manually.

As for the hardware, I would like to integrate the wireless bridge inside the box. Wi-Fi cards are available in SD/MMC format, but it would probably be impossible to access the secret recipe to enable such a card.

The LPC2138 was definitely the right microcontroller for this project. I enjoyed its ability to run off the integrated flash memory and RAM without any wait states. The 60-MIPS budget allowed me to focus more time on the development of algorithms than on counting cycles in critical sections of code. Finally, for someone like me who hates polling, the interrupt handler is certainly one of the most flexible I have seen.

Author's note: I would like to thank Guy Grotke for reviewing my contest entry and this article.

Bernard Debbasch (bernard.debbasch@mail.com) has more than 20 years of experience in engineering and management (mostly in the semiconductor industry). He graduated from the Ecole Nationale Supérieure des Télécommunications de Bretagne in France and later moved to Southern California. In addition to his interest in electronics, Bernard enjoys jogging, flying model airplanes, and sailing with his wife, Christina.

PROJECT FILES

To download the code and additional files, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2006/190.

RESOURCES

A. Dunkels, uIP embedded TCP/IP stack, www.sics.se/~adam/uip/.

J. Randolph, "Low Cost Telephone Line Interface (DAA, FXO)," TN 98, Randolph Telecom, Inc., 2005, www.midcom-inc.com/Tech/tn98.asp.

T. Sailer, "DSP Modems," HB9JNX, 1995, www.baycom.org/~tom/ham/da95/d_dspmod.pdf.

Transformer and DAA information, Midcom, Inc., www.midcom-inc.com.

B. Watson, "FSK: Signals and Demodulations," WJ Communications, Inc. www.wj.com/pdf/technotes/FSK_signals_demod.pdf.

SOURCES

MCB2130 Development board
Keil Software, Inc.
www.keil.com

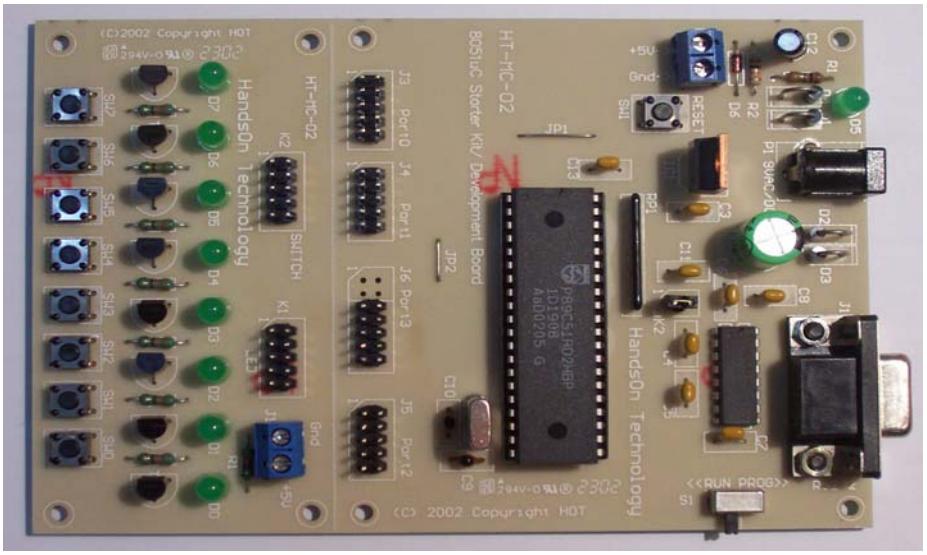
Elite Pro SD card
Kingston Technology Company, Inc.
www.kingston.com

ENC28J60 Ethernet controller
Microchip Technology, Inc.
www.microchip.com

LPC2138 Microcontroller
Philips Semiconductors, Inc.
www.semiconductors.philips.com

Low Cost 8051μC Starter Kit/ Development Board *HT-MC-02*

HT-MC-02 *is an ideal platform for small to medium scale embedded systems development and quick 8051 embedded design prototyping.* ***HT-MC-02*** *can be used as stand-alone 8051μC Flash programmer or as a development, prototyping and educational platform*



Main Features:

- 8051 Central Processing Unit.
- On-chip Flash Program Memory with In-System Programming (ISP) and In Application Programming (IAP) capability.
- Boot ROM contains low level Flash programming routines for downloading code via the RS232.
- Flash memory reliably stores program code even after 10,000 erase and program cycles.
- 10-year minimum data retention.
- Programmable security for the code in the Flash. The security feature protects against software piracy and prevents the contents of the Flash from being read.
- 4 level priority interrupt & 7 interrupt sources.
- 32 general purpose I/O pins connected to 10pins header connectors for easy I/O pins access.
- Full-duplex enhanced UART **–** Framing error detection Automatic address recognition.
- Programmable Counter Array (PCA) & Pulse Width Modulation (PWM).
- Three 16-bits timer/event counters.
- AC/DC (9~12V) power supply – easily available from wall socket power adapter.
- On board stabilized +5Vdc for other external interface circuit power supply.
- Included 8x LEDs and pushbuttons test board (free with ***HT-MC-02*** while stock last) for fast simple code testing.
- Industrial popular window ***Keil*** C compiler and assembler included (Eval. version).
- Free ***Flash Magic*** Windows software for easy program code down loading.

PLEASE READ ***HT-MC-02 GETTING STARTED MANUAL*** BEFORE OPERATE THIS BOARD
INSTALL ACROBAT READER (AcrobatReader705 Application) TO OPEN AND PRINT ALL DOCUMENTS