by Larry Martin

$1 Wireless Interface

Do you want a wireless interface for your next project? Got some wire? With a coil, a capacitor, and a transistor, you can make your next project emulate a radio frequency identification device (RFID), commonly called a "tag" or an "RFID tag." In this article, Larry shows you how.

The RFID industry has been in the news lately with big support from Gillette, Wal-Mart, and the U.S. Department of Defense. With that sort of heavy money lined up, the technology is poised to take off. The widespread adoption of RFID will have effects beyond the RFID industry itself. People are already discussing RFID tags as the leaf nodes in an "Internet of things," suggesting that RFID is also poised to generate a flood of data, driving the sale of routers, storage, and processing capacity in an effort to keep up. The Internet holds a growing body of RFID information. One good place to start reading is the *RFID Journal* (www.rfidjournal.com).

In general, RFID systems follow a simple model. A device usually called a "reader" creates a field. One or more tags communicate with the reader by varying the amount of energy reflected back to the reader by the tag. This process is called "backscatter," and it's the really neat thing about RFID technology because it enables extremely cheap devices to communicate wirelessly (see Figure 1).

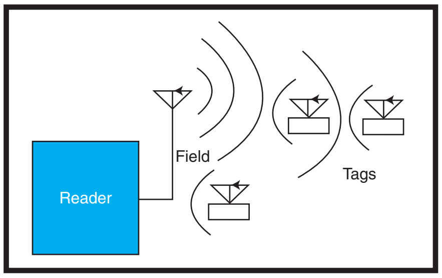

Figure 1—*The RFID system includes a reader with an antenna, a field, and several tags with antennas. The big arcs going to the right show the field generated by the reader, or the forward link. The smaller arcs going the left indicate the tags' replies, or the return link. If the tags are active tags, the return link will be radio signals driven by the tags. If the tags are passive tags, the return link will be forward link energy backscattered by the tags' circuitry, as this project illustrates.*

WHAT'S IN A TAG?

The general mechanism for creating backscatter is to detune a tuned element. The tuned element can be a resonant circuit or a dipole antenna, depending on the tag's frequency range (more on that later). The point is this: your tag can be tuned, or it can be detuned, without creating much of a stir. But when the tag changes from one state to the other, it creates a disturbance in the field. A well-timed series of such disturbances starts to look like a datastream. An RFID tag contains the minimum possible circuitry needed to produce that datastream: the antenna itself, a diode and capacitor to scavenge power from the field, another diode or transistor to detune the antenna, and a little intelligence.

There are many types of RFID tags. Some fill unique market niches; some embody really cool technology. Others are just there because somebody thinks they can make a buck on them. The different tag types can be divided in a few ways, the most useful of which are passive versus active, tag talk first versus reader talk first, and frequency range.

PASSIVE VS. ACTIVE

When people talk about passive tags, they usually mean tags that get their power entirely from the RF field generated by the reader. The same people use the term "active tag" to mean tags that get their power from a battery and use the reader's field only for communications. But that's not technically correct, and it fails to cover some of the most economically important devices. The correct usage of the terms is this: passive tags communicate by backscatter, so a tag can be passive even if it has a battery backup. Active tags do not use backscatter, but they incorporate low-power radios something like radar IFF systems. Of course, active tags need a battery or some other power source—they cannot scavenge enough power from the reader's field to run that radio.

California-based Savi Technology has an active tag, readers, and LAN protocol that together provide a complete vertical solution for tracking shipping containers. Savi's big application is military shipping containers, which were tried out in the first Gulf War. Now every container bound for Iraq and Afghanistan is identified with a Savi active tag. For another application, most RFID-based toll payment systems work with active tags.

At the moment, active tags are inherently more expensive than passive tags because of the battery. But

Range	Frequency
LF	125 and 135 kHz
HF	13.56 MHz
UHF	Approximately 915 MHz
MW	Approximately 2.4 GHz

Table 1—*RFID operates in four frequency bands. The bands are different outside of the United States.*

that gap is closing, too, because the difficulties of passive antenna design and bonding tag silicon to antenna take center stage as the most expensive aspects of producing tags.

The biggest commercial application for passive tags is car immobilization. The large plastic knob at the end of new car keys typically holds a passive tag. The embedded computer of cars equipped with this kind of system will not allow the car to be started or driven unless it sees data it recognizes on the key's RFID tag. Each car company has its favorite tag, but the market leader is Microchip Technology, the company that brings us the PIC processor. Another familiar application is the key fob that lets you buy gas and fast food without pulling out your credit card.

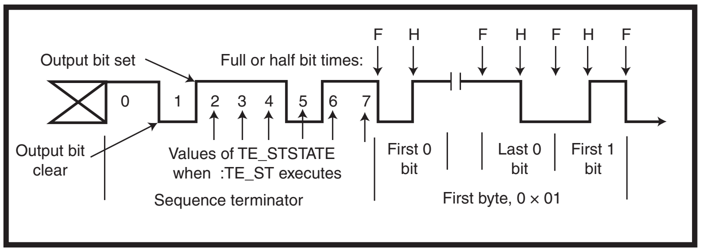

Figure 2—*Take a look at the sequence terminator and first byte of the sample message (hex 01). The bold line is the transistor base voltage (output on port B pin 4). When the output bit is set, the transistor conducts and the field is modulated. When the output bit is clear, the transistor shuts off and the field is not modulated. In Manchester encoding, the sequence terminator is a pair of on-off-on-on sequences, each 0 bit is an off-on sequence, and each 1 bit is an on-off sequence.*

WHO TALKS FIRST?

The RFID world has intermittent religious wars over which component of the system should talk first, the reader or the tag. Readers typically communicate with tags by creating timed gaps in the field. The gaps have to be long enough to be sensed at a distance, but short enough that tags do not lose power during the gap interval.

Most tags wait for a command—a series of gaps—from the reader before modulating the field. These are known as "reader-talk-first" tags. Some tags, however, start modulating as soon as they power up (or, in the case of active tags, as soon as they detect a reader's field). These are known as "tag-talk-first" tags. Each approach has its own strengths and weaknesses, but it's obvious that tag-talk-first is the simplest approach, and it's the one I used in this demo project.

FREQUENCY RANGE

In the Unite States, RFID operates in four frequency bands set aside for industrial, scientific, and medical (ISM) purposes (see Table 1). Although the terms have other meanings, within the RFID industry, the terms LF, HF, UHF, and Microwave are used as shorthand for the ISM bands used for RFID in those larger regions of the spectrum.

Outside of the United States, the bands differ slightly, mainly in the UHF range. European UHF is defined in the middle 800-MHz range. In Japan, UHF is higher in the 900-MHz range. Other countries also have different rules for hopping, duty cycle, power, and bandwidth.

LF and HF tags couple to the readers inductively, using coils as antennas and enabling the readers to be simple electrically. UHF and MW tags couple radiatively, requiring more complex reader electronics but providing more range in return.

e5551 AIR INTERFACE

You can add an extremely cheap wireless interface to your next project by making it act like an RFID tag. The Atmel e5551, which is a tag-talk-first passive tag, is the tag best suited to this application for a few reasons: as an LF tag, its inductive interface is easy to duplicate; it can run slowly for good range from a weak inductor; as one of the older tag protocols, many readers support it; because it is tag-talk-first, you don't have to decode traffic from the reader; and its air interface is posted on the 'Net by the manufacturer (many RFID protocols are available only under NDA).

The e5551's default behavior is to backscatter its data on the air interface whenever it is energized by the field from a reader. So, you can set up your hardware to continually backscatter the data you want to transmit. If a reader is there, the data will be transferred. If no reader is there, then you are not losing anything beyond a few compute cycles.

The e5551 has many configuration options. You don't have to worry about handling all of the configurations. Just pick one, and set up your system to emulate it.

Referring to the tag's configuration word in the e5551's datasheet, you'll want to emulate a Manchester tag with sequence termination running at 128 cpb. You can emulate a MAXBLOCK of 1 through 7 based on how much data you want to handle. In the language of low frequency tags, "cpb" stands for (carrier) cycles per bit. At 125-kHz carrier, each cycle takes 8 µs, so 128 cpb translates to 128 × 8 µs, or 1 ms, per bit. Why so slow? There are a couple of reasons.

First, a slow interface minimizes the effect on your project's compute cycles. Second, slow transitions on the air interface are easier to detect, which means you will have additional range and more reliable communications. Finally, the reader used to verify this design reads the Atmel tag configured this way by default, and it does-

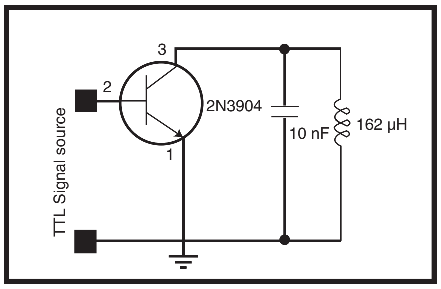

Figure 3—*This has got to be one of the simplest diagrams ever presented in an electronics magazine. The transistor type is unimportant, as long as it is an NPN device. The capacitor and coil together must resonate at 125 kHz. When the base voltage is low, the transistor is shut off, and the LC tank is free to resonate whenever an RFID reader is in the vicinity. When the base voltage is high, the transistor swamps the LC tank. Each transition between resonating and swamped states produces an edge in the reader's detection circuits. The edges are generally decoded in software to reconstruct the original message. The coil is hand wound from magnet wire (30 turns 2.5″ diameter, 0.05″ length).*

n't require set-up commands to do so.

I will skip over the question of how to modulate bits onto the air interface and go directly to the bit pattern you must create. I originally had planned to show an RS-232-to-RFID air interface converter that translates your keystrokes into RFID signals, but the overhead of that application tended to obscure the simplicity of the air interface. So, this article just shows the transmission of canned data (this sequence emulates a MAXBLOCK setting of 4):

0x01010101
0x02020202
0x03030303
0xdeadbeef

Simple, eh? Look for this pattern later in the RFID reader's output.

The key to understanding the e5551 air interface is understanding the data encoding method. This example uses the Manchester encoding method, which is illustrated in the e5551's datasheet. The tag is expected to modulate ("damp") the field in the first half of a bit time for a 1 and in the second half of a bit time for a 0. The second part of Figure 2 shows the signal sequence for the first data word, 0x01010101. Assume that the field is modulated when the MOD signal is high.

So, to send data, simply shift it onto the air interface, right? Well, not quite. If you are going to repeatedly send your data without being asked for it, you should provide some way for the listener to know when you are done with one round and starting another. The e5551 provides a sequence terminator (ST) for just this purpose. For RFID, the main feature of the ST is typically an intentional violation of Manchester timing. The first part of Figure 2 shows the ST sequence followed by the first data bits. If you send this bitstream into your antenna and put an RFID reader close to your antenna, you will see your data in the RFID reader's output.

That leaves only the question of how to modulate or damp the field. Well, LF and HF tags both work through inductance. The air interface

Listing 1—*This is the most interesting part of the Scenix program.*

```
;; TagEmulator
TagEm        =   serial

TE_STState   ds  1        ; TagEmulator Sequence Terminator State
TE_BBState   ds  1        ; TagEmulator Bit Boundary state:  0 = full
                          ; bit, ff = half bit
TE_ClkCnt    ds  1        ; TagEmulator Clock Interrupt Count
TE_BitCount  ds  1        ; TagEmulator bit count 0..7 in byte
TE_ByteAcc   ds  1        ; TagEmulator Byte Accumulator, copy from
                          ; data and shift out
TE_BytePtr   ds  1        ; TagEmulator Byte Pointer, offset from
                          ; start of tag data
TE_tmpds     1            ; TagEmulator temp byte
TE_OutBit    =   rb.4
;; end TagEmulator
  org 0
;
;
; Interrupt routine - virtual peripherals
;
interrupt bank timers    ;1
  ;; TagEmulator:interrupt every 3.12 uS for 19.2 kbaud soft UART
  bank      TagEm
  decsz     TE_ClkCnt
  jmp           :TE_Done
:TE_512Us
  ;; set up TE_ClkCnt for next half-bit: 157 interrupts * 3.12 uS
  ;; equals 512 uS
  mov           W,#157
  mov           TE_ClkCnt,w
  ;; Are we doing the ST or not?
  mov           w,TE_STState
  mov           TE_tmp,w
  mov           w,#8
  sub           TE_tmp,w ; subtract W from FR
  snb           C        ; carry set for FR values 0..8 (nine states)
  jmp           :TE_Bit
:TE_ST
  ;; select action based on Sequence Terminator State 0 through 7
  ;; a jump table would save a few words here
  mov           w,TE_STState
  mov           TE_tmp,w
  snb           Z
  setb      TE_OutBit    ; 0
  dec           TE_tmp
  snb           Z
  clrb      TE_OutBit    ; 1
  dec           TE_tmp
  snb           Z
  setb      TE_OutBit    ; 2
  dec           TE_tmp
  snb           Z
  setb      TE_OutBit    ; 3
  dec           TE_tmp
  snb           Z
  setb      TE_OutBit    ; 4
  dec           TE_tmp
  snb           Z
  clrb      TE_OutBit    ; 5
  dec           TE_tmp
  snb           Z
setb TE_OutBit           ; 6
dec   TE_tmp
snb   Z
setb TE_OutBit           ; 7
dec   TE_tmp
snb   Z
setb TE_OutBit           ; 8
;; set up for next time, first byte in case this is the last time
inc   TE_STState
clr   TE_ByteAcc
clr   TE_BytePtr
```

(continued)

Listing 1—*Continued.*

```
dec   TE_BytePtr         ; to -1
clr   TE_BBState
not   TE_BBState         ; to FF
mov   W,#8
mov   TE_BitCount,W      ; trick byte read logic into getting the
                         ; first byte
jmp   :TE_Done
:TE_Bit
;; Are we on half or full bit boundary?
not   TE_BBState
sz
jmp   :TE_HalfBit
:TE_FullBit
;; get next bit
inc   TE_BitCount
sb   TE_BitCount.3       ; quick test for 8
jmp   :TEFB_ShiftBitAcc
:TEFB_RefreshBitAcc
;; get the next byte from program memory
inc   TE_BytePtr
mov   w,#_RFID_Data
add   W,TE_BytePtr
xor   W,#_RFID_DataEnd
sz
jmp   :TEFB_GetNextByte
;; we've hit the end of data, double back to the ST code
clr   TE_BytePtr
clr   TE_STState
jmp   :TE_ST
:TEFB_GetNextByte
mov   m,#0
mov   w,#_RFID_Data
add   w,TE_BytePtr
IREAD                    ; get instruction word at m:W, store in m:W
mov   TE_ByteAcc,W
clr   TE_BitCount
jmp   :TEFB_BitAccIsSetUp
:TEFB_ShiftBitAcc
clrb  C
rl   TE_ByteAcc
:TEFB_BitAccIsSetUp
;; shift it out
sb   TE_ByteAcc.7
clrb TE_OutBit
snb   TE_ByteAcc.7
setb TE_OutBit
jmp   :TE_BitDone
:TE_HalfBit
sb   TE_ByteAcc.7
setb TE_OutBit
snb   TE_ByteAcc.7
clrb TE_OutBit
:TE_BitDone
:TE_Done

[snip Soft Peripheral code]

:rxdone
  mov w,#-163            ;1  ;interrupt every 163 clocks, 3.26 uS:
                             ;serial baud rate depends on this
  retiw                  ;3
;
; Data
;
_hellodw     13,10,13,10,'SX Virtual Peripheral Demo',13,10,0
_prompt      dw  13,10,'>',0
_errordw     'Error!',13,10,0
_hex         dw  '0123456789ABCDEF'
_RFID_Data   dw  1,1,1,1, 2,2,2,2, 3,3,3,3, $de,$ad,$be,$ef
                         ; TagEmulator
_RFID_DataEnd   dw 0     ; TagEmulator
```

is basically a loosely coupled transformer with the reader's antenna as the primary and each tag's antenna serving as a secondary winding. (Yes, there can be more than one tag present in some protocols. The e5551 supports that to some extent, but it's too much to get into at this point.) Your output circuit is a parallel LC tank circuit tuned to 125 kHz. To support modulating the field, place a transistor across the tank. When the transistor is off, the circuit is tuned. When the transistor is on, the circuit is detuned, and the air interface is damped.

Figure 3 shows the general scheme with typical component values. With all the elements in place, you're ready to look at the project.

THE SAMPLE PROJECT

The signal input in Figure 3 is labeled "TTL Signal Source." I used a Scenix SX28 prototype board. Listing 1 shows the interesting part of the Scenix program. Most of the listing is the sample program provided with the SX28 development kit. The TagEmulator string sets off everything added to this project. You may download the full source code from the *Circuit Cellar* ftp site.

The Scenix processors are interesting because they have few dedicated peripherals. Scenix (now Ubicom) designers made a decision to provide a fast chip with a lot of I/O, so that peripherals could be implemented in software rather than hardware. With clock rates up to 100 MHz, the performance of Scenix processors approaches that of programmable logic. For example, the program as posted still includes the software UART, which bit-bangs RS-232 data through any I/O pins you want to use for that purpose.

The program's canned output string is at label `_RFID_Data`. The ISR (label `:TE_512Us` through `:TE_Done`, where `TE` stands for tag emulator) gets output bytes from program memory and shifts bits out to the air interface in 512-µs intervals. The interval timing is controlled by the two lines at `:rxdone`, which push the immediate value `-163` into the real-time clock at the end of the interrupt. That value leads to an interrupt every 3.12 µs, which is important for the software UART timing. To generate the 512-µs intervals for the air interface, code at `:TE_512Us` checks the value of `TE_ClkCnt`. When that register counts down to zero, you are at a 512-µs boundary and ready to handle the air interface. One of the first things the code does is reset `TE_ClkCnt` to 157, because 157 × 3.12 µs is as close to 512 µs as possible.

The Scenix board has two outputs, a circuit common and the transistor's base. The base signal is on port B, bit 4. When that signal is low, the transistor is off and the LC tank resonates at its natural frequency. Note that there is no power connection to the tuned circuit. This interface takes almost no power from your project, and it adds noise only when an RFID reader is close by. When the transistor base signal is high, the transistor conducts, swamping the tuned circuit. As you know, neither state is particularly memorable. But if you carefully change states at the right time intervals, you can create data, which is exactly what happens in Photo 1.

With the tag emulator circuit running, an RFID reader placed nearby detects and displays your data. Listing 2 is the output from the SAMSys LF reader used to verify my prototype.

Listing 2—*The output from the SAMSys LF reader is used to verify the prototype.*

```
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
{Rd,d:0101010102020202030303DEADBEEF,t:T128M;04
```

NEXT STEP

Where to go from here? Well, you could easily modify the sample program to run, say, RS-232 data to the air interface. You could experiment with different data lengths and formats, or you could cycle your data so that everything you want to say is presented over a set of reader output lines. As you experiment, you may notice that the data coming out of the RFID reader is sometimes different from what you had intended. Most RFID tags implement a CRC to keep that from happening. One of the great things about the e5551 is that the tag enforces no CRC, so you can implement readers (and they can implement tags) fairly easily. The downside is that your data has no error protection. Implementing error protection at the application layer would be another fun project. The real adventure will be implementing the forward link, but that will require hardware changes.

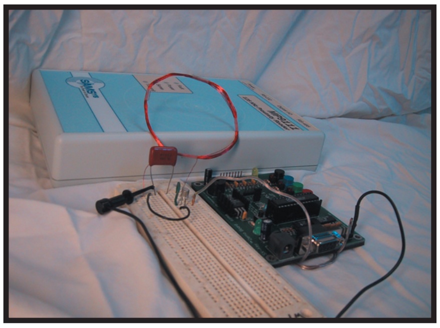

Photo 1—*The RFID reader was used to read the tag producing the output shown in Listing 2. The circuit board is the Scenix SX28 demo board by Parallax. That board is the TTL signal source. Black wires carry common. The clear speaker wire provides base voltage from Scenix rb.4 to the transistor. The green wire carries collector current from the top of the LC tank.*

Larry Martin is an embedded system programmer in Research Triangle Park, North Carolina. He got into embedded systems after working for 10 years as an electronics technician. He has worked in the nuclear, medical, aerospace, and telecom industries, and is now the senior programmer for RFID startup SAMSys Technologies. You may contact Larry at Larry@GlueLogix.com.

PROJECT FILES

To download the code, go to ftp.circuitcellar.com/pub/Circuit_Cellar/2004/163.

RESOURCE

Atmel Corp., "e5551: Standard R/W Identification IC with Anticollision," rev. A3, October 2000.

SOURCES

e5551
Atmel Corp.
(408) 441-0311
www.atmel.com

LF reader
SAMSys Technologies, Inc.
(905) 707-0404
www.samsys.com

Scenix SX28 Prototype board
Ubicom, Inc.
(650) 210-1500
www.scenix.com